powerGate

coolOrange s.r.l

Mar 20, 2024

POWERGATE

1 Installation **1**
- 1.1 Requirements . . . 1
- 1.2 Setup . . . 1
- 1.3 After the Setup . . . 2
- 1.4 Install Locations . . . 2
- 1.5 Updates . . . 2
- 1.6 Uninstall . . . 3

2 Activation and Trial limitations **5**
- 2.1 Trial limitations . . . 5
- 2.2 Activation . . . 5
- 2.3 License Information . . . 5
- 2.4 Command-line . . . 6
- 2.5 Licensing Options . . . 6

3 Getting Started **7**
- 3.1 Using the powerGate Cmdlets . . . 7
- 3.2 Using the powerGate .NET library . . . 9
- 3.3 Demo ERP system . . . 11
- 3.4 View ERP data in tabs . . . 12
- 3.5 Transfer ERP data manually with tabs . . . 13

4 Connecting Autodesk & ERP **15**
- 4.1 Sample.ConnectToERP . . . 15
- 4.2 Sample.Tab-File-ErpBom . . . 16
- 4.3 Sample.Tab-File-ErpItem . . . 19
- 4.4 Sample.Tab-Item-ErpBom . . . 21
- 4.5 Sample.Tab-Item-ErpItem . . . 24
- 4.6 ERP integrations . . . 27
- 4.7 Errors . . . 30

5 BOM Window **35**
- 5.1 Customization . . . 35
- 5.2 Status . . . 38
- 5.3 BOM Tab . . . 39
- 5.4 Item Tab . . . 40
- 5.5 Errors . . . 41

6 Code Reference **43**
- 6.1 Cmdlets . . . 43
- 6.2 .NET Library . . . 105

6.3 UI Components . . . 146

7 Logging **151**
7.1 Log requests and responses . . . 151
7.2 LogFile . . . 153
7.3 PowerShell IDE . . . 153

8 Change logs **155**
8.1 powerGate v24 . . . 155
8.2 powerGate v23 . . . 161
8.3 powerGate v22 . . . 165
8.4 powerGate v21 . . . 166
8.5 powerGate v20 . . . 169
8.6 powerGate v19 . . . 171
8.7 powerGate v18 . . . 175
8.8 powerGate v17 . . . 176
8.9 powerGate 2016 . . . 183

9 Features **185**
9.1 Pull and Transfer material information . . . 185
9.2 Pull project information . . . 186
9.3 Transfer BOM to the ERP system . . . 186

CHAPTER

ONE

INSTALLATION

1.1 Requirements

As powerGate allows to automate data synchronization between Vault and ERP systems, the Vault system requirements defined by Autodesk leads.

Operating System: 64-bit only

- Microsoft Windows 10
- Microsoft Windows 11

.NET Framework: 4.7 or higher

Windows PowerShell: PowerShell 4.0 or higher

1.1.1 Workstations

Autodesk Vault Client: 2024 / 2023 / 2022 / 2021

- Vault Professional

coolOrange powerJobs Client: powerJobs Client is installed automatically *(optional)*

- powerEvents is needed to easily realize integrations between Vault Client, Inventor and the ERP system (see *ERP integration sample*).

1.1.2 Job Processor

coolOrange powerJobs Processor: powerJobs Processor *(optional)*

- Needed to perform automated data synchronization tasks between Vault and ERP via the Job Processor.

1.2 Setup

The powerGate setup is delivered as an executable and accepts the standard windows installer arguments.
To accept the products EULA when starting the setup in silent mode pass the ***ACCEPT_EULA=1*** argument.
This installs all components needed on a *workstation* by default:

```
"\\path\to\networklocation\powerGate24.0_Vault2024.exe" -silent ACCEPT_EULA=1
```

To install only the main components (*Cmdlets* and *.NET Library*) on the *Job Processor* machine, pass the ***MAIN_COMPONENTS_ONLY=1*** argument.
This will NOT install any sample files and no *powerJobs Client*:

```
"\\path\to\networklocation\powerGate24.0_Vault2024.exe" -silent ACCEPT_EULA=1 MAIN_
↪COMPONENTS_ONLY=1
```

1.3 After the Setup

After installing powerGate on a development environment, the provided *sample files* can be used to implement your own Vault ERP integration.
It is recommended to disable all PowerShell scripts that are not used and have been replaced with your own version.

1.4 Install Locations

powerGate is installed in the following locations on your system:

- The Cmdlets will be installed to *C:\Program Files\coolOrange\Modules\powerGate*
- All PowerShell scripts and XAML files relevant for the ERP integration are placed in *C:\ProgramData\coolOrange\Client Customizations* (only on *workstations*)

Following shared libraries are installed in *GAC*:

- powerGate.Erp.Client.dll
- coolOrange.Logging.dll

Following shortcuts are added in the start menu:

- **powerGate Console** - Opens the PowerShell Console and loads the powerGate module
- **powerGate Information** - Opens the About dialog with product related information
- **powerGate License Information** - Opens the License Information dialog to activate the product
- **powerGate Logs** - Opens the log file location

1.5 Updates

To install a newer version of powerGate on *workstations*, just execute the setup file of the new version.
ERP integrations will then continue to work as usual, with all delivered *sample files* also beeing updated (see *C:\ProgramData\coolOrange\Client Customizations\Disabled* folder).
Improvements in these files will automatically take effect only for those PowerShell scripts that were also previously enabled.

Note: On the *Job Processor*, Vault Workgroup (unsupported) and Vault 2020 or older environments, powerGate must be updated with the previously used `MAIN_COMPONENTS_ONLY=1` argument.

Please note that for such installations that contain only the main components (*Cmdlets* and *.NET Library*), future updates can also be performed **only with this argument**!
Otherwise the update will be prevented by the installer. A full installation on workstations is then unfortunately only possible by *uninstalling* the old version before installing the new one.

1.6 Uninstall

In case you want to remove powerGate from your computer you can:

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

CHAPTER

TWO

ACTIVATION AND TRIAL LIMITATIONS

2.1 Trial limitations

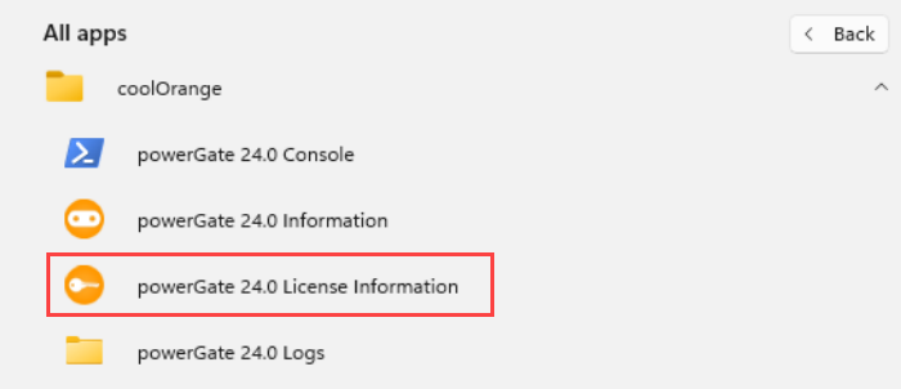
There is no difference in functionality between the trial version and the fully licensed product.
After the installation the product is available as a trial version for 30 days.
During this time, the sample *Vault ERP Integration* can also be tested against our *Demo ERP* and all your test data will be retained.

2.2 Activation

The product can be activated during or after the trial period.
For manually activating the product following Dialog can be used:

2.3 License Information

Open the Start Menu and navigate to "*All Apps → coolOrange → powerGate 24.0 License Information*" shortcut.



2.4 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\powerGate\License.exe" --StandAlone --Serialnumber=
↪"XXXXX-XXXXX-XXXXX-XXXXX"
```

For more information about activating the product, see Licensing.

2.5 Licensing Options

2.5.1 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, powerGate will show a windows notification and the *ERP Integration* notifies about the license error within the Connection Error Dialog, BOM Window Error statuses and powerEvents restrictions.
The *Connect-ERP* cmdlet and the *IErpClient.ConnectErp* function throw a *LicenseException* when attempting to connect to an OData server.
Any subsequently executed *ERP cmdlets* or *.NET Library* functions will fail and prevent the user and configured automatisms from working with the ERP.

This also applies if the OData service is a powerGateServer.
In this case, all Vault workstations will be informed about the **expired powerGateServer** license.

2.5.2 Offline activation

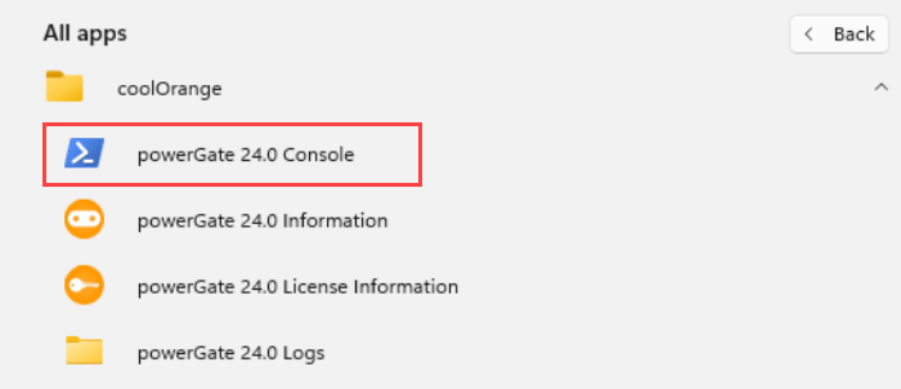
The serial number of the license and the machine code are required to generate an activation file.
The activation file for an offline activation can be generated and downloaded on the following site: powerGate - Activation file generator

CHAPTER

THREE

GETTING STARTED

3.1 Using the powerGate Cmdlets



3.1.1 Start the PowerShell environment

In order to get started either open any PowerShell IDE and load the powerGate Module by calling `Import-Module powerGate` or open the *powerGate Console* shortcut in the start menu, which already loads powerGate for you.

3.1.2 Connect with ERP system

Before you are able to work with your ERP-System you have to connect to powerGateServer or directly your ERP system (if it support's an OData interface).
This can be done by calling *Connect-Erp*.

Some public accessible OData-Services for testing can be found here.

3.1.3 Get multiple entities from ERP

After you are connected, you want to work with entities. In order to get the entities you have to know the appropriate **EntitySet**.
The list of available EntitySet's can be retrieved with the *Get-ERPEntitySets* Cmdlet.

Note: If you want to get the names of the EntitySet's you can access them via *$entitySet.Name*

Now you know the name of the different entitiyTypes, therefore you can finally get some entities with Cmdlet *Get-ErpObjects*.

Note: *Get-ErpObjects* has many optional parameters like **-Top -Filter -Expand**. See detailed documentation for them.

3.1.4 Get a specific entity from ERP

In order to get a specific entity you have to know their key properties which identifies them.
The metadata information for the **key properties** of the entity can be retrieved using following:

```
$itemsEntityType = Get-ERPEntityTypes -EntitySet $entitySet.Name
$keyProperties = $itemsEntityType.Keys
```

First set up the keys for your entity as a Hashtable:

```
$keys = @{ 'Number'='100001' }
```

Execute *Get-ErpObject* with the mandatory arguments EntitySet and Keys.

```
$entity = Get-ERPObject -EntitySet $entitySet.Name -Keys $keys
```

3.1.5 Update an existing entity

In case you want to change some values of the entity you can achieve this with the *Update-ErpObject* Cmdlet.
First setup a Hashtable, with the values you want to update.

Note: **Key** properties can **NOT** be modified!

```
$updatedProperties = @{ 'Material'='uranium'; 'UnitOfMeasure'='Bq' }
```

Execute the Cmdlet with the entity-keys and properties and it returns the updated entity.

```
$updatedEntity = Update-ERPObject -EntitySet $entitySet.Name -Keys $keys -Properties
↪$updatedProperties
```

3.1.6 Add an entity to ERP

The *New-ERPObject* cmdlet allows you to create a new and empty instance of the required EntityType and pre-fill it with data that can be passed to the *Add-ERPObject* to create a new entity in ERP.

```
$newEntityProperties = New-ERPObject -EntityType $itemsEntityType.Name -Properties @{
↪'Number'='100002'; 'Material'='Einsteinium'; 'UnitOfMeasure' = "Bq"}
Add-ERPObject -EntitySet $entitySet.Name -Properties $newEntityProperties
```

3.1.7 Upload file to ERP

You may want to attach/link a Pdf file to an entity. This operation is called *Add-ErpMedia*.
It will create the entity on the server side and upload and link the file with the newly created entity.

3.2 Using the powerGate .NET library

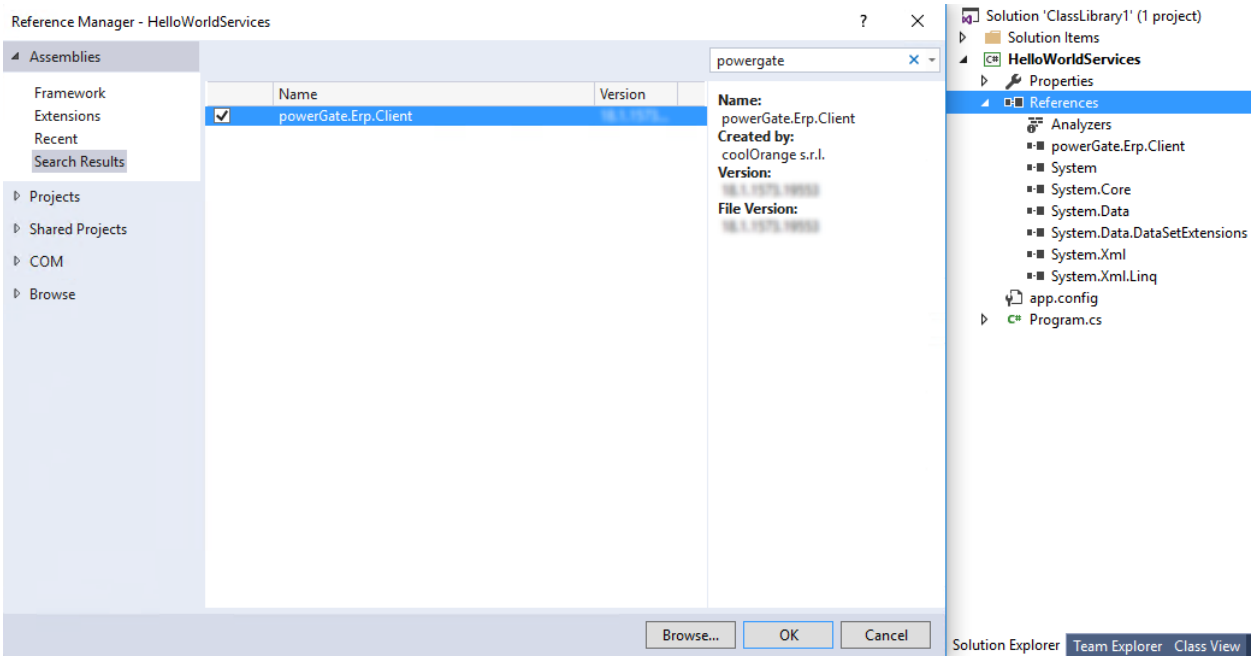
3.2.1 Follow these steps to use the powerGate library in your C# project

With the *installation* of powerGate on your development machine, the required .NET library will be installed for all users.
It contains the relevant API's to communicate with an ERP System via OData.

The powerGate .NET library requires your project to target at least **.NET framework v4.7**!

1. Reference the *powerGate.Erp.Client* assembly

In Visual Studio right-click on *References* and click "*Add References*".
Search for the assembly "*powerGate.Erp.Client*" in Assemblies-tab and add it to your project.



The assembly will be referenced from the **GAC**, therefore set *"Copy Local"* to *"false"* (when using Visual Studio 2017 this should be done automatically).

2. Create an ErpClient instance and connect to a service

In order to gain access to the powerGate API's, the following **namespace** must be imported:

```
using powerGate.Erp.Client;
```

Root entry point is the class *ErpClient*:

```
var client = new ErpClient();
```

Call the **ConnectErp()** method in order to have access to the prefered Service:

```
var northwindService = client.ConnectErp(new Uri("http://services.odata.org/V4/Northwind/
↪Northwind.svc/"));
```

3. Communicate with the service

If we have successfully connected to a *Service (service.Available will return true)*, then we are able to make CRUD operations on the prefered *EntitySet*.

See the following example with a **single get**, which retrieves exactly one entry:

```
var categories =  client.Services["Northwind.svc"].EntitySets["Categories"];
var categoryKeys = new Dictionary<string,object> { {"CategoryID", 3} };
var category = categories.GetErpObject(categoryKeys);
Console.WriteLine("Successful retrieved category '{0}' with Id '{1}'.",
category["CategoryName"], (int)category["CategoryID"]);
```

4. Release the service when done

When you are done working with your ERP System, the *ErpClient* should be **disposed**, in order to disconnect from all the connected *services* and release all the resources.

```
client.Dispose();
```

We recommend using the "using" statement on the *ErpClient*, so *Dispose()* will be called in each situation, also when unexpected exceptions are thrown!

Install powerGate on customer machine

When shipping your projects binaries to your customer, **also the customers machine requires a powerGate installation**.
Therefore delivering the *powerGate.Erp.Client* assembly within your project should be avoided, in order to not lose the benefits from powerGates *Update strategy*.

See the **complete example**:

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace HelloWorldServices
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var client = new ErpClient())
            {
                var northwindService = client.ConnectErp(new Uri("http://services.odata.
↪org/V4/Northwind/Northwind.svc/"));
                var categories = client.Services["Northwind.svc"].EntitySets["Categories
↪"];

                var categoryKeys = new Dictionary<string, object>{ {"CategoryID", 3} };
                var category = categories.GetErpObject(categoryKeys);

                Console.WriteLine("Successful retrieved category '{0}' with Id '{1}'.",␣
↪category["CategoryName"], (int)category["CategoryID"]);
            }

            Console.ReadLine();
        }
    }
}
```

3.3 Demo ERP system

For evaluation and demo purposes public web services are used by the delivered sample *Vault ERP Integration*, where **all Vault users that are connected to the same Vault** on the same Vault Server (ADMS) **share the same test data**.

Testdata will be removed

As this ERP system should only be used for demo and testing purposes, the stored data **will be be removed** after about 30 days.

The following OData service is based on the powerGateServer ERP plugin and are publicly accessible:

- DemoService - `http://demo.powergate.online/PGS/ERP/DemoService`

Requirement

The following domain needs to be accessible on TCP port 80 *'demo.powergate.online'*

3.3.1 Testing

This demo system can be used to try out the sample *Vault ERP Integrations*, the *.NET Library* or the *Cmdlets*:

1. Open the *powerGate Console* shortcut
2. **Connect to the Vault** for which you want to start a new test session:

```
Import-Module powerVault
Open-VaultConnection -Server '<Your Vault Server>' -Vault '<Your Vault name>' -User
↪'Guest' ...
```

3. **Connect to the Demo ERP system** using the provided `$demoErpConnect` variable, which enables a new test session for your Vault:

```
Connect-ERP -Service 'http://demo.powergate.online/PGS/ERP/DemoService' -OnConnect
↪$global:demoERPconnect
```

4. Retrieve the available entity sets:

```
Get-ERPEntitySets
<#
 Service                                          Name       EntityType
 -------                                          ----       ----------
 http://demo.powergate.online/PGS/ERP/DemoService Items      Item
 http://demo.powergate.online/PGS/ERP/DemoService BomRows    BomRow
 http://demo.powergate.online/PGS/ERP/DemoService BomHeaders BomHeader
#>
```

If you are new to powerGate, topics in this section will help you quickly start getting the best of the product:

- *Using the Cmdlets*
- *Using the .NET library*

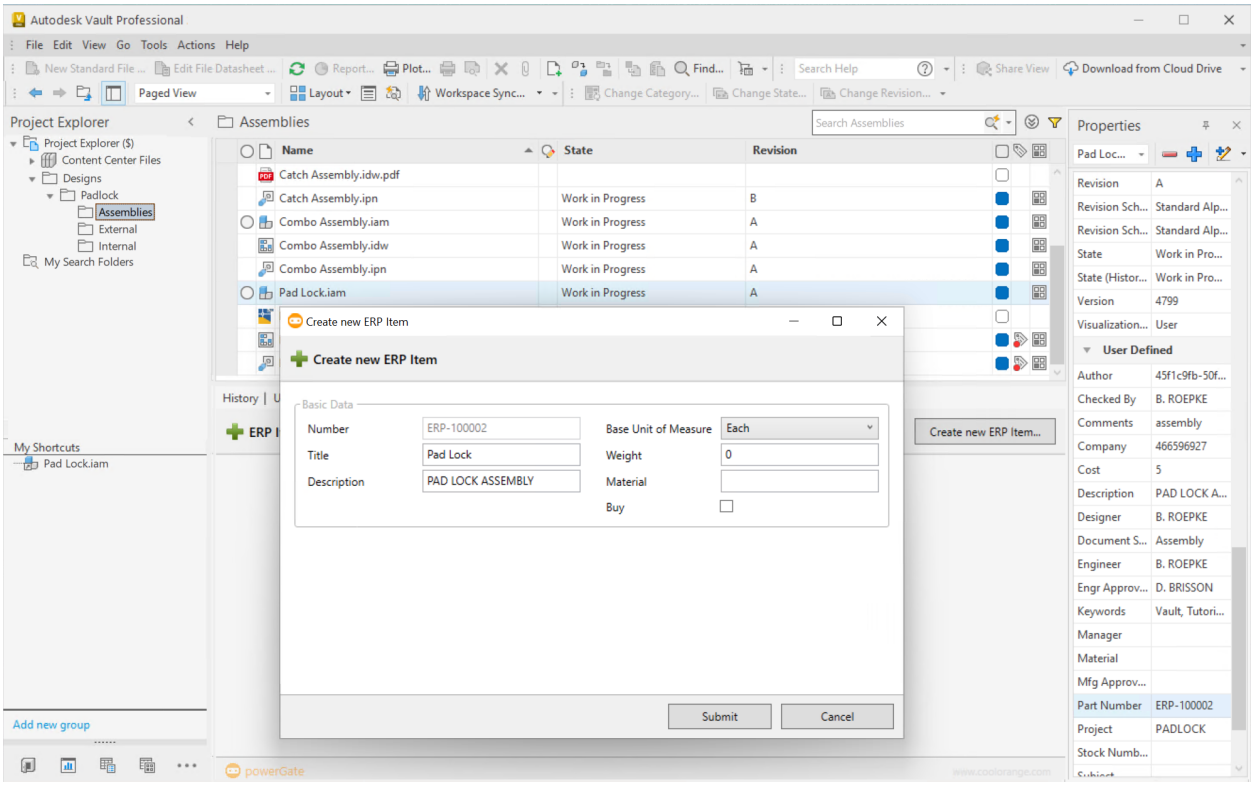
For evaluation purposes, working samples of *ERP tabs* are delivered, which connect to a public *Demo ERP system*.
All you need to try it out is an **internet connection**. Also, **no special Vault permissions** are necessary as the tabs themselves do not change any data in your production Vault.
For 30 days, your **test data will remain secured** in our *Demo ERP* just for you and for other employees who also log into your Vault.

3.4 View ERP data in tabs

Open the Vault Client and login to your Vault. Afterwards select an Inventor file within the Vault Client.

Activate the tab “ERP Item”, which automatically connects to the Demo ERP system and displays the current data for this file.
As its used for the first time currently no item exists for the selected file in the Demo ERP system.

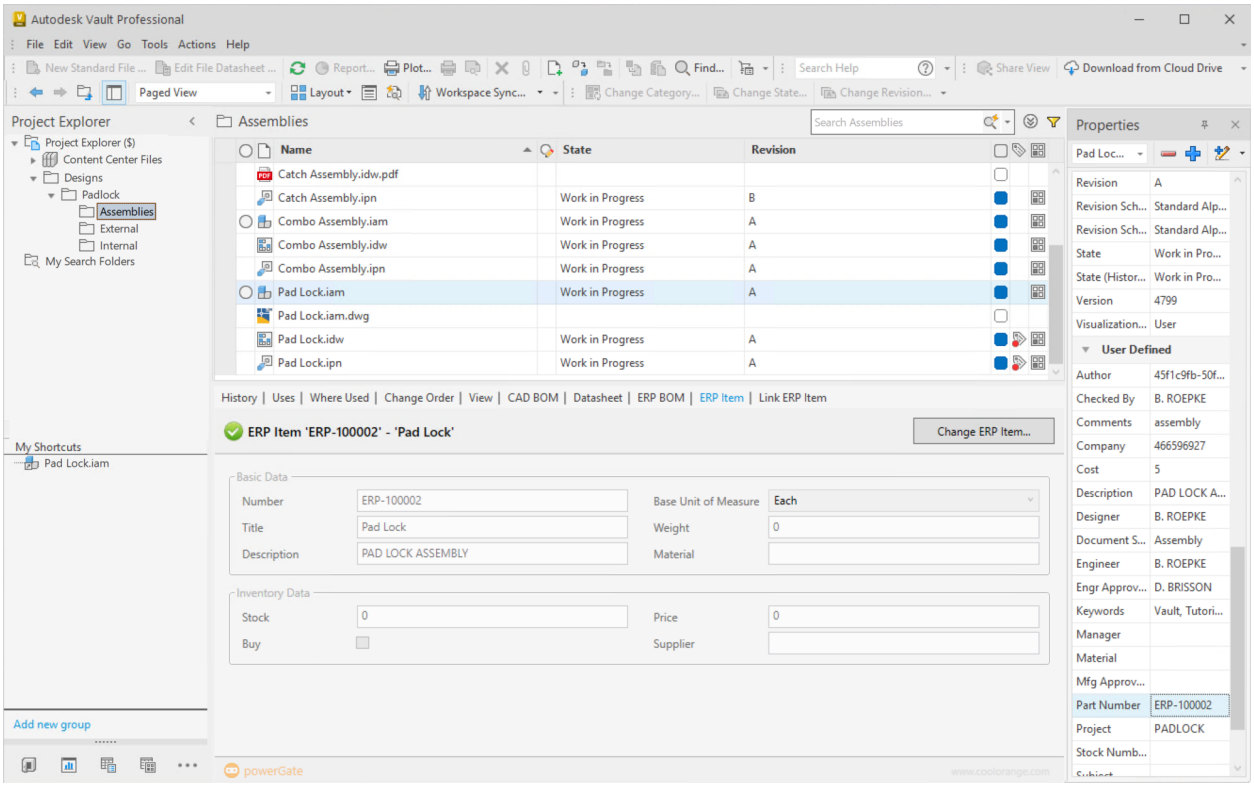
You can now click on the “*Create new ERP Item...*” button which opens a dialog with pre-filled data from the file and the supplied default configuration:



Similarly, also the additional “ERP BOM” tab indicates that no BOM exists in the ERP system for the selected file.

3.5 Transfer ERP data manually with tabs

After you have entered all the required data, click on the “Submit” button to create a new item in the ERP system. If the creation is successful, the tab displays the data of the new ERP Item.



The ERP Item can be changed simply by clicking the "*Change ERP Item...*" button, which opens a dialog for modifiying the data.

Similarly, the "ERP BOM" tab provides access to the *BOM Window* for the current selection. It allows to view, create and update the BOM and the items of the BOM in the ERP system.

Note: The two previously mentioned tabs are by default also available for Vault items and can be accessed by selecting an item in the Item Master view.

CHAPTER

FOUR

CONNECTING AUTODESK & ERP

4.1 Sample.ConnectToERP

This client customization shows how to automatically connect to all *configured ERP Services* when Vault users log into Vault from within the Vault Client or Inventor.
It adds a menu item in the Tools menu of the Vault Client to open the *ERP Integration Settings* dialog, where these connection settings can be viewed and changed.

Evaluating sample ERP features is possible out-of-the-box. No manual configuration in the Vault is required for this, as our public *Demo ERP system* is used by default.

4.1.1 Modifications

To modify this script to your needs, it is recommended to create a copy of the sample script, customize it accordingly and then disable this sample script.

The sample script registers to the `LoginVault_Post` event and tries to connect to all configured ERP services using *`Connect-ERP -UseSettingsFromVault`*.
But also after saving changes in the ERP Integration Settings dialog, this cmdlet is invoked again.

The following script section can be adapter if connections to these services might not be supported by default or additional connection procedures are required:

```
function ConnectToConfiguredERPServices(){
   Disconnect-ERP

   # multiple SAP Gateway services are configured as 'Direct connection to an OData-capable ERP'
   Connect-ERP -UseSettingsFromVault -OnConnect {
      param($settings)

      $settings.Credentials = New-Object System.Net.NetworkCredential('EX_DEMO', 'secret')

      if($settings.Service.EndsWith('/sap/opu/odata/arcona6/MATERIAL_SRV')) {
         $global:sapConnect.Invoke($settings)

         # Other custom -OnConnect logic ...
      }elseif(...){ ... }
   }
}
```

4.1.2 Disabling the script

After the *installation* on new environments this script is **enabled** by default for the Vault Client and Inventor.
To globally disable this client customization, move the script file to the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.

Note: Please note that after **updates** from v23.0.10 and older, this script is automatically disabled and installed in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.
Scripting guys can then enable and review this customization in a test environment before deploying the script in production ERP integrations.

4.2 Sample.Tab-File-ErpBom

The "ERP BOM" tab from this sample script allows the designer to select a file in the Vault Client and immediately see if a BOM already exists in the ERP system.
If it does, all the position numbers, item numbers, quantities, and other relevant BOM row details are displayed live from the ERP system.
With the *BOM Window* it is possible to automatically generate and update all required materials and BOMs in the ERP system.

Requirements

This Vault Client customization is designed to be used with the "*DemoService*" from the ***Demo ERP system***:

- An **internet connection** is required and the domain *'demo.powergate.online'* needs to be accessible on TCP port 80.
- An entity set named "**BomHeaders**" is expected as well as the entityset **"BomRows"** and **"Items"**
- Vault files are uniquely assigned to an ERP BOM via their item number stored in the Vault property **"Part Number"** and the ERP field "ParentNumber".
- Individual BOM rows are also identified by the item number, but their **position in the CAD BOM** and the ERP BOM "Position" must also match.

This tab can also be activated at any time when you are already using your own ERP integration.

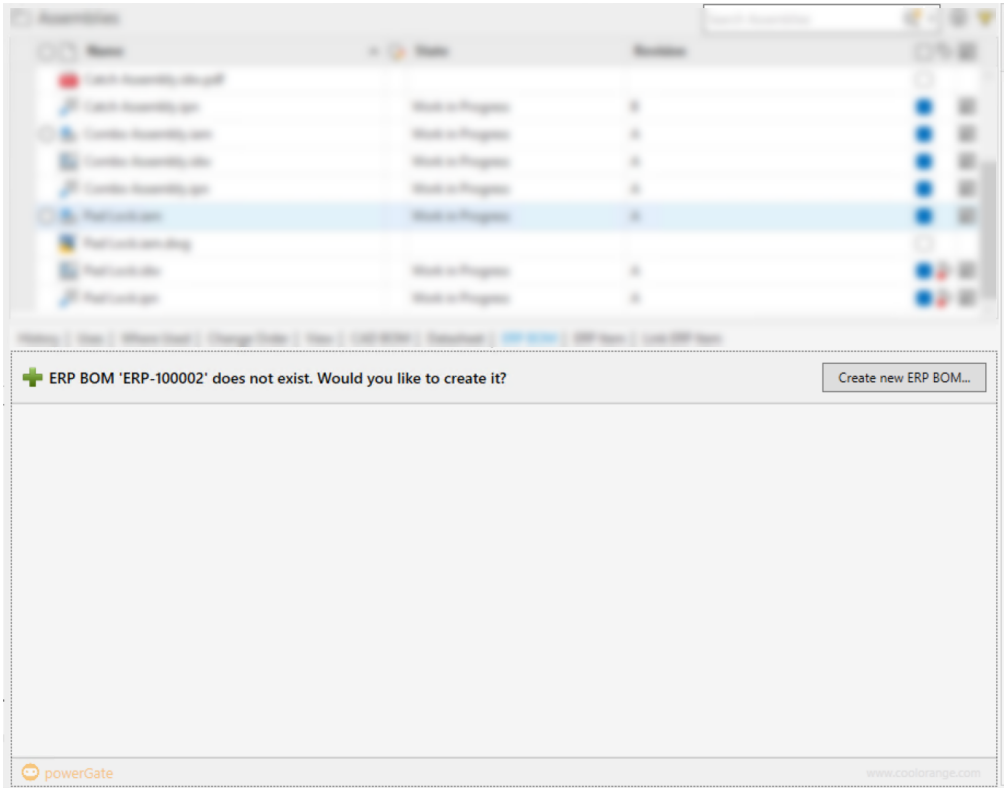
In the *ERP Integration Settings* dialog, it is possible to specify from which *Vault Properties* (or *fixed-values*) the data of created Items comes from (see type-mapping *Vault 'File' -> ERP 'Item'*).
Additionally, also the possible values for the ERP field *"UnitOfMeasure"* are retrieved from this configuration (see ERP field *'UnitOfMeasure' -> list values*).

4.2.1 Testing

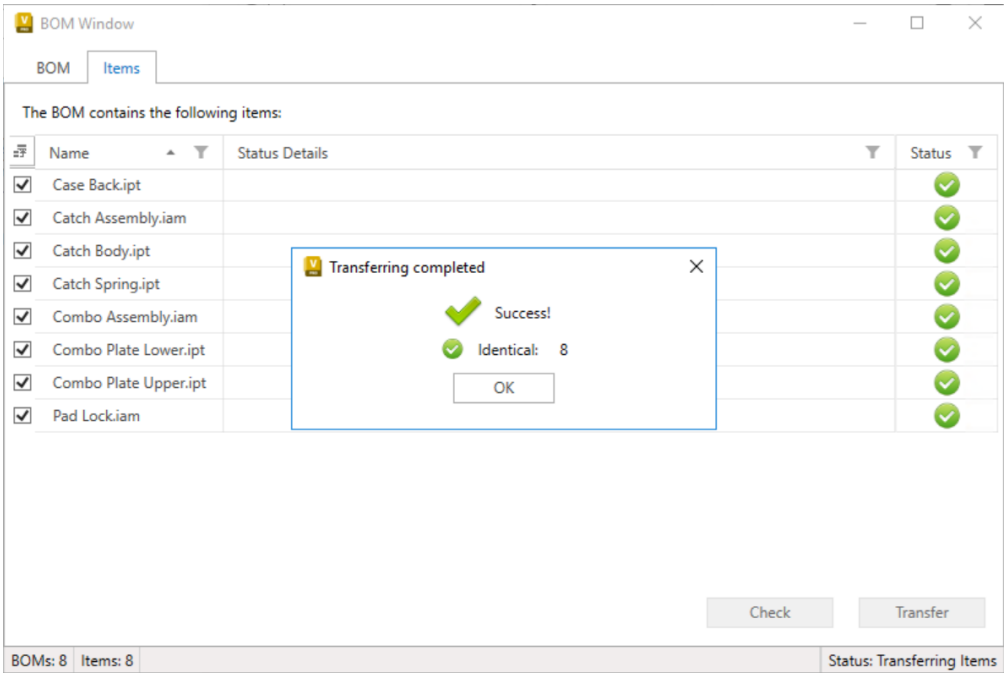
The script can be tested by doing the following steps on your test-environment:

1. Open the Vault Client and log in to your *Vault*.
2. Navigate to a file and select it.
3. Click on the tab with the name "ERP BOM".

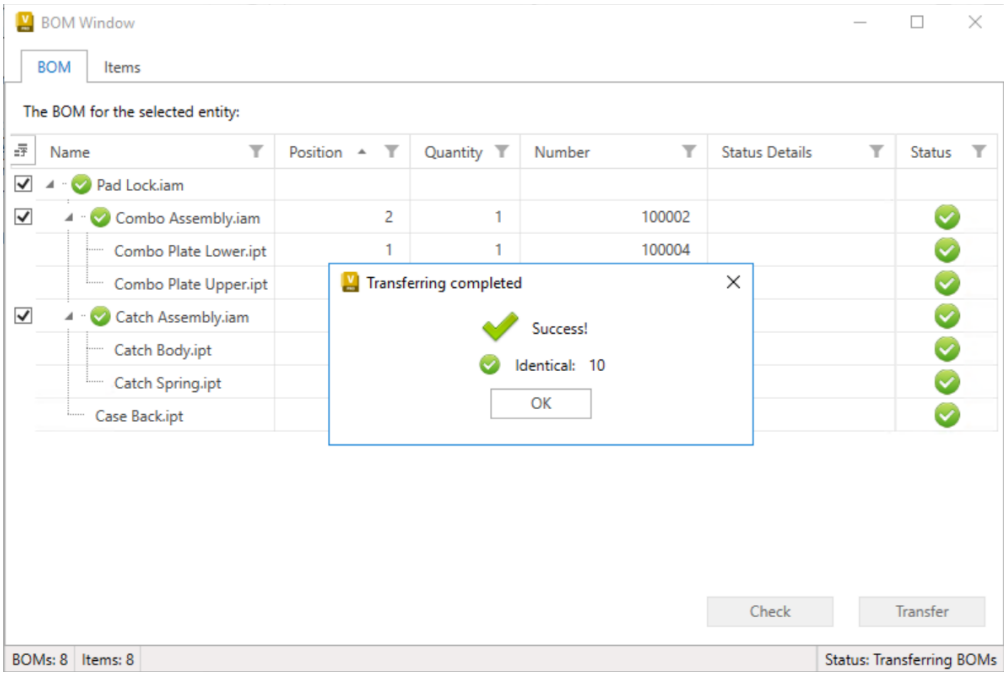
4. Since no BOM with the “Part Number” of the selected file exists in the *Demo ERP system*, a new one can be created.



5. Click on “Create new ERP BOM…”. The *BOM Window* opens and the complete CAD-BOM is displayed.
6. Use the respective Check- and Transfer buttons in the “Items” tab to first create or update all the required Items in ERP.

