powerLoad (bcpToolkit)

coolOrange s.r.l.

May 16, 2024

POWERLOAD (BCPTOOLKIT)

CHAPTER

ONE

INSTALLATION

1.1 Requirements

Operating System: 64-bit only

- Microsoft Windows 7 SP1
- Microsoft Windows 8.1
- Microsoft Windows 10

.NET Framework: 4.7 or higher

Windows PowerShell: PowerShell 4.0 or higher

1.2 Setup

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

1.3 Install locations

powerLoad (bcpToolkit) tools are installed in the following locations:

- All program libraries and executable files are placed in *C:\Program Files\coolOrange\bcpToolkit*
- All PowerShell libraries are placed in *C:\Program Files\coolOrange\Modules\bcpToolkit*

Following shared libraries are installed in *GAC*:

- bcpDevKit.dll
- coolOrange.Logging.dll

Following shortcuts are added in the start menu:

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

1.4 Updates

To install a newer version of powerLoad (bcpToolkit) execute the setup file of the new version. This will automatically update the files in the existing installation.

Tip: When upgrading from bcpChecker versions earlier than *17.0* (e.g. bcpChecker 2015 or 2016) we recommend *uninstalling* the old bcpChecker version *before upgrading*

1.5 Uninstall

In case you want to remove powerLoad (bcpToolkit) from your computer you can:

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

CHAPTER

TWO

ACTIVATION AND TRIAL LIMITATIONS

2.1 Trial limitations

2.1.1 bcpViewer

Only the *first 50 elements* will display full informations (names, properties, references). The Folder structure is shown completely.
No access to additional Checks e.g. File existence check, etc.

2.1.2 .NET library

In trial mode, BCP-packages can be created and all its *metadata* can be fully imported into Vault.
Please note that since *no real files* will be imported into Vault, opening them will fail!

2.1.3 Cmdlets

In trial mode, BCP-packages can be opened and manipulated without limitation.
However the *export* of a package will contain only the *metadata*, independent of a specified *-NoSourceFiles* parameter.

```
Export-BcpPackage -To ...

# in trial mode same result as:
Export-BcpPackage -To ... -NoSourceFiles
```

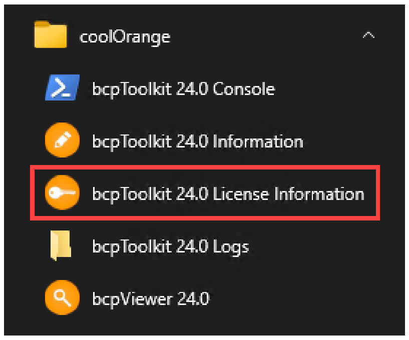
Keep in mind that since links to your *real files get removed* in the exported package, opening the files later in Vault will fail!

2.2 Activation

In order to activate the product you have following options:

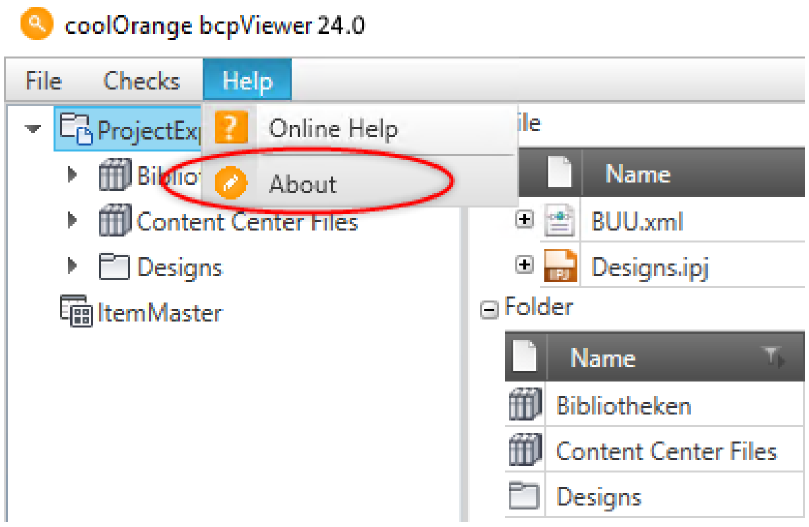
2.2.1 License Shortcut

Open the Start Menu and use the '*bcpToolkit 24.0 License Information*' shortcut:



2.2.2 bcpViewer Menu

Launch the *bcpViewer 24.0* shortcut, open the 'Help' Menu and click the 'About' button:



2.2.3 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\bcpToolkit\License.exe" --StandAlone --Serialnumber="XXXXX-
↪XXXXX-XXXXX-XXXXX"
```

For more information about activating the product, see Licensing.

Activate powerLoad (bcpToolkit) on export machine

The environment where your project, that uses the bcpDevKit *.NET Library*, gets *executed* must be activated in order to create productive BCP packages.

The same applies when automating operations via PowerShell scripts with the bcpToolkit *Cmdlets*.
However build- and test-machines do not need to be activated.

2.3 Licensing Options

2.3.1 Stand Alone Licensing

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

License expired

When the the license expires, the bcpViewer will switch back to Trial mode, the .NET library will throw an exception and the Cmdlets also switch back to Trial mode.

Offline activation