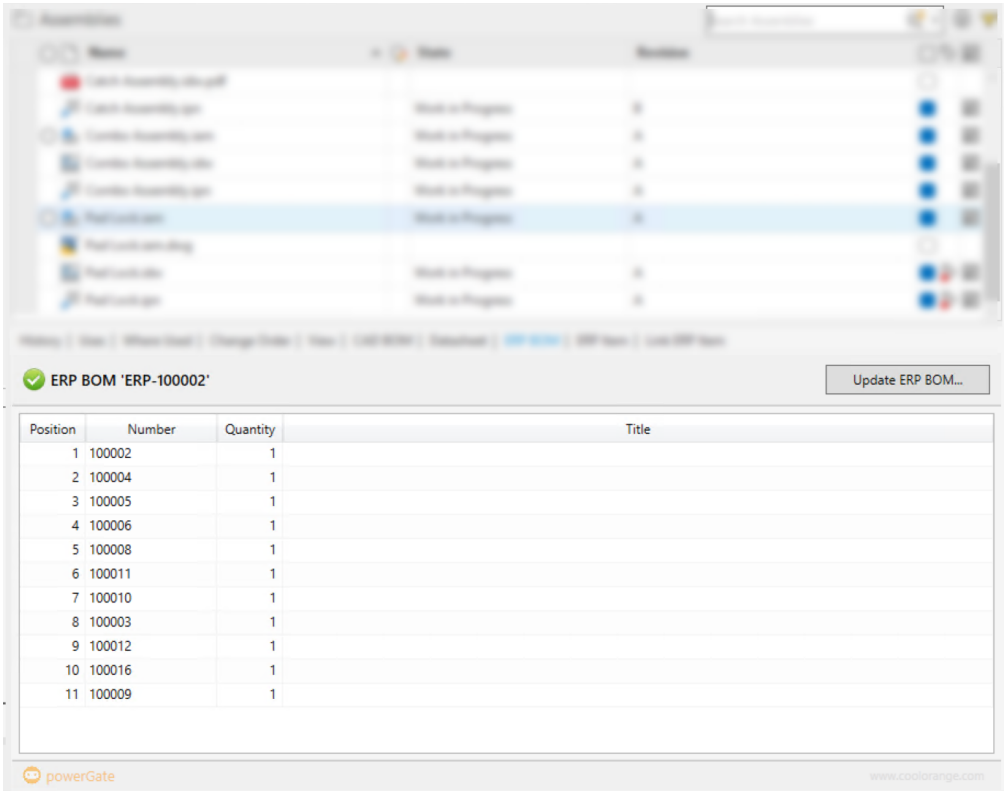


7. Click the Check- and Transfer buttons in the “BOM” tab.



The BOMs have now been successfully generated in the ERP system and the BOM Window can be closed.

8. The position number, item number, quantity, and other relevant information are displayed for all rows in the newly created ERP BOM:



4.2.2 Disabling the script

After the *installation* on new environments, this script by default **enables** the Vault "ERP BOM" tab for files.
If you work exclusively with Vault Items, or if you do not use this tab, the file *%PROGRAMDATA%\coolOrange\Client Customizations\Sample.Tab-File-ErpBom.ps1* can be moved to the directory *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled*.

Note: Please note that after **updates** from v23.0.4 and older, this script is automatically disabled and installed in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.
Scripting guys can then enable and review the "ERP BOM" tab in a test environment before deploying their own version of the script in production ERP integrations.

4.3 Sample.Tab-File-ErpItem

During the design process, it is crucial for the engineer to access material information from the ERP system.

Using the "ERP Item" tab from this sample script, the designer can select a file in the Vault Client and view the item number and other relevant information live from the ERP system.
If desired, the displayed ERP data can also be modified directly via this tab.
If no item exists yet, all necessary information for the creation of a new ERP Item can be entered manually to create it in the ERP system afterwards.

Requirements

This Vault Client customization is designed to be used with the "*DemoService*" from the ***Demo ERP system***:

- An **internet connection** is required and the domain *'demo.powergate.online'* needs to be accessible on TCP port 80.
- An entity set named "**Items**" is expected
- Vault files are uniquely assigned to an ERP item via their item number stored in the Vault property **"Part Number"** and the ERP field "Number".

This tab can also be activated at any time when you are already using your own ERP integration.

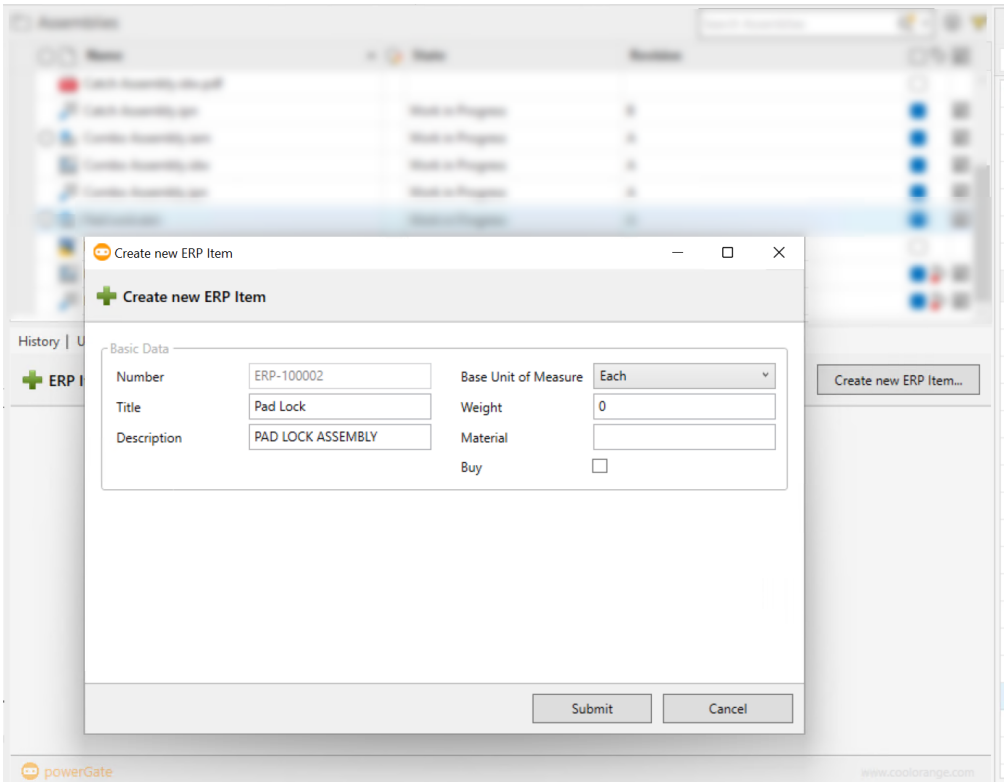
In the *ERP Integration Settings* dialog, it is possible to specify from which *Vault Properties* (or *fixed-values*) the data of new Items comes from (see type-mapping *Vault 'File' -> ERP 'Item'*).
Additionally, also the values for the *"Unit of Measure"* selection list are retrieved from this configuration (see ERP field *'UnitOfMeasure' -> list values*).

4.3.1 Testing

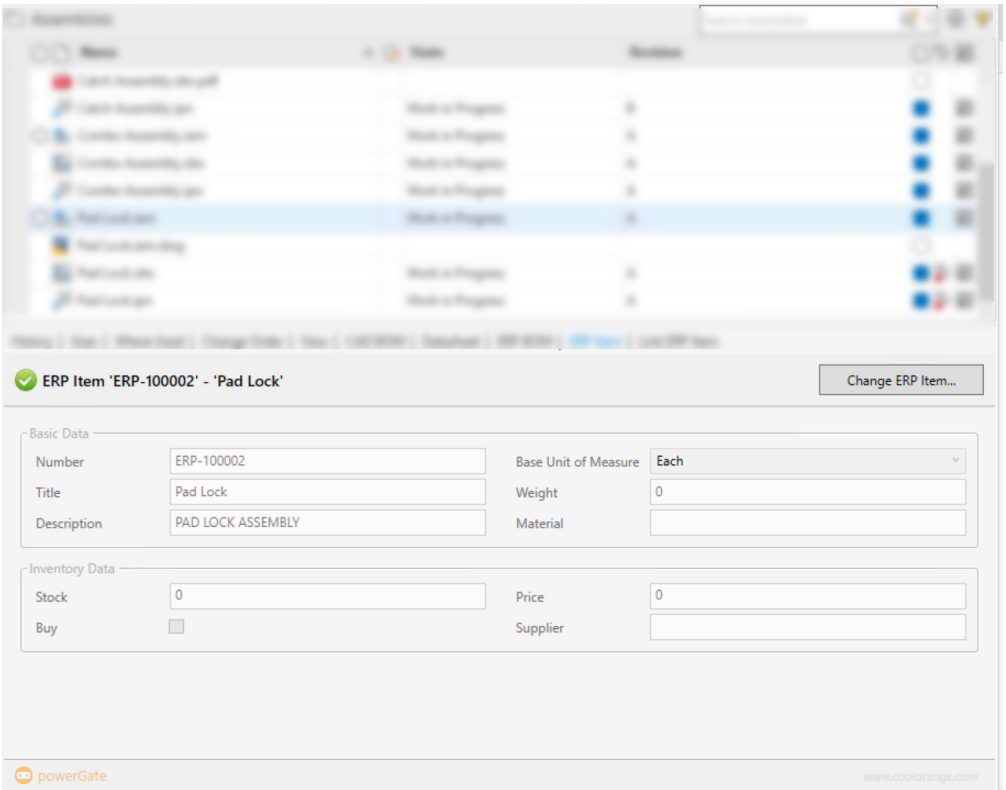
The script can be tested by doing the following steps on your test-environment:

1. Open the Vault Client and log in to your *Vault*.
2. Navigate to a file and select it.
3. Click on the tab with the name "ERP Item".
4. Since no item with the "Part Number" of the selected file exists in the *Demo ERP system*, a new one can be created.

5. Click on “Create new ERP Item…” which opens a Dialog with pre-filled data.



6. Press *Submit* and a new ERP Item gets created in the ERP system. Its item number and other metadata are displayed:



This information can also be changed after clicking on "*Change ERP Item...*".

4.3.2 Disabling the script

After the *installation* on new environments, this script by default **enables** the Vault "ERP Item" tab for files.
If you work exclusively with Vault Items, or if you do not use this tab, the file *%PROGRAMDATA%\coolOrange\Client Customizations\Sample.Tab-File-ErpItem.ps1* can be moved to the directory *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled*.

Note: Please note that after **updates** from v23.0.4 and older, this script is automatically disabled and installed in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.
Scripting guys can then enable and review the "ERP Item" tab in a test environment before deploying their own version of the script in production ERP integrations.

4.4 Sample.Tab-Item-ErpBom

The "ERP BOM" tab from this sample script allows the designer to select an item in the Vault Client and immediately see if a BOM already exists in the ERP system.
If it does, all the position numbers, item numbers, quantities, and other relevant BOM row details are displayed live from the ERP system.
Using the *BOM Window* it is possible to automatically generate and update all required materials and BOMs in the ERP system.

Requirements

This Vault Client customization is designed to be used with the "*DemoService*" from the ***Demo ERP system***:

- An **internet connection** is required and the domain *'demo.powergate.online'* needs to be accessible on TCP port 80.
- An entity set named "**BomHeaders**" is expected as well as the entityset **"BomRows"** and **"Items"**
- Vault items are uniquely assigned to an ERP BOM via their item number stored in the Vault property **"Number"** and the ERP field "ParentNumber".
- Individual BOM rows are also identified by the item number, but their **position number in the Vault BOM** and the ERP BOM "Position" must also match.

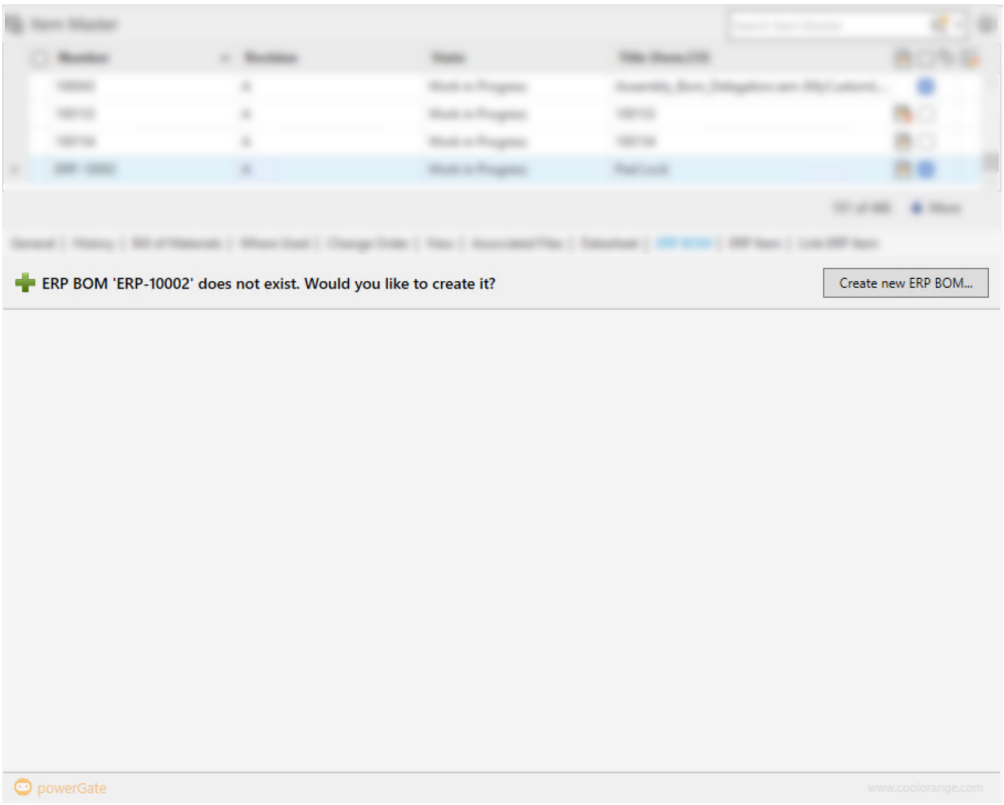
This tab can also be activated at any time when you are already using your own ERP integration.

In the *ERP Integration Settings* dialog, it is possible to specify from which *Vault Properties* (or *fixed-values*) the data of created Items comes from (see type-mapping *Vault 'Item' -> ERP 'Item'*).
Additionally, also the possible values for the ERP field *"UnitOfMeasure"* are retrieved from this configuration (see ERP field *'UnitOfMeasure' -> list values*).

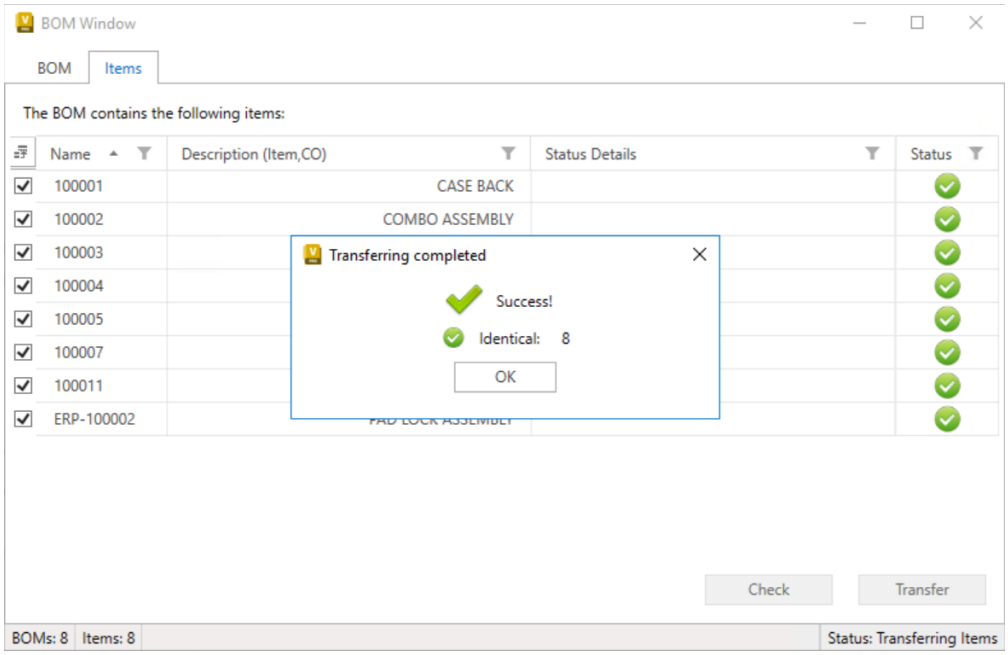
4.4.1 Testing

The script can be tested by doing the following steps on your test-environment:

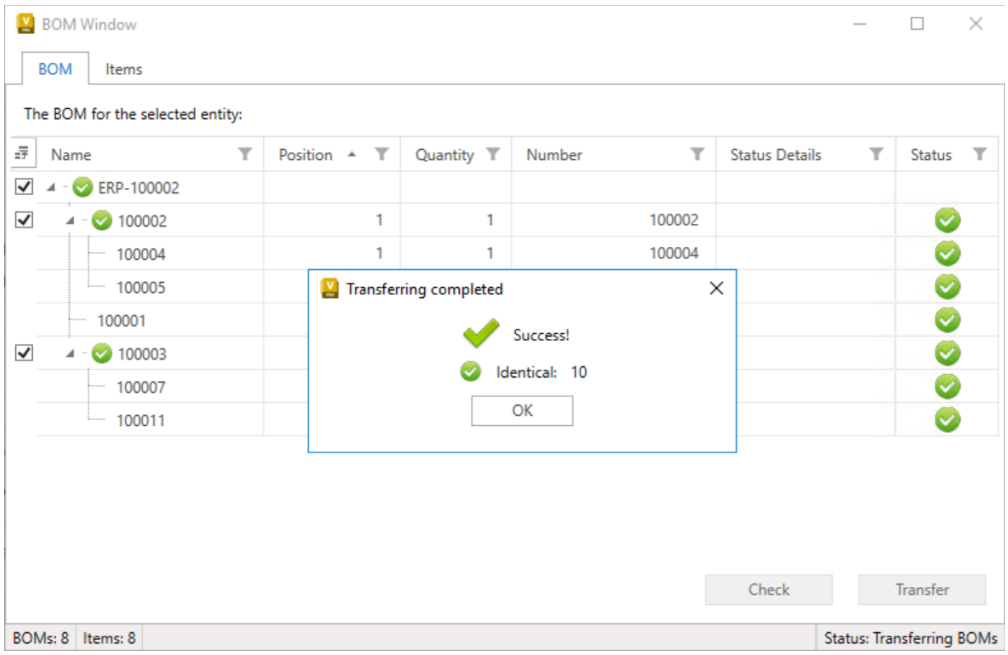
1. Open the Vault Client and log in to your *Vault*.
2. Navigate to an item in the Item Master view and select it.
3. Click on the tab with the name “ERP BOM”.
4. Since no BOM with the “Number” of the selected item exists in the *Demo ERP system*, a new one can be created.



5. Click on “Create new ERP BOM…”. The *BOM Window* opens and the complete Vault BOM is displayed.
6. Use the respective Check- and Transfer buttons in the “Items” tab to first create or update all the required Items in ERP.



7. Click the Check- and Transfer buttons in the “BOM” tab.



The BOMs have now been successfully generated in the ERP system and the BOM Window can be closed.

8. The position number, item number, quantity, and other relevant information are displayed for all rows in the newly created ERP BOM.

4.4.2 Disabling the script

After the *installation* on new environments, this script by default **enables** the Vault "ERP BOM" tab for items.
If you work exclusively with Vault files, or if you do not use this tab, the file *%PROGRAMDATA%\coolOrange\Client Customizations\Sample.Tab-Item-ErpBom.ps1* can be moved to the directory *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled*.

Note: Please note that after **updates** from v23.0.4 and older, this script is automatically disabled and installed in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.
Scripting guys can then enable and review the "ERP BOM" tab in a test environment before deploying their own version of the script in production ERP integrations.

4.5 Sample.Tab-Item-ErpItem

During the design process, it is crucial for the engineer to access material information from the ERP system.
Using the "ERP Item" tab from this sample script, the designer can select an item in the Vault Client and view the item number and other relevant information live from the ERP system.

Requirements

This Vault Client customization is designed to be used with the "*DemoService*" from the ***Demo ERP system***:

- An **internet connection** is required and the domain *'demo.powergate.online'* needs to be accessible on TCP port 80.
- An entity set named "**Items**" is expected
- Vault items are uniquely assigned to an ERP item via their item number stored in the Vault property **"Number"** and the ERP field "Number".

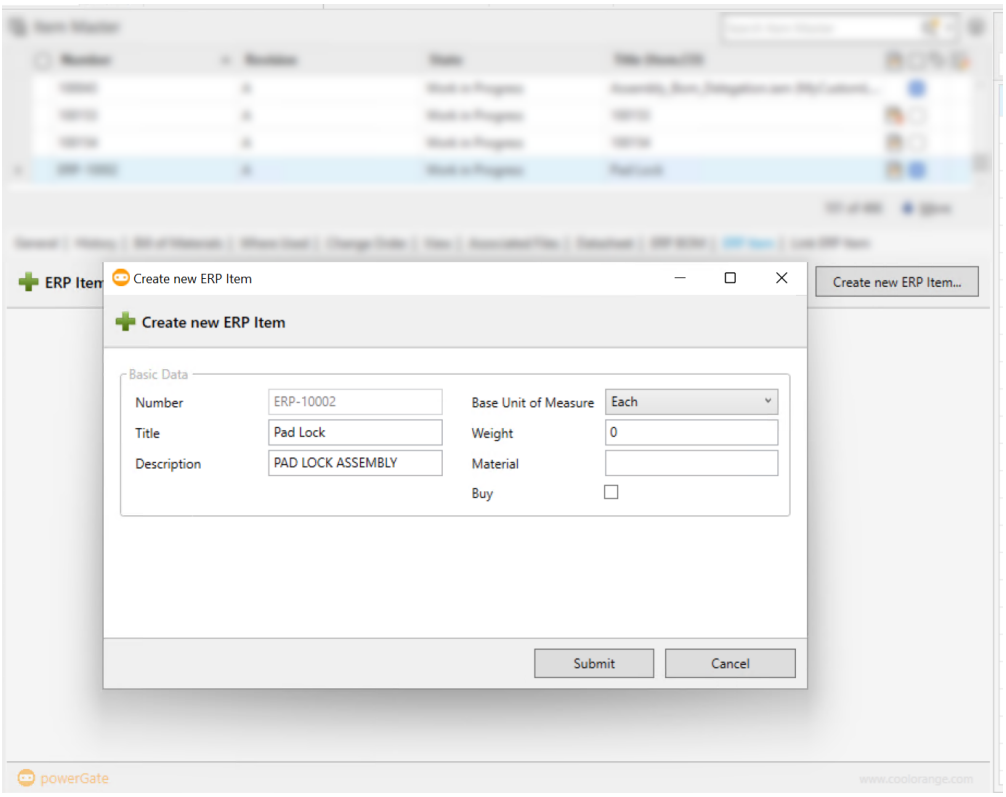
This tab can also be activated at any time when you are already using your own ERP integration.

In the *ERP Integration Settings* dialog, it is possible to specify from which *Vault Properties* (or *fixed-values*) the data of new Items comes from (see type-mapping *Vault 'Item' -> ERP 'Item'*).
Additionally, also the values for the *"Unit of Measure"* selection list are retrieved from this configuration (see ERP field *'UnitOfMeasure' -> list values*).

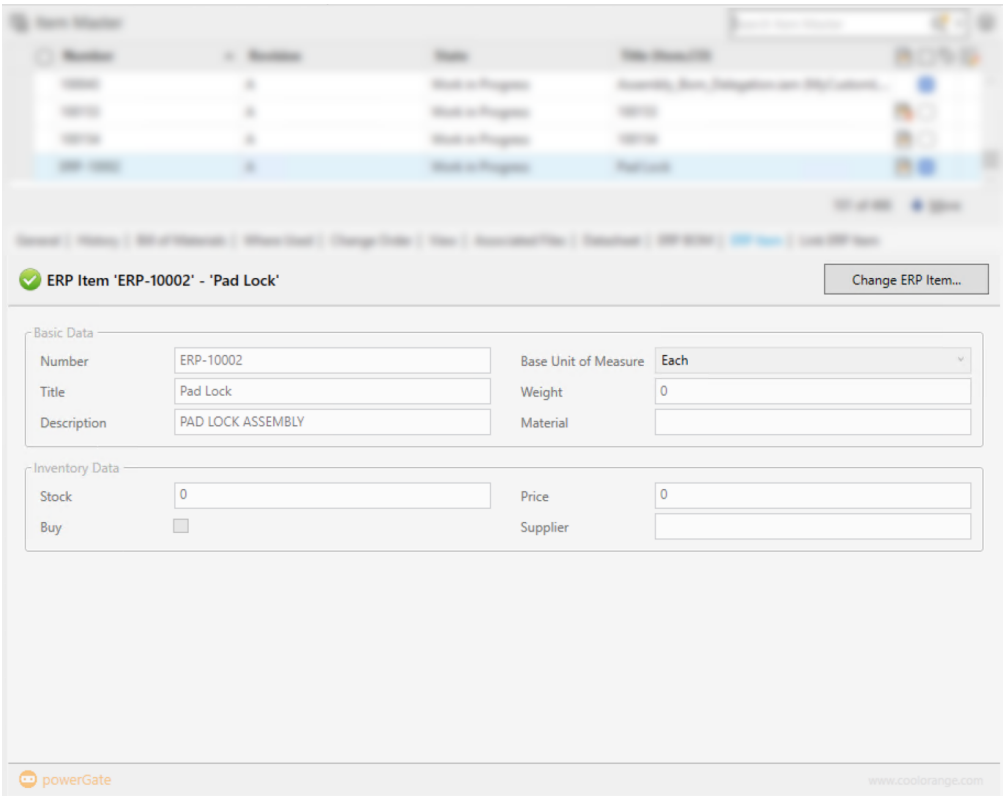
4.5.1 Testing

The script can be tested by doing the following steps on your test-environment:

1. Open the Vault Client and log in to your *Vault*.
2. Navigate to an item in the Item Master view and select it.
3. Click on the tab with the name "ERP Item".
4. Since no item with the "Number" of the selected item exists in the *Demo ERP system*, a new one can be created.
5. Click on "Create new ERP Item…" which opens a Dialog with pre-filled data.



6. Press *Submit* and a new ERP Item gets created in the ERP system. Its item number and other metadata are displayed:

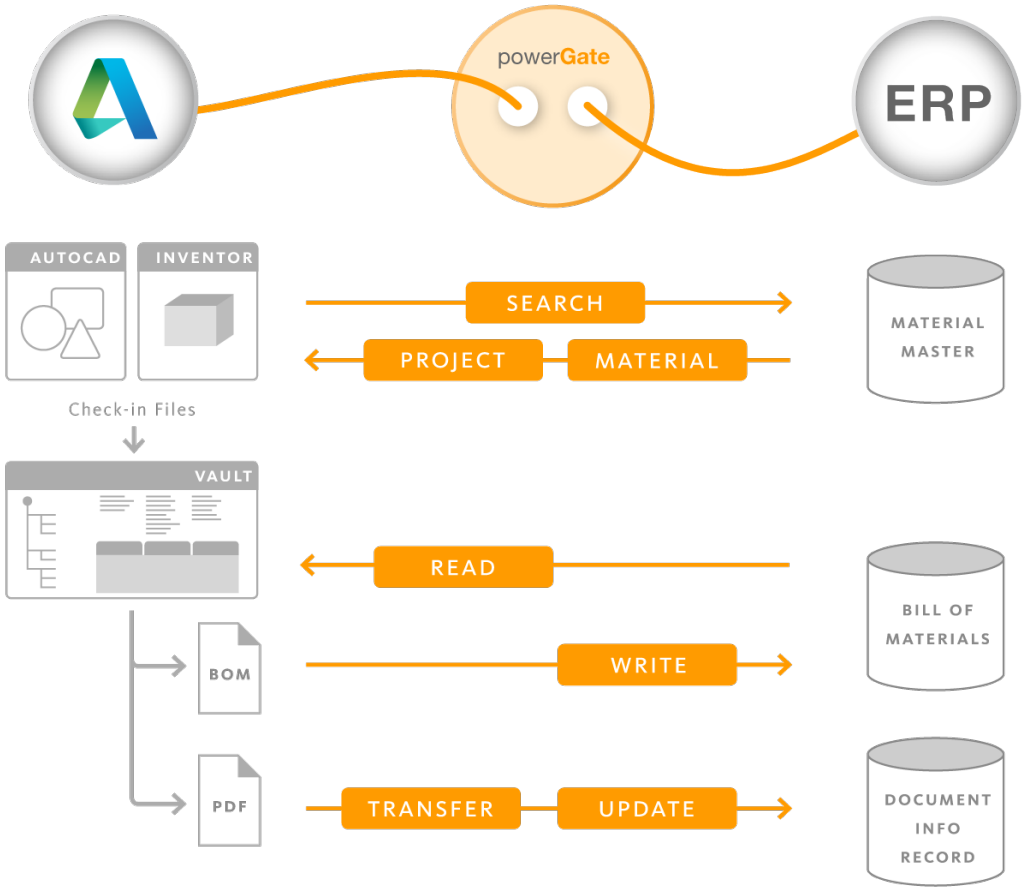


This information can also be changed after clicking on "*Change ERP Item...*".

4.5.2 Disabling the script

After the *installation* on new environments, this script by default **enables** the Vault "ERP Item" tab for items.
If you work exclusively with Vault files, or if you do not use this tab, the file *%PROGRAMDATA%\coolOrange\Client Customizations\Sample.Tab-Item-ErpItem* can be moved to the directory *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled*.

Note: Please note that after **updates** from v23.0.4 and older, this script is automatically disabled and installed in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.
Scripting guys can then enable and review the "ERP Item" tab in a test environment before deploying their own version of the script in production ERP integrations.



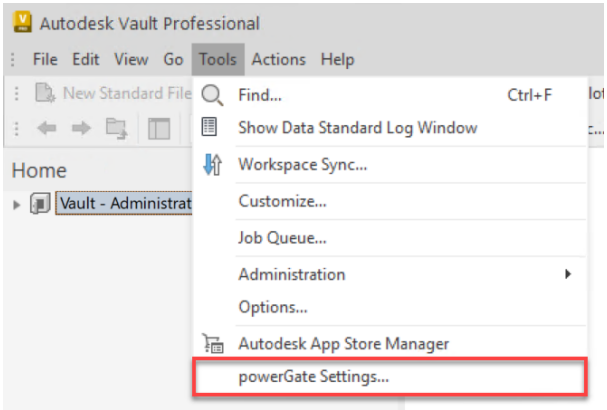
4.6 ERP integrations

Vault Clients, Inventor and AutoCAD applications can be easily connected to ERP systems using the powerGate *Cmdlets* or the *.NET library*.
Depending on whether data synchronizations must be fully automated, partially automated or performed manually, an ERP integration can be realized using powerEvents, Autodesk Vault Data Standard and additionally with powerJobs Processor.

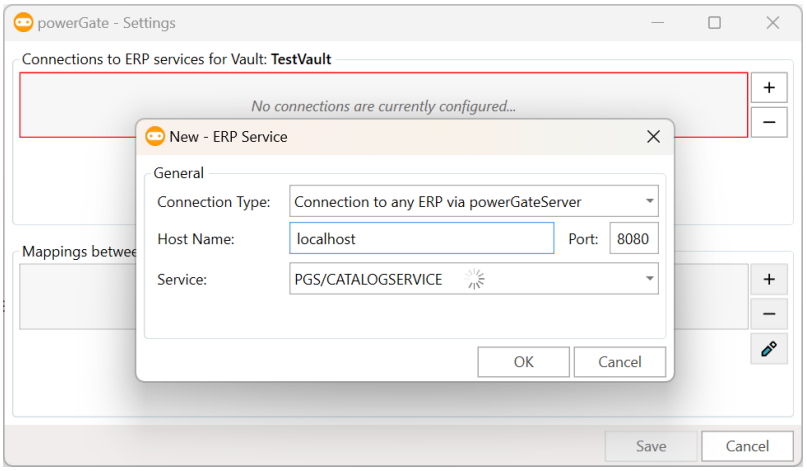
4.6.1 Configuration

Vault administrators can configure settings for their entire ERP integration, and these settings are specific to each Vault.

To do this, start the Vault Client and **login** to the respective Vault.
Afterwards, the "ERP Integration Settings" dialog can be opened in the **Tools** → **powerGate Settings...** menu (see *Sample.ConnectToERP*).



Within the dialog it is possible to configure the necessary **Services** required for connecting to the ERP system.



For some ERP systems, a single service can be configured that provides all the necessary functionalities for the transfer of Items and BOMs.
Other ERP systems, such as SAP Gateway, however requires multiple URL endpoints for this purpose.

Connection Types:

- *Direct connection to an OData-capable ERP:*
  ERP systems that provide an OData interface can be directly connected from Vault applications.

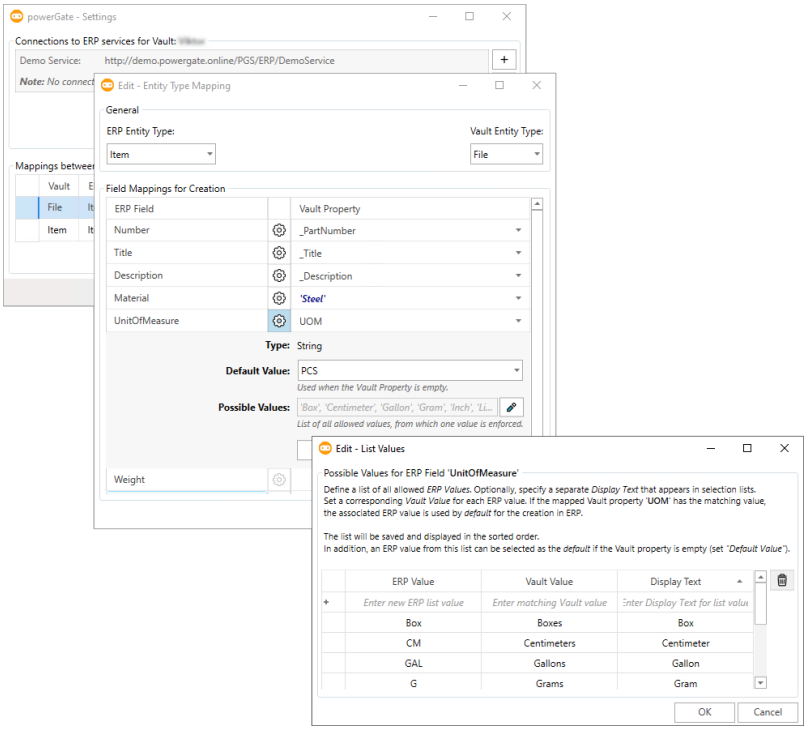
- *Connection to any ERP via powerGateServer:*
  Also ERP systems which doesn't expose a web API can be connected using powerGateServer, who then acts as a middleman and translator.
  New Services can be configured quickly because *Host Name*, *Port* and *Service* paths are proposed based on the following recommendations.

  Recommendations:
  If you are using a test-Vault, then the services from **local** ERP plugins should be configured (they may still be under development). Please ensure they actually connect to the **test-ERP** system!
  For production Vaults, on the other hand, the powerGateServer should be installed directly on the **Vault Server** machine.

  Note that powerGateServer installations include an ERP Plugin by default, but it's not needed for real ERP integrations; thus, install the powerGateServer without 'ERP Plugin' on test and productive environments!
  Then, a single *CatalogService* connection can be configured, allowing workstations to automatically connect to all installed services, regardless of their names.

  Otherwise, please configure the individual **services separately** to avoid name-conflicts with the services of the ERP Plugin.

Additionally, the dialog provides the option to configure mappings between **Vault** and **ERP entity types**.
Each mapping allows you to specify, for each ERP field, whether data from Vault properties or fixed-values should to be transferred.



Once settings are changed, the changes can be saved across the entire Vault by clicking the **Save** button.

Requirements

In order to change settings, the logged-in Vault user must have the respective **permissions** to edit the *$/powerGate.settings* file.
Otherwise, the Save button is disabled and the status bar of the dialog provides according information.

When applying the same configuration to other Vaults, be sure to check all settings (e.g. Service URLs) for correctness.

Potential errors must then be resolved, especially if a Vault Property or an ERP Field is missing (perhaps it has been renamed), or when mapped value types suddenly no longer match.

4.6.2 Scripts

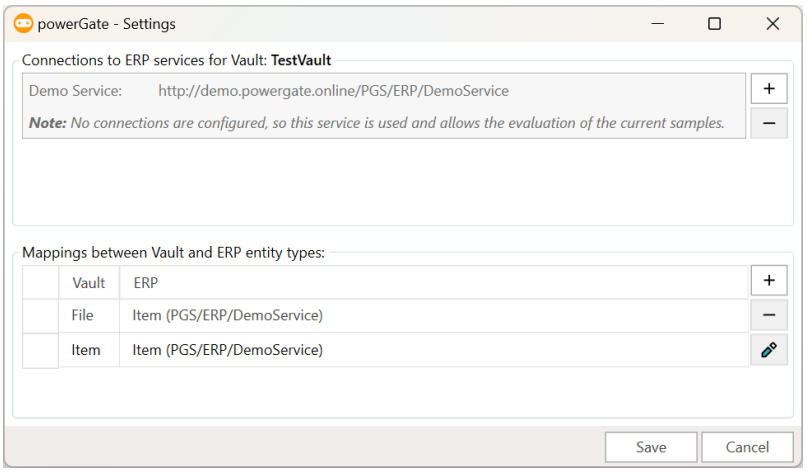
Several PowerShell scripts are delivered with the product, all starting with the name *'Sample.'*.
Their purpose is mainly to help getting started with implementing a new ERP integration for the Vault Client and Inventor.

For this purpose, powerEvents is used on the workstations to execute the individual Client Customizations in *%ProgramData%\coolOrange\Client Customizations*:

- *Sample.ConnectToERP.ps1*
- *Sample.Tab-File-ErpItem.ps1* & *Sample.Tab-Item-ErpItem.ps1*
- *Sample.Tab-File-ErpBom.ps1* & *Sample.Tab-Item-ErpBom.ps1*

These samples cover the most basic scenarios for connecting and exchanging data between Vault and common ERP systems.

When starting Vault applications and logging into Vault, the *Sample.ConnectToERP* script automatically connects to the configured services of the ERP system.
To be able to evaluate the individual functions of our sample ERP integration, our public *Demo ERP system* is used.



In the Vault Client, additional "ERP Item" and "ERP BOM" tabs are displayed for selected files and items.
For files, the *Sample.Tab-File-ErpItem* script displays the information of the corresponding ERP Item and the *Sample.Tab-Item-ErpItem* script displays the current bill of materials from the ERP.
For items, the *Sample.Tab-File-ErpBom* and *Sample.Tab-Item-ErpBom* scripts display the corresponding information from the ERP system.
In addition, data can also be created or changed live in the ERP system, considering all configurations made.

Since normally either Vault Items or Vault Files are used, the PowerShell scripts that are not needed should be *disabled*.

The Tabs are delivered with corresponding .xaml files (Extensible Application Markup Language) which describe their layout, including the controls used (label, text field, combo box, etc.).

Note: For all sample scripts and the according xaml files it is recommended to create a **copy** when they need to be **modified**.
Then disable the particular sample script by simply moving it to the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory.

4.6.3 Modules

The *powerGate_Connections.psm1* module is delivered in the Cmdlets installation directory *%ProgramFiles%\coolOrange\Modules\powerGate*.
It gets automatically imported with the *powerGate module* and provides global variables and extensions:

- The **$sapConnect** variable for *connecting to SAP systems*.
- The **$demoErpConnect** variable for *connecting to the Demo ERP system*.
- The **$global:Host.PrivateData.OnNonTerminatingError** extension ensures that Vault users are notified about *connection problems* and can better understand whether the problem is caused by their client-machine, by powerGateServer or directly triggered by the ERP system.

In addition, the *powerGate_Configuration.psm1* module provides an **$ERPSettings** variable and ERP Cmdlet extensions, necessary for working with *configurations* made for the ERP integration in the current Vault.

Import powerGate module

Before the global variables can be used in *-OnConnect* of the *Connect-ERP*, they must be imported using `Import-Module powerGate`.
Otherwise it may happen that no successful connection to the ERP system can be established.

4.7 Errors

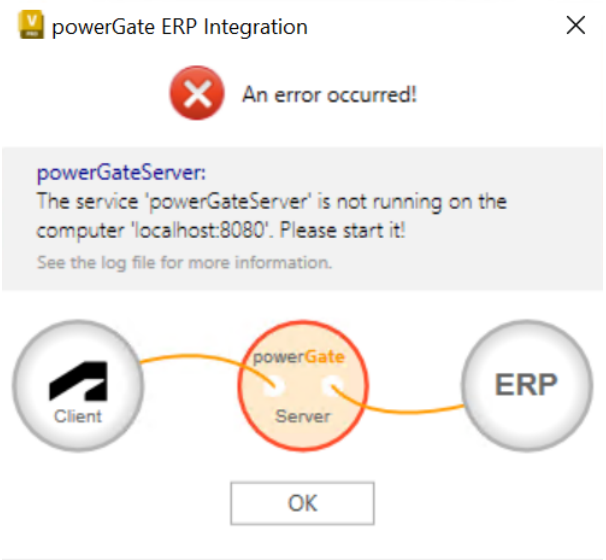
PowerShell hosting applications inform developers and Vault users in different ways about Terminating Errors that arise during script or module executions :

- powerEvents displays Error Message Boxes
- *BOM Window* displays Error Dialogs during Check- and Transfer operations or failure Icons while loading the BOM tree
- powerJobs Processor jobs fail
- Autodesk Vault Data Standard writes to the Log File and shows Message Boxes in several customization areas
- PowerShell IDE's display the error message and its stack trace with red text in the terminal when the script execution stops

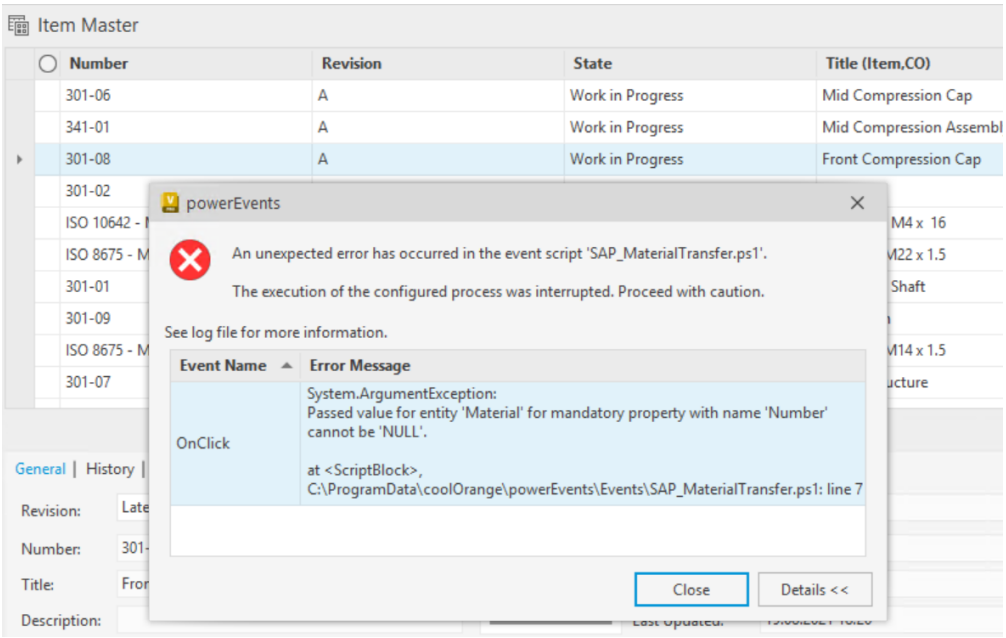
Instead, when Non-Terminating Errors occur within *Cmdlets*, the script execution is not stopped by default (see `-ErrorAction:Continue` parameter) and further information is available in the global `$Error` variable.
Only with ERP integrations, which are loaded as powerEvents Client Customizations, many errors are also managed automatically and no additional error handling needs to be implemented:

- In applications such as **Vault Client, Inventor or AutoCAD**, dialogs are displayed that provide information about the nature of the problem.

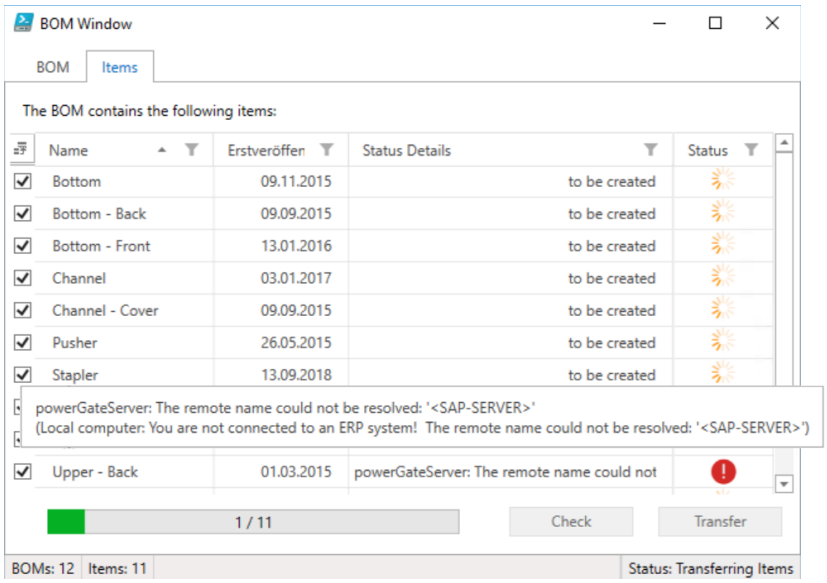
  For connection problems, these would be *WebRequestExceptions* occurring in *ERP cmdlets*, a Connection Error Dialog is displayed to the Vault user.
  This helps to understand whether the problem was caused by the client-machine (no request was send), the powerGateServer or directly by the ERP system (error response):



Other types of Non-Terminating Errors are incorrectly used *ERP cmdlets* or problematic parameters where no web requests are sent at all.
In this case, the developer can see all the details of the exception in an Error Message Box, which helps him to quickly identify the erroneous line:

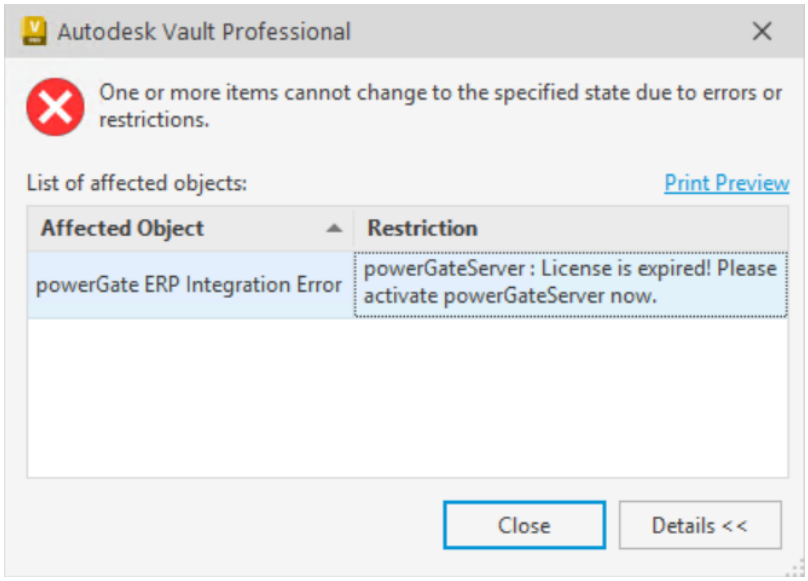


- Such modal error dialogs are not displayed in the **BOM Window** when problems with ERP cmdlets occur during *Check- or Transfer functions*.
  Instead, the current processed *$bom*, *$bomRow* and *$item* objects are automatically assigned with an *Error Status*:



The automatically set `_StatusDetails` help the Vault user to localize the cause of connection problems more precisely.
Unfortunately the operations are not automatically *terminated* after problematic ERP cmdlet usages or incorrect parameter values, but also in this case additional stack trace information allows the scripting guy to find the problematic line faster.

- All **Vault Restriction events** that encounter ERP cmdlet errors are automatically restricted to prevent the configured process from passing successfully, after connection or cmdlet problems have occurred:



Unfortunately, the executed Restrictions event actions are not automatically aborted directly after problematic ERP cmdlet usages or incorrect parameter values.
Still, additional stack trace information helps the script guy to quickly find and correct the affected ERP cmdlet line.

Autodesk Vault Data Standard - Non-Terminating Errors

For ERP integrations implemented via Vault Data Standard, non-terminating errors in ERP cmdlets are only logged in the Log Window.

To prevent connection or cmdlet issues beeing swallowed, the customization should therefore check for non-terminating errors after ERP cmdlet invocations and handle them if necessary. See the *"Error handling"* examples of the individual cmdlets for this.

Please note that debugging such customizations within PowerShell IDE's may also be affected, as the observed error behavior may differ from that in the Vault Client or Inventor.
Just execute the following line in the debugging session to ensure that no modal dialogs etc. are displayed in the PowerShell IDE either:

```
$global:Host.PrivateData.OnNonTerminatingError = [Action[System.Management.Automation.
↪RuntimeException]] { }
```

However, the automatic handling of errors can also be **suppressed** for individual *ERP cmdlet* invocations by passing the `-ErrorAction SilentlyContinue` parameter.
This allows the script execution to continue so that special error handling can be implemented. For example, personalized messages can be displayed to the Vault user or specific 4xx responses can be intercepted (see example *"Check SAP Material does not exist or other 400 response occurred"*).
Error responses can then be handled as desired using the information provided by the *WebRequestException*, which is stored in the global `$Error` variable.

In any case, all error details are logged and can be found in the *logfile*.

CHAPTER

FIVE

BOM WINDOW

5.1 Customization

Localization

The names of the tabs, buttons and columns in the BOM Window will be either displayed in *English* or *German* language.
The Window uses the current threads UI culture to determine the language to be displayed.
In general this is determined by the system's regional settings.

For changing the UI culture of the current thread within PowerShell, see the available *example*.

5.1.1 Check and Transfer

In order to open the BOM Window in a PowerShell session, the cmdlet *show-bomWindow* should be used.
According *functions* can be implemented that will be executed when performing Check or Transfer operations for Items, as well as for the BOMs and their rows.
In order to add, remove or manipulate such entities within these functions, there are according PowerShell *cmdlets* available as well.

5.1.2 Customizing the layout

The layouts of the BOM and Items table can be modified individually so only columns relevant to the user are shown.
The *BOM Tab* has the following columns that are always available:

- Name
- Position
- Quantity
- Number
- *Status Details*
- *Status*

Additionally all *BOM and entity properties* are available.

The *Item Tab* has the following standard columns:

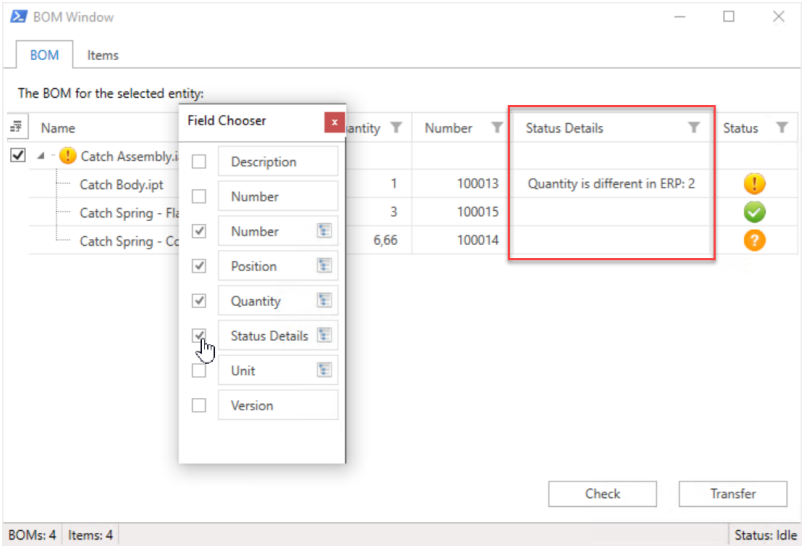
- Name

- Usages Count (not enabled by default)
- *Status Details*
- *Status*

Additionally all entity properties are available.

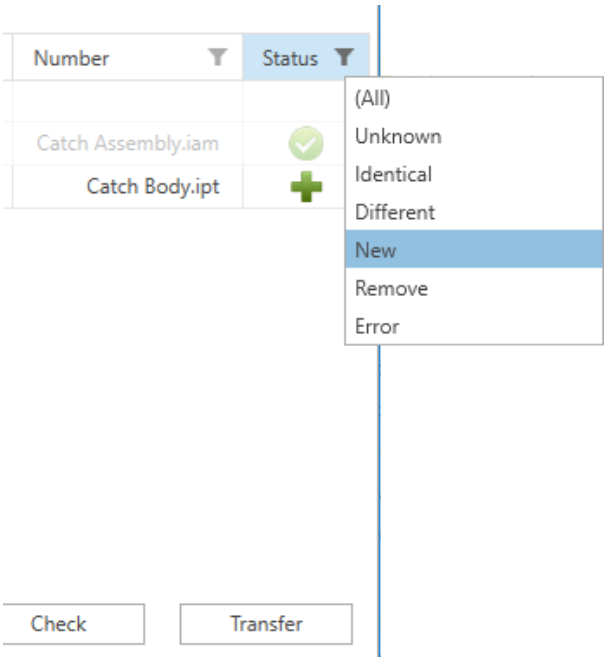
Add or remove columns

Clicking on the Field Chooser button ( ) will open a new window where you can select the BOM properties (marked with the icon:) and entity properties that should be displayed.



Filtering rows

Clicking on the small funnel ( ) next to the column name will present a dropdown menu. This menu allows you to filter all rows based on the value the row has for the filtered column.
Only rows which match the filter condition are shown. On the BOM tab, all parents of a matching row will stay visible but will be grayed out.
An active filter on a column is indicated by a blue filter icon ( ) in the column header. The filter can be cleared by selecting the (All) entry.
It is possible to set filters on multiple columns at the same time, only rows that match all filters will be shown.



Please note

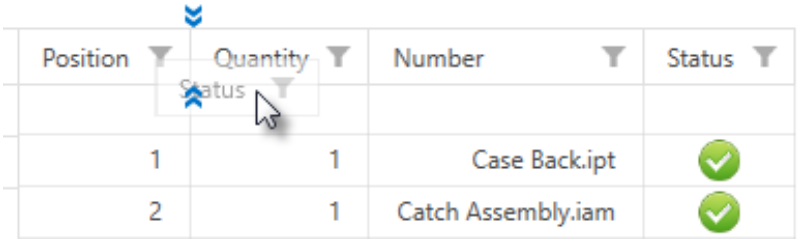
On the BOM tab, filtering the Status column will filter the BOM status as well as the BomRow status.
Filtered columns are not automatically un-selected so clicking Check or Transfer might include entities that are hidden by the filters.

Sorting rows

Clicking on a column name will order the rows by this column values. A column that with ordering enabled has a small arrow next to the column name to indicate whether the column is ordered in ascending (▲) or descending (▼) order. You can order by one column at a time.

Change column position and size

You can move a column by dragging it to the desired location. The placement of the column is indicated by the blue arrows.



The size of the columns can be changed.
By double clicking on the right edge of a column header the width gets auto sized based on its contents.

Saving and restoring layouts

The configuration files are saved in the following directory: *%LOCALAPPDATA%\coolOrange\powerGate*:

- For the *BOM Window*: *StoredLayout_Window.xml*
- For the *Bom Tab*: *StoredLayout_BomsView.xml*
- For the *Item Tab*: *StoredLayout_ItemView.xml*

The size and position of the BOM window itself is automatically saved when the window is closed and is restored when the window is opened.

The view settings for the Item and BOM tables are automatically saved when changed and will be restored when the window is opened. The size of the columns however is only saved after specific view settings are changed, such as Sorting, Filtering, Adding/Removing columns.

5.2 Status

The Status of BOMs, rows and Items is displayed by one of the following icons:

Icon	Name	Description	Default Status Details
	**Unknown**	Entities in this state have not been checked yet.	Unknown
	**Identical**	Entities in this state exist on the ERP side and are identical, or have been successfully transferred to ERP.	Identical
	**Different**	Entities in this state have been compared against the ERP and have some differences.	Differences found!
	**New**	Entities in this state have been checked but do no exists on the ERP side and will be created when transferred.	Will be added
	Remove	Entities in this state will be removed when transferred.	Will be removed
	**Error**	For entities in this state an error is occured during the Check/Transfer.	Error occurred!

5.2.1 Status Details

The Status Details is designed to provide the description of the current *Status* of an entity.

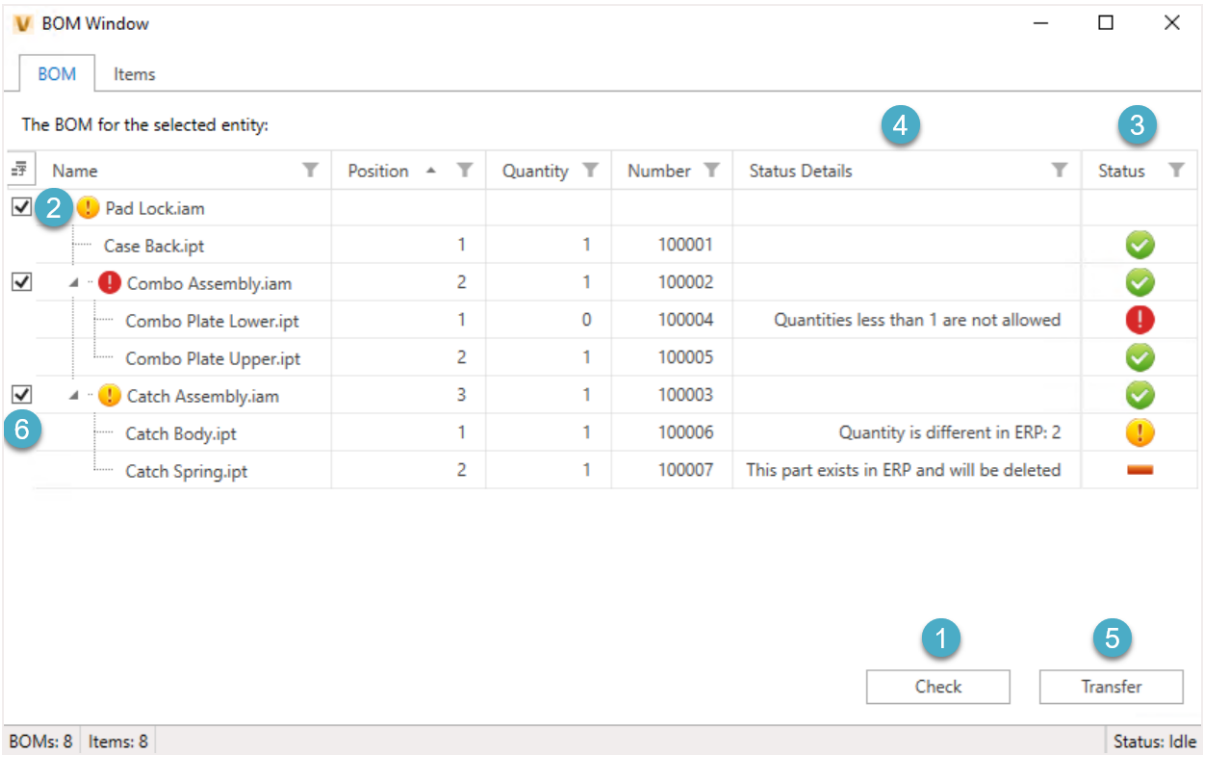
The Status Details are displayed as tooltip when hovering over the Status icon of an entity and for *BomRows* and *Items* in the dedicated Status Details column of the *BOM Window*.

It is possible to retrieve the Status Details programmatically by accessing the `_StatusDetails` member of the according *entity*.

The BOM Window exposes a mulit-level bill of material and each item, which is included in the BOM.

5.3 BOM Tab

The BOM Tab displays the BOM with all its rows including sub-level BOMs.



When the whole BOM tree finished loading, the **Check**[1] button can be pressed in order to compare the selected BOMs with the ones on the ERP side.
It shows us whether the BOM exists, has differences or is identical to the one on the ERP side.
The result will be displayed in the **Icon**[2] on the left of the BOM name (No icon next to the BOM name means the status is *Unknown*).
Same goes for the BomRows. All rows of the BOMs will be compared to the ones on the ERP side.
The according result will be displayed as icon in the **Status**[3] column and the according description of the status will be displayed in the **Status Details**[4] column. The *Status Details* can also be retrieved when hovering over the status icon.
For more details about the different Status Details- and Icons see *here*.

After the BOMs have been checked, they can be transferred to the ERP System by pressing the **Transfer**[5] Button.
Please note: BOMs can only be transferred when they have been updated to a status other than *Unknown*.
Depending on the result of the check, the transfer action handles the selected BOMs and its rows differently:

- BOMs and BomRows which do not exist in the ERP are getting created.

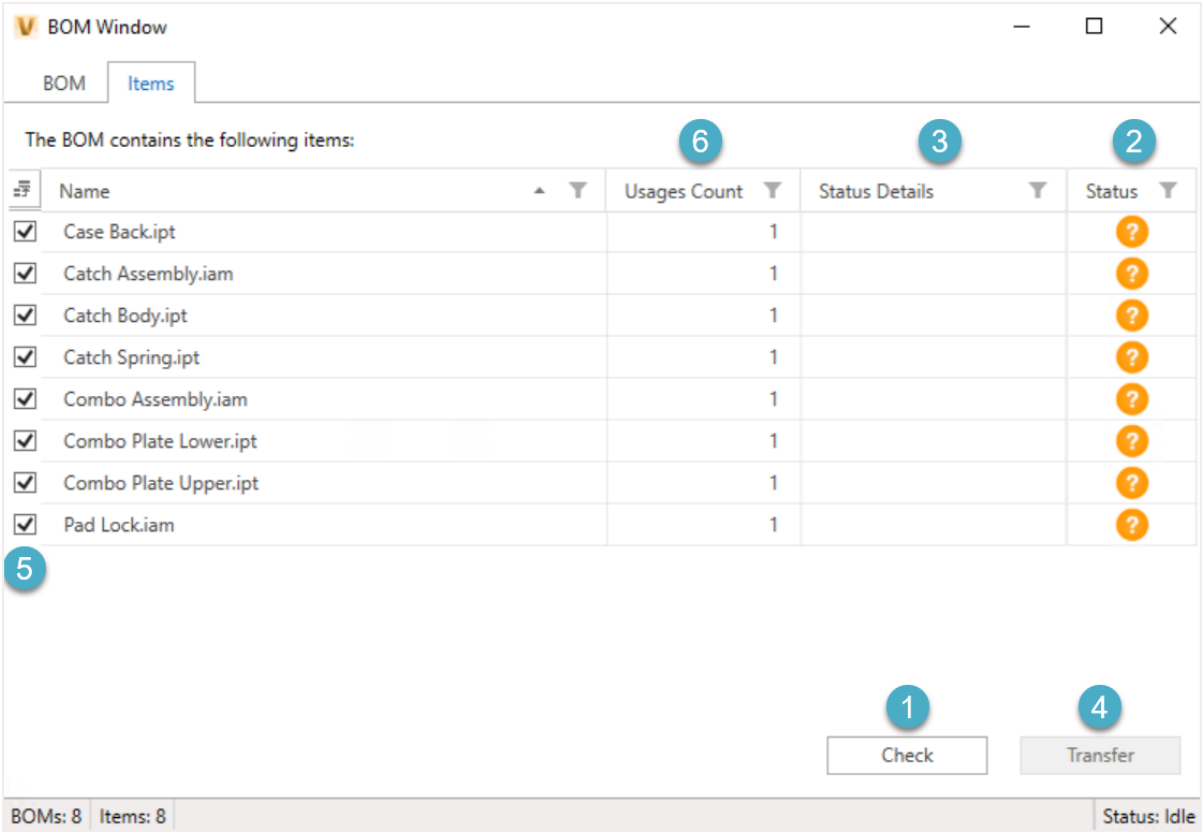
- BOMs and BomRows which already exist in the ERP are ignored during the tranfer action.
- For BOMs and BomRows marked as different the transfer action performs an update.

The result will then again be displayed.

By clicking on the **Checkbox**[6] a BOM can be skipped from the Check and Transfer operation.
The items of those BomRows will be automatically disabled in the *Item Tab*.

5.4 Item Tab

Clicking on the item Tab will display all the items of the BOM including the RootItem.
If an item is located more then once in the BOM Window, it will be displayed just once.



By pressing the **Check**[1] Button the selected Items will be compared with the ones on the ERP side.
It shows us whether the Item does not exist, whether it has differences or it is identical.
The according result will be displayed in the **Status**[2] column and the according description of the status will be displayed in the **Status Details**[3] column. The *Status Details* can also be retrieved when hovering over the status icon.
For more details about the different Status Details- and Icons see *here*.

After the Items have been checked, they can be transfered to ERP System by pressing the **Transfer**[4] Button.
Depending on the result of the check, the transfer action handles the selected items differently:

- Items which do not exist in the ERP are getting created.
- Items which already exist in the ERP are ignored during the transfer action.
- For items marked as different the transfer action performs an update.

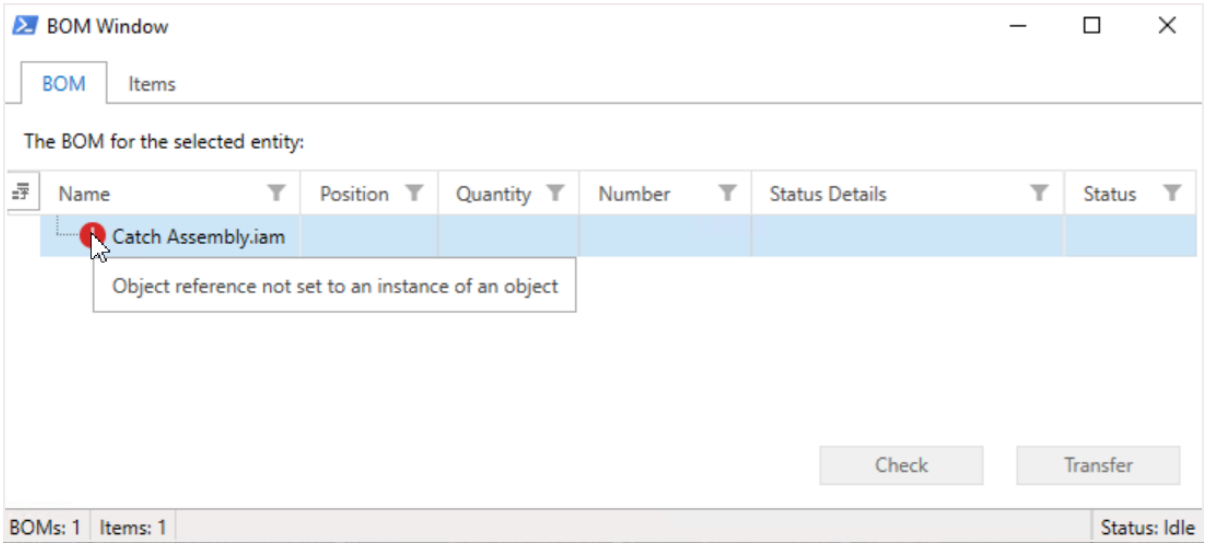
The result will then again be displayed as for the Check operation.

By clicking on the **Checkbox**[5] an Item can be skipped from the Check and Transfer operation.

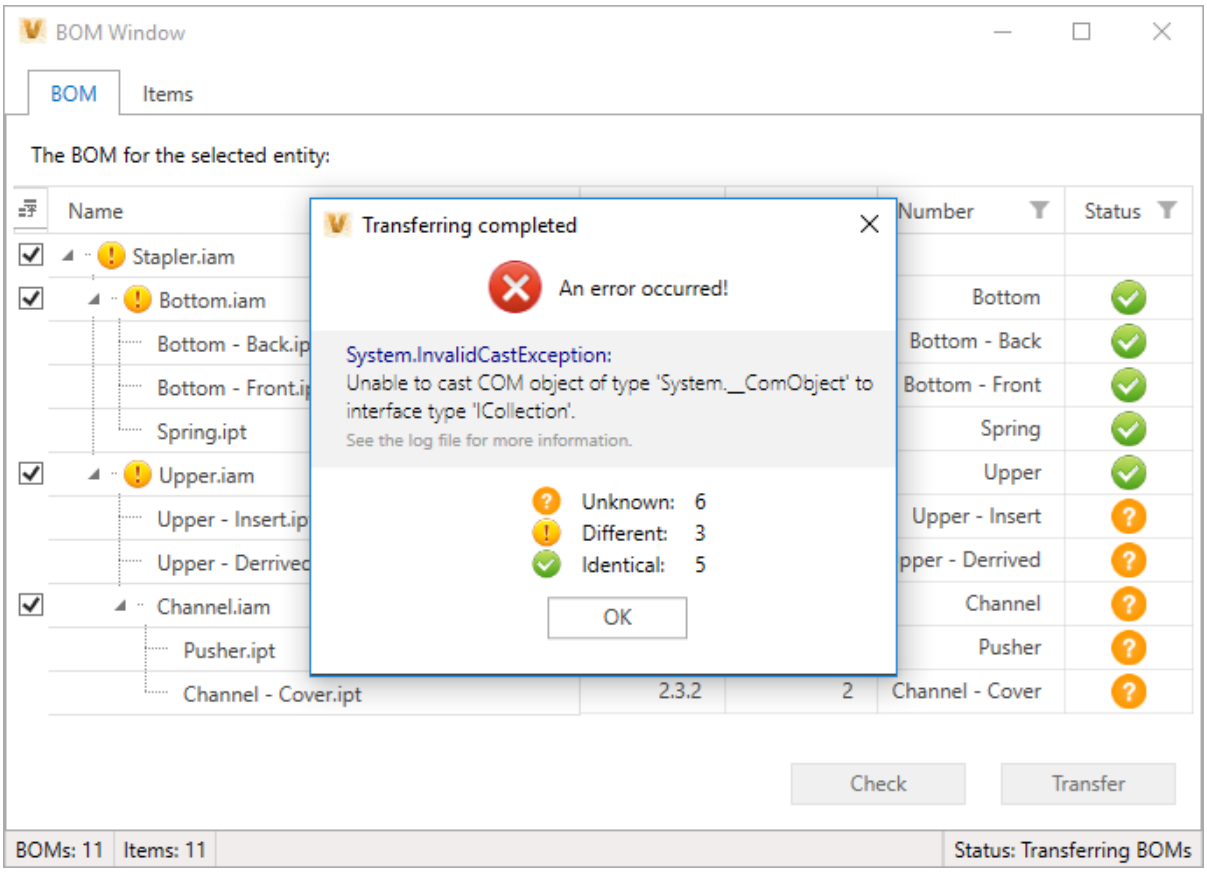
The **Usages Count**[6] (in German: Anzahl Verwendungen) can be displayed using the *FieldChooser*.
It provides us information about how often an item is located in the BOM Window.

5.5 Errors

The BOM Window can handle errors that occur when loading the BOM tree.
The affected BOMs are marked with an *Error Icon*, indicating that retrieving its rows failed.



Also errors occurring during Check or Transfer operations are handled by the BOM Window.
When an operation aborts, the remaining BOMs, rows or Items that where not processed, are automatically marked with an *Unknown Icon*.


BOM Window
BOM
Items
The BOM for the selected entity:
Name
Number
Status
Stapler.iam
Bottom.iam
Bottom - Back.ip
Bottom - Front.ip
Spring.ipt
Upper.iam
Upper - Insert.ip
Upper - Derrived
Channel.iam
Pusher.ipt
Channel - Cover.ipt
2.3.2
2
Bottom
Bottom - Back
Bottom - Front
Spring
Upper
Upper - Insert
pper - Derrived
Channel
Pusher
Channel - Cover
Transferring completed
An error occurred!
System.InvalidCastException:
Unable to cast COM object of type 'System.__ComObject' to interface type 'ICollection'.
See the log file for more information.
Unknown: 6
Different: 3
Identical: 5
OK
Check
Transfer
BOMs: 11
Items: 11
Status: Transferring BOMs

CHAPTER

SIX

CODE REFERENCE

6.1 Cmdlets

6.1.1 Objects

Entity

An Entity object is of type PSObject and represents an entity of a specific type from the ERP-System.

The $entity object is dynamically created depending on the composition of the EntityType.
Each property is attached to the PSObject as NoteProperty member with the same name and value as defined in the ERP-System.

Syntax

```
$entity._Keys
```

Following read-only properties are always available:

Type	Name	Description
PSObject	_Keys	A PSObject containing only the Key properties and its values.
PSObject	_Properties	A PSObject containing Properties, NavigationProperties and its values.

Examples

Properties of the dynamic created Entity:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
$entity = Get-ERPObject -EntitySet "Customers" -Keys @{ "CustomerID" = "ANTON" }

<#
        CustomerID : ANTON
        CompanyName : Antonio Moreno Taquería
        ContactName : Antonio Moreno
        ContactTitle : Owner
        Address : Mataderos 2312
        City : México D.F.
```

(continues on next page)

(continued from previous page)

```
        Region :
        Postalcode : 05023
        Country : Mexico
        Phone : (5) 555-3932
        Fax :
#>
```

Accessing _Keys and _Properties of the Entity:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
$entity = Get-ERPObject -EntitySet "Customers" -Keys @{ "CustomerID" = "ANTON" }
$entity._Keys

<#
        CustomerID : ANTON
#>

$entity._Properties
<#
        CompanyName : Antonio Moreno Taquería
        ContactName : Antonio Moreno
        ContactTitle : Owner
        Address : Mataderos 2312
        City : México D.F.
        Region :
        PostalCode : 05023
        Country : Mexico
        Phone : (5) 555-3932
        Fax :
#>
```

EntitySet

The EntitySet object is of type *PSObject* and represents the metadata information of a specific set of entities from the ERP-System.

Syntax

```
$entitySet.Name
```

Following properties are available:

Type	Name	Description
System.Uri	Service	The Url of the Service.
string	Name	The name of the EntitySet.
string	EntityType	The name of the associated EntityType.

EntityType

The EntityType object is of type *PSObject* and provides access to the metadata information of a specific entity from the ERP-System.

Syntax

```
$entityType.Name
```

Following properties are available:

Type	Name	Description
string	Name	The name of the EntityType.
string	Namespace	The namespace where the EntityType is located in.
System.Uri	Service	The Url of the Service.
string	EntitySet	The name of the EntitSet which the Entity belongs to.
Property[]	Keys	The Key properties which uniquely identifies the Entity.
Property[]	Properties	The available properties for the Entity.
NavigationProperty[]	Navigationproperties	The available navigation properties for the Entity.

ErpService

The ErpService object is of type *PSObject* and represents the metadata information of a specific service from the ERP-System.

Syntax

```
$erpService.Url
```

Following properties are available:

Type	Name	Description
System.Uri	Url	The Url of the Service.
string	Name	The name of the Service.
bool	Available	The current availability of the service.

Remarks

When accessing the property **Available**, powerGate will retry to retrieve the service metadata of the according service. Therefore each time when accessing the property, a new request will be send to the server in order to provide live data.

Multiplicity

The Enum of type *powerGate.Erp.Client.Multiplicity* specifies the multiplicity for *NavigationProperty* objects.

Syntax

```
[powerGate.Erp.Client]:Multiplicity
```

Note: For a complete list of supported multiplicities and for more informations see: *Multiplicity Enumeration*.

NavigationProperty

A NavigationProperty object is of type *Property* and represents the name of the relationship property from one entity to another within the ERP-System.

Syntax

```
$navigationProperty
```

Following properties are available in addition to the *Property* members:

Type	Name	Description
bool	IsCollection	Is true if the navigationproperty is a collection.
Multiplicity	Multiplicity	the multiplicity of the property.
string	TargetEntityType	Name of the target *EntityType*.

Property

The Property object is of type *string* and represents the name of specific property of an entity from the ERP-System.

Syntax

```
$property
```

Following properties are additionally available:

Type	Name	Description
string	DefaultValue	The default property value.
Type	Type	The .Net representation of the property type.
bool	IsNullable	Nullable properties are allowed to have a null value

SapConnect

The SapConnect object is of type ScriptBlock and can be used with *Connect-ERP* to work with SAP systems.

The $sapConnect variable is based on the *SapConnect* implementation from the .NET library.

Syntax

```
$sapConnect
```

Remarks

The $sapConnect variable is available *globally* in every PowerShell session where the powerGate Module is imported. It only supports connecting to the services from a single SAP system, by retrieving the **X-CSRF-Token** and **cookie** just once.

Examples

Connecting to a SAP service with CSRF protection

```
Connect-ERP -Service "http://sap.coolorange.com" -User "EX_DEMO" -Password "secret" -
↪OnConnect $global:sapConnect
```

Connecting multiple SAP systems with different CSRF-Token expiration intervals:

```
$sapConnect_System2 = {
param($settings)
        # tokens expire every 24h on this system
        $reconnectInterval = 86400 * 1000
        ([Action[powerGate.Erp.Client.ErpClientSettings]](New-Object powerGate.Erp.
↪Client.SapConnect($reconnectInterval))).Invoke($settings)
}

Connect-ERP "http://sap.coolorange.com/CATALOGSERVICE" -OnConnect $Global:sapConnect
Connect-ERP "http://sap.some_other_company.com/CATALOGSERVICE" -OnConnect $sapConnect_
↪System2
```

Customization

In case the default implementation does not fit your needs (`Download Example`) the PowerShell module and extract it to your projects location:

```
.\Modules\connections_sap.psm1
```

Freely customize the module to your needs!
To make use of the module in your current PowerShell environment it has to be imported:

```
Import-Module ".\Modules\connections_sap.psm1"
```

In order to use the customized SAP Connect for the authentication with powerGate, use the variable *$extendedSapConnect*:

```
$connected = Connect-ERP "https://sap.coolorange.com/CATALOGSERVICE" -OnConnect
↪$global:extendedSapConnect
```

Attention

If you are setting *$settings.OnCreateMessageHandler* then the *$settings.OnApplyClientHandler* will **not** be called anymore and the $extendedSapConnect will not work anymore. Therefore be sure to invoke it afterwards with the returned handler:

```
$global:erpSettings = $settings
$settings.OnCreateMessageHandler = {
        $customHandler = New-Object Your.Custom.Handler
        # Your other custom code
        $erpSetting.OnApplyClientHandler.Invoke($customHandler)
        return $customHandler
}
```

Settings

The ERPSettings object allows reading and writing the configuration from the *C:\ProgramData\coolOrange\powerGate.settings* file.
This settings file corresponds to the Vault file located at *$/powerGate.settings* (see *Settings File*).

Changing Settings

Since these settings apply to the entire *Vault ERP Integrations* for all Vault users, the settings should be changed by Vault administrators using the *Settings Dialog* (*Editor* object).

In exceptional cases the *C:\ProgramData\coolOrange\powerGate.settings* file can be edited in external tools and then checked back into Vault. This is useful when working in multiple Vaults with slightly different settings.
The Vault History tab can be used to track changes and also to revert to working versions in case of configuration problems.

Syntax

```
$ERPSettings
```

The following properties are available in addition to the derived ApplicationSettingsBase type:

Type	Name	Description
System.Collections.Gen	Services	Provides the settings for all configured OData services used to establish the connection to ERP (e.g. the `Service` URLs).For connections routed through the powerGate-Server, other *additional settings* are possible than e.g. for SAP Gateway services.
System.Collections.Gen	TypeMappings	Grants access to mapping configurations between Vault entity types and ERP entity types.This includes details such as where each ERP `Field`'s data comes from (e.g. from a `VaultProperty` or fixed-value).
Window	Editor	*ERP Integration Settings* dialog for displaying and changing the configuration Vault-wide.

In addition, ApplicationSettingsBase also provides methods for reading and writing to the *powergate.settings* file (e.g. `Reload()`, `Reset()` and `Save()`).

Examples

Retrieving all configured Services for connecting the ERP system:

```
$global:ERPSettings.Services | Format-List -Property *,@{'Label'='Type'; 'Expression'={$_
↪.GetType().Name}}

<#
Service          : http://MyADMS:8080/PGS/my_company/ErpService
Host             :
Port             :
ServicePath      :
IsDefaultSetting : False
Type             : PowerGateServerServiceSettings

Service          : http://sap.coolorange.com/sap/opu/odata/arcona6/DOCUMENT_INFO_RECORD_
↪SRV
IsDefaultSetting : False
Type             : ServiceSettings
#>
```

Extending ERP field mapping configurations for complex and calculated PowerShell values:

```
$vaultFile2ErpItemSettings = $global:ERPSettings.GetTypeMapping('File', 'Item')
$descriptionFieldSettings = $vaultFile2ErpItemSettings.FieldMappingsForCreate | Where-
↪Object { $_.ErpField -eq 'Description' }
$descriptionFieldSettings.ConvertToErpValue = {
        param($vaultFile)

        $entireDescription = $vaultFile._Description.Trim([Char]'.') # remove dots at␣
↪the beginning and end
        if($vaultFile.Description_2) {
            $entireDescription += ' - '
            $entireDescription += $vaultFile.Description_2.Substring(0,17)
        }

        foreach($invalidCharacter in  @('<', '>', ':', '*', '/', '\')) {
            if($entireDescription.Contains($invalidCharacter)) {
```

(continues on next page)

(continued from previous page)

```
                throw "The translation of the Vault descriptions failed because the␣
↪resulting ERP description can not contain the character '$invalidCharacter'"
            }
        }
        return $entireDescription
    }
}

$global:ERPSettings.GetTypeMapping('FileBomRow', 'BomRow'). `
    FieldMappingsForCreate = (
        [powerGate.Erp.Client.Properties.FieldSettings] @{

            ERPField = 'ChildNumber'
            ConvertToErpValue = {
                param($vaultBomRow)

                if(-not $vaultBomRow._PartNumber) {
                    return $vaultBomRow.'Raw Material Number'
                }
                return $vaultBomRow._PartNumber
            }
        },
        [powerGate.Erp.Client.Properties.FieldSettings] @{

            ERPField = 'ParentNumber'
            ConvertToErpValue = { param($vaultBomRow) $vaultBOM._PartNumber.ToUpper() }
        }
    )


$global:ERPSettings.TypeMappings | Select-Object -Property *, @{'Label'=
↪'FieldMappingsForCreate'; 'Expression'={ '[{0}]' -f ($_.FieldMappingsForCreate |␣
↪format-list | Out-String) }} -ExcludeProperty 'FieldMappingsForCreate' `
    | Format-List

<#
VaultEntityType        : File
ErpEntityType          : ErpServices.Services.Entities.Item
FieldMappingsForCreate : [
                             ErpField          : Number
                             VaultProperty     : File Name
                             DefaultValue      :
                             ConvertToErpValue : System.Func`2[System.Object,System.
↪Object]

                             ErpField          : Description
                             VaultProperty     :
                             DefaultValue      :
                             ConvertToErpValue : { param($vaultFile) ... }
                         ]
IsDefaultSetting       : False
```

(continues on next page)

(continued from previous page)

```
VaultEntityType        : FileBomRow
ErpEntityType          : ErpServices.Services.Entities.BomRow
FieldMappingsForCreate : [
                             ErpField          : ChildNumber
                             VaultProperty     :
                             DefaultValue      :
                             ConvertToErpValue : { param($vaultBomRow) ... }

                             ErpField          : ParentNumber
                             VaultProperty     :
                             DefaultValue      :
                             ConvertToErpValue : { param($vaultBomRow) $vaultBOM._
↪PartNumber.ToUpper() }
                           ]
IsDefaultSetting       : False
#>
```

Retrieve values for drop-down lists live from ERP and cache them for performance reasons:

This example also helps to quickly create an initial configuration with all possible ERP values, especially if the allowed ERP keys are unknown and have to be queried via API.
Then, the actual list-values to be used can easily be further adapted and saved by the Vault administrator via the Settings Dialog.

```
# e.g. directly after line: Connect-ERP -UseSettingsFromVault

$matlGroupFieldSettings = $global:ERPSettings.GetTypeMapping('File', 'BasicData'). `
    FieldMappingsForCreate | Where-Object { $_.ErpField -eq 'MatlGroup' }

if(-not $matlGroupFieldSettings.ListValues) {
    $allSapMaterialGroups = Get-ERPObjects -EntitySet 'MatlGroupLookupCollection'

    foreach ($sapMaterialGroup in $allSapMaterialGroups) {
        $matlGroupFieldSettings.ListValues.Add( (New-Object powerGate.Erp.Client.
↪Properties.ListValueEntry -Property @{
            Erp     = $sapMaterialGroup.MatlType
            Display = $sapMaterialGroup.Description
        }) )
    }
    $matlGroupFieldSettings.DefaultValue = $allSapMaterialGroups[0].MatlType
}

$matlGroupFieldSettings | Format-List
<#
ErpField          : MatlGroup
VaultProperty     :
DefaultValue      : 06
ListValues        : [
                        Display : Oil Products
                        Erp     : 06
                        Vault   :
```

(continues on next page)

(continued from previous page)

```
                        Display : SSR-Gasoline Prods
                        Erp     : 02
                        Vault   :

                        Display : Material group 2
                        Erp     : 03
                        Vault   :

                        ...
                    ]
ConvertToErpValue : System.Func`2[System.Object,System.Object]
#>
```

6.1.2 Show-BOMWindow

Cmdlets

Add-BomWindowEntity

Creates and adds a new entity to the *BOM Tab*.

Syntax

```
Add-BomWindowEntity [[-Type] <BomWindowEntityType>] [[-Properties] <Object>] [[-Parent]
↪<PSObject>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
BomRow / *Material*	Type	Specifies whether adding a new *BomRow* or an `Material (Item) </code_reference/commandlets/show-bomwindow/objects/bomrow>`	no
PSObject	Parent	For `BomRow's </code_reference/commandlets/show-bomwindow/objects/bomrow>` it's important to specify the parent BOM, where the new row will be added to	yes (not when *Type* is 'BomRow')
Hashtable / PSObject	Properties	The properties for the entity being created	yes

Return type

BomRow / *Item* ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.

Remarks

The Cmdlet's purpose is to add a new *BomRow* or *Material (Item)* to the *BOM Window*.
The entity's default *State* is "*Unknown*".

BomRow:
In general two categories of *Properties* can be provided for all newly created BomRows:

- BOM properties are properties prefixed with Bom (e.g. `@{'Bom_Number'= ..., 'Bom_Quantity'= ..., 'Bom_PositionNumber'= ...}`). They are displayed only in the BOM-Tab without the prefix.
- Entity properties are all other properties (e.g. `@{'Number'= ...}`). They are available in the BOM-Tab as well.

Custom *BOM properties* (e.g. `@{'Bom_Unit'= ... ;'Bom_ItemQuantity'= ...}`) and *Entity properties* (e.g. `@{'Description'= ... ; 'SomEntityProperty'= ...}`) can be passed and later *displayed as columns* in the BOM Window.

Material:
It is possible to create the new Material directly with *Properties*.

- The name for the new *Item* can be specified as `@{'Number'= ...}`.
- Other *custom properties* can be specified like `@{'Description'= ... ; '_Title(Item,CO)'= ...}`.

A newly added *Material* has no associated BomRow.

Examples

Adding a new material in the BOM Window:

```
$newMaterial = Add-BomWindowEntity -Type Material -Properties @{'Number'='100001'}
```

Adding a new BomRow to a BOM in the BOM Window:

```
$newBomRow = Add-BomWindowEntity -Type BomRow -Parent $bom -Properties @{'Bom_Number'=
↪'PAD LOCK';'Bom_Quantity'= 23.4;'Bom_PositionNumber'=6}
```

Adding BomRows with information from the ERP as BOM properties:

```
$erpBomRow = Get-ERPObject -EntitySet "BomItems" -Keys @{ "ChildNumber"="IWillBeDeleted";
↪"ParentNumber"="CheckBoms";"Position"=1 }
$erpBomProperties = @{}
foreach ($property in $erpBomRow._Keys.PSObject.Properties) {  #Adding _Keys with "Bom_"␣
↪prefix to pass them as Bom properties
        $erpBomProperties["Bom_"+$property.Name] = $property.Value
}
foreach ($property in $erpBomRow._Properties.PSObject.Properties) {  #Adding _Properties␣
↪with "Bom_" prefix to pass them as Bom properties
        $erpBomProperties["Bom_"+$property.Name] = $property.Value
}
```

(continues on next page)

(continued from previous page)

```
Add-BomWindowEntity -Type BomRow -Parent $bom -Properties $erpBomProperties
<#
        Bom_Number          :
        Bom_PositionNumber : 1
        Bom_Quantity        : 1
        Bom_ChildNumber     : IWillBeDeleted
        Bom_ParentNumber    : CheckBoms
        _Status             : Unknown
        _StatusDetails      :
#>
```

Adding new BomRows that exists in the ERP BOM but not in the Vault BOM and mark them with Status 'Remove':

```
function Check-Boms($boms) {
        foreach($vaultBom in $boms) {
                $erpBom = Get-ERPObject -EntitySet "Boms" -Keys @{ "ParentNumber"=
↪$vaultBom.Bom_Number } -Expand "Children"
                foreach($erpBomRow in $erpBom.Children) {
                        $vaultBomRow = $vaultBom.Children | Where-Object { $_.Bom_Number␣
↪-eq $erpBomRow.ChildNumber -and $_.Bom_PositionNumber -eq $erpBomRow.Position } |␣
↪select -First 1
                        if($vaultBomRow -eq $null) {
                                $erpBomRow_to_remove = Add-BomWindowEntity -Type BomRow -
↪Parent $vaultBom
                                $erpBomRow_to_remove | Update-BomWindowEntity -Status
↪'Remove' -StatusDetails 'Row will be deleted in ERP.'
                        }
                }
        }
}
```

Remove-BomWindowEntity

Removes an entity from the *BOM Window*.

Syntax

```
Remove-BomWindowEntity [[-InputObject] <PSObject>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
BomRow / *Item*	InputObject	An entity displayed in the BOM Window. The argument accepts pipeline input	no

Return type

empty ← On failure the Exception/ErrorMessage can be accessed using $Error.

Remarks

The Cmdlets's purpose is to remove a *BomRow* or an *Item* from the *BOM Window*.
When removing a BomRow, the row and the associated Item will be removed together!
The same applies when removing an Item that depends on a BomRow: the item and the associated *BomRow* will be removed at the same time.

Restrictions:

- *Items* that are used in multiple *BOM's* can not be removed
- Only *BomRows* without any children can be removed

Examples

Removing an item from the BOM Window:

```
$item | Remove-BomWindowEntity
```

Removing rows from a BOM:

```
foreach($bomRow in $bom.Children) {
        $bomRow | Remove-BomWindowEntity
}
```

Deletes an item when marked as "Remove" and when successfully deleted in ERP:

```
function Transfer-Items($items) {
        foreach($item in $items) {
                if($item._Status -eq 'Remove'){
                        $removed = Remove-ERPObject -EntitySet 'Materials' -Keys @{'Number'=$item._PartNumber}
                        if($removed) {
```

(continues on next page)

(continued from previous page)

```
                                $item | Remove-BomWindowEntity
                            }
                        }
                    }
}
```

Update-BomWindowEntity

Updates an entity in the *BOM Window*.

Syntax

```
Update-BomWindowEntity [[-InputObject] <PSObject>] [[-Status] <Status>] [[-
↪StatusDetails] <string>] [[-Properties] <Object>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
BOM / *BomRow* / *Item*	InputObject	An entity displayed in the `BOM Window </bom_window>`. The argument accepts pipeline input	no
Status	Status	The new `/bom_window/status` of the entity	yes
String	StatusDetails	The new `Status Details </bom_window/status>` of the entity	yes
Hashtable / PSObject	Properties	The properties to add or update for the entity	yes

Return type

empty ← On failure the Exception/ErrorMessage can be accessed using $Error.

Remarks

The Cmdlet's purpose is to update the *Properties*, *Status* and *StatusDetails* of *Bom*, *BomRow* and *Item* in the *BOM Window*:

Status and StatusDetails:
When updating the *Status* the according custom *StatusDetails* is reset when no *-StatusDetails* is specified. However, the tooltip of the *Status Icon* will hold a default value.
Additionally, when updating the *Status* of an entity, the progress in the *BOM Window* will be automatically incremented for that entity.

Properties:
It is possible to pass new *Properties*, update existing ones or remove them from the entity as required.

BomRow:
In general two categories of *Properties* can be provided for BomRows:

- *BOM properties* are properties prefixed with Bom (e.g. `@{'Bom_Number'= ..., 'Bom_Quantity'= ..., 'Bom_PositionNumber'= ...}`). They are displayed only in the BOM-Tab without the prefix.
- *Entity properties* are all other properties (e.g. `@{'_Name'= ...}`). They are displayed in both the BOM-Tab and the Items-Tab.

Custom *BOM properties* (e.g. `@{'Bom_Unit'= ... ;'Bom_ItemQuantity'= ...}`) and *Entity properties* (e.g. `@{'Description'= ... ; 'Title (Item,CO)'= ...}`) can be passed and later *displayed as columns* in the BOM Window.
They are replacing the current set of *BomRow* properties or the ones of the associated *Item*.

Material:

- The *standard property* `@{'_Name'= ...}` can be passed in order to update the item number if required.
- Other properties are replacing the current set of *custom properties* for instance `@{'Description'= ... ; '_Title(Item,CO)'= ...}`.

Examples

Updating the Status and StatusDetails of a BOM :

```
$bom | Update-BomWindowEntity -Status 'New' -StatusDetails 'BOM Header does not exist in␣
↪ERP and will be created!'
```

Setting and updating properties of an Item:

```
$item | Update-BomWindowEntity -Properties @{"SomeCustomProperty" = "true";
↪"SomeOtherCustomProperty" = 6.66}
$item.'_Title(Item,CO)' = "A new title"
$item | Update-BomWindowEntity -Properties $item
```

Setting new properties to a BomRow by adding a note property member to the object:

```
$bomRow | Add-Member -MemberType NoteProperty -Name "Bom_SomeNewProperty" -Value "A new␣
↪BOM property"
$bomRow | Add-Member -MemberType NoteProperty -Name "SomeNewProperty" -Value "A new␣
↪Entity property"
$bomRow | Update-BomWindowEntity -Properties $bomRow
```

Updating the Item's Status and StatusDetails, depending on the existence and equality to the material in ERP:

```
function Check-Items($items) {
        foreach($item in $items) {
                $erpItem = Get-ERPObject -EntitySet 'Materials' -Keys @{'Number'=$item._
↪PartNumber}
                if($erpItem -eq $null) {
                        $item | Update-BomWindowEntity -Status 'New'
                }else{
                        if( $item._Description -cne $erpItem.Description) {
                                $item | Update-BomWindowEntity -Status 'Different' -
↪StatusDetails "Description is Different!"
                        }else{
```

(continues on next page)

(continued from previous page)

```
                                $item | Update-BomWindowEntity -Status 'Identical'
                            }
                        }
                }
}
```

Updating a standard BOM property on a BomRow:

```
$bomRow | Update-BomWindowEntity -Properties @{"Bom_PositionNumber" = 4}
```

The *BOM Tab* consists of *BOMs* and *BomRows* and the *Item Tab* consists of the *Items* instead.

Availability restricted to certain functions

The Cmdlets are **only** available in the context of the *Show-BOMWindow* cmdlet.
This means, that they can only be used in the functions *Check-Boms* , *Transfer-Boms* , *Check-Items* , *Transfer-Items*.

To add, change or remove entities in the BOM Window the following Cmdlets should be used:

Name	Description
Add-BomWindowEntity	Adds a new entity to the BOM Window.
Update-BomWindowEntity	Updates an existing entity in the BOM Window.
Remove-BomWindowEntity	Removes an entity from the BOM Window.

Objects

Bom

The BOM object is of type PSObject and gets generated based on the result of the function *Get-BomRows* .
Therefore $boms are usually extended powerVault Files or powerVault Items.

Syntax

```
$bom.Children
```

Following properties are always available :

Type	Name	Description
Bom-Row[]	Children	All the child rows returned from the function *Get-BomRows* as Array of *BomRows* .
Status	_Status	The current status of the BOM in the *BOM Tab*.
string	_Status-Details	The BOM's *Status Details* which are displayed in the *Status Details* column and when hovering over the BOM status in the *BOM Tab*.
string	_Name	The name of the BOM header, that gets displayed in the *BOM Tab* column 'Name'.

Remarks

A list of $boms gets passed into the functions:

- *Check-Boms*
- *Transfer-Boms*

Entity Properties

The BOM object directly provides all the information of the Header *Item* , for instance the *Name* of the according root item.
These are all the properties declared with simple names (not starting with the prefix *'Bom_'*).

At least following **standard properties** should always be available: `_Name`.

All the other properties are **custom properties** that can be *displayed* in both the *BOM Tab* and the *Item Tab*.
For example `Description`, `_Category` or all the other dynamically generated members of a powerVault File or powerVault Item.

BOM Properties

This are all the properties with the prefix *'Bom_'*, like e.g. `Bom_Number`, `Bom_PositionNumber`, `Bom_Unit`, ...
In contrast to *BomRows* , the BOM object can only provide **custom properties** and there are **no standard properties** at all!
BOM properties can only be *displayed* in the *BOM Tab*.

BomRow

The BomRow object is of type PSObject and represents an individual row in a *BOM* .

$bomRows are generated based on the result of the *Get-BomRows* function and therefore they are usually extended powerVault FileBomRows or powerVault ItemBomRows.

Syntax

```
$bomRow.Bom_Number
```

Following properties are always available :

Type	Name	Description
Status	_Status	The current status of the BomRow that is displayed in the 'Status' column in the *BOM Tab*.
string	_StatusDetails	The BomRow's *Status Details* which are displayed in the *Status Details* column and when hovering over the status icon in the *BOM Tab*.
string	Bom_Number	The number of the row, that gets displayed in the *BOM Tab* column 'Number'.
double / string	Bom_PositionN	The position in the BOM, that gets displayed in the *BOM Tab* column 'Position'.
double	Bom_Quantity	The quantity of the row, that gets displayed in the `*BOM Tab* column 'Quantity'.
string	_Name	The name of the according item of the row, that gets displayed in the *BOM Tab* column 'Name'.

Remarks

$bomRows of a specific BOM can be retrieved directly via the `Children` property on the according *$bom* object.

BOM Properties

BOM properties can only be displayed in the *BOM Tab*, because they can provide additional information for an *Item* instance in the BOM.
This could simply be the different *positions* of the same instance of an *Item* in the whole BOM.
These properties are declared with the *'Bom_'* prefix.

At least following **standard properties** should always be available: `Bom_Number`,`Bom_PositionNumber` and `Bom_Quantity`.

All the other BOM properties are **custom properties** that can be *additionally displayed* in the *BOM Tab* without the *'Bom_'* prefix.
For example `Bom_Unit` or all the other dynamically generated *'Bom_'* members of a powerVault FileBomRow or powerVault ItemBomRow.

Entity Properties

Entity properties are displayed in both: the *BOM Tab* and the *Item Tab*, and they provide directly the information of the according *Item* .
This could be e.g. the item's *name*, which is always the same for all the different instances of an item in the BOM.
Entity properties are all the properties declared with simple names (not starting with the prefix *'Bom_'*) e.g. `_Name`, `Description`, `_Category`, ...

BomRows with a linked *Item* do provide only one **standard properties**: `_Name`.

All the other properties are **custom properties** that can be *additionally displayed* in both the *BOM Tab* and *Item Tab*
For example `Description`, `_Category` or all the other dynamically generated members of a powerVault File or powerVault Item.

Item

The Item object is of type *PSObject* and gets generated based on the Item information from the result of the *GetBomRows* function.

Therefore $items are usually extended powerVault Files or powerVault Items.

Syntax

```
$item._Name
```

Following properties are always available :

Type	Name	Description
Status	_Status	The current status of the Item that is displayed in the 'Status' column in the *Item Tab*.
string	_StatusDetails	The Item's *Status Details* which are displayed in the *Status Details* column and when hovering over the status icon in the *Item Tab*.
string	_Name	The name of the Item, that gets displayed in the *Item Tab* column 'Name'.

Remarks

A list of $items gets passed into the functions:

- *Check-Items*
- *Transfer-Items*

Entity Properties

Entity properties provide all the properties of an entity and are displayed in the *Item Tab* as well as on the according rows in the *BOM Tab*.

At least following **standard property** should always be available: `_Name`.

All the other properties are **custom properties** that can be *additionally displayed* in the *Item Tab*.
For example `Description`, `_Category` or all the other dynamically generated members of a powerVault File or Item .

Types which are used in the *Required Functions* of the Bom-Window:

- *BOM*
- *BomRow*
- *Item*

Required Functions

Check-Boms

This function is executed when clicking the "Check" button within the *BOM Tab*.
All the *BOMs* from the BOM Window can be checked against ERP.

Syntax

```
Check-Boms [-Boms] <Bom[]>
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
Bom []	Boms	The Boms which should be checked	input	yes	

Return type

void

Remarks

Each **unique** *BOM* shown in the BOM Window is passed to this function.

It's recommended to update the *status* of all the BOMs and their rows with the *Update-BomWindowEntity* cmdlet. Only in case of *connection problems* with *ERP cmdlets* the status of the according BOMs and rows is automatically set to *Error*.

When an exception is thrown within this function, the BOM Window *shows the Exception* message of the terminated Check operation.
The *status* of all the *BOMs* and their *Children* that where not updated, gets automatically changed to *Unknown*.

Examples

Checking whether Vault file BOMs exist in the ERP system: BOMs are marked as "New", "Identical" and with "Error" icons:

```
function Check-Boms($boms) {
   foreach($vaultBom in $boms) {
      if($vaultBom._CategoryName -eq 'Base') {
         $vaultBom | Update-BomWindowEntity -Status 'Error' -ToolTip "BOMs for files of␣
↪the category group 'Base' should not be processed"
      }

      $erpBom = Get-ERPObject -EntitySet 'Boms' -Keys @{ 'ParentNumber'= $vaultBom.Bom_
↪Number }

      if($? -and -not $erpBom) {
         Update-BomWindowEntity -InputObject $vaultBom -Status New -StatusDetails 'BOM␣
↪does not exist in ERP'
      }
      if($erpBom) {
         Update-BomWindowEntity -InputObject $vaultBom -Status Identical -StatusDetails
↪'BOM exists in ERP'
      }
   }
}
```

Note: In case of connection problems during the *Get-ERPObject* cmdlet, the status of the currently iterated `$vaultBom` object is automatically set to *'Error'*.
Further exception details can be retrieved via automatic variables like `$?` and `$Error[0]`.

Checking whether the Vault file BomRows are "Identical" to those of the ERP BOMs, whether there are "New" once or rows that should be "Removed":

```
function Check-Boms($boms) {
   foreach($vaultBom in $boms) {
      $erpBom = Get-ERPObject -EntitySet 'Boms' -Keys @{ 'ParentNumber'= $vaultBom.Bom_
↪Number } -Expand 'Children'

      #iterating ERP-BOM rows and checking if they exist in the Vault-BOM
      foreach($erpBomRow in $erpBom.Children) {
         $vaultBomRow = $vaultBom.Children | Where-Object { $_.Bom_Number -eq $erpBomRow.
↪ChildNumber -and $_.Bom_PositionNumber -eq $erpBomRow.Position } | select -First 1
         if($vaultBomRow -eq $null) {
            $bomRow = Add-BomWindowEntity -Type BomRow -Parent $vaultBom -Properties @{
               'Bom_Number'=$erpBomRow.ChildNumber;
               'Bom_PositionNumber'=$erpBomRow.Position;
               'Bom_Quantity'=$erpBomRow.Quantity;
            }
            Update-BomWindowEntity -InputObject $bomRow -Status Remove
         }
         elseif($vaultBomRow.Bom_Quantity -cne $erpBomRow.Quantity) {
            Update-BomWindowEntity -InputObject $vaultBomRow -Status Different -
↪StatusDetails "'Quantity' is different:`nVault '$($vaultBomRow.Bom_Quantity)'`nERP '$(
↪$erpBomRow.Quantity)'"
         }
         else {
            Update-BomWindowEntity -InputObject $vaultBomRow -Status Identical
         }
      }
      #iterating Vault-BOM rows and checking if they exist in the ERP-BOM
      if($vaultBom.Children) {
         foreach($vaultBomRow in $vaultBom.Children) {
            $erpBomRow = $erpBom.Children | Where-Object { $_.ChildNumber -eq
↪$vaultBomRow.Bom_Number -and $_.Position -eq $vaultBomRow.Bom_PositionNumber } |␣
↪select -First 1
            if( $erpBomRow -eq $null) {
               Update-BomWindowEntity -InputObject $vaultBomRow -Status New
            }
         }
      }
   }
}
```

Check-Items

This function is executed when clicking the "Check" button within the *Item Tab*.
All the *Items* from the BOM Window can be checked against ERP.

Syntax

```
Check-Items [-Items] <Item[]>
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
Item []	Items	The Items which should be checked	input	yes	

Return type

void

Remarks

Each **unique** *Item* shown in the BOM Window will be passed to this function.

It's recommended to update the *Status* of all the Items with the *Update-BomWindowEntity* cmdlet.
Only in case of *connection problems* with *ERP cmdlets* the status of the according items is automatically set to *Error*.

When an exception is thrown within this function, the BOM Window *shows the Exception* message of the terminated Check operation.
The *Status* of all the *Items* that where not updated, gets automatically changed to *Unknown*.

Examples

Checking whether Vault Items exist in the ERP system: Items are marked as "New", "Identical" and with "Error" icons:

```
function Check-Items($items) {
   foreach($item in $items) {
      if(-not $item._PartNumber) {
         $item | Update-BomWindowEntity -Status 'Error' -StatusDetails 'Number is empty'
      }

      $erpItem = Get-ERPObject -EntitySet 'Materials' -Keys @{'Number' = $item._
↪PartNumber} -Expand 'Descriptions'

      if($? -and -not $erpItem) {
        $item | Update-BomWindowEntity -Status 'New' -StatusDetails 'Item does not exist␣
```

(continues on next page)

(continued from previous page)

```
↪in ERP'
      }
      if($erpItem) {
         $item | Update-BomWindowEntity -Status 'Identical' -StatusDetails 'Item exists␣
↪in ERP'
      }
   }
}
```

Note: In case of connection problems during the *Get-ERPObject* cmdlet, the status of the currently iterated `$item` object is automatically set to *'Error'*.
Further exception details can be retrieved via automatic variables like `$?` and `$Error[0]`.

Get-BomRows

This function is called recursively for loading the BOM tree.
The returned *BomRow* are the children of the *BOMs* shown in the BOM Window.

Syntax

```
Get-BomRows [-BomHeader] <Item>
```

Parameters

Type	Name	Description	in-put/output	Manda-tory	Default value
Bom-Row	Bom-Header	The parent element for which bom rows must be returned	input	yes	

Return type

PSObject[] ← typically an array of powerVault FileBomRows or ItemBomRows or *empty*.
It is recommended for every BomRow to provide at least the following *properties* :

- `_Name` ← serves as the unique identifier for every *BOM* and *Item*
- `Bom_Number`
- `Bom_PositionNumber`
- `Bom_Quantity`

Remarks

The first time the function is called for the root entity, which is passed to the *Show-BomWindow* *-Entity* parameter. For each returned *BomRow*, the function is then invoked again recursively until no more new rows are returned and the complete BOM structure is loaded.
In order to signal a leaf of the BOM tree (a BomRow that has no more children) the function should return an *empty array*.

Note: The function will only be called once for BOMs that are located multiple times in the BOM tree.

In general two categories of *properties* can be provided for all the BomRows:

- BOM properties are properties prefixed with 'Bom_' (e.g. `Bom_PositionNumber`, `Bom_Unit`, ...). They are displayed only in the BOM-Tab without the prefix.
- Entity properties are all other properties (e.g. `Description`, `_Category`, ...). They are available in both the BOM-Tab and the Items-Tab.

Since every BomRow is a *PSObject*, custom *BOM properties* and *Entity properties* can be attached and later *displayed as columns* in the BOM Window.

When an exception is thrown within this function, the passed *$bomHeader* element is *marked as failed* in the BOM Window.

Examples

Return a single BOM row with custom properties:

```
function Get-BomRows($bomHeader){
   if($bomHeader._Name -eq 'BomRow'){
      return @()
   }
   $bomRow = New-Object -TypeName PSObject -Property @{
      #Recommended properties
      _Name='BomRow'
      Bom_Quantity=10
      Bom_PositionNumber=1
      Bom_Number='BomRow'
      _Category='Part' #Additional entity property
      Bom_Unit='Each' #Additional BOM property
   }
   return @($bomRow)
}
```

Return all BOM rows of a Vault Item:

```
function Get-BomRows($vaultItem) {
   $bomRows = Get-VaultItemBom -Number $vaultItem._Number
   $bomRows | foreach {
      if($vaultItem.Bom_RowOrder) {
         Add-Member -InputObject $_ -Name Bom_RowOrder -Value ("{0}.{1}" -f $vaultItem.
↪Bom_RowOrder,$_.Bom_RowOrder) -MemberType NoteProperty -Force
      }
   }
   return $bomRows
}
```

Return all BOM rows of a Vault file and warn about disabled Structured View:

```
function Get-BomRows($file) {
   $fileBom = Get-VaultFileBom -File $file._FullPath

   if(($fileBom | select -ExpandProperty 'Bom_PositionNumber' -Unique) -eq '') {
      throw "The BOM of file '$($file._Name)' contains $($fileBom.Count) rows without␣
↪position number! Please checkout the file in Inventor, enable the Structured View and␣
↪re-checkin the file to Vault!"
   }
   return $fileBom
}
```

Return all BOM rows of a Vault Item or File, by handling purchased, virtual and non-master model state components:

```
function Get-BomRows($bomHeader) {
   if($bomHeader._EntityTypeID -notin 'ITEM','FILE') {
      return @()
   }
   if($bomHeader._EntityTypeID -eq "ITEM") {
      $bomRows = Get-VaultItemBom -Number $bomHeader._Number
   }
   if($bomHeader._EntityTypeID -eq "FILE") {
      if($bomHeader.Bom_Structure -eq 'Purchased') {
         return @()
      }
      $bomRows = Get-VaultFileBom -File $bomHeader._FullPath -ModelStateType $bomHeader.
↪Bom_ModelState
   }

   foreach($bomRow in $bomRows) {
      if($null -eq $bomRow._EntityTypeID) {
         # Virtual components - have no file properties
         Add-Member -InputObject $bomRow -Name "BomType" -Value "Virtual" -MemberType␣
↪NoteProperty -Force
      }
      if($bomHeader.Bom_RowOrder) {
         Add-Member -InputObject $bomRow -Name Bom_RowOrder -Value ("{0}.{1}" -f
↪$bomHeader.Bom_RowOrder,$bomRow.Bom_RowOrder) -MemberType NoteProperty -Force
      }
   }
   return $bomRows
}
```

Transfer-Boms

This function is executed when clicking the "Transfer" button within the *BOM Tab*.
All the *BOMs* from the BOM Window can be transferred to ERP.

Syntax

```
Transfer-Boms [-Boms] <Bom[]>>
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
Bom []	Boms	The BOMs which should be transfered	input	yes	

Return type

void

Remarks

Each **unique** *BOM* shown in the BOM Window with another *Status* then *Unknown* is passed to this function.

It's recommended to update the *Status* of all the BOMs and their rows with the *Update-BomWindowEntity* cmdlet.
Only in case of *connection problems* with *ERP cmdlets* the status of the according BOMs and rows is automatically set to *Error*.

When an exception is thrown within this function, the BOM Window *shows the Exception* message of the terminated Transfer operation.
The *Status* of all the *BOMs* and their *Children* that where not updated, gets automatically changed to *Unknown*.

Examples

Creating all the Vault Item BOMs with Status "New" in ERP: Upon successful deep-creation they are marked as "Identical" whereby BOMs that should not be transferred are skipped:

```
function Transfer-Boms($boms) {
   foreach($vaultBom in $boms) {
      if($vaultBom._Status -eq 'Identical' -or $vaultBom._Status -eq 'Error'){
         $vaultBom | Update-BomWindowEntity -Status $vaultBom._Status -StatusDetails
↪$vaultBom._StatusDetails
      }
      elseif($vaultBom._Status -eq 'New'){
         $result = Add-ERPObject -EntitySet 'Boms' -Properties @{
            'ParentNumber'= $vaultBom._PartNumber
            'Children' = @( $vaultBom.Children | foreach-object {
```

(continues on next page)

(continued from previous page)

```
                    @{
                        'ParentNumber' = $vaultBom._PartNumber
                        'ChildNumber' = $_.Bom_Number
                        'Position' = [int]($_.Bom_PositionNumber)
                        'Quantity' = $_.Bom_Quantity
                    }})
            }
            if($result){
                $bom | Update-BomWindowEntity -Status 'Identical' -StatusDetails 'BOM␣
↪created in ERP'
            }
        }
    }
}
```

Note: In case of connection problems during the *Add-ERPObject* cmdlet, the status of the currently iterated `$vaultBom` object is automatically set to *'Error'*.
Further exception details can be retrieved via automatic variables like `$?` and `$Error[0]`.

Updating all BomRows with Status "Different" in ERP: setting Status and StatusDetails on the according row:

```
function Transfer-Boms($boms) {
    foreach($vaultBom in $boms) {
        foreach($vaultBomRow in $vaultBom.Children){
            if($vaultBomRow._Status -eq 'Different'){
                $erpBomRow = Update-ERPObject -EntitySet 'BomItems' -Keys $vaultBomRow._Keys␣
↪-Properties @{'Quantity'=$vaultBomRow.Bom_Quantity}
                if($erpBomRow){
                    $vaultBomRow | Update-BomWindowEntity -Status 'Identical' -StatusDetails
↪'BomRow updated in ERP'
                }
            }
        }
    }
}
```

Note: In case of connection problems during the *Update-ERPObject* cmdlet, the status of the currently iterated `$vaultBomRow` object is automatically set to *'Error'*.
Further exception details can be retrieved via automatic variables like `$?` and `$Error[0]`.

Transfer-Items

This function is executed when clicking the "Transfer" button within the *Item Tab*.
All the *Items* from the BOM Window can be transferred to to ERP.

Syntax

```
Transfer-Items [-Items] <Item[]>
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
Item []	Items	The Items which should be transfered	input	yes	

Return type

void

Remarks

Each **unique** *Item* shown in the BOM Window with another *Status* then *Unknown* will be passed to this function.

It is recommended to update the *Status* of all the Items with the *Update-BomWindowEntity* cmdlet.
Only in case of *connection problems* with *ERP cmdlets* the status of the according items is automatically set to *Error*.

When an exception is thrown within this function, the BOM Window *shows the Exception* message of the terminated Transfer operation.
The *Status* of all the *Items* that where not updated, gets automatically changed to *Unknown*.

Examples

Removing all the Vault Items with Status "Remove" in ERP: after successful deletion the Item disappears in the BOM Window, otherwise it is marked as "Error":

```
function Transfer-Items($items) {
   foreach($item in $items) {
      if($item._Status -eq 'Remove'){
         $removed = Remove-ERPObject -EntitySet 'Materials' -Keys @{'Number'=$item._
↪PartNumber}
         if($removed) {
            $item | Remove-BomWindowEntity
         }
         else {
            $item | Update-BomWindowEntity -Status 'Error' -StatusDetails "Error occured␣
↪when removing item: $($Error[0].Exception.Message)"
```

(continues on next page)

(continued from previous page)

```
        }
    }
  }
}
```

Note: In case of connection problems during the *Remove-ERPObject* cmdlet, the status of the currently iterated `$item` object is automatically set to *'Error'* but it can still be assigned manually.
Exception details can be retrieved via automatic variables like `$?` and `$Error[0]`.

Following functions have to be available within the current PowerShell Runspace, in order to implement the behaviour of the BOM Window:

- *Check-Boms*
- *Check-Items*
- *Get-Bomrows*
- *Transfer-Boms*
- *Transfer-Items*

Opens the *BOM Window* for checking and transferring BOM's and materials of Vault Files or Items.

Syntax

```
Show-BOMWindow [-Entity] <PSObject> [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
powerVault File / powerVault Item	Entity	The main Vault entity for which the BOM will retrieved and displayed.	no

Return type

void

Remarks

The *BOM Window* retrieves the BOM tree for the passed **-Entity** by calling the method *Get-BomRows* recursively. Such root elements must provide at least *entity properties* as those provided by powerVault File or Item objects. However also *custom BOM properties*, such as those available on powerVault FileBomRows and ItemBomRows, can be passed and *displayed as columns*.

In order to customize the Check- and Transfer operations for BOMs and Items, the following functions are required additionally and must be implemented with custom logic:

Required Functions

- *Check-Boms*
- *Check-Items*
- *Get-Bomrows*
- *Transfer-Boms*
- *Transfer-Items*

Examples

Showing the BOM of a Vault Item:

```
$item = Get-VaultItem -Number '100001'

function Get-BomRows($vaultItem) {
   return (Get-VaultItemBom -Number $vaultItem._Number)
}

function Check-Items($items) {

}

function Transfer-Items($items) {

}

function Check-Boms($boms) {

}

function Transfer-Boms($boms) {

}

Show-BomWindow -Entity $item
```

Opening the Bom-Window for a Vault file with simple implementation of Check- and Transfer operations:

```
$file = Get-VaultFile -Properties @{'File Name'='MSB-Weld.iam'}
if($file._HasModelState) {
   $allModelStates = Get-VaultFileBom -File $file._FullPath -ModelStateType All
   $file = $allModelStates | Where-Object { $_.Bom_ModelState -eq 'Face Machining +␣
↪Treading' }
}

function Get-BomRows($file) {
   return (Get-VaultFileBom -File $file._FullPath)
}

function Check-Items($items) {
   foreach($item in $items) {
      $item | Update-BomWindowEntity -Status 'New'
```

(continues on next page)

(continued from previous page)

```
    }
}

function Transfer-Items($items) {
    foreach($item in $items) {
        $item | Update-BomWindowEntity -Status 'Identical'
    }
}

function Check-Boms($boms) {
    foreach($bom in $boms) {
        $bom | Update-BomWindowEntity -Status 'New'
    }
}

function Transfer-Boms($boms) {
    foreach($bom in $boms) {
        $bom | Update-BomWindowEntity -Status 'Identical'
    }
}

Show-BomWindow -Entity $file
```

Force the Bom-Window to be displayed in german culture:

```
[CultureInfo]::DefaultThreadCurrentUICulture = [CultureInfo]::CreateSpecificCulture('de-
↪DE')
$file= Get-VaultFile -File '$/Designs/Pad Lock/Assemblies/Catch Assembly.iam'

Show-BomWindow -Entity $file
```

6.1.3 Add-ERPMedia

Cmdlet to create a streamable entity with a given media file.

Syntax

```
Add-ERPMedia [[-EntitySet] <String>] [[-File] <String>]] [[-Properties] <Object>] [[-
↪ContentType] <String>]] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	EntitySet	The EntitySet name where the Media Link Entity is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
String	File	Path to the file to upload	no
Hashtable / PSObject	Properties	The properties for the entity being created. Those are passed as Slug-Header to the server	yes
String	ContentType	Specifies the content type of the web request. If no content type is provided Add-ERPMedia sets the content type depending on the file extension. (e.g .txt 'text/plain', .pdf 'application/pdf' ...)	yes

Return type

Entity ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

The Cmdlet is used to create Media Link Entries (MLEs) with the request body containing the Media Resource (MR) and the Content-Type header indicating its media type.
In other words it will create a streamable entity (Media Link Entry) and upload it together with the specified file (Media Resource).

The **ContentType** is used to specify the nature of the file currently being handled. With the appropriate content type the web browser can open the file with the proper extension/plugin.

- If no content type is set the Cmdlet will determine the content type depending on the file type.
- If the content type contains text (e.g text/plain, text/html...) as type or json , xml (e.g application/json, application/xml...) as subtype, then the content of the file is uploaded to the server as UTF-8 Encoded text.

The **Properties** are passed as Slug-Header to the server, in the augmented BNF syntax.
Please note, that the field-values are passed in JSON-format (depending on the OData-version) to the server. The Slug header is send as defined in Atom Publishing Protocol by encoding the data to UTF-8 and later using percent encoding (for all octets outside the ranges %20-24 and %26-7E)!

Note: All Properties are formatted in **following format**: `Property1='SomeText',Property2=666`
This format is supported by SAP and powergateserver. Note that other ERP systems could expect data in different format!

Since OData-Servers can create the Media Link Entry completely without using the Slug-Header, the server will respond with the created Media Link Entry, or at least by passing a link to the created MLE in the Location header.
In both cases the cmdlet will take care about returning the newly created Media Link Entry.

Examples

In the following example we are using the public OData Services (http://services.odata.org) for demonstration purposes:

Create Media Link Entry (MLE)

```
Connect-Erp -Service "http://services.odata.org/V4/OData/(S(du4oaehbpzqh2eznhygi1xkg))/
↪OData.svc"
Add-ERPMedia -EntitySet "Advertisements" -File "C:\Temp\TestMedia.txt"
<#
        ID : 7fb23d89-f6b0-4010-b789-2d6e39b62187
        Name :
        AirDate : 01.01.2000 00:00:00 +00:00
#>
```

In the following examples we are using a custom plugin for the `powergateserver:index`:

Create Media Link Entry (MLE) with specific properties

```
Connect-Erp -Service "http://localhost:8080/powerGate.Tests/TestService"
Add-ERPMedia -EntitySet "Files" -Properties @{"Id"=2;"FileName"="SampleFile";"Size"=999}␣
↪-File "C:\Temp\TestMedia.txt"
<#
        Id : 2
        Created : 01/01/0001 00:00:00
        Size : 999
        Value : {}
        FileName : SampleFile
#>
```

Error handling, analyze why the Media Link Entry could not be created, by using $Error

```
Connect-ERP https://services.odata.org/V3/Northwind/Northwind.svc
$mediaEntity = Add-ERPMedia -EntitySet Categories -File C:\Temp\CategoryIcon.png

if(-not $mediaEntity){
        $Error[0].Exception #"EntityType Category is not streamable!"
}
```

6.1.4 Add-ERPObject

Cmdlet to create a new entity and transfers it to an ERP-System.

Syntax

```
Add-ERPObject [[-EntitySet] <String>] [[-Properties] <Object>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Properties	The properties for the entity being created	no

Return type

Entity ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

This Cmdlet is used to create an entity in the ERP-System.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Create an orange Juice

```
Connect-Erp -Service "http://services.odata.org/V4/OData/(S(fa2g1e4kjczcfnog0asxmvhh))/
↪OData.svc"
$entity = Add-ERPObject -EntitySet "Products" -Properties @{
        "ID "= 5;
        "Name" = "Orange Juice";
        "Description" = "The original Orange Juice. Refreshing!";
        "ReleaseDate" = "2006-08-04T00 =00 =00Z";
        "DiscontinuedDate" = $null;
        "Rating" = 3;
        "Price" = 22.8
}
```

Create a person with details (deep create)

```
Connect-Erp -Service "http://services.odata.org/V4/OData/(S(fa2g1e4kjczcfnog0asxmvhh))/
↪OData.svc"
```

(continues on next page)

(continued from previous page)

```
$entity = Add-ERPObject -EntitySet "Products" -Properties @{
        "ID" = 3;
        "Name" = "Augustin Hodorsson";
        "PersonDetail" = @{
                "PersonID" = 3;
                "Age" = 23;
                "Gender" = $true;
                "Phone" = "(307) 555-4680123"
        }
}
```

Create a Product entity using *New-ErpObject*

```
Connect-Erp -Service "http://services.odata.org/V4/OData/(S(fa2g1e4kjczcfnog0asxmvhh))/
↪OData.svc"
$product = New-ERPObject -EntityType 'Product'-Properties @{"ID"=11;"Name"="Forst beer"}
Add-ERPObject -EntitySet "Products" -Properties $product

<#
        ID : 11
        Name : Forst beer
        Description :
        ReleaseDate : 01.01.0001 00:00:00 +00:00
        DiscontinuedDate :
        Rating : 0
        Price : 0
#>
```

Error handling, analyze the AggregateException **why the entity could not be created due to several errors, by using $Error**

```
Connect-ERP "https://services.odata.org/V3/(S(fa2g1e4kjczcfnog0asxmvhh))/OData/OData.svc"
$entity = Add-ERPObject -EntitySet "PersonDetails" -Properties @{"Age"=$true;"Gender"=
↪"Female"; "Photo"="This is a Photo"}

if(-not $entity){
        $Error[0].Exception #System.AggregateException: "One or more errors occurred."
        $Error[0].Exception.InnerExceptions
        <#
                Following mandatory properties in entity 'PersonDetail' are missing:
↪'PersonID'
                Passed value for entity 'PersonDetail' for property with name 'Photo' has a␣
↪wrong data type 'String', but should be of type 'Stream'.
                Passed value for entity 'PersonDetail' for property with name 'Gender' has␣
↪a wrong data type 'String', but should be of type 'Boolean'.
        #>
}
```

6.1.5 Connect-ERP

Allows to connect to powerGateServer services or directly to the OData endpoints of ERP systems.

These can be either the services configured in the Vault within the *ERP Integration Settings* dialog, or alternatively all the necessary connection details can be provided via parameters.

Syntax

Configuration from Vault:

```
Connect-ERP -UseSettingsFromVault [-OnConnect <Scriptblock | String>] [<CommonParameters>
↪]
<#
PARAMETER
    -UseSettingsFromVault
        Required                true

    -OnConnect
        Required                false

    <CommonParameters>
        This cmdlet supports the common parameters: ErrorAction, ErrorVariable
#>
```

Providing connection details directly:

```
Connect-ERP [[-Service] <Uri>] [[-User] <String>] [[-Password] <String>] [-
↪IgnoreCertificates] [-OnConnect <Scriptblock | String>] [<CommonParameters>]
<#
PARAMETER
    -Service
        Required                false

    -User
        Required                false

    -Password
        Required                false

    -IgnoreCertificates
        Required                false

    -OnConnect
        Required                false

    <CommonParameters>
        This cmdlet supports the common parameters: ErrorAction, ErrorVariable
#>
```

Parameters

Type	Name	Description	Default value
SwitchParameter	UseSettingsFromVault	Connect to services configured in *ERP Integration Settings* dialog for the currently connected Vault	False
Scriptblock / String	OnConnect	This script block or function will be executed before connecting to the service. More details below	
Url	Service	Url to a specific OData service	http://localhost:8080/PGS/CatalogService
String	User	Username which should have permission to read the Service	
String	Password	Password which matches with username	
SwitchParameter	IgnoreCertificates	If Flag is set the connection is set up in order to trust all certificates	False

Return type

Bool:
$true ← on success.
$false ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

If the service to be connected is a *CatalogService* (available with powerGateServer or SAP), then all its known services are automatically connected too.
Thereby the same connection details are used as for the CatalogService itself, e.g. the same authentication.

OnConnect

The script block or the function name that is passed to the **-OnConnect** parameter enables connections to services that have very specific requirements.
The `$settings` parameter (see *connection settings*) allows to manipulate certain settings or even attaching to the BeforeRequest/AfterResponse handler.
Please note that when connecting to a CatalogService, the specified script block or function is also invoked for all contained services that will be connected automatically.
When an exception is thrown no connection to the services is established.

Configuration from Vault

When using the **-UseSettingsFromVault** switch, the cmdlet connects to all services that are configured for the currently **connected Vault** via the *$ERPSettings* variable.
The connection settings can be changed for each Vault individually using the *ERP Integration Settings* dialog.

The cmdlet fails when the connected Vault is not properly configured:
For instance, in production Vaults, it's essential to manually setup the connection to the correct ERP system.
When working with test-Vaults, administrators often overlook to add the necessary configurations to their productive Vaults, which leads to ERP integration issues after go-live.
However, problems can also arise from sudden changes to the ERP-endpoint itself.

During the trial period or evaluation phase - when no specific configuration has been made - the default settings for a connection to a public *Demo ERP system* are used.
This allows all the delivered *ERP integration samples* to be used out of the box.

Settings File

The cmdlet relies on a hidden Vault file *'$/powerGate.settings'* which stores all *settings* of the ERP integration.
A Vault administrator can set the file **permissions** so that only selected users can check-in changes.
Read and download permissions are required by all Vault users working with the ERP integration in Vault Client, Inventor or the Job Server.

The the cmdlet downloads the current settings to *C:\ProgramData\coolOrange\powerGate.settings*.

Providing connection details directly:

The cmdlet can connect directly to a specified **-Service** URL.
Basic Authentication is used for the connection when passing **-User** and **-Password** credentials.

The cmdlet can also be called again for an individual service using different authentication details, e.g. if this service requires different authentication than the CatalogService.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

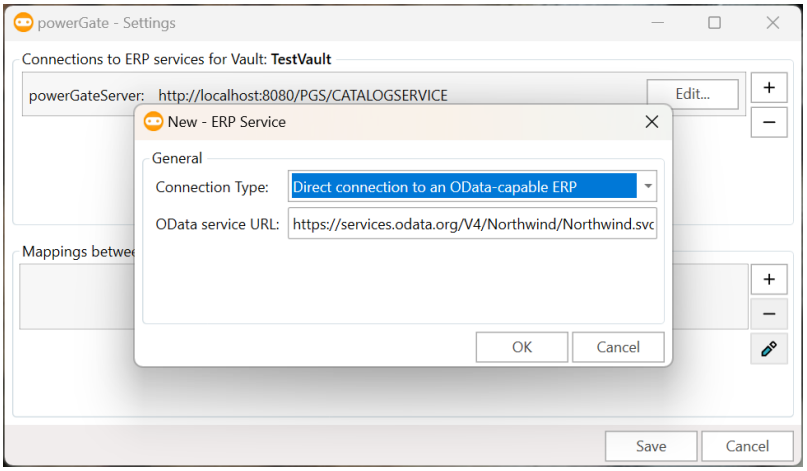
Connect to the Northwind Service

```
Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc"
```

Connect to a SAP service **that requires authentication**

```
Connect-ERP -Service "http://sap.coolorange.com" -User "EX_DEMO" -Password "secret" -
↪OnConnect $global:sapConnect
```

Connect to all services which are *configured* for the connected Vault




```
Import-Module powerVault
Open-VaultConnection

Connect-ERP -UseSettingsFromVault
```

Note: When using the cmdlet in applications that may connect to different Vaults, ensure only ERP connections configured for the current Vault are utilized, such as in the following powerJobs Processor script:

```
Import-Module powerGate
Disconnect-ERP
Connect-ERP -UseSettingsFromVault -ErrorAction Stop
```

Connect to a service via secured SSL connection and trust all certificates

```
Connect-ERP -Service "https://services.odata.org/V4/Northwind/Northwind.svc" -
↪IgnoreCertificates
```

Using the -OnConnect parameter to add a header to each request

```
Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc" -OnConnect {
param($settings)
$settings.BeforeRequest = [Action[System.Net.Http.HttpRequestMessage]] {
        param($request)
        $request.Headers.Add("Accept","application/json")
}
}
```

Using multiple script blocks in -OnConnect parameter

```
$outputService = {
        param($settings)
        Write-host "Connecting to the Service $($settings.BaseUri)"
}

$outputPreferredUpdateMethod = {
        param($settings)
        Write-host "Current PreferredUpdateMethod is $($settings.PreferredUpdateMethod)"
}

```

(continues on next page)

(continued from previous page)

```
Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc" -OnConnect {
        param($settings)
        $outputService.Invoke($settings)
        $outputPreferredUpdateMethod.Invoke($settings)
        $sapConnect.Invoke($settings)
}
```

Error handling, analyze the *WebRequestException* why the connection to server could not be established, using $Error:

```
$result = Connect-ERP "https://services.odata.org/NotExisting.svsc"

if(-not $result){
        $Error[0].Exception.StatusCode #404
        $Error[0].Exception.Source #Local computer
        $Error[0].Exception.Message #"The service is not available or could not be found␣
↪on the server."
        $Error[0].Exception.RawResponse #"The resource you are looking for has been␣
↪removed, had its name changed, or is temporarily unavailable."
}
```

6.1.6 Disconnect-ERP

Disconnects from a ERP system.

Syntax

```
Disconnect-ERP  [[-Service] <Uri>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
Url	Service	URL to the service that should be disconnected.	yes

Return type

empty ← On failure the Exception/ErrorMessage can be accessed using $Error.

Remarks

The cmdlet can be invoked **without arguments** to disconnect all connected *ErpServices*.
Afterwards they are no longer available in the current session.

Alternatively, the **-URL** parameter can be used to disconnect individual services by specifying the URL of the respective service.
If it is a *Catalog Service* (powerGateServer or SAP systems), the connection to this service and all the contained services are automatically disconnected.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Disconnect all connected Services

```
Disconnect-ERP
```

Disconnect the NorthwindService

```
Disconnect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc"
```

Disconnect all not available services

```
(Get-ERPServices) | where { -not $_.Available } | foreach { Disconnect-ERP -Service $_.
↪Url}
```

6.1.7 Get-ERPEntitySets

Cmdlet to retrieve metadata informations about EntitySets.

Syntax

```
1 Get-ERPEntitySets [[-Service] <String>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	Service	Retrieve only EntitySets for this service	yes

Return type

EntitySet [] ← on success
empty array ← on failure. Exception/ErrorMessage can be accessed using $Error.

Remarks

When calling the Get-ERPSEntitySets without arguments the EntitySets of all available services are returned.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Getting all EntitySets

```
Get-ERPEntitySets
```

Get EntitySets from the Service:

```
Get-ERPEntitySets -Service "Northwind.svc"
```

Error handling: Check if an error appeared by using $? and analyze the *WebRequestException* to understand why the error occurred, by using $Error:

```
$entitySets = Get-ERPEntitySets -Service "SomeService"
if(-not $entitySets){
        if($? -eq $false){
                $Error[0].Exception #Could not find any service for: 'SomeService'
        }
        else{
                Write-Host("The Service 'SomeService' does not have any EntitySets!")
        }
}
```

6.1.8 Get-ERPEntityTypes

Cmdlet to retrieve metadata informations about EntityTypes.

Syntax

```
Get-ERPEntityTypes [[-Service] <String>] [[-EntitySet] <String>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	Service	Retrieve all EntityTypes from the specified service	yes
String	EntitySet	Retrieve all EntityTypes from the specified entitySet	yes

Return type

EntityType [] ← on success
empty array ← on failure. Exception/ErrorMessage can be accessed using $Error.

Remarks

When calling the Get-ERPEntityTypes without arguments the EntityTypes of all available services are returned.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Getting all available EntityTypes

```
Get-ERPEntityTypes
```

Getting all EntityTypes from the specified Service "OData.svc"

```
$entityTypes = Get-ERPEntityTypes -Service "OData.svc"
$entityTypes #returns Products, Persons...
```

Getting all available EntityTypes from the EntitySet "Customers"

```
$entityTypes = Get-ERPEntityTypes -EntitySet "Customers"
$entityTypes[0].Name #returns Customer
```

Error handling: Check if an error appeared by using $? and analyze the *WebRequestException* to understand why the error occurred, by using $Error:

```
$entityTypes = Get-ERPEntityTypes -Service "SomeService"

if(-not $entityTypes){
        if($? -eq $false){
                $Error[0].Exception #Could not find any service for: 'SomeService'
        }
        else{
                Write-Host("The Service 'SomeService' does not have any EntityTypes!")
        }
}
```

6.1.9 Get-ERPMedia

Cmdlet to download the attached media link of a specified entity from the ERP-System.

Syntax

```
Get-ERPMedia [[-EntitySet] <String>] [[-Keys] <Object>] [[-File] <String>]] [
↪<CommonParameters>]
```

Parameters

Type	Nam	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Keys	The reference properties for the searching item	no
String	File	Path where the file should be downloaded (e.g C:\Temp\myFile.txt). In case a file with the same name already exists it will be overwritten.	no

Return type

Bool:
$true ← on success.
$false ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

The Cmdlet is used to download the Media Link Entries (MLEs) from the specified entity to the specified location. In case a file with the same name already exists in the specified location it will be overwritten. Also the appropriate folder structure will be created if it doesn´t exists.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Download media from the Advertisement entity

```
Connect-Erp -Service "http://services.odata.org/V3/OData/OData.svc"
Get-ERPMedia -EntitySet "Advertisements" -Keys @{"ID" = [Guid]"db2d2186-1c29-4d1e-88ef-
↪a127f521b9c6"} -File "C:\Temp\TestMedia.txt"
```

Error handling, analyze why the Media Resource could not be downloaded to the specified directory, by using $Error

```
Connect-Erp -Service "https://services.odata.org/V3/OData/OData.svc"
$result = Get-ERPMedia -EntitySet "Advertisements" -Keys @{"ID" = [Guid]"db2d2186-1c29-
↪4d1e-88ef-a127f521b9c6"} -File "C:\Temp\AdvertismentText.txt"

if(-not $result){
        $Error[0].Exception #"Access to the path 'C:\Temp\AdvertismentText.txt' is denied.
}
```

6.1.10 Get-ERPObject

Cmdlet to retrieve a specific entity from the ERP-System.

Syntax

```
Get-ERPObject [[-EntitySet] <String>] [[-Keys] <Object>] [[-Expand] <String[]>] [[-
↪Select] <String[]>] [<CommonParameters>]
```

Parameters

Type	Nam	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Keys	The reference properties for the searching item	no
String[]	Expand	The name/s of the Navigation Property which should be expanded	yes
String[]	Select	The name/s of the Property or Navigation Property which should explicitly be requested and returned	yes

Return type

Entity ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

This Cmdlet is used to retrieve one specific object from ERP.

Per default navigation properties will **not** be returned unless explicitly specified by using the **-Expand** parameter with the name of the property as argument.
The **-Select** argument lets you receive only those properties or navigation properties which you want to have in the result.

Error Handling

It is a common use case to utilize this Cmdlet to verify the existence of an object in the ERP system. Therefore it is not expected that the Cmdlet fails (e.g. throws an error) when the requested object does not exist and handles responses of type '**404 Resource not found**' not as errors.

As a result, no warnings or errors will be logged, the $Error variable remains unchanged and the $? Automatic Variable returns `$true`.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Request the Category with Id '1'

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObject -EntitySet "Categories" -Keys @{ "CategoryID" = 1 }
<#
        CategoryID   : 1
        CategoryName : Beverages
        Description  : Soft drinks, coffees, teas, beers, and ales
        Picture      : {21, 28, 47, 0...}
        _Keys        : @{CategoryID=1}
        _Properties  : @{CategoryName=Beverages; Description=Soft drinks, coffees, teas,␣
↪beers, and ales; Picture=System.Byte[]}
#>
```

Request a Category with its Navigation Property Products

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObject -EntitySet "Categories" -Keys @{"CategoryID"=1} -Expand "Products"
<#
        Products     : {Chai, Chang, Guaraná Fantástica, Sasquatch Ale...}
        CategoryID   : 1
        CategoryName : Beverages
        Description  : Soft drinks, coffees, teas, beers, and ales
        Picture      : {21, 28, 47, 0...}
        _Keys        : @{CategoryID=1}
        _Properties  : @{Products=powerGate.Erp.Cmdlets.Cmdlets.Results.PsEntity[];␣
↪CategoryName=Beverages; Description=Soft drinks, coffees, teas, beers, and ales;␣
↪Picture=System.Byte[]}
#>
```

Request only the 'CategoryID' of a Category

```
$result = Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc"
Get-ERPObject -EntitySet "Categories" -Keys @{"CategoryID"=1} -Select "CategoryID"
<#
        CategoryID : 1
#>
```

Create an empty entity with *New-ERPObject* **, set the Key and use it to search the entity in the ERP System**

```
Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc"
$category = New-ERPObject -EntityType "Category"
$category.CategoryID = 1

Get-ERPObject -EntitySet "Categories" -Keys $category._Keys
<#
CategoryID   : 1
CategoryName : Beverages
Description  : Soft drinks, coffees, teas, beers, and ales
Picture      : {21, 28, 47, 0...}
_Keys        : @{CategoryID=1}
_Properties  : @{CategoryName=Beverages; Description=Soft drinks, coffees, teas, beers,
↪and ales; Picture=System.Byte[]}
#>
```

Check whether an SAP Material does not exist or if another 400 error response occurred (only for older SAP Gateway integrations):

```
Connect-ERP -Service 'https://sap_environment/sap/opu/odata/arcona6/MATERIAL_SRV' -
↪OnConnect $global:sapConnect

$materialReallyDoesNotExist = $false
$material = Get-ERPObject -EntitySet 'MaterialContextCollection' -Keys @{ Material =
↪'000000000200314159'; Plant = '1001'; ValuationArea=''; ValuationType=''} -Expand @(
↪'Description','PlantData','BasicData') -ErrorAction SilentlyContinue

if($? -and $material -eq $null){
   # For older SAP Gateway interfaces this code block is never executed because
↪unfortunately 400 instead of 404 responses are mistakenly returned when materials doe
↪not exist
   $materialReallyDoesNotExist = $true
}

# That's why the response message must be checked. Attention to different languages!
if($? -eq $false){
   if($Error[0].Exception.StatusCode -eq 400) {
      $sapResponse = $Error[0].Exception.RawResponse | ConvertFrom-Json
      $sapResponse.error.code #SY/530

      if($sapResponse.error.message.lang -eq 'de' -and $sapResponse.error.message.value -
↪eq 'Das Material 200314159 ist nicht vorhanden oder nicht aktiviert') {
         $materialReallyDoesNotExist = $true
      }
      else {
         Write-Message "A real error response was returned from SAP: $($sapResponse.
```

(continues on next page)

(continued from previous page)

```
↪error.message.value)"
      }
   }
   else {
      Write-Message "Also in this case, SAP replied with an error: $($Error[0])"
   }
}

if($materialReallyDoesNotExist) {
   # create a new material
}
```

Error handling: Check if an error appeared by using $? and analyze the *WebRequestException* to understand why the object could not be retrieved, by using $Error:

```
Connect-ERP -Service "https://services.odata.org/V4/(S(zhhrvnffwxy1zfd0uy2aeye0))/
↪TripPinServiceRW/"
$entity = Get-ERPObject -EntitySet "People" -Keys @{"UserName"="scottketchum"} -Expand
↪"Trips" -Select "Friends"

if(-not $entity) {
   if($? -eq $false) {
      $Error[0].Exception.StatusCode #500
      $Error[0].Exception.Message
      <#
        The entity instance value of type 'Microsoft.OData.SampleService.Models.TripPin.
↪Person' doesn't have a value for property 'UserName'.
         To compute an entity's metadata, its key and concurrency-token property values␣
↪must be provided.
      #>
      $Error[0].Exception.Response.ErrorCode #"InternalServerError"
   } else {
      Write-Host("No object found with keys `UserName = scottketchum'!")
   }
}
```

6.1.11 Get-ERPObjects

Cmdlet to search for specific entities depending on the passed arguments.

Syntax

```
Get-ERPObjects [[-EntitySet] <String>] [[-Top] <int>] [[-OrderBy] <Object>] [[-Filter]
↪<String>] [[-Expand] <String[]>] [[-Select] <String[]>] [<CommonParameters>]
```

Parameters

Type	Nam	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Integer	Top	The amount of how many items should be shown	yes
String	Filter	Query what should be executed as a filter	yes
String / Hashtable / Hashtable[]	OrderBy	Items getting ordered by passed property name and/or direction	yes
String[]	Expand	Name/s of the related Navigation Properties which should be included	yes
String[]	Select	Name/s of the properties or Navigation Properties which should explicitly be requested and returned	yes

Return type

Entity [] ← on success
empty array ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

This Cmdlet is used search for specific entities from the ERP.
The **Filter** argument allows you to specify a filter with OData syntax. More informations about the OData Filter syntax can be found here.
The **OderBy** argument can be specified as String, as hashtable or as an array of hashtables.

- If the argument is specified as String then the entities are orderd ascending by the passed property name.
- If the argument is specified as hashtable then the entities are orderd by the passed property name and direction ('Ascending' or 'Descending')
- If the argument is specified as array of hashtables, the entities can be orderd by multiple properties, in the specified direction (See example below)

The **Expand** allows you to expand multiple navigation properties. As default if the expand argument is **not** specified then the entity will be returned without navigation properties.
The **Select** argument lets you receive only those properties or navigation properties which you want to have in the result.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Search for customers from the company 'Alfreds Futterkiste':

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
$entity = Get-ERPObjects -EntitySet "Customers" -Filter "CompanyName eq 'Alfreds␣
↪Futterkiste'"
```

Search for customers with a companyName starting with 'A':

```
Connect-Erp -Service "http://services.odata.org/V3/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Customers" -Filter "startswith(CompanyName,'A')"
```

Get categories:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
$entity = Get-ERPObjects -EntitySet "Categories"
```

Get first 3 Products:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Products" -Top 3
```

Get objects ordered by Name:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Products" -OrderBy 'Name'
```

Get objects ordered by the ProductName <Descending>:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Products" -OrderBy @{'ProductName'='Descending'}
```

Get objects ordered by the ProductName <Descending> and then the CategoryId <Ascending>:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Products" -OrderBy @(@{'ProductName'='Descending'},@{
↪'CategoryID'='Ascending'} )
```

Get objects ordered by the ProductName <Ascending> and then the CategoryId <Descending>:

```
Connect-Erp -Service "http://services.odata.org/V4/Northwind/Northwind.svc/"
Get-ERPObjects -EntitySet "Products" -OrderBy @('ProductName',@{'CategoryID'='Descending
↪'} )
```

Get all OrderID's of the available 'Orders':

```
Connect-ERP -Service "http://services.odata.org/V4/Northwind/Northwind.svc"
Get-ERPObjects -EntitySet "Orders" -Select "OrderID"
```

Error handling: Check if an error appeared by using $? and analyze the *WebRequestException* to understand why the error occurred, by using $Error:

```
Connect-ERP -Service "https://services.odata.org/V4/Northwind/Northwind.svc/"
$filter = "startswith(CustomerID,1234567)"
$entities = Get-ERPObjects -EntitySet "Invoices" -Filter $filter

if(-not $entities){
if($? -eq $false){
   $Error[0].Exception.StatusCode #400
   $Error[0].Exception.Message
   <#
     No function signature for the function with name 'startswith' matches the specified arguments.
     The function signatures considered are: startswith(Edm.String Nullable=true, Edm.String Nullable=true).
   #>
   $Error[0].Exception.Response.InnerError.StackTrace
   <#
     at Microsoft.OData.Core.UriParser.Parsers.FunctionCallBinder.MatchSignatureToBuiltInFunction(String functionName, SingleValueNode[] argumentNodes, FunctionSignatureWithReturnType[] signatures)
     ...
     at Microsoft.OData.Service.Parsing.RequestExpressionParser.ParseFilter()
   #>
   }
   else{
     Write-Host("No objects found for the `$filter '$filter'!")
   }
}
```

6.1.12 Get-ERPServices

Cmdlet to retrieve metadata informations about the services.

Syntax

```
Get-ERPServices [-Available] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
SwitchParamter	Available	Retrieves only the available services	yes

Return type

ErpService [] ← on success
empty array ← on failure. Exception/ErrorMessage can be accessed using $Error.

Remarks

When calling the Get-ERPServices without arguments all the services are retrieved.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Get all services

```
$services = Get-ERPServices
```

Get all available services

```
$availableServices = Get-ERPServices -Available
```

6.1.13 New-ERPObject

Returns a new and by default empty ERP object of the specified ERP type.

Thereby its field values can be passed directly, or alternatively they are determined automatically.
For Vault entities, for example, based on the mappings configured in the *ERP Integration Settings* dialog, or from the $metadata of the ERP system.

Syntax

Configuration from Vault:

```
New-ERPObject [-EntityType] <string> [-VaultEntity] <PSObject> [<CommonParameters>]
<#
PARAMETER
    -EntityType
        Required                true

    -VaultEntity
        Required                true
        Accepts pipeline input: true

    <CommonParameters>
        This cmdlet supports the common parameters: ErrorAction, ErrorVariable
#>
```

Providing field values directly:

```
New-ERPObject [-EntityType] <string> [[-Properties] <Hashtable / PSObject>] [
↪<CommonParameters>]
<#
PARAMETER
    -EntityType
        Required                true

    -Properties
        Required                false

    <CommonParameters>
        This cmdlet supports the common parameters: ErrorAction, ErrorVariable
#>
```

Parameters

Type	Name	Description
String	EntityType	The EntityType of the ERP object that should be returned.It is possible to additionaly specify a part or the whole namespace (e.g 'ErpServices.Services.Entities.Item')
Hashtable / PSObject	Properties	The field values that should be set on the returned ERP object.
powerVault Object	VaultI	The Vault entity, from which mapped Properties in the *ERP Integration Settings* dialog are set on the corresonding Fields of the ERP object, that is returned.Typically, a powerVault File, Item, FileBomRow or ItemBomRow object is passed.

Return type

Entity ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.

Remarks

With only an **-EntityType** parameter and no *-EntitySet* or *-Service* parameters, the cmdlet searches for the specified type in all services, considering namespace information.

It analyzes the $metadata for all ERP *Properties* and *NavigationProperties* to create a new ERP object.
As a result, the returned field values have the following *default values*:

- the default field value that is specified in the $metadata (see *DefaultValue*)
- Empty for nullable properties (see *IsNullable*).
  Note that also *NavigationProperties* with a target *multiplicity* of *ZeroOrOne* or *Many* are nullable.
- the required target instance for *NavigationProperties* with a target *multiplicity* of *One*, even recursively on multiple levels

Configuration from Vault

A **-VaultEntity** can be passed after calling *Connect-ERP -UseSettingsFromVault* (configuration for the currently connected Vault is then provided by the *$ERPSettings* variable).
The prerequisite is that a type mapping has been set up for the specified Vault- and ERP entity type via the *ERP Integration Settings* dialog (e.g. `$vaultFile` and `-EntityType 'Item'`).

Based on the configured *Field Mappings* the returned ERP object is automatically filled with data.
For this, the respective ERP fields provide the values of the mapped *Vault Properties* or alternatively the configured *fixed-values*.
Additional settings are also taken into account, such as special *default values* if the Vault Property is empty, matching ERP values from a *possible value* list, and complex or calculated value determinations.

Providing field values directly

When **-Properties** are passed, the cmdlet assigns them directly to the fields of the newly created ERP object.
Note that the parameter only accepts existing properties, and the passed values are not checked for validity!

Examples

In the following examples we are using the public OData Northwind Services for demonstration purposes:

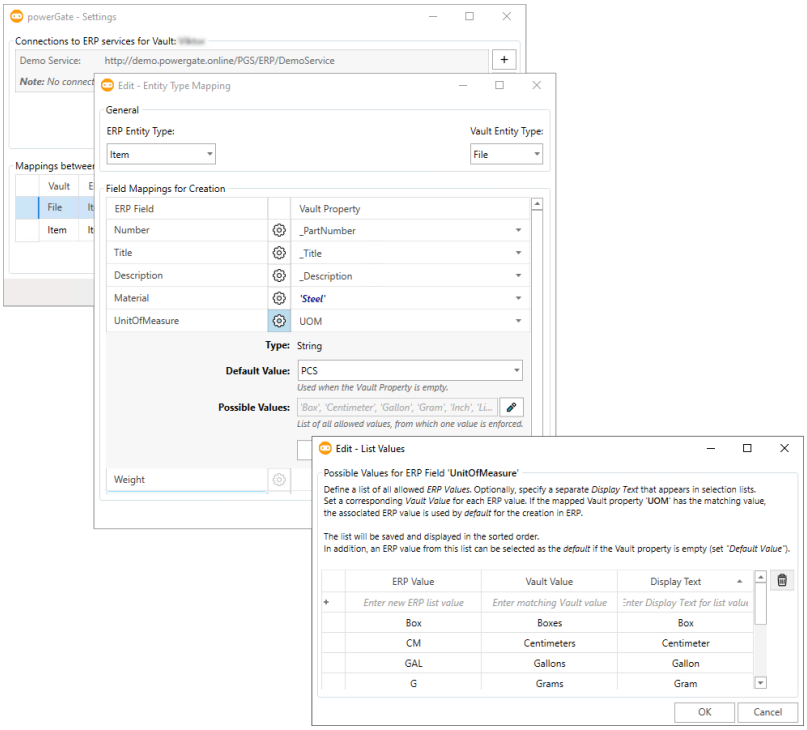
Create a new instance by specifying the EntityType

```
$territory = New-ERPObject -EntityType 'Territory'
```

Create a new instance by specifying the whole EntityType with namespace

```
$shipper= New-ERPObject 'NorthwindModel.Shipper'
```

Create a new ERP object from a Vault file for which mappings are *configured*




```powershell
Import-Module powerVault
...
$file = Get-VaultFile -Properties @{'File Name'='Pad Lock.iam'}

Connect-ERP -UseSettingsFromVault
$erpObject = $file | New-ERPObject -EntityType 'Item'

$erpObject | Format-List Number,Title,Description,Material,UnitOfMeasure,Weight
<#
Number          : ERP-17425549
Title           : Pad Lock
Description     : PAD LOCK ASSEMBLY
Material        : Steel
UnitOfMeasure   : CM
Weight          : 0.130
#>
```

Create a new ERP object instance with specific field values

```powershell
$region = New-ERPObject 'Region' -Properties @{'RegionID'=1}
$territory = New-ERPObject -EntityType 'Territory' -Properties @{'TerritoryID'='66';
↪'Region'=$region}
$region.Territories = @($territory)
```

Dynamically create new instances for all the EntityTypes of a specific service

```powershell
$allEntityTypes = Get-ERPEntityTypes -Service 'http://services.odata.org/V3/Northwind/
↪Northwind.svc'
$allEntityTypes | foreach {
        New-ERPObject ($_.Namespace+'.'+$_.Name)
```

(continues on next page)

(continued from previous page)

```
}
```

Error handling, analyze why no default ERP object is created, by using $Error

```
Connect-ERP -Service "http://services.odata.org/V3/Northwind/Northwind.svc"
$entity = New-ERPObject -EntityType "People"

if(-not $entity){
        $Error[0].Exception #"No EntityType found with the given name: People"
}
```

6.1.14 Remove-ERPObject

Cmdlet to remove a specific entity from the ERP-System.

Syntax

```
Remove-ERPObject [[-EntitySet] <String>] [[-Keys] <Object>] [<CommonParameters>]
```

Parameters

Type	Nam	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Keys	The reference properties for the searching item	

Return type

Bool:
$true ← on success.
$false ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

This Cmdlet is used to delete a specific object from the ERP System.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Remove Product with Id 1

```
Connect-ERP -Service "http://services.odata.org/V3/(S(xnnohcch2jddbn1z1ytpfni2))/OData/
↪OData.svc/"
Remove-ErpObject -EntitySet "Products" -Keys @{"ID"=1}
```

Search for Product and remove it

```
Connect-ERP -Service "http://services.odata.org/V3/(S(xnnohcch2jddbn1z1ytpfni2))/OData/
↪OData.svc/"
$product = Get-ERPObject -EntitySet "Products" -Keys @{"ID"=10}
Remove-ErpObject -EntitySet "Products" -Keys $product._Keys
```

Error handling, analyze why the ERP object could not be removed, by using $Error

```
Connect-ERP -Service "http://services.odata.org/V3/(S(xnnohcch2jddbn1z1ytpfni2))/OData/
↪OData.svc/"
$result = Remove-ErpObject -EntitySet "Products" -Keys @{}

if(-not $result){
  $Error[0].Exception #"Following mandatory properties in entity 'Product' are missing: 'ID'
↪"
}
```

6.1.15 Update-ERPMedia

Cmdlet to update the Media Resource of an existing streamable entity with a given media file.

Syntax

```
Update-ERPMedia [[-EntitySet] <String>] [[-File] <String>]] [[-Keys] <Object>] [[-
↪ContentType] <String>]] [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	EntitySet	The EntitySet name where the Media Link Entity is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Keys	The reference properties for the entity beeing updated	no
String	File	Path to the file to upload	no
String	ContentType	Specifies the content type of the web request. If no content type is provided, the cmdlet sets the content type depending on the file extension. (e.g .txt 'text/plain', .pdf 'application/pdf' ...)	yes

Return type

Bool:
$true ← on success.
$false ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

The Cmdlet is used to update the Media Resource of an existing Media Link Entry (MLE) by uploading the specified file.

The **ContentType** is used to specify the nature of the file currently being handled. With the appropriate content type the web browser can open the file with the proper extension/plugin.

- If no content type is set the Cmdlet will determine the content type depending on the file type.
- If the content type contains text (e.g text/plain, text/html...) as type or json , xml (e.g application/json, application/xml...) as subtype, then the content of the file is uploaded to the server as UTF-8 Encoded text.

Examples

In the following example we are using the public OData Services (http://services.odata.org) for demonstration purposes:

Updating the Media Resource of a Media Link Entry (MLE)

```
Invoke-WebRequest -Uri 'https://randompokemon.com/sprites/normal/143.gif' -OutFile 'C:\
↪Temp\TestMedia.gif'
Connect-Erp -Service "http://services.odata.org/V4/OData/(S(du4oaehbpzqh2eznhygi1xkg))/
↪OData.svc"
Update-ERPMedia -EntitySet "Advertisements" -Keys @{ID=[Guid]'f89dee73-af9f-4cd4-b330-
↪db93c25ff3c7'} -File 'C:\Temp\TestMedia.gif'
```

Note: The cmdlet automatically detects the **ContentType** of the file to be '*image/gif*'. The browser is able to use this information and directly show the image. You can test the updated Media Resouce by clicking here.

In the following examples we are using a custom plugin for the powerGateServer :

Updating the Media Resource of a Media Link Entry (MLE) with specific ConentType and download it again

```
Invoke-WebRequest -Uri 'https://randompokemon.com/sprites/normal/6.gif' -OutFile 'C:\
↪Temp\TestMedia.gif'
Connect-Erp -Service "http://localhost:8080/powerGate.Tests/TestService"
Update-ERPMedia -EntitySet "Files" -Keys @{"Id"=2} -File "C:\Temp\TestMedia.gif" -
↪ContentType 'image/x-xbitmap'
Get-ERPMEdia -EntitySet "Files" -Keys @{Id=2} -File "C:\Temp\TestMedia_2.gif"
```

Error handling, analyze the *WebRequestException* why the Media Resource could not be updated, by using $Error

```
Connect-Erp -Service "https://services.odata.org/V3/(S(o2bnti0dqsoaosxxnt5pci1q))/OData/
↪OData.svc/"
$result = Update-ERPMedia -EntitySet "Advertisements" -Keys @{"ID"=[guid]"f89dee73-af9f-
↪4cd4-b330-db93c25ff3c7"} -File "C:\Temp\MyPictures\GT220 Change Proposal.png"

if(-not $result){
        $Error[0].Exception.StatusCode #413
        $Error[0].Exception.Message #"Request Entity Too Large"
}
```

6.1.16 Update-ERPObject

Cmdlet to update an existing entity in the ERP-System.

Syntax

```
Update-ERPObject [[-EntitySet] <String>] [[-Keys] <Object>] [[-Properties] <Object>] [
↪<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	EntitySet	The EntitySet name where the item is located. It is also possible to specify additional namespaces or the whole url (e.g MaterialService/Materials, http://localhost:8080/PGS/ERP/MaterialService/Materials)	no
Hashtable / PSObject	Keys	The reference properties for the updating item	no
Hashtable / PSObject	Properties	The properties which should be changed	no

Return type

Entity ← on success
empty ← on failure. Exception/ErrorMessage can be accessed using $Error.
If the cmdlet fails due to error responses returned by the ERP system, the $Error variable provides a *WebRequestException* .

Remarks

This Cmdlet is used to update an existing entity in ERP.
Per default it uses a **HTTP MERGE** request.

Method: PUT

When using **PUT** as PreferredUpdateMethod please note that you have to pass **all** properties, otherwise it will replace/reset the properties which where not passed.
For more information see: 2.6 Updating Entries

Examples

In the following examples we are using the default "ERP" plugin from the powerGateServer. Additionally they assume that objects with following entities exist:

- On **Materials**: *"Number" = "3169"*
- On **Descriptions**: *"Number" = "6931", "Language" = "EN"*

Update type in Material

```
Connect-Erp -Service "http://localhost:8080/pgs/ERP/MaterialService/"
$entity = Update-ERPObject -EntitySet "Materials" -Keys @{ "Number" = "3169" } -
↪Properties @{ "Type" = "updated Type"}
```

Update description in MaterialDescription

```
Connect-Erp -Service "http://localhost:8080/pgs/ERP/MaterialService/"
$entity = Update-ERPObject -EntitySet "MaterialDescriptions" -Keys @{ "Number" = "6931";
↪"Language" = "EN" } -Properties @{ "Description" = "changed by powerGate"}
```

In the following example we are using the public OData Services (http://services.odata.org):
Search for Product and update it

```
Connect-ERP -Service "http://services.odata.org/V3/(S(xnnohcch2jddbn1z1ytpfni2))/OData/
↪OData.svc/"
$product = Get-ERPObject -EntitySet "Products" -Keys @{"ID"=9}
Update-ERPObject -EntitySet "Products" -Keys $product._Keys -Properties @{"Rating"=9;
↪"Price"=1.99}

<#
        ID : 9
        Name : Lemonade
        Description : Classic, refreshing lemonade (Single bottle)
        ReleaseDate : 01.01.1970 00:00:00
```

(continues on next page)

(continued from previous page)

```
        DiscontinuedDate :
        Rating : 9
        Price : 1,99
#>
```

Update description using PUT as PreferredUpdateMethod

```
Connect-Erp -Service "http://localhost:8080/pgs/ERP/MaterialService/" -OnConnect {
        param($settings)
        $settings.PreferredUpdateMethod = "PUT"
}
$entity = Update-ERPObject -EntitySet "MaterialDescriptions" -Keys @{ "Number" = "6931";
↪"Language" = "EN" } -Properties @{ "Description" = "change with PUT request"}
```

Error handling, analyze the *WebRequestException* **why the entity could not be updated, by using $Error**

```
Connect-Erp -Service "https://services.odata.org/V3/OData/OData.svc"
$entity = Update-ERPObject -EntitySet "Persons" -Keys @{"ID"=1} -Properties @{"Name"=
↪"Sepp Meißnor"}

if(-not $entity){
        $Error[0].Exception.StatusCode #403
        $Error[0].Exception.Message #"You are connected to a read-only data session.␣
↪Update operations are not permitted for your session"
}
```

All the following OData cmdlets support OData version v1, v2, v3, v4.
They all try to perform as less server requests as possible, in order to ensure high performance when communicating over the network or the internet with your ERP system.

6.1.17 Connection management

Cmdlets for connection establishment with OData services:

Name	Description
Connect-ERP	Connects to an ERP-System.
Disconnect-ERP	Disconnects from an ERP-System.

Cmdlets to read metadata of connected OData services:

Name	Description
Get-ERPServices	Returns information about the connected services
Get-ERPEntitySets	Returns information about the available entitySets
Get-ERPEntityTypes	Returns information about the available EntityTypes.

Metadata

Requests to retrieve the service `$metadata` are only performed when needed and only for the required services. When services are not available, powerGate will try to retrieve there metadata as long as they become available! Warnings will be logged in order to inform about the downtime of the service.

After retrieving metadata once, no further metadata request is required as the server results are cached.

Performance

When some of the services are permanently not available, you can speed up your process by disconnecting them. Please see example *Disconnect all not available services*.

The following Cmdlets **require a successful connection** between the current application and the ERP system or powerGateServer.
For them to work properly, all the necessary ERP services (or a single CatalogService) must have been previously connected by the *Connect-ERP* cmdlet.

6.1.18 Entity transfer

Cmdlets for reading, adding, updating and removing OData entities:

Name	Description
Get-ERPObject	Returns a specific entity from the ERP-System.
Get-ERPObjects	Searches for entities depending on the passed arguments.
Update-ERPObject	Updates an existing entity on the ERP-System.
Add-ERPObject	Creates a new entity and transfers it to a ERP-System.
Remove-ERPObject	Deletes a specific entity from the ERP-System.
New-ERPObject	Returns a new default instance of the desired ERP entity.

6.1.19 Media exchange

Cmdlets for downloading, uploading and updating OData media files such as images, text files or even Autodesk file formats:

Name	Description
Add-ERPMedia	Uploads media files to the ERP-System.
Get-ERPMedia	Downloads media files from the ERP-System.
Update-ERPMedia	Updates media files on the ERP-System.

6.2 .NET Library

6.2.1 ConnectionSettings Class

SapConnect Class

Class which provides a default implementation for connecting to SAP services and handling CSRF protection.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.SapConnect

Syntax

```
public class SapConnect
```

The SapConnect type exposes the following members.

Constructors

Type	Description
SapConnect(long reconnectInterval = 1500 * 1000)	Initializes a new instance of the SapConnect class. The default reconnectInterval parameter, expressed in milliseconds, is set to 25 minutes.

Operator

Type	Description
Implicit(SapConnect to Action<*ErpClientSettings*>)	Converts the SapConnect object to an action of type Action<*ErpClientSettings*>.

Remarks

Since most of the SAP systems require **PUT requests** as the preferred method to *update* entities in SAP, this class adapts the according *PreferredUpdateMethod* to *UpdateMethod.PUT*.

Additionally many SAP systems require by default CSRF protection.
The Netweaver Gateway extension requires for every update/write operation the **X-CSRF-Token** header to be set and a **cookie**.
Because the X-CSRF-Token expires after a certain amount of time, the Token will be automatically refreshed, on the next request after the expiration!
Depending on the SAP Release or the security session management configuration, the Token is valid for 24 hours or

30 minutes by default.
The default behaviour to retrieve a new token every 25min can be changed the reconnectInterval parameter in the constructor.

Examples

Connect to a SAP service using SapConnect

```
using (var erpclient = new ErpClient())
{
        var connectionSettings = new ConnectionSettings
        {
                Service = new Uri("http://sap.coolorange.com/CATALOGSERVICE"),
                Credentials = new System.Net.NetworkCredential("Administrator", "<secret password>"),
                OnConnect = new SapConnect() //SAP Connection
        };
        using (var service = erpclient.ConnectErp(connectionSettings))
        Console.Write("Connected to SAP service: {0}", service.Name);
}
```

Connecting multiple SAP systems with different CSRF-Token expiration intervals:

```
using (var erpclient = new ErpClient())
{
        var connectionSettings1 = new ConnectionSettings
        {
                Service = new Uri("http://sap.coolorange.com/CATALOGSERVICE"),
                OnConnect = new SapConnect()
        };
        var connectionSettings2 = new ConnectionSettings
        {
                Service = new Uri("http://sap.some_other_company.com/CATALOGSERVICE"),
                OnConnect = new SapConnect(86400 * 1000) //tokens expire every 24h on this system
        };
        using (var service1 = erpclient.ConnectErp(connectionSettings1))
        using (var service2 = erpclient.ConnectErp(connectionSettings2))
        {
                Console.Write("Connected to SAP-1: {0}", service1.Name);
                Console.Write("Connected to SAP-2: {0}", service2.Name);
        }
}
```

See also

Reference

- *$sapConnect*
- *powerGate.Erp.Client namespace*

Settings used to connect with the ERP Service.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.ConnectionSettings

Syntax

```
public class ConnectionSettings
```

The ConnectionSettings type exposes the following members.

Constructors

Name	Description
ConnectionSettings()	Initializes a new instance of the ConnectionSettings class.

Properties

Type	Name	Description
Uri	Service	Gets or sets the Url of the service to connect with.
ICredentials	Credentials	Gets or sets the credentials to connect with the services.
bool	IgnoreCertificates	Gets or sets the property whether to trust all certificates.
Action<*ErpClientSetting*	OnConnect	Gets or sets the Action which will be invoked before connecting to the service. More details below.

Remarks

The **OnConnect**-Action will be executed before connecting to the service.
In case of the CatalogService, the OnConnect will be called for each service it connects to.
Setting the Action will allow you to manipulate certain *settings* or even attaching to the BeforeRequest/AfterResponse handler.

Examples

The following example creates an instance of the ConnectionSettings class.

Create settings with Credentials

```
var connectionSettings = new ConnectionSettings {
        Service = new Uri("http://services.odata.org/V4/Northwind/Northwind.svc"),
        Credentials = new NetworkCredential("MyUserName", "1234"),
        IgnoreCertificates = true
};
```

Create settings using the OnConnect property to add a header to each request

```
var connectionSettings = new ConnectionSettings {
        Service = new Uri("http://services.odata.org/V4/Northwind/Northwind.svc"),
        OnConnect = settings => {
                //Set a 5 seconds request timeout
                settings.RequestTimeout = TimeSpan.FromSeconds(5);
                //Add header to each request
                settings.BeforeRequest = requestMessage => { requestMessage.Headers.Add(
↪"Accept", "application/json"); };
        }
};
```

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.2 ErpClient Class

Provides a class to connect with ERP services.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.ErpClient

Syntax

```
public class ErpClient : IErpClient
```

The ErpClient type exposes the following members.

Constructors

Name	Description
ErpClient()	Initializes a new instance of the ErpClient class.

Properties

Type	Name	Description
IErpServices	Services	Gets the list of connected services.

Methods

Type	Name	Description
IErpService	ConnectErp(*ConnectionSettings* connectionSettings)	Creates a connection to a service using the specified *ConnectionSettings*.
void	Dispose()	Disconnects from all services and releases all resources used by the ErpClient.

Extension Methods

Type	Name	Description
IErpService	ConnectErp(Uri service, ICredentials credentials = null, bool ignoreCertificates = false, Action<*ErpClientSettings*> onConnect = null)	Overloaded. Creates a connection to a service using the specified parameter values.

Remarks

The **ConnectErp(..)** function is able to either connect to a CatalogService or directly to a Service.
When calling the function passing settings having an url to a CatalogService, the client automatically connects to all the registered services with the same authentication as used for the login to the CatalogService.
The function can be recalled for a single service with different authentication if that service is not available by using the authentication of the CatalogService.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:
Connect to the NorthwindService

```
using System;
using powerGate.Erp.Client;

namespace HelloWorldServices
{
        class Program
        {
                static void Main(string[] args)
                {
                        //Create Instance of client
                        using (var erpclient = new ErpClient())
                        {
                                //Connect to the service
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V4/Northwind/Northwind.svc")))
                                        Console.Write("Connected to service: {0}",␣
↪service.Name);
                        }
                }
        }
}
```

Connect to a service via secured SSL connection and trust all certificates

```
using System;
using powerGate.Erp.Client;

namespace HelloWorldServices
{
        class Program
        {
                static void Main(string[] args)
                {
                        var erpclient = new ErpClient();
                        var service = erpclient.ConnectErp(new Uri("https://services.
↪odata.org/V4/Northwind/Northwind.svc"), ignoreCertificates: true);
                        //Dispose Service and Client
                        service.Dispose();
                        erpclient.Dispose();
                }
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Connect to a SAP service **that requires authentication**

```
using System;
using System.Net;
using powerGate.Erp.Client;

namespace HelloWorldServices
{
        class Program
        {
                static void Main(string[] args)
                {
                        using(var erpclient = new ErpClient())
                        {
                                //Create special ConnectionSettings for connecting to the SAP service
                                var connectionSettings = new ConnectionSettings
                                {
                                        Service = new Uri("http://sap.coolorange.com"),
                                        Credentials = new NetworkCredential("EX_DEMO", "secret"),
                                        OnConnect = new SapConnect()
                                };
                                using(var service = erpclient.ConnectErp(connectionSettings))
                                        Console.Write("Connected to service: {0}", service.Name);
                        }
                }
        }
}
```

Accessing the last server response

```
using System;
using System.Net.Http; //assembly has to be referenced
using powerGate.Erp.Client;

namespace HelloWorldServices
{
        class Program
        {
                static void Main(string[] args)
                {
                        var onConnect = new Action<ErpClientSettings>(settings =>
                        {
                                settings.AfterResponse = (response =>
                                {
                                        if (response.IsSuccessStatusCode)
```

(continues on next page)

(continued from previous page)

```
                                                return;
                                        Console.WriteLine("Request Url: " + response.
↪RequestMessage.RequestUri);
                                        Console.WriteLine("Status Code: " + response.
↪StatusCode);
                                        Console.WriteLine("Body: " + response.Content.
↪ReadAsStringAsync().Result);
                                } + settings.AfterResponse;
                        });
                        using (var erpclient = new ErpClient())
                                try
                                {
                                        erpclient.ConnectErp(new Uri("https://www.
↪coolorange.com"), onConnect: onConnect);
                                }
                                catch (Exception e)
                                {
                                }
                }
        }
}

/* Console Output
        Request Url: https://www.coolorange.com/$metadata
        Status Code: NotFound
        Body: <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
        <html><head>
        <title>404 Not Found</title>
        </head><body>
        <h1>Not Found</h1>
        <p>The requested URL /$metadata was not found on this server.</p>
        <hr>
        <address>Apache Server at www.coolorange.com Port 443</address>
        </body></html>
*/
```

Error handling, analyze why entity could not be updated

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace HelloWorldServices {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                try
                                {
                                        using (var service = erpclient.ConnectErp(new␣
```

(continues on next page)

(continued from previous page)

```
↪Uri("http://services.odata.org/V4/Northwind/Northwind.svc")))
                                        {
                                                var employees = service.EntitySets[
↪"Employees"];
                                                employees.UpdatErpObject(new Dictionary
↪<string, object>{{"EmployeeID", 66}},
                                                        new Dictionary<string, object> {
↪{ "FirstName", "Franz" }, { "LastName", "VomBerg" } });
                                        }
                                }
                                catch (WebRequestException e)
                                {
                                        Console.WriteLine("Message: " + e.Message);
                                        Console.WriteLine("Status Code: " + e.
↪StatusCode);
                                        Console.WriteLine("Body: " + e.RawResponse);
                                }
                        }
                }
        }

}

/* Console
        Message: Not Implemented
        Status Code: 501
        Body: {"error":{"code":"","message":"Not Implemented"}}
*/
```

See also

Reference

- *powerGate.Erp.Client namespace*
- *Connect-ERP cmdlet*
- *Disonnect-ERP cmdlet*

6.2.3 ErpClientSettings Class

Settings used to manipulate requests, timeouts etc. send by the client.
Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.ErpClientSettings

Syntax

```
public class ErpClientSettings
```

The ErpClientSettings type exposes the following members.

Constructors

Type	Description
ErpClientSettings()	Initializes a new instance of the ErpClientSettings class.

Properties

Type	Name	Description
Uri	BaseUri	Gets or sets the service Url.
Action<HttpRespon	AfterResponse	Gets or sets the action executed after the OData request.
Action<HttpRequest	BeforeRequest	Gets or sets the action executed before the OData request.
Action<HttpClientH	OnApplyClientHandler	Gets or sets the action on HttpClientHandler.
Func<HttpMessa	OnCreateMessageHandler	Called before every request and the returned handler is used to send the the requests. **ATTENTION**: If set, then 'OnApplyClientHandler' will not be called anymore!
ICredentials	Credentials	Get or set the credentials to connect with the services.
TimeSpan	RequestTimeout	Gets or sets the time period to wait before the request times out.
PreferredUpdateMethod	PreferredUpdateMethod	Gets or sets the HTTP method (PUT or MERGE) used for updating OData entities.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.4 ErpObject Class

Provides the data for the ERP entity.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
System.Collections.Generic.Dictionary<string,object>
powerGate.Erp.Client.ErpObject

Syntax

```
public class ErpObject : Dictionary<string, object>
```

The ErpObject type exposes the following members.

Properties

Type	Name	Description
IErpEntityType	EntityType	Gets the EntityType of the ErpObject.

Methods

Type	Name	Description
Dictionary<string, object>	GetKeys()	Returns only the Key properties of the Entity.
Dictionary<string, object>	GetProperties()	Returns the properties and navigation properties (except Key properties) of the Entity.

Remarks

The **GetKeys()** function returns a Dictionary of Key properties with its values which uniquely identifies the Entity.
The **GetProperties()** function returns a Dictionary of properties and navigation properties (except Key properties) of the Entity.

Readonly Results

The **GetKeys()** and the **GetProperties()** function result **can not be modified**!
Instead modifications on the ErpObject will be returned by both methods.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.5 IErpClient Interface

Provides the base interface for the *ErpClient* class.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpClient : IDisposable
```

Properties

Type	Name	Description
IErpServices	Services Gets the list of connected services.	

Methods

Type	Name	Description
IErpService	ConnectErp(*ConnectionSettings* connectionSettings)	Creates a connection to a service using the specified *ConnectionSettings*.
void	Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.(Inherited from IDisposable.aspx).)

Extension Methods

Type	Name	Description
IErpService	ConnectErp(Uri service = null, ICredentials credentials = null, bool ignoreCertificates = false, Action<*ErpClientSettings*> onConnect = null)	Overloaded. Creates a connection to a service using the specified parameter values.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.6 IErpEntitySet Interface

Provides the interface for working with an EntitySet: Adding, Getting, Updating and/or Removing entities.

: \

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpEntitySet
```

Properties

Type	Name	Description
IErpEntityType	EntityType	Gets the EntityType of the EntitySet.
IMediaResources	MediaResources	Gets the instance of IMediaResources to Add, Get and/or Update MediaResources.
string	Name	The name of the EntitySet.
IErpService	Service	Gets the service of the EntitySet.
bool	SupportMediaResources	Indicates whether the EntitySet supports MediaResources.

Methods

Type	Name	Description
ErpObject	AddErpObject(Dictionary<string,object> properties)	Creates a new ErpObject for the EntitySet in the ERP-System.
ErpObject	GetErpObject(*SearchOptions* options)	Retrieves the specified ErpObject from the ERP-System
IEnumerable<*ErpObject*>	GetErpObjects(*QueryOptions* options)	Searches for ErpObject's depending on the passed options in the ERP-System.
void	RemoveErpObject(Dictionary<string,object> keys)	Removes the specified ErpObject from the EntitySet in the ERP-System.
ErpObject	UpdatErpObject(Dictionary<string,object> keys, Dictionary<string,object> properties)	Updates the specified ErpObject for the EntitySet in the ERP-System.

Extension Methods

Type	Name	Description
ErpObject	GetErpObject(Dictionary<string,object> keys, IEnumerable<string> expand = null, IEnumerable<string> select = null)	Overloaded. Retrieves the specified ErpObject from the ERP-System using the specified parameter values.
IEnumerable<*ErpO*	GetErpObjects(string filter = null, int top = 0, IEnumerable<string> expand = null, IEnumerable<string> select = null, IEnumerable<*OrderBy*> orderBy = null)	Overloaded. Searches for ErpObject's from the ERP-System using the specified parameter values.

Exceptions

In case of an invalid request the above methods will throw a *WebRequestException*.

Remarks

The property **MediaResources** con only be used on EntitySets that are supporting streaming.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Add a new ErpObject

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace EntitySetSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/OData/OData.svc")))
                                {
                                        var products = service.EntitySets["Products"];
                                        //Add a new ErpObject
                                        var orangeJuice = products.AddErpObject(new␣
↪Dictionary<string, object>
                                        {
                                                {"ID", 5},
                                                {"Name", "Orange Juice"},
                                                {"Description", "The original Orange␣
↪Juice. Refreshing!"},
                                                {"ReleaseDate", "2006-08-04T00 ,00 ,00Z"}
↪,
                                                {"DiscontinuedDate", null},
```

(continues on next page)

(continued from previous page)

```
                                                {"Rating", 3},
                                                {"Price", 22.8}
                                        });
                                        Console.Write("Created new Product: {0}",␣
↪orangeJuice["Name"]);
                                }
                        }
                }
        }
}
```

Get a specific ErpObject

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace EntitySetSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V4/Northwind/Northwind.svc")))
                                {
                                        var categories = service.EntitySets["Categories
↪"];

                                        //Get the customer with ID CACTU
                                        var category = categories.GetErpObject(new␣
↪SearchOptions
                                        {
                                                Keys = new Dictionary<string, object> {
↪{ "CategoryID", 1 } },
                                                Expand = new[] { "Products" },
                                                Select = new[] { "CategoryID" }
                                        });
                                        Console.Write("Category ID: {0}", category[
↪"CategoryID"]);
                                }
                        }
                }
        }

}
```

Search for ErpObject's with a filter

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;
```

(continues on next page)

(continued from previous page)

```

namespace EntitySetSample {

        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/Northwind/Northwind.svc/")))
                                {
                                        var customers = service.EntitySets["Customers"];
                                        //Get all customers with a company starting with
↪'A'
                                        var foundCustomers = customers.
↪GetErpObjects(filter: "startswith(CompanyName,'A')");
                                        foreach (var customer in foundCustomers)
                                                Console.Write("Customer {0} with Company
↪{1}", customer["CustomerID"], customer["CompanyName"]);
                                }
                        }
                }
        }

}
```

Remove an ErpObject

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace EntitySetSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/OData/OData.svc")))
                                {
                                        var products = service.EntitySets["Products"];
                                        //Remove Product with ID '1'
                                        products.RemoveErpObject(new Dictionary<string,␣
↪object> { { "ID", 1 } });
                                }
                        }
                }
        }

```

(continues on next page)

(continued from previous page)

```
}
```

Update an ErpObject

```
using System;
using System.Collections.Generic;
using powerGate.Erp.Client;

namespace EntitySetSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http://services.odata.org/V3/OData/OData.svc")))
                                {
                                        var products = service.EntitySets["Products"];
                                        //Get Product to update
                                        var product = products.GetErpObject(new Dictionary<string, object> { { "ID", 9 } });
                                        //Updating Rating and Price properties
                                        var updatedProduct = products.UpdatErpObject(product.GetKeys(), new Dictionary<string, object> { { "Rating", 9 }, { "Price", 1.99 } });
                                        Console.Write("New Price: {0}", updatedProduct["Price"]);
                                }
                        }
                }
        }

}
```

See also

Reference

- *powerGate.Erp.Client namespace*
- *Add-ERPObject cmdlet*
- *Get-ERPObject cmdlet*
- *Get-ERPObjects cmdlet*
- *Remove-ERPObject cmdlet*
- *Update-ERPObject cmdlet*

6.2.7 IErpEntitySets Interface

Provides the interface for working with a list of EntitySets.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpEntitySets : IEnumerable<IErpEntitySet>
```

Properties

Type	Name	Description
IErpEntitySet	Item[string name]	Gets the EntitySet with the specified name.

Methods

Type	Name	Description
IEnumerable<*IErpEntitySets*>	Find(string entityTypeName)	Searches for EntityTypes having the specified name.
IEnumerator<*IErpEntitySets*>	GetEnumerator()	Returns an enumerator that iterates through the collection.(Inherited from IEnumerable<T>).

See also

Reference

- *powerGate.Erp.Client namespace*
- *Get-ERPEntitySets cmdlet*

6.2.8 IErpEntityType Interface

Provides the interface holding data about the EntityType.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpEntityType
```

Properties

Type	Name	Description
IErpEntitySet	EntitySet	Gets the EntitySet of the EntityType.
IErpService	Service	Gets the EntitySet of the Service.
string	Name	The name of the EntityType.
string	Namespace	The namespace of the EntityType.
IErpProperties	Keys	Gets the Key properties which uniquely identifies the EntityType.
IErpProperties	Properties	Gets the properties for the EntityType.
IErpNavigationProperties	Navigationproperties	Gets the navigation properties for the EntityType.

Methods

Type	Name	Description
ErpObject	NewErpObject()	Creates a new and empty *ErpObject* instance of the current EntityType. The properties will be filled with the default values.

Remarks

The property **EntitySet** returns the EntitySet where this EntityType is assigned to. Some EntityTypes are not assigned to an EntitySet an the property will return Null for them.

The **NewErpObject()** creates a new *ErpObject* instance of the current EntityType, by analyzing the $metadata for required and optional Properties and NavigationProperties.
The properties will be filled with the default values (see __DefaultValue__ in *Property*). For nullable properties (see IsNullable in *Property*) the value will be null.
For NavigationProperties with a target *Multiplicity* of One, the function is able to create the required target instance as well (recursive on multiple levels).
NavigationProperties can therefore safely be casted to type *ErpObject* or type Dictionary<string,object> (or IEnumerable of the described Type when they are collections).

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Create a new empty ErpObject instance

```
using System;
using powerGate.Erp.Client;

namespace EntityTypeSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/Northwind/Northwind.svc")))
                                {
                                        //Get the Territory EntityType
                                        var entityType = service.EntityTypes[
↪"NorthwindModel.Territory"];
                                        //Create a new empty ErpObject instance from the␣
↪Territory EntityType
                                        var territory = entityType.NewErpObject();
                                        Console.Write("Region ID: {0}", territory[
↪"RegionID"]);
                                }
                        }
                }
        }
}
```

See also

Reference

- *powerGate.Erp.Client namespace*
- *New-ERPObject cmdlet*

6.2.9 IErpEntityTypes Interface

Provides the interface for working with a list of EntityTypes.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpEntityTypes : IEnumerable<IErpEntityType>
```

Properties

Type	Name	Description
IErpEntityType	Item[string name]	Gets the EntityType with the specified name.

Methods

Type	Name	Description
IEnumerable<*IErpEntityType*>	Find(string entityTypeName)	Searches for EntityTypes having the specified name.
IEnumerator<*IErpEntityType*>	GetEnumerator()	Returns an enumerator that iterates through the collection.(Inherited from IEnumerable<T>).

Remarks

The **Item[]** property and **Find()** function are supporting to pass additionaly parts of the namespace (e.g. NorthwindModel.Territory instead of Territory).

See also

Reference

- *powerGate.Erp.Client namespace*
- *Get-ERPEntityTypes cmdlet*

6.2.10 IErpNavigationProperties Interface

Provides the interface for working with a list of NavigationProperties.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpNavigationProperties : IEnumerable<NavigationProperty>
```

Properties

Type	Name	Description
NavigationProperty	Item[string name]	Gets the NavigationProperty with the specified name.

Methods

Type	Name	Description
IEnumerator<*NavigationProperty*>	GetEnumerator()	Returns an enumerator that iterates through the collection.(Inherited from IEnumerable<T>).

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.11 IErpProperties Interface

Provides the interface for working with a list of ERP Properties.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpProperties : IEnumerable<Property>
```

Properties

Type	Name	Description
Property	Item[string name]	Gets the Property with the specified name.

Methods

Type	Name	Description
IEnumerator<*Property*>	GetEnumerator()	Returns an enumerator that iterates through the collection.(Inherited from IEnumerable<T>).

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.12 IErpService Interface

Provides the interface for working with an ERP Service.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpService : IDisposable
```

Properties

Type	Name	Description
IErpEntitySets	EntitySets	Gets the EntitySets of the service.
IErpEntityTypes	EntityTypes	Gets all the EntityTypes of the service.
bool	IsAvailable	Indicates whether the service is available.
string	Name	The name of the service.
Uri	Url	The Url of the service.

Methods

Type	Name	Description
void	Dispose()	Disconnects the service from the ErpClient. Also Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.(Inherited from IDisposable.)

Remarks

When calling **Dispose()** the service is disconnected from the *ErpClient*.
If the service is a CatalogService, the CatalogService and all its known services get disconnected.

See also

Reference

- *powerGate.Erp.Client namespace*
- *Get-ERPServices cmdlet*

6.2.13 IErpServices Interface

Provides the interface for working with a collection of ERP Services.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public interface IErpServices : IEnumerable<IErpService>
```

Properties

Type	Name	Description
IErpService	Item[Uri url]	Gets the service with the specified Uri.
IErpService	Item[string url]	Gets the service with the specified url as string.

Methods

Type	Name	Description
IEnumerable <*IErpService*>	Find(string serviceName)	Searches for services having the specified serviceName.
IEnumerator <*IErpService*>	GetEnumerator()	Returns an enumerator that iterates through the collection.(Inherited from IEnumerable<T>).

Remarks

The **Item[]** property and **Find()** function are supporting to pass only parts of the service Url (e.g. MaterialService instead of http://localhost:8080/PGS/ERP/MaterialService).)

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes.

Get service by url

```
//ErpClient already connected to Odata services
var service = erpclient.Services[new Uri("http://services.odata.org/V4/Northwind/
↪Northwind.svc")]
Console.Write("Service: {0}", service.Name);
```

Find service by name

```
//ErpClient already connected to powerGateServer Catalogservice
var services = erpclient.Services.Find("MaterialService");
foreach (IErpService service in services)
        Console.Write("Service: {0}", service.Name);
```

See also

Reference

- *powerGate.Erp.Client namespace*
- *Get-ERPServices cmdlet*

6.2.14 IMediaResources Interface

Provides the interface for working with MediaResources: Adding, Getting and/or Updating -ErpMedias.

Namespace: powerGate.Erp.Client\ **Assembly:** powerGate.Erp.Client.dll

Syntax

```
public interface IMediaResources
```

Properties

Type	Name	Description
IErpEntitySet	EntitySet	Gets the EntitySet of this MediaResources.

Methods

Type	Name	Description
void	GetErpMedia(Dictionary<strir keys, Stream stream)	Downloads the Media Resource of an existing Media Link Entry (MLE) and writes it to the passed Stream.
ErpObject	AddErpMedia(*MediaCreateOp* options)	Creates a new Media Link Entry (MLE) with the request body containing the Media Resource (MR) and the Content-Type header indicating its media type. In other words it will create a streamable entity (Media Link Entry) and upload it together with the specified file (Media Resource).
void	UpdateErpMedia(*MediaUpdateO options*)	Updates the Media Resource of an existing Media Link Entry (MLE).

Extension Methods

Type	Name	Description
ErpObject	AddErpMedia(Stream data, Dictionary<string,object> properties=null, string contentType = "application/octet-stream")	Overloaded. Creates a new Media Link Entry (MLE) with the request body containing the Media Resource (MR) using the specified parameter values.
void	UpdateErpMedia(Dictionary<string, object> keys, Stream data, string contentType = "application/octet-stream")	Overloaded. Updates the Media Resource of an existing Media Link Entry (MLE) using the specified parameter values.

Exceptions

In case of an invalid request the above methods will throw a *WebRequestException*.

Remarks

The **GetErpMedia(...)** function copies the downloaded binary data into the passed Stream object, and therefore the Stream needs to be writable.
For Streams that support seeking, the position is automatically set to starting position in order to directly allow reading it's content.

The **CreateErpMedia(...)** and **UpdateErpMedia(...)** functions reading and uploading the binary data from the passed Stream object, and therefore the Stream has to be readable.

Examples

In the following examples we are using public OData Services (http://services.odata.org) for demonstration purposes:

Download ErpMedia

```
using System;
using System.Collections.Generic;
using System.IO;
using powerGate.Erp.Client;

namespace MediaResourcesSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http://services.odata.org/V3/OData/OData.svc")))
                                {
                                        var advertisments = service.EntitySets["Advertisements"];
                                        //Check if the EntitySet supports MediaResources
                                        if (advertisments.SupportsMediaResources)
                                                using(var downloadData = new FileStream("C:\Temp\DownloadedFile.txt", FileMode.Create,FileAccess.Write))
                                                {
                                                        //Download MediaResource
                                                        advertisments.MediaResources.GetErpMedia(new Dictionary<string, object> { { "ID", Guid.Parse("db2d2186-1c29-4d1e-88ef-a127f521b9c6") } }, downloadData);
                                                }
                                }
                        }
                }
        }

}
```

Add a new ErpMedia

```
using System;
using System.Collections.Generic;
using System.IO;
using powerGate.Erp.Client;

namespace MediaResourcesSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
```

(continues on next page)

(continued from previous page)

```
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/OData/OData.svc")))
                                {
                                        var advertisments = service.EntitySets[
↪"Advertisements"];
                                        //Check if the EntitySet supports MediaResources
                                        if (advertisments.SupportsMediaResources)
                                                using(var uploadData = File.OpenRead(@
↪"C:\Temp\TestMedia.txt"))
                                                {
                                                        var mediaCreateOptions = new␣
↪MediaCreateOptions
                                                        {
                                                                Data = uploadData ,
                                                                ContentType = "text/plain
↪",
                                                                Properties = new␣
↪Dictionary<string, object> { { "Name", "My new Advertisment, Yeaahh!" } }
                                                        };
                                                        //Create a new Media Link Entry␣
↪with a MediaResource
                                                        var newAdvertisment =␣
↪advertisments.MediaResources.AddErpMedia(mediaCreateOptions);
                                                        Console.Write("Advertisment ID:
↪{0}", newAdvertisment["ID"]);
                                                }
                                }
                        }
                }
        }

}
```

Update existing ErpMedia

```
using System;
using System.Collections.Generic;
using System.IO;
using powerGate.Erp.Client;

namespace MediaResourcesSample {
        class Program
        {
                static void Main(string[] args)
                {
                        using (var erpclient = new ErpClient())
                        {
                                using (var service = erpclient.ConnectErp(new Uri("http:/
↪/services.odata.org/V3/OData/OData.svc")))
                                {
                                        var advertisments = service.EntitySets[
↪"Advertisements"];
```

(continues on next page)

(continued from previous page)

```
                                        //Check if the EntitySet supports MediaResources
                                        if (advertisments.SupportsMediaResources)
                                                using(var uploadData = File.OpenRead(@
↪"C:\Temp\TestMedia.txt"))
                                                {
                                                        //Update MediaResource of Media␣
↪Link Entry
                                                        advertisments.MediaResources.
↪UpdateErpMedia(
                                                                keys: new Dictionary
↪<string, object> { { "ID", Guid.Parse("f89dee73-af9f-4cd4-b330-db93c25ff3c7") } },
                                                                data: uploadData );
                                                }
                                        }
                                }
                        }
                }
        }
}
```

See also

Reference

- *powerGate.Erp.Client namespace*
- *Add-ERPMedia cmdlet*
- *Get-ERPMedia cmdlet*
- *Update-ERPMedia cmdlet*

6.2.15 MediaCreateOptions Class

Specifies options to use for creating Media Resources.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.MediaCreateOptions

Syntax

```
public class MediaCreateOptions
```

The MediaCreateOptions type exposes the following members.

Constructors

Name	Description
MediaCreateOptions()	Initializes a new instance of the MediaCreateOptions class.

Properties

Type	Name	Description
string	Content-Type	Specifies the content type of the HTTP request.
Stream	Data	Stream of Data to upload to the ERP System.
Dictionary<string, object>	Properties	The properties for the entity beeing created. Those are passed as Slug-Header to the ERP System.

Remarks

The **ContentType** is used to specify the nature of the Data being uploaded. With the appropriate content type the web browser can open the Data with the proper extension/plugin.
If the content type contains text (e.g text/plain, text/html...) as type or json , xml (e.g application/json, application/xml...) as subtype, then the content of the Data is uploaded to the server as UTF-8 Encoded text.

The **Properties** are passed as Slug-Header to the server, in the augmented BNF syntax.
Please note, that the field-values are passed in JSON-format (depending on the OData-version) to the server. The Slug header is send as defined in Atom Publishing Protocol by encoding the data to UTF-8 and later using percent encoding (for all octets outside the ranges %20-24 and %26-7E)!

Note: All Properties are formatted in **following format**: `Property1='SomeText',Property2=666`
This format is supported by SAP and powergateserver. Note that other ERP systems could expect data in different format!

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.16 MediaUpdateOptions Class

Specifies options to use for updating Media Resources.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.MediaUpdateOptions

Syntax

```
public class MediaUpdateOptions
```

The MediaUpdateOptions type exposes the following members.

Constructors

Name	Description
MediaUpdateOptions()	Initializes a new instance of the MediaUpdateOptions class.

Properties

Type	Name	Description
Dictionary<string, object>	Keys	The reference properties for searching the item which uniquely identifies the Entity.
string	ContentType	Specifies the content type of the HTTP request.
Stream	Data	Stream of Data to upload to the ERP System.

Remarks

The **ContentType** is used to specify the nature of the Data being uploaded. With the appropriate content type the web browser can open the Data with the proper extension/plugin.
If the content type contains text (e.g text/plain, text/html...) as type or json , xml (e.g application/json, application/xml...) as subtype, then the content of the Data is uploaded to the server as UTF-8 Encoded text.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.17 Multiplicity Enumeration

Enumerates the multiplicities of navigation properties.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public enum Multiplicity
```

Members

Member name	Description
OneToOne	The Multiplicity of the association ends is one to one.
OneToZeroOrOne	The Multiplicity of the association ends is one to zero or one to one.
ZeroOrOneToOne	The Multiplicity of the association ends is zero to one or one to one.
OneToMany	The Multiplicity of the association ends is one to many.
ManyToOne	The Multiplicity of the association ends is many to one.
ManyToMany	The Multiplicity of the association ends is many to many.
ManyToZeroOrOne	The Multiplicity of the association ends is many to zero or many to one.
ZeroOrOneToZeroOrOne	The Multiplicity of the association ends is zero or one to zero or one.
ZeroOrOneToMany	The Multiplicity of the association ends is zero or one to many.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.18 NavigationProperty Class

Provides metadata informations about the NavigationProperty.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.Property
powerGate.Erp.Client.NavigationProperty

Syntax

```
public class NavigationProperty : Property
```

The NavigationProperty type exposes the following members.

Constructors

Name	Description
NavigationProperty()	Initializes a new instance of the NavigationProperty class.

Properties

Type	Name	Description
bool	IsCollection	Indicates whether the navigation property is a collection.
Multiplicity	Multiplicity	Gets the the multiplicity of the navigation property.
IErpEntityType	TargetEntityType	Gets the TargetEntityType of the navigation property.
string	DefaultValue	Gets the default value of the navigation property.
bool	IsNullable	Indicates whether the navigation property is nullable.
string	DefaultValue	Gets the name of the navigation property.
Type	Type	Gets the Type of the navigation property.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.19 OrderBy Class

Specifies options to use for ordering ERP entities.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.OrderBy

Syntax

```
public class OrderBy
```

The OrderBy type exposes the following members.

Constructors

Name	Description
OrderBy(string propertyName, OrderDirection? direction = null)	Initializes a new instance of the OrderBy class by passing the property name that should be ordered.

Properties

Types	Name	Description
string	PropertyName	The property used to order a collection of entities.
OrderDirection?	Direction	Nullable. The order of a sequence (ascending or descending).

Remarks

When the **OrderDirection** property is not set, the entities are ordered in the direction defined on the Erp-side (this should be ascending order by default).

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.20 OrderDirection Enumeration

Specifies the order of a sequence (ascending or descending).

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public enum OrderDirection
```

Members

Member name	Description
Ascending	The items are sorted in ascending order.
Descending	The items are sorted in descending order.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.21 Property Class

Provides metadata informations about the Property.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.Property

Syntax

```
public class Property
```

The Property type exposes the following members.

Constructors

Name	Description
Property()	Initializes a new instance of the Property class.

Properties

Type	Name	Description
string	DefaultValue	Gets the default value of the property.
bool	IsNullable	Indicates whether the property is nullable.
string	DefaultValue	Gets the name of the property.
Type	Type	Gets the Type of the property.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.22 QueryOptions Class

Specifies options to use for searching entities in ERP.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.QueryOptions

Syntax

```
public class QueryOptions
```

The QueryOptions type exposes the following members.

Constructors

Name	Description
QueryOptions()	Initializes a new instance of the QueryOptions class.

Properties

Type	Name	Description
IEnumerable<string>	Expand	The Navigation property name(s) which should be expanded.
string	Filter	The OData filter which will be executed.
IEnumerable<*OrderBy*>	OrderBy	The order of the Item's. They can be ordered by the property name and/or direction.
IEnumerable<string>	Select	Specifiy the properties which should be explicitly requested and returned by the client.
int	Top	The amount of items which should be returned.

Remarks

The **Filter** property allows you to specify a filter with OData syntax. More informations about the OData Filter syntax can be found here.
The **OrderBy** property allows you to order the list of entities by property name(s) and/or direction (Ascending or Descending).
The **Expand** property allows you to expand multiple navigation properties. By default, when the expand property is **not** specified then most Erp-systems reurn the entity without navigation properties.
The **Select** property lets you receive only those properties which you want to have in the result.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.23 SearchOptions Class

Specifies options to use for retrieving a single entity from ERP.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
powerGate.Erp.Client.SearchOptions

Syntax

```
public class SearchOptions
```

The SearchOptions type exposes the following members.

Constructors

Name	Description
SearchOptions()	Initializes a new instance of the SearchOptions class.

Properties

Type	Name	Description
Dictionary<string, object>	Keys	The reference properties for searching the item which uniquely identifies the Entity.
IEnumerable<string>	Expand	The Navigation property name(s) which should be expanded.
IEnumerable<string>	Select	Specifiy the properties which should be explicitly requested and returned by the client.

Remarks

The **Expand** property allows you to expand multiple navigation properties. By default, when the expand property is **not** specified then most Erp-systems reurn the entity without navigation properties.
The **Select** property lets you receive only those properties which you want to have in the result.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.24 UpdateMethod Enumeration

Specifies the HTTP method used for updating.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Syntax

```
public enum UpdateMethod
```

Members

Member name	Description
MERGE	Represents an HTTP MERGE protocol method that is used to update an entity.
PUT	Represents an HTTP PUT protocol method that is used to update an entity.

See also

Reference

- *powerGate.Erp.Client namespace*

6.2.25 WebRequestException Class

The exception that is thrown when an error occurs while sending a request.

Namespace: powerGate.Erp.Client
Assembly: powerGate.Erp.Client.dll

Inheritance Hierarchy

System.Object
System.Exception
powerGate.Erp.Client.WebRequestException

Syntax

```
public class WebRequestException : Exception
```

The following members which allow to determine why the Web server response was not successful:

Properties

Type	Name	Description
string	Message	Gets a human-readable message that describes the current exception (Inherited from System.Exception).
string	RawResponse	Gets the plain message body from the server response.
object	Response	Gets the error data from the OData response or null for generic HTTP error responses.
int	StatusCode	Gets the HTTP status code of the server response.
string	Source	Indicates in which part of the ERP integration the error was caused (by the *"Local computer"*, *"powerGateServer"* or the "*ERP system*"). (Inherited from System.Exception).
string	StackTrace	Gets a string representation of the immediate frames on the call stack (provided by the powerGateServer Plugin or Erp system). (Inherited from System.Exception).

Remarks

Exceptions of this type are thrown by the *ErpClient*, *ErpEntitySet* and *MediaResources* when requesting data from the ERP Services.

The **Message** property returns the error message from within the OData response (the message of the most *InnerError* is provided, when such data is available in the OData response).
For generic HTTP error responses a returned reason phrase provides further information about the nature of the problem.

The **Response** property provides access to the data within OData error responses (see OData v3 and OData v4):

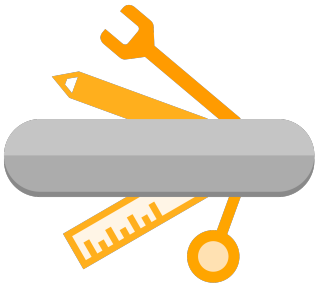
- The property `Message` contains a human-readable representation of the error.
- `ErrorCode` returns a service-defined error code which serves as a sub-status for the HTTP *StatusCode*.
- `InnerError` data can be available and contains information that will help to debug the service.

See also

Reference

- *powerGate.Erp.Client namespace*

The powerGate .NET library contains a set of classes, interfaces, and value types that provide the functionality to automate data synchronization with ERP systems.



The library supports:

- .Net Framework 4.7 or higher
- OData Version v1, v2, v3, v4.

The library tries to perform **as less server requests as possible**, in order to keep a **high performance** when communicating over the network or the internet.

Metadata

This means that e.g. the service **metadata** will be retrieved **only when required**, and only for the required services.

After retrieving metadata once, **no further metadata request is required**, because powerGate caches the server results!

6.2.26 Classes

Class	Description
ConnectionSettings	Settings used to connect with the ERP Service.
ErpClientSettings	Settings used to manipulate requests, timeouts etc. send by the client.
ErpClient	Class used to connect with ERP services.
ErpObject	Provides the data for the ERP entity.
MediaCreateOptions	Specifies options to use for creating Media Resources.
MediaUpdateOptions	Specifies options to use for updating Media Resources.
NavigationProperty	Provides metadata informations about the NavigationProperty.
OrderBy	Specifies options to use for ordering ERP entities.
Property	Provides metadata informations about the Property.
QueryOptions	Specifies options to use for searching entities in ERP.
SearchOptions	Specifies options to use for retrieving a single entity from ERP.
SapConnect	Class which provides prepared connection to use for connecting with SAP services.
WebRequestException	The exception that is thrown when an error occurs while sending a request.

6.2.27 Interfaces

Interface	Description
IErpClient	Provides the base interface for the *ErpClient* class.
IErpEntitySet	Provides the interface for working with an EntitySet: Adding, Getting, Updating and/or Removing -ErpEntities.
IErpEntitySets	Provides the interface for working with a list of EntitySets.
IErpEntityType	Provides the interface holding data about the EntityType.
IErpEntityTypes	Provides the interface for working with a list of EntityTypes.
IErpNavigationProperties	Provides the interface for working with a list of NavigationProperties.
IErpProperties	Provides the interface for working with a list of ERP Properties.
IErpService	Provides the interface for working with an ERP Service.
IErpServices	Provides the interface for working with a list of ERP Services.
IMediaResources	Provides the interface for working with MediaResources: Adding, Getting and/or Updating -ErpMedias.

6.2.28 Enumerations

Enumeration	Description
Multiplicity	Enumerates the multiplicities of navigation properties.
OrderDirection	Specifies the order of a sequence (ascending or descending).
UpdateMethod	Specifies the HTTP method used for updating.

6.3 UI Components

6.3.1 ERPComboBox

A WPF ComboBox control that automatically displays all *configured* list values for a bound ERP field.

Namespace: powerGate.UI.Components
Assembly: powerGate.UI.dll

Inheritance Hierarchy

ComboBox
powerGate.UI.Components.ERPComboBox

Syntax

```
<ERPComboBox SelectedValue="{Binding *}"/>
```

Properties

All properties inherited from ComboBox are available, whereby only the assignment of `SelectedValue` is required:

Property	Usage	Default value
SelectedValue	Binding to an ERP field of *Entity*.The value of this field is then selected by default.	
ItemsSourc		All list-values which are configured for the ERP field bound in `SelectedValue`.
SelectedValuePath		'Erp' (the underlying ERP value controls the display, and when selection changes, this ERP value is written back to the Entity field)
DisplayMemberPath		'Display' (the configured Display Text is shown for the selected value and dropdown items instead of the ERP values)
VerticalAlignment		'Center'
Padding		4

Remarks

The control displays all *configured* ERP values for a given field or, if configured, their corresponding display texts. This is allowed by the default values of `ItemsSource` and `DisplayMemberPath`.

In combination with *New-ERPObject -VaultEntity*, this combobox aids displaying the respective ERP value for the mapped Vault Properties value.

SelectedValue

The control can only provide the above functionalities automatically when `SelectedValue` is **bound** to an ERP field of an *Entity* returned by **New-ERPObject -VaultEntity**.

So the only requirement is a type mapping *configuration* between the ERP and Vault Entity type.
The customization development can also be continued without configured list-values. Once **Possible Values** are defined for the Item creation, these become visible in the combobox dropdown.

In contrast to a regular WPF ComboBox, a **clear error** tooltip displays to the Vault user when the ERP field has a value that is not available in its `ItemsSource`.

The list-values themselves are displayed in **sorted order** by default, allowing them to quickly find and pick the ERP value of their choise.
If descending sorting is desired, the list-values can be ordered differently in the *ERP Integration Settings* dialog.

> **Warning:** The `ERPComboBox` does not work correctly if the parent Window uses the `SizeToContent` attribute. This is caused by a WPF bug that affects the `Loaded` event which is required internally for the `ERPComboBox` to work.

Examples

Using a `ERPComboBox` requires an explicit XAML namespace declaration:
Assuming the StackPanel's or ErpComboBox's `DataContext` provides an *Entity* with a `UOM` property.

```
<StackPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:pg="clr-namespace:powerGate.UI.Components;assembly=powerGate.UI">
        <pg:ERPComboBox SelectedValue="{Binding UOM}"/>
</StackPanel>
```

Displaying Configured List Values from another (custom) configuration section:
The ERPComboBox control in this example cannot automatically determine the configured list-values because:

- the assigned *DataContext* is not a *New-ERPObject* `-VaultEntity` result (because Inventor)
- the Vault admin has only configured the language codes once for the ERP type 'Description', but does not want to configure an additional redundant ERP type mapping for 'BasicDataText'
- instead of a simple field a currently unsupported navigation property is bound

```
 $window = [Windows.Markup.XamlReader]::Load( (New-Object System.Xml.XmlNodeReader @'
<Window Title="Inventor - Create ERP Item"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:pg="clr-namespace:powerGate.UI.Components;assembly=powerGate.UI">

        <GroupBox Header="SAP Basic Data">
```

(continues on next page)

(continued from previous page)

```
                <StackPanel Orientation="Vertical">
                        ...
                        <pg:ERPComboBox Name="LanguageComboBox"
                                        SelectedValue="{Binding BasicData.BasicDataText.
↪LanguageISO}" />
                        <TextBox Text="{Binding BasicData.BasicDataText.MatlDesc}" />
                </StackPanel>
        </GroupBox>
</Window>
'@) )

# programmatically set the ItemsSource to the configured list-value that are also used␣
↪for Vault Items. Possible Language Codes are 'DE','IT','EN'
$vaultItem2SapDescription = $global:ERPSettings.GetTypeMapping('Item', 'material_srv.
↪Description')
$languageCodeField = $vaultItem2SapDescription.FieldMappingsForCREATE | Where-Object { $_
↪.ErpField -eq 'LanguageISO' }

$window.FindName('LanguageBasicTextComboBox').ItemsSource = $languageCodeField.
↪ListValues # Possible Language Codes are 'DE','IT','EN'

$partNumber_iProperty = $document.PropertySets.Item('Design Tracking Properties')['Part␣
↪Number']
$materialContext = New-ERPObject -EntityType 'material_srv.MaterialContext' -Properties @
↪{ Material = $partNumber_iProperty.Value }
$materialContext.Description = @( (New-ERPObject -EntityType 'material_srv.Description')␣
↪)
$materialContext.BasicData = New-ERPObject -EntityType 'material_srv.BasicData' -
↪Properties @{ BasicDataText = @() }
$materialContext.BasicData.BasicDataText += New-ERPObject -EntityType 'material_srv.
↪BasicDataText'
$window.DataContext = $materialContext
```

Disable other control in case of a Data Error:
When the *Entity* field holds invalid data, an automatically assigned DataErrorValidationRule ensures that the control shows these Validation.Errors:

```
<Grid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:pg="clr-namespace:powerGate.UI.Components;assembly=powerGate.UI">
        <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <pg:ERPComboBox Name="Categories" SelectedValue="{Binding Category}" Grid.Row="0
↪"/>

        <TextBox Text="{Binding Inventory, ValidatesOnDataErrors=True}" Grid.Row="1"> <!-
↪- error validation may also supported for other fields -->
                <TextBox.Style>
                        <Style TargetType="{x:Type TextBox}">
                                <Setter Property="IsEnabled" Value="False" />
```

(continues on next page)

(continued from previous page)

```
                                <Style.Triggers>
                                    <MultiDataTrigger>
                                        <MultiDataTrigger.Conditions>
                                            <Condition Binding="{Binding␣
↪ElementName=Categories, Path=(Validation.HasError)}" Value="False" />
                                            ...
                                        </MultiDataTrigger.Conditions>
                                        <Setter Property="IsEnabled" Value="True
↪" />
                                    </MultiDataTrigger>
                                </Style.Triggers>
                            </Style>
                        </TextBox.Style>
                    </Button>
</Grid>
```

powerGate provides custom WPF components to simplify the implementation and customization of Tabs and dialogs for ERP integrations.

6.3.2 Components

The following components are provided:

Component	Description
ERPComboBox	ComboBox pre-populated based on ERPIntegration *Configuration*

CHAPTER

SEVEN

LOGGING

powerGate 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\powerGate\Logs\powerGate.log'* and it contains only Warnings and Errors.
Perhaps you can find backups of previous logfiles in this directory.

The log4net settings file is located in C:\Program Files\coolOrange\Modules\powerGate\powerGate.log4net.
Further information about log4Net Configurations can be found here.

7.1 Log requests and responses

PowerGate has the opportunity to customize the format of the traced requests that are send to the server, and the received responses.

7.1.1 Log Level

powerGate has a special logging level called **TRAFFIC**. In this level all **requests** and **responses** will be logged. The level is located in the order between **DEBUG** and **INFO**, but you can even adjust this by changing the value '30001' in these lines:

```
<level>
        <name value="TRAFFIC" />
        <value value="30001" />
</level>
```

In the following link you will find the default log levels associated with the numeric values they have at the bottom: Level.cs

If you want to change the logging level for all appenders e.g. to "TRAFFIC", please visit the root logger and change the level in the lines:

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

you can configure the required logging level. You could set the level to "DEBUG", than all the levels above Debug and also Debug will be logged.

Note: When increasing the loglevel for the root logger, please make sure your favorite appenders minlevel is configured to allow logging messages with the specified level.

7.1.2 When to change the logging behavior?

When you have issues or when you want to get a more detailed knowledge about what powerGate is doing, you can increase the logging level.

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

---

Additionally you can change the logfile location or integrate the logging mechanism into your administrative environment by using build in EventLogMessages etc.

7.1.3 TrafficPatternLayout

Each appender has its own layout that defines how the log-messages are formatted. By using the special Layout **powerGate.Erp.Client.Traffic.TrafficPatternLayout** you have the possibility to configure two new ConverstionPatterns: **RequestConversionPattern**, **ResponseConversionPattern**.

The **TrafficPatternLayout** has all the functionality of a simple PatternLayout too. That means all logs that are not Requests or Responses can be configured by using the default **ConversionPattern** node.
Therefore all the available conversion pattern names from PatternLayout are available for the TrafficPatternLayout too.

7.1.4 RequestConversionPattern

This pattern applies when a **request** gets logged and additionally you have access to following options:

```
%Request{Protocol}
%Request{ProtocolVersion}
%Request{Method}
%Request{Url}
%Request{Headers}
%Request{Body}
```

For instance you are able to log something like this:

```
<RequestConversionPattern value="Called: %Request{Method}% on %Request{Url}" />
```

This will result in logs like:
Called: GET on localhost:8080/pgs/ERP/MaterialService/Materials

7.1.5 ResponseConversionPattern

This pattern applies when a **response** gets logged and additionally you have access to following options:

```
%Response{Protocol}
%Response{ProtocolVersion}
%Response{StatusCode}
%Response{Status}
%Response{Headers}
%Response{Body}
```

For instance you are able to log something like this:

```
<ResponseConversionPattern value="Received: %Response{StatusCode}% %Response{Status}" />
```

This will result in logs like:
Received: 200 OK

7.2 LogFile

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

```
<appender name="FileAppender" type="log4net.Appender.RollingFileAppender">
```

In the lines

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

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

In the line

```
<param name="File" value="${LOCALAPPDATA}\coolOrange\powerGate\Logs\powerGate.log" />
```

you can configure the outputpath and name of the logfile.

7.3 PowerShell IDE

PowerShell IDE's like PowerShell console (and PowerShell ISE) are configured to show the logging levels in a different color.

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

(continues on next page)

(continued from previous page)

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

These and many other options can be configured in the appender named **ColoredConsoleAppender**.

CHAPTER

EIGHT

CHANGE LOGS

8.1 powerGate v24

8.1.1 v24.0.15

20-03-2024

Fixed

- Issue in the *ERP Integration Settings* dialog where changes to list values for ERP fields were not saved when Vault Data Standard (VDS) is installed.
- Display problem in the *ERP Integration Settings* dialog where changing the Vault Entity Type caused the selected value to disappear.

8.1.2 v24.0.14

06-03-2024

General

- Updated powerEvents to version: 24.0.12
  This fixes occasional issues with *Sample.ConnectToERP* scripts on Vault 2021 environments, where the Vault Client may freeze on login.

8.1.3 v24.0.13

23-02-2024

Features

- The *ERP Integration Settings* dialog now allows to configure a list of values that are allowed for individual ERP fields, when creating Items in ERP.
  Especially for the manual Item creation, this makes it easy to control which values are displayed in selection lists. And this also applies if some ERP values don't directly match the data in Vault.
- New UI component: *ERPComboBox*
  - displays all the configured list values for the bound ERP field
  - shows the configured *DefaultValue* if no ERP Item exists
  - clear error display if no suitable ERP value is configured for the mapped Vault property

General

- Sample "ERP Item" tabs now display the list values in their *UnitOfMeasure* comboboxes, which are configured in the *ERP Integration Settings* dialog.
  (*Sample.Tab-ErpItem.xaml* and *Sample.ErpItemCreate.xaml*)
- The sample BOM Window "Item Transfer" now also takes these list value configurations into account.
  (*Sample.Tab-File-ErpBom* and *Sample.Tab-Item-ErpBom* scripts)
- Removed sample script *Sample.ManagePowerGateConfiguration.ps1* and the associated configuration file *Sample.PowerGateConfiguration.xml*, because these are no longer relevant after the changes described above.
  (*Sample.Menu-Inventor-CreateErpItem.ps1* hard-codes the original `GetPowerGateConfiguration 'UnitOfMeasures'` configuration)
- Updated powerEvents to version: 24.0.11
  This fixes possible performance problems caused by too frequent *Connect-ERP* calls during Vault Client or Inventor startup, especially when connecting multiple slow ERP services.

Fixed

- Failing ERP item creation when mapping Vault *Enum* Properties such as *Classification*, *Visualization Attachment*, *Property Compliance* or *File Link State*.
  The issue was caused by *New-ERPObject* returning a non-transferable `Autodesk. ... .PropertyDefinition+EnumeratedValue` object instead of the underlying Vault value.
- Incorrect error displays in the *ERP Integration Settings* dialog, when mapping numeric Vault Properties to number fields

8.1.4 v24.0.9

`16-01-2024`

Features

- The *ERP Integration Settings* dialog now allows to configure via mappings, how Items are created in ERP.
  For individual fields, it is now possible to specify whether data from Vault properties or fixed-values should to be transferred.
- Extended *New-ERPObject* with a `-VaultEntity` parameter that converts passed powerVault Entities to the corresponding ERP format, based on the configured mappings

General

- Sample "ERP Item" tabs and the BOM Window "Item Transfer" also take these mapping configurations into account.
  (*Sample.Tab-File-ErpItem*, *Sample.Tab-Item-ErpItem*, *Sample.Tab-File-ErpBom* and *Sample.Tab-Item-ErpBom* scripts)
- Removed default `<PropertyMappings>` for Vault Files and Items from *Sample.PowerGateConfiguration.xml*.
  During evaluation they can now be configured directly via the *ERP Integration Settings* dialog (see *`$ErpSettings.TypeMappings`*)
- UI improvements in the *ERP Integration Settings* dialog:
  - a red border clearly indicates that connection settings are missing or mappings are erroneous, e.g. if a Vault property or an ERP field was removed or renamed or when the mapped value types do not match.
  - a wait cursor is displayed while saving.
- Updated powerJobs Client to version: 24.0.5
  - This fixes compatibility issues with the LoginVault_Post event of the *Sample.ConnectToERP* script, where now also Vault Properties are retrieved for display in the *ERP Integration Settings* dialog.

  - In addition, the included powerVault version 24.0.7 solves the Show-Inspector issue where no Inspector window opened within *BOM Window functions*

Fixed

- Vault Client crash when invalid characters are entered as powerGateServer *Host Name* in the *ERP Integration Settings* dialog
- Display problems in the *ERP Integration Settings* dialog, which did not show connection settings in some exceptional situations:
  - during evaluation the used *Demo ERP system* was unfortunately only displayed on first opening
  - after manually importing a *powerGate.settings* file into Vault, the current settings are now displayed immediately, even without logging into Vault again

8.1.5 v24.0.7

`05-10-2023`

Features

- A new *ERP Integration Settings* dialog allows Vault administrators to configure ERP services on a Vault-wide basis.
  This enables all common powerGateServer situations as well as direct OData connections to be easily set up.
- The *Sample.ConnectToERP* script automatically connects to all these configured services when logging into the respective Vault in Vault Client and Inventor
- Extended *Connect-ERP* with a `-UseSettingsFromVault` parameter, which e.g. makes this easily possible also on the Job Processor

General

- Renamed sample script *Sample.ConnectToPowerGateServer.ps1* to *Sample.ConnectToERP.ps1*
- Instead of the *Sample.ManagePowerGateConfiguration* script, now the *Sample.ConnectToERP* script adds a Tools menu item to the Vault Client for opening the current Vault - ERP Integration Settings

Fixed

- Issue in the *BOM Window* where context menu buttons to select/unselect disappeared when custom display scaling settings are used

8.1.6 v24.0.5

`11-08-2023`

General

- Updated Licensing to version: 18.3.1
- Updated powerEvents to version: 24.0.4
  - This solves the problem where changes to ERP integration scripts being incorrectly reloaded on MTA background threads, resulting in a smoother debugging and development experience:
    Connection Error Dialogs are displayed again if problems occur with new changes in such scripts.
  - Changes can now be saved even while modal dialogs are open. It is no longer necessary to restart the Vault Client if, for example, Error Message Boxes, *Connection Error Dialog*, *BOM Windows*, Inspector dialogs or simple Message Boxes are displayed.
    Advantageously, for scripts based on our *ERP BOM* tab samples, changes in *BOM Window functions* now

take effect immediately without the need to re-open the window, reloading tab contents, or waiting for any Check-operations to complete.

- In addition, all sample ERP integration files are now installed in *%ProgramData%\coolOrange\Client Customizations*.

Breaking Changes

Change paths in Customization distribution mechanisms

Please note that the simplification of the folder structure for Client Customizations may require adjusting paths in automatic distribution mechanisms (e.g. setups or IT tools).
For more details, see the "Breaking Change" section in powerEvents v24.0.4

Fixed

- Issue in sample scripts for *ERP BOM* tabs where failed BOM row creations were incorrectly marked as successfully transferred in the BOM Window.
  BOMs are now again created with deep-create, which ensures that they are transferred as a whole and marked as erroneous, even if there are problems with the creation of individual BOM rows.

Note: For ERP interfaces implemented with a *ERP BOM* tab based on *v24.0.3* or *v24.0.4*, it is recommended to update the section for creating BOM headers and rows in the *Transfer-Boms* function.

8.1.7 v24.0.4

`30-05-2023`

Fixed

- Issue in sample script for Inventor Menu item "*Create/Update ERP Item*" and Inventor Menu item "*Link ERP Item*" where the parent assembly was used when a part or assembly was opened within an opened assembly

8.1.8 v24.0.3

`28-05-2023`

Features

- The user interface for the "ERP Item" and "ERP BOM" tabs in the Vault Client have been redesigned
- Added "*Link ERP Item*" tabs for searching and linking an existing ERP Item to a selected Vault File or a Vault Item
- Added "*Create/Update ERP Item*" Inventor Menu item for creating or updating an Item in ERP using iProperties of the active Part- or Assembly-Document
- Added "*Insert ERP Item*" Inventor Menu item for searching and inserting an existing ERP Item as Virtual Component or as Raw Material for the active Part- or Assembly-Document
- Added "*Link ERP Item*" Inventor Menu item for searching and linking an existing ERP Item to the active Part- or Assembly-Document

General

- The *Sample.ConnectToPowerGateServer* script is enabled by default on new environments and automatically connect to the public *Demo ERP system* by default

- Renamed sample scripts *Sample.TransferERPItemViaFileTab.ps1* and *Sample.TransferERPItemViaItemTab.ps1* to *Sample.Tab-File-ErpItem.ps1* and *Sample.Tab-Item-ErpItem.ps1*
- Renamed used *Sample.TransferERPItemTab.xaml* to *Sample.Tab-ErpItem.xaml*
- Renamed sample scripts *Sample.TransferERPBOMViaFileTab.ps1* and *Sample.TransferERPBOMViaItemTab.ps1* to *Sample.Tab-File-ErpBom.ps1* and *Sample.Tab-Item-ErpBom.ps1*
- Renamed used *Sample.TransferERPBOMTab.xaml* to *Sample.Tab-ErpBom.xaml*
- Renamed sample script *Sample.SyncERPTabConfiguration.ps1* to *Sample.ManagePowerGateConfiguration.ps1*
- Renamed used configuration *Sample.DefaultERPTabConfiguration.xml* to *Sample.PowerGateConfiguration.xml*

8.1.9 v24.0.2

`18-05-2023`

Features

- Easy and secure upgrade of already productive ERP integrations implemented with *v21* (or older) and Vault Data Standard, so that working can be continued as usual on all workstations

General

- *Updating* the product disables all sample ERP integration *scripts* by default if they were not already present on the workstation before.
  So for *v22* upgrades (or older) also all *Sample.TransferERP...Tab.ps1* and *Sample.SyncERPTabConfiguration.ps1* files are installed to the *.\Disabled* subdirectory, so they no longer need to be disabled manually for compatibility reasons.
- *Automatic display of Non-Terminating Errors*:
  - For Vault Data Standard integrations, connection errors and incorrect *ERP cmdlet* usages or parameters are no longer displayed by default.
    This eliminates the need to disable possible duplicate messages after updating from *v21* or older. Such errors are now clearly displayed only in powerEvents-based integrations (and PowerShell IDE's).
  - In powerEvents-based integrations the automatic display of these errors can be suppressed for individual *ERP cmdlet* calls by passing `-ErrorAction SilentlyContinue`

Fixed

- Issue with *Updates* on Job Processor environments (and workstations with unsupport Vault versions) when only main components (Cmdlets and .NET Library) were installed, but simple setup executions incorrectly installed sample files and powerJobs Client

Breaking Changes

Setup argument renamed and value changed
The *Setup argument* for installing only the main components (Cmdlets and .NET Library) has been renamed to ***MAIN_COMPONENTS_ONLY*** and the required value was changed to **1**.
Therefore updates on Job Processor environments (and workstations with unsupport Vault versions) should be performed using the adapted command-line argument:

Previous	Now
`"\\path\to\networklocation\powerGate24.0_Vault2024.exe" -silent ACCEPT_EULA=1 VAULT_ERP_INTEGRATION=0`	`"\\path\to\networklocation\powerGate24.0_Vault2024.exe" -silent ACCEPT_EULA=1 MAIN_COMPONENTS_ONLY=1`

Vault Data Standard integrations for v22.0.1 and later - automatic Error display removed
ERP integrations implemented with Vault Data Standard (e.g. powerGateTemplate based projects) do not contain any additional error checking or handling logic after *ERP cmdlet* invocations.
So that non-terminating connection errors (*v22.0.1*) and incorrect ERP cmdlet usages (*v23.0.1*) are not swallowed, the following line should be added after importing the `powerGate` module:

```
Import-Module powerGate
$global:Host.PrivateData.OnNonTerminatingError = $global:deprecated_
↪defaultErrorBehaviorForVDS
```

Note: Earlier VDS integrations implemented for *v21* or older, should already contain the required error handling logic and are therefore not affected (see *"Autodesk Vault Data Standard - Non-Terminating Errors"*).

8.1.10 v24.0.1

`27-04-2023`

General

- Added support for Vault 2024
- Updated Licensing to version: 18.2.29
- End User License Agreement (EULA) has changed
- Updated minimum required .NET Framework version to 4.7
- Removed DEPRECATED support for simple updates of ERP interfaces that are compiled against the *20.0* version of the *.NET library*.
  The policy files for v20 versions of the *powerGate.Erp.Client* assembly are no longer installed in the GAC, which requires a rebuild against the latest version.

Breaking Changes

Projects targeting .NET Framework versions lower than 4.7 :
Projects using the powerGate *.NET library* that target a lower .NET Framework version than 4.7 need to be upgraded to target at least 4.7 to compile when referencing the latest `powerGate.Erp.Client` assembly.

8.2 powerGate v23

8.2.1 v23.0.13

18-04-2023

General

- Updated powerEvents to version: 23.0.20.
  This fixes terminating errors that prevented the use of `finally` statements in *-Action* parameters of cmdlets like Register-VaultEvent, Add-VaultTab, Add-VaultMenuItem and Add-InventorMenuItem,
  which affect e.g. ERP integrations based on the powerGateTemplate project.

Fixed

- Display issue in the Error Message Box where *System.Management.Automation.RuntimeException* was displayed for all *non-terminating errors* instead of their actual exception type

8.2.2 v23.0.12

04-04-2023

Features

- The sample *Vault - ERP integration* can be tried out immediately after installing powerGate, as it automatically connects to a public *Demo ERP system*.
  This way, no more powerGateServer installation or "TestVault" configuration is required for the evaluation.

General

- Disabled the *Sample.ConnectToERP* script by default on new environments, as a connection to the powerGateServer ERP Plugin for evaluation purposes is no longer performed

Fixed

- Issue with *Sample.SyncERPTabConfiguration* where the script failed and an error dialog was displayed, when a user logged in to Vault with no `Vault Get Options` or `Vault Set Options` permission

8.2.3 v23.0.10

17-03-2023

Fixed

- Issue with *Update-ERPObject* where the cmdlet did not return a result even though the update was successful, when the *BeforeRequest/AfterResponse* action gets set in *-OnConnect* of the *Connect-ERP*

8.2.4 v23.0.9

07-03-2023

General

- Updated powerEvents to version: 23.0.19.
  This prevents crashes of the Vault Client that can occur in custom ERP Tabs after *scripting errors* in UI events or when using the `Show-Inspector` cmdlet,
  especially in scripts based on *Sample.TransferERPBOMViaFileTab* or *Sample.TransferERPBOMViaItemTab*.

8.2.5 v23.0.8

08-02-2023

Features

- Extended *Setup* with additional command-line argument for installing product without *sample files* and without powerJobs Client on Job Processor and unsupported Vault workstations

Fixed

- Issue with *Setup* that after upgrading it was not longer possible to uninstall via Programs & Features, when the Setup file of the previous installation was renamed or removed

8.2.6 v23.0.7

12-01-2023

Fixed

- Issue in BOM Window when multiple positions of the same item are displayed in the *BOM Tab* and the *custom BOM properties* of the first bomRow were incorrectly displayed for all rows and also transferred to ERP
- In the *Field Chooser* all BOM properties can be correctly identified with a corresponding BOM icon
- Issue with *Add-ERPMedia* where uploading a file to an EntitySet of an Odata v4 service failed when using the *-Properties* argument

> **Warning:** When upgrading to this version with saved layout files, the *columns* and their order on the *Item Tab* and *BOM Tab* are reset to the default value.

8.2.7 v23.0.6

19-12-2022

Features

- Added *Sample.SyncERPTabConfiguration* script that provides Tools Menu items in the Vault Client for uploading -and downloading configurations for the *ERP integrations*

8.2.8 v23.0.5

14-12-2022

Features

- The "ERP Item" tabs in the Vault Client allow to create new materials in the ERP system, with information from the selected *Vault file*, *Vault item* or manually entered data. In addition, these tabs also allow material information to be updated in the ERP system directly from the Vault Client.
- Enhanced "ERP BOM" tabs with the ability to batch transfer material- and BOM data to the ERP system. With the *BOM Window* the complete CAD-BOM of the selected *Vault file* or *Vault item* can be displayed and its status can be compared with the ERP system.
  BOMs, individual positions or items that do not yet exist or need to be updated can than be automatically transferred to ERP.

General

- Renamed sample scripts *Sample.DisplayERPItemInFileTab.ps1* to *Sample.TransferERPItemViaFileTab.ps1* and *Sample.DisplayERPItemInFileTab.ps1* to *Sample.TransferERPItemViaItemTab.ps1*
- Renamed used *Sample.DisplayERPItemTab.xaml* file to *Sample.TransferERPItemTab.xaml*
- Renamed sample scripts *Sample.DisplayERPBOMInFileTab.ps1* to *Sample.TransferERPBOMViaFileTab.ps1* and *Sample.DisplayERPBOMInFileTab.ps1* to *Sample.TransferERPBOMViaItemTab.ps1*
- Renamed used *Sample.DisplayERPBOMTab.xaml* file to *Sample.TransferERPBOMTab.xaml*

8.2.9 v23.0.4

06-12-2022

Features

- Added *Sample.ConnectToPowerGateServer* script that automatically connects to the test/productive powerGateServer environment depending on the logged in Vault
- Added *ERP Item tab* that displays the material number and other relevant information from the ERP system for a selected Vault file
- Added *ERP BOM tab* that displays the complete ERP bill of materials (BOM) for the selected Vault file
- Added *ERP Item tab* that displays the material number and other relevant information from the ERP system for a selected Vault item
- Added *ERP BOM tab* that displays the complete ERP bill of materials (BOM) for the selected Vault item

Fixed

- Issue with *Disconnect-ERP* and *IErpService.Dispose()* that performed unnecessary requests which terminated with the unclear error message *"Unexpected WebException encountered"* when closing connections to unavailable *Catalog Services* (powerGateServer and SAP systems)

Breaking Changes

ERP integrations without powerGateServer (SAP)
The *Sample.ConnectToPowerGateServer* script is active by default and will, depending on the Vault name try to connect to a powerGateServer on the localhost or ADMS.
For ERP integrations where no powerGateServer is used (e.g. with SAP systems), *Error Message Boxes* are displayed after logging into Vault.
The sample script must therefore be disabled.

8.2.10 v23.0.3

`10-11-2022`

Fixed

- Issue with *Update-ERPObject* that *-Keys* were missing in request body when using *PUT* as PreferredUpdateMethod

8.2.11 v23.0.2

`17-10-2022`

Fixed

- Issue with *Connection Error Dialog* not displayed in Vault Data Standard customizations (e.g. powerGateTemplate) and powerEvents Client Customizations

8.2.12 v23.0.1

`05-10-2022`

Features

- powerJobs Client is automatically installed and therefore adds support for *Vault Applications*:
  Vault Client- and Inventor 2023, 2022 and 2021
- An *Error Message Box* informs about erroneous *ERP cmdlet* usage and parameters by which no web requests can be sent
  (e.g. missing *Connect-ERP* invocations, $null values for *-Keys* or for not-nullable *-Properties*, wrong property names, property values of invalid type…)
- Also in the *BOM Window*, where multiple BOMs, rows and Items are checked or transferred, automatic *Error states* are displayed if *ERP cmdlets* with problematic parameter values are used.
- powerEvents Restriction events are automatically restricted when such issue occur in *ERP cmdlets*.

Breaking Changes

Requires Vault Professional Client
The product can only be installed on environments where Autodesk Vault Professional Client 2023, 2022 or 2021 is installed.

8.3 powerGate v22

8.3.1 v22.0.1

20-09-2022

Features

- A *Connection Error Dialog* informs Vault users about erroneous web requests, responses and their exact cause
- Similarly, in the *BOM Window*, BOMs, rows and Items are automatically marked with an *Error status* when connection problems occur during Check or Transfer operations
- powerEvents Restriction events are automatically restricted when ERP connection problems occur

General

- *WebRequestException Class* provides following properties:
  - `Source` informs about where the issue occurred. This can be either the local computer, powerGateServer or the ERP system
  - `StackTrace` provides details about the actual error occured in powerGateServer Plugins or on the web server of the ERP system
- Connection problems are displayed more clearly on all workstations when beeing caused by:
  - the local computer if no request can be send to the server (e.g. no internet connection available, proxy blocks access…) or if the *license has expired*
  - expired powerGateServer licenses (requires powerGateServer v21.0.5 or later)
  - the powerGateServer or the ERP system when no response is returned (e.g. server not responding in time, powerGateServer service not running…)
  - incorrect data input by the Vault user or the ERP integration (e.g. invalid credentials, execution of operations for non-existing entities, bad syntax in requests…) causing the server to respond with 4xx client errors
  - the powerGateServer or the ERP system while processing `$metadata` requests
- All Cmdlets now respond to non-terminating errors by additionally displaying the error messages in PowerShell consoles before the execution continues (common parameter *-ErrorAction:Continue*)
- Improved performance of *ERP Entity- and Media cmdlets* by reducing unnecessary `$metadata` requests to unavailable *Catalog Service*-services (powerGateServer and SAP systems).
  An attempt to first find the passed *-EntitySet* or *-EntityType* only within previously available services is performed.

Fixed

- Issue with *Connect-ERP* and *IErpClient.ConnectErp* which did not fail with *WebRequestExceptions* on failed connections to *Catalog Services* (powerGateServer and SAP systems)
- Follow-up errors after failed or missing *Connect-ERP* cmdlet calls in all *ERP Entity- and Media cmdlets* which terminated with the unclear error message *"No EntitySet found with the given name: …"*

- Issue with *Get-ERPObject* and *Get-ERPObjects* which failed when *-Expand* and *-Select* argument were used together on same navgation property
- Typo in the error message of *New-ERPObject* when multiple entity types match the passed *-EntityType* parameter

8.4 powerGate v21

8.4.1 v21.0.16

`09-05-2022`

Features

Bom Window

- Custom *Bom properties* can also be displayed for the root element in the *BOM Window*.
  With the help of powerVault v23.0.2, for example, model state information can also be displayed for Inventor main assemblies (see *full example*).

General

- The *Get-BomRows* *-BomHeader* argument provides also the `Bom_` properties of the root entity which got passed to the *Show-BOMWindow* *-Entity* parameter

8.4.2 v21.0.15

`04-03-2022`

Features

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

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

8.4.3 v21.0.14

`18-10-2021`

General

- Updated Licensing to version: 18.2.27

Features

Bom Window

- The *BOM Window* provides a dedicated column *Status Details* which holds the *Status Details* of a *BomRow* and *Item* entitity.

Fixed

Bom Window

- Issue with *Update-BomWindowEntity* where not passing a value for *-Status* resets the *Status* of the entity

Breaking Changes

Cmdlets: Update-BomWindowEntity renamed -Tooltip parameter
The *-Tooltip* parameter of the *Update-BomWindowEntity* Cmdlet was renamed to *-StatusDetails* as it provides are more appropriate description of the newly introduced behaviour of the parameter.

Objects: _Tooltip property renamed to _StatusDetails
The `_Tooltip` property of the *Bom*, *BomRow* and *Item* objects has been renamed to `_StatusDetails` as it holds both the values of the *Status Icon Tooltip* and the *Status Details Column* of the BOM Window.

8.4.4 v21.0.13

`11-08-2021`

Features

- *WebRequestException* provides more readable error messages from ERP systems.
  Its `Response` property grants direct access to the OData error response.
- Non-terminating errors within *Cmdlets* can now be retrieved more easily using $Error. When the cmdlets fail due to error responses from the ERP system, the variable provides the *WebRequestException*.
- *Get-ERPObject*: "Not Found" warnings are no longer logged when the ERP object cannot be found with the specified keys

Breaking Changes

Cmdlets: Different return value in error situation
The following cmdlets: *Get-ERPEntityTypes*, *Get-ERPEntitySets*, *Get-ERPServices*, *Get-ERPObjects* which previously returned $null on error, now return empty array.

.NET Library: GetErpObject method no longer throws exception
The `GetERPObject()` method and the corresponding extension method of the *IErpEntitySet Interface* interface from the .NET Library no longer throw exceptions if the ERP object cannot be found.

8.4.5 v21.0.11

`10-05-2021`

General

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

Features

Bom Window

- The size and position of the window is automatically *saved/restored*.

8.4.6 v21.0.8

17-02-2021

Features

Bom Window

- *Add-BomWindowEntity* with *-Type BomRow*:
  - *Entity properties* and *BOM properties*
  - Passed *Entity properties* (e.g. "Name") can be displayed in the *BOM Tab*

Fixed

Bom Window

- *Update-BomWindowEntity* updates the passed *Entity properties* in the BOM Window even for BomRows created by *Add-BomWindowEntity*, instead ignoring them.

Breaking Changes

Add-BomWindowEntity -Type BomRow

BOM properties have to be passed with prefix
The behavior that all passed properties are treated as *BOM properties* changed, as the cmdlet now distinguishes between *Entity properties* and BOM properties.
Therefore **BOM properties** have to be passed with a 'Bom_' prefix for keeping them displayed in the already configured BOM Window columns.

BOM properties returned with prefix
All the BOM properties in the *BomRow* result are returned with a 'Bom_' prefix.
When accessing BomRow property values, extend the property names with this prefix in order prevent accesses to not existing or wrong members.

8.4.7 v21.0.7

20-01-2021

General

- Updated Licensing to version: 18.1.22

8.4.8 v21.0.6

21-12-2020

General

- Updated Licensing to version: 18.1.21
- Copyright notices have changed

Fixed

- Issue that led to an unusable machine and failing Jobs after running Job Processor for a long time

8.4.9 v21.0.3

22-09-2020

Fixed

Bom Window

- BomRows of identical BOM's were not update correctly

8.4.10 v21.0.2

29-05-2020

General

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

Fixed

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

8.5 powerGate v20

8.5.1 v20.0.7

05-11-2019

General

- Updated Licensing to version: 18.0.10

8.5.2 v20.0.6

20-08-2019

General

- Updated Licensing to version: 18.0.8

8.5.3 v20.0.5

09-08-2019

Fixed

- Increasing memory usage when connecting ERP systems by using *$sapConnect* or *SapConnect*
- ParameterBindingException when using *Connect-Erp* multiple times with *$sapConnect*

8.5.4 v20.0.4

23-07-2019

Fixed

- Issue that host application gets unresponsive when License information is retrieved

8.5.5 v20.0.3

30-05-2019

General

- Updated Licensing to version: 18.0.7

8.5.6 v20.0.2

21-05-2019

Features

- Added support for *Stand-Alone Licensing*

General

- Updated Licensing to version: 18.0.6

8.5.7 v20.0.1

04-04-2019

Features

- Added support for *Token Licensing*
- *Trial mode* expires after 30 days

General

- End User License Agreement (EULA) has changed
- Updated Licensing to version: 18.0.3

8.6 powerGate v19

8.6.1 v19.0.14

28-11-2018

General

BomWindow

- Column values are immediately refreshed when updating custom properties

Features

BomWindow

- It is now possible to update standard properties with *Update-BomWindowEntity*:
  - For BomRows: *_Name, Bom_Number, Bom_PositionNumber, Bom_Quantity*
  - For Items: *_Name*

Fixed

BomWindow

- *BOM loading errors* occurred during Check or Transfer operations because *Get-BomRows* got executed after using *Add-BomWindowEntity*
- Column values did not get refreshed when updating custom properties that exist on the root entity
- The *BomRow* properties *Bom_PositionNumber* and *Bom_Quantity* were always of type 'string'
- *Update-BomWindowEntity* removed columns even if property still exists on different Items/BomRows
- Progressbar did not update correctly when items were added or removed during Check or Transfer operations
- The Check button was enabled before the *BomWindow* finished loading BOM

> **Warning:** When upgrading to this version with stored layout files, in the *Item Tab* the *columns* and their order will be resetted to default.

Breaking Changes

Show-BomWindow

When calling *Add-BomWindowEntity*, the function *Get-BomRows* is no longer executed for the newly added entity. This could cause issues in some very rare cases where Get-BomRows was expected to be called.

8.6.2 v19.0.11

06-11-2018

General

BomWindow

- Improved resizing behavior of columns and applied minimum width
- *Status* column is pinned to the right side

Fixed

BomWindow

- First column *Name* gets to small when many columns are added
- Some standard properties got removed after updating *-Properties* with *Update-BomWindowEntity*
  - For BomRows: standard properties *Bom_Number, Bom_PositionNumber, Bom_Quantity and _Name* were removed
  - For Items: *_Name* property was removed

Warning: It is recommended to remove the *stored layout* files when upgrading to this version, otherwise the *Status* column might not be pinned.

8.6.3 v19.0.9

17-10-2018

Features

BomWindow

- *Filter and order* BOMs and items by their *status*

Fixed

BomWindow

- Sorting and Filtering the *Name* column was not possible in the *BOM Tab*
- Applied *Filters* were not saved immediately
- *Name* column has now the same height as all other columns

Note: When upgrading to this version with *stored layout* files, the position of some columns might change. The column order has to be manually fixed.

8.6.4 v19.0.8

08-10-2018

General

- Updated Licensing to version: 17.0.3

Fixed

BomWindow

- Existing rows are getting duplicated when collapsing and expanding BOMs in the *BOM Tab*

8.6.5 v19.0.7

21-09-2018

Features

BomWindow

- After Check or Transfer operation completed, Message Box informs directly about *Success*, *Caution* and *Error* depending on the States of the processed BOMs and Items
- Improved Message Box to show a report of the occurred States within the operation, also when operation *aborted with exception*
- Tooltip remains visible while mouse is located on *Status* icon
- BOM and entity properties are displayed in the *BOM Tab* and the *field chooser*, also when they have the same name

General

- Assemblies *System.Windows.Interactivity* and *Microsoft.Expression.Interactions* are getting installed into the GAC

Fixed

BomWindow

- Stored columns are getting removed and the BOM layout gets reset when using entity properties and BOM properties with the same name
- Sorting settings of columns did not get saved correctly
- Crash with error message: "*Binding cannot be changed after it has been used*" for certain stored layouts
- Crash when using *Add-BomWindowEntity* with a property that already exists as entity property
- *show-BomWindow* crashed when invoked for the first time in a Vault session within DataStandard

> **Warning:** It is recommended to remove the *stored layout* files when upgrading to this version.

8.6.6 v19.0.5

10-08-2018

Features

BomWindow

- Re-designed the *tooltip and status* information for BOMs and BomRows:
  - Moved the BOM status from the *Status* column into the *Name* column (on the left of the BOM name). It provides only the status and tooltip for the BOM.
  - The *Status* column now instead displays the state and tooltip for the BomRows.
  - In case of an error during the Check or Transfer operation the BomRows, which were not *updated* during that process, will have the status *Unknown*.
  - Switched *Number* column with *Name* column in *BOM Tab*.
  - Renamed *Number* column to *Name* in *Item Tab*.

> **Warning:** The *Columns* and their order will be resetted to default.

8.6.7 v19.0.4

25-04-2018

Features

- powerGate *cmdlets* can be used in every IDE

General

- Updated to PowerShell 4.0
- Removed powerGate ISE shortcut from startmenu

Breaking Changes

Previous	Now
The cmdlet was automatically creating a new runspace that was cleaned up when closing the Window.	The cmdlets will **NOT** create a new runspace. This has the advantage that the required functions do no have to be imported as PowerShell Module anymore. **Attention**: When some sort of caching is made in the runspace, the cmdlet does not clean that up. Also pay attention that variables could be overwritten.

8.6.8 v19.0.3

22-02-2018

General

- Assemblies *coolorange.licensing* and *coolorange.Utils.UI* now gets installed in the GAC
- Applications using *powerGate.Erp.Client* library are still running after update without having to recompile them
- Replaced Log4PostSharp with PostSharp Diagnostics for extended Debug logging

Fixed

- Issue where logging did not work when powerGate was used in a 32-Bit process
- Issue where BomWindow did not display values of properties containing whitespace, caret (^) or comma (,) characters.

8.7 powerGate v18

8.7.1 v18.1.3

13-10-2017

General

- Cmdlets: require *.Net framework 4.5*
- Cmdlets: Moved *SapConnect* implementation to *.NETLibrary*

Features

- Added powerGate *.NETLibrary* in order to take full advantage of the powerGate features directly in your .NET application

Fixed

- Issue with SapConnect not refreshing the CSRF-Token properly, when additional Certificates are used for the communication

8.7.2 v18.0.4

21-04-2017

- Official Release

General

- Standardized Logging same as for other products
- Changed registry keys to "HKLM\Software\coolOrange s.r.l.\powerGate": Location and Version

8.8 powerGate v17

8.8.1 v17.1.68 beta

10-03-2017

Features

BomWindow

- Item Tab now also contains Item of the RootBom.
- BomRows now have different stylings depending on their Status.

Fixed

- *Connect-ERP*: Issue when connecting to service which requires authentication. Connection request was send twice.
- *Update-BomWindowEntity*: Issue when updating BomRow and Item properties, like "*@{'BOM_Test'='Test'}*" was added as "*BOM_BOM_Test*".
- *Update-BomWindowEntity*: When the item of a bomRow was updated, the changed properties were not available on the bomRow.
- *$sapConnect*: Issue when multiple cookies are required from SAP.

8.8.2 v17.1.60 beta

10-02-2017

Fixed

- Fixed Issue where BOM Window was crashing when updating a bomRow property
- Unhandled exceptions occured in the BOM Window are now handled internally. Therefore the calling application like *Vault* or *Poweshell_ISE* will not crash anymore because of unhandled exceptions occurred in the BOM Window.

8.8.3 v17.1.57 beta

12-01-2017

Features

BomWindow

- *Show-BomWindow* can be called also in other PowerShell IDE's
- Merged all *required functions* to single Check/Transfer functions
- *Required functions* are called a single time by passing all the entities as list (except *Get-BomRows*)
- New cmdlet: *Add-BomWindowEntity*
- New cmdlet: *Remove-BomWindowEntity*
- New cmdlet: *Update-BomWindowEntity*
- Introduced *status* and Tooltip. Those are also available arguments for *required functions*
- extended UI to show in MessageBox the thrown errors from PowerShell functions

Fixed

- Getting response with state *403 Forbidden* when using *$global:SapConnect_Authentication*
- *Performance issue*: when running an Item/BOM check/transfer, all entities seem to be busy for a long time and suddenly all entities complete very fast at the exact same time. In the background the requests are send to the server in the meantime, but UI does not update with the results quickly.
- *Performance issue*: After clicking 'Check' or 'Transfer' button, it takes very long time until first request is send to the server when a huge amount of entities becomes processed from the Window.

Breaking Changes

: Renamed Cmdlet Show-ErpBom to Show-BomWindow.

v16	v17
Convert-ToErpMaterial	**removed**, because the Bom-Window has no knowledge about ERP anymore
Convert-FromPsMateri	**removed**, because the differences are settable via *Update-BomWindowEntity* Cmdlet and are not automatically detected anymore.
Get-ErpMaterial	Code **moved** to *Check-Items* function. Use *Update-BomWindowEntity* to change the *Status* of the entity to New when not existing in ERP.
Compare-Material	Code **moved** to *Check-Items*. Use *Update-BomWindowEntity* to change the *Status* of the entity to Different (set Tooltip with differences manually) or Identical when it exists in ERP.
Create-ErpMaterial	Code **moved** to *Transfer-Items*. Create entity in ERP when the *Status* of the entity is New.
Update-ErpMaterial	Code **moved** to *Transfer-Items*. Update entity in ERP when the *Status* of the entity is Different.
Get-ErpBom	Code **moved** to *Check-Boms*. Use *Update-BomWindowEntity* to change the *Status* of the BOM-Header to New when the BOM does not exist in ERP.
Compare-BomHeader	Code **moved** to *Check-Boms*. Use *Update-BomWindowEntity* to change the *Status* of the BOM-Header to Different (set Tooltip with differences manually) or Identical when the BOM does not exist in ERP.
Compare-VaultBomRow	Code **moved** to *Check-Boms*. Use *Update-BomWindowEntity* to change the *Status* of the BOM-Row to Remove (when Row does not exist in ERP-BOM) or Different (set Tooltip with differences manually).
Compare-ErpBomRow	Code **moved** to *Check-Boms* function. Use *Update-BomWindowEntity* to change the *Status* of the BOM-Row to New (when ERP-Row does not exist in Children) or Identical.
Create-ErpBom	Code **moved** to *Transfer-Boms*. Create BOM with all it's Children in ERP when the *Status* of the BOM-Header is New.
Update-ErpBom	Code **moved** to *Transfer-Boms*. Update BOM-Header in ERP when the *Status* of the BOM-Header is Different. Additionally check for all it's Children whether there *Status* is New, Remove or Different in order to create/remove/update those rows in the ERP-BOM.
Begin-GetErpMateri	**removed**, because the operation can be performed at the beginning of *Check-Items* function.
End-GetErpMateri	**removed**, because the operation can be performed at the end of *Check-Items* function.
Begin-Create/Update	**removed**, because the operation can be performed at the beginning of *Transfer-Items* function.
End-Create/Update	**removed**, because the operation can be performed at the end of *Transfer-Items* function.
Begin-GetErpBoms	**removed**, because the operation can be performed at the beginning of *Check-Boms* function.
End-GetErpBoms	**removed**, because the operation can be performed at the end of *Check-Boms* function.
Begin-Create/Update	**removed**, because the operation can be performed at the beginning of *Transfer-Boms* function.
End-Create/Update	removed, because the operation can be performed at the end of *Transfer-Boms* function.

8.8.4 v17.0.46

29-11-2016

General

- Removed Datastandard Integration support from documentation

Features

- Re-designed Cmdlets (see below *Breaking Changes*)
- Support for OData v1 - v4 standards
- Installed "powerGate 17.0 Logs" shortcut in start-menu section of powerGate

BomWindow

- Changed status-icons to more meaningful icons
- Introduced Localization for German environments

Fixed

- Moved LogFile to %LOCALAPPDATA%/coolOrange/Logs in order that Non-Admin users have write-access to the files.
- Crash in BOM Window when customizing the columns with the field chooser
- BomWindow shows error "Object reference not set to an instance of an object" when following BomRow properties are $null: Bom_PositionNumber, Bom_Number, Bom_Quantity etc. .

Breaking Changes

: **Connect-Erp**

v16	v17
-ServiceUri <String>	renamed to *Service*. Additionally changed Type to <Uri>
-Username <String>	renamed to *User*
-Startcache	**removed** because Services are loaded on demand.
Login-Window	**removed**

When connecting to SAP servers v17 requires additional *-OnConnect* argument for SAP endpoint.

Disconnect-Erp

v16	v17
Return type bool	changed to void

Get-ERPServices

v16	v17
-Refresh <String>	removed because Services are loaded on demand.
Return type List<Service>	changed to *ErpService[]*

$service: Use *ErpService* instead.

v16	v17
-Path <String>	replaced with Url. Changed type to <Uri>
-Loaded <bool>	removed, use Get-ERPServices instead and check if the service is contained.
-EntityContainer <Collection<IEntityTyp>>	removed, use Get-ERPEntitySets and Get-ERPEntityTypes instead.

$entitytype: use EntitySet and EntityType instead

v16	v17
-Service <String>	Changed type to <Uri> in EntitySet
-PropertyRefs <Collection<ISapProperty>>	replaced with Keys in Get-ERPEntityTypes. Changed type to <String[]>
-Properties <Collection<ISapPropert>>	replaced with Properties in EntityType. Changed type to <String[]>
-NavigationProperties <Collection<ISapNavigationPropert>>	replaced with NavigationProperties in EntityType. Changed type to <String[]>
-PropNames Collection<String>	removed, use Properties instead
-PropRefNames Collection<String>	removed, use Keys instead
-AllProperties Collection<ISapProperty>	removed, use Keys,Properties and NavigationProperties instead
-AllPropertiesNames Collection<String>	removed, use Keys,Properties and NavigationProperties instead
-AllMandatoryErpProperties Collection<ISapProperty>	removed, use Keys,Properties and NavigationProperties instead and filter for mendatory once.

Get-ERPObject

v16	v17
-Entity <String>	renamed to EntitySet. Additionally it was possible to specify the EntitySet by giving the bundle name, separated with '.'. Now it is possible to specify the EntitySet by providing the full or a part of it's whole Url.
-Key <String>	renamed to Keys. Changed to to <Hashtable> or <PSObject>
-Exists	removed
-Expand <String>	changed type to String[]. In past it was possible to specify multiple properties separated by ','. Pass array now.
-File <String>	removed, use *Get-ERPMedia* instead.
Return type <PSObject>	Expanded navigation property lists where accessible via ".results". Now the lists are directly accessible.
	empty on error

Get-ERPObjects

v16	v17
-Entity <String>	renamed to EntitySet. Additionally it was possible to specify the EntitySet by giving the bundle name, separated with '.'. Now it is possible to specify the EntitySet by providing the full or a part of it's whole Url.
-Top <String>	changed to <int>
-OrderBy <String>	changed to <String> or <hashtable> or <hashtable[]>.
-Expand <String>	changed to <String[]>. Note that it was possible to specify multiple properties separated by ','.
Return type <PSObject>	Expanded navigation property lists where accessible via ".results". Now the lists are directly accessible.
	empty on error

Remove-ERPObject

v16	v17
-Entity <String>	renamed to EntitySet. Additionally it was possible to specify the EntitySet by giving the bundle name, separated with '.'. Now it is possible to specify the EntitySet by providing the full or a part of it's whole Url.
-Key <String>	renamed to Keys. Changed to <Hashtable> or <PSObject>
Return type <bool>	<bool> with an additional property 'Error' containing the Exception/ErrorMessage.

Update-ERPObject

v16	v17
-Entity <String>"	renamed to EntitySet. Additionally it was possible to specify the EntitySet by giving the bundle name, separated with '.'. Now it is possible to specify the EntitySet by providing the full or a part of it's whole Url.
-Properties <String>	Splited now to two parameters called Keys and Properties. Changed to <Hashtable> or <PSObject>
-Deep switchparameter	removed
HTTP request **PUT**	HTTP request **MERGE**
Return type <bool>	empty on error.

Add-ERPObject

v16	v17
-Entity<String	renamed to EntitySet. Additionally it was possible to specify the EntitySet by giving the bundle name, separated with '.'. Now it is possible to specify the EntitySet by providing the full or a part of it's whole Url.
-Properties <Hashtable>	Changed to <Hashtable> or <PSObject>
Return type <PSObject>	empty on error.

Removed Cmdlets

Get-ERPConnectionStatus

Use result of *Connect-ERP* instead, e.g.:

```
$global:connected = Connect-ERP -Service ...
```

When running in different runspace, or $global:connected variable is not accessible because of the scope, you can use the following:

```
if( (Get-ErpServices) | where { $_.Name -eq '...' } ) {
        #already connected
}
```

Resolve-ERPObject

Get-ERPEntities

Use cmdlets Get-ERPEntitySets and *Get-ERPEntityTypes* instead.

Add-ERPDir

Use cmdlet Add-ERPObject for creating the DirContext and *Add-ERPMedia* for creating the MLE entity and the Media Resource.
Note that Add-ERPDir did not transmit the Slug-Header in the specified way. This is fixed with *Add-ERPMedia* and could therefore cause server side fixes too.

Update-ERPDir

Use cmdlet *Update-ERPMedia* for updating the Media Resource, and *Update-ERPObject* for updating the MLE.

powerGateErrors

removed
The last server response can be retrieved by using following example instead: *Accessing the last server response from your script*

8.9 powerGate 2016

8.9.1 v16.0.73

05-08-2015

Features

BomWindow:

- Disable specific boms
- Update exisintg Boms and Materials if they exists in Erp

General

- Remove Get-,Update- and Create-ErpMaterialForBomRow PowerShell functions from Communication.psm1
- Added Begin/End PowerShell function for Check/Process Item/BOM
- Removed powerJobs integration

BomWindow:

- Allow to Enable / Disable Items /Boms to be able to either processs them or not
- Disable and unselect Item when all BomRows with this item are unselected
- Bom view has Check functionality like the materials view
- do not show checkBoxes on empty boms in BomView
- Check,Process Button one control for each tab
- Merge column status and progress in Bom and Material View
- MessageBoxes should be shown when processing / checking is finished

Fixed

- improved PowerShell profile installation
- $powerGateErrors was not set in certain scenarios
- Setup installs ProgramData stuff on wrong drive in special cases
- localization: default only working on english vault environment
- improved Upgrade functionality
- When WindowsPowershell folder does not exists setup aborted
- Create-ErpItem: created invalid json-request when data contains special characters
- Wrong installation, if there are multiple installations of datastandard
- BomWindow: same BomRow on multiple levels in the Bom is not handled as same item
- BomView: process Bom for item that is located multiple times in bom is not processed once
- Add-ErpObject crashes when pasing null values
- BOM with Virtual components are handled correctly
- Assembly with one Component does not show components in window
- Component which is used in multiple Boms can not be checked / processed
- Material Update on Items generates new MaterialNumber for each Item

- “usage count” column in BOM window is not set correctly
- BOM Window shows different Row Order than Vault: should show hierarchical number e.g. 1.1.3
- Save Window settings failes and crashes BomWindow if no rights on ProgramData folder
- BOMtransfer not working in PowerShell v2
- removed unnecessary logs: powerGateLogger - ERROR
- licensing

8.9.2 v16.0.54

18-06-2015

Features

- **Bom Window** for bomcreation
- new **entity model**

General

- Removed XamlIncluder from CAD / Vault Datastandard Integration
- Generate new number - When creating an item, powerGate shall suggest the next free ERP number
- Changed logfile location to “%AllUsersProfile%/coolorange/powerGate”
- Changed login to adress full uri to catalogservice
- Renamed cmdlets from “SAP” to “ERP” (e.g. Get-SAPItem => Get-ErpItem)
- Renamed cmdlets from “Item” to “Object” (e.g. Add-ERPItem => Add-ERPObject)
- Renamed Tools Shortcut´s in StartMenu

Fixed

- Get-ErpItem with entity-properties of type numeric
- Add-erpitem fails with navigationentity, which exists more then once

powerGate, our .NET and PowerShell-extension, makes it possible to create scripts and applications that automate data synchronization with ERP systems.
It is designed for companies in which the engineering department works with Autodesk tools such as AutoCAD, AutoCAD Mechanical and Inventor for authoring data, as well as Vault for managing data.

In addition, with the help of powerEvents and powerJobs Processor the integration between these Autodesk applications and the ERP system can be realized in a simple way.
For example, it is possible to also introduce new workflows were the presence of ERP materials and BOMs is automatically checked after Lifecycle State changes in Vault.

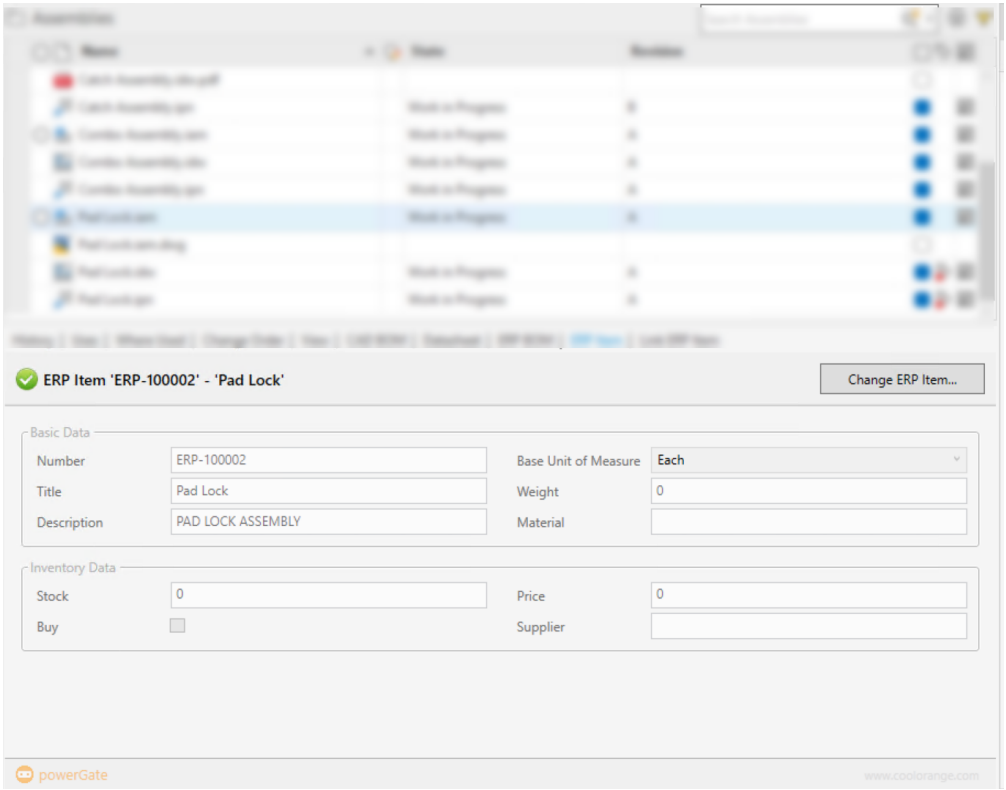
CHAPTER

NINE

FEATURES

9.1 Pull and Transfer material information

During the design process, it is crucial for the engineer to access material information from the ERP system.
With powerGate, the designer can assign the material number and other relevant information from the ERP right from the beginning to the component he is creating.
In the Vault Client this ERP data can be viewed, created or modified in specially provided *ERP Item tabs*:

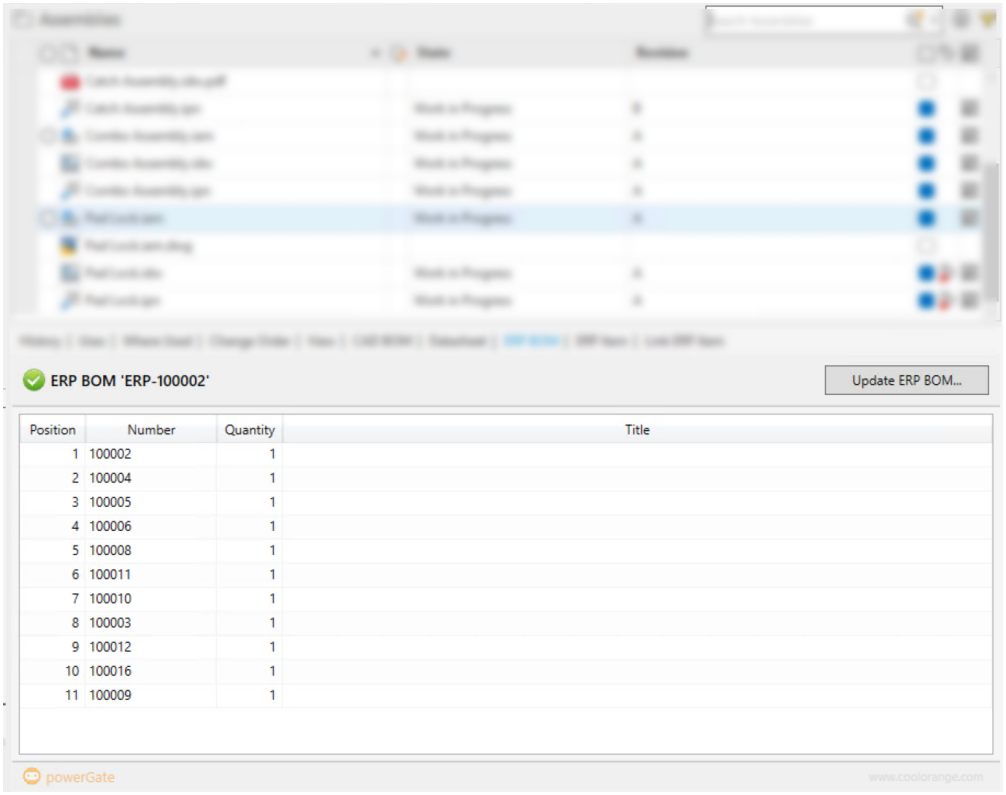


9.2 Pull project information

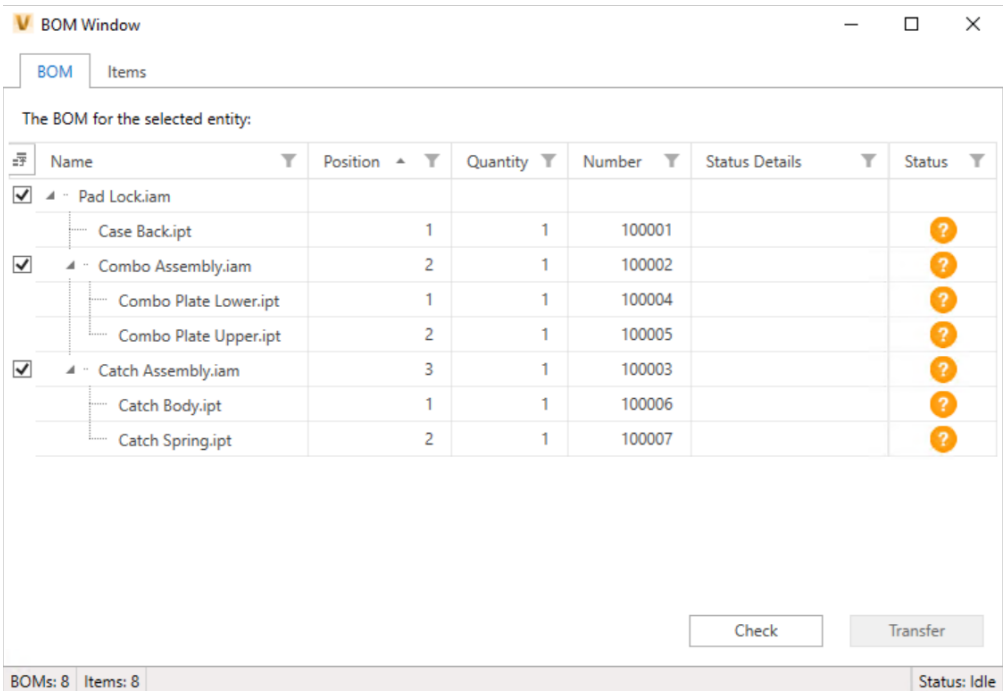
In order to archive newly created designs in the right project, it is necessary to access this information from the ERP system.
As with material information, powerGate can provide the correct project number.

9.3 Transfer BOM to the ERP system

As soon as the design takes shape, a bill of materials (BOM) will be derived from the CAD data, which must later be transferred to the ERP system.
The designer can determine whether a BOM already exists in the ERP system by using the *ERP BOM tabs* in the Vault Client:



Subsequently powerGate also enables the engineer to complete the CAD BOM within Vault and transfer it manually or automatically back to the ERP system, using the *BOM Window*:


BOM Window
BOM
Items
The BOM for the selected entity:
Name
Position
Quantity
Number
Status Details
Status
Pad Lock.iam
Case Back.ipt 1 1 100001
Combo Assembly.iam 2 1 100002
Combo Plate Lower.ipt 1 1 100004
Combo Plate Upper.ipt 2 1 100005
Catch Assembly.iam 3 1 100003
Catch Body.ipt 1 1 100006
Catch Spring.ipt 2 1 100007
Check
Transfer
BOMs: 8
Items: 8
Status: Idle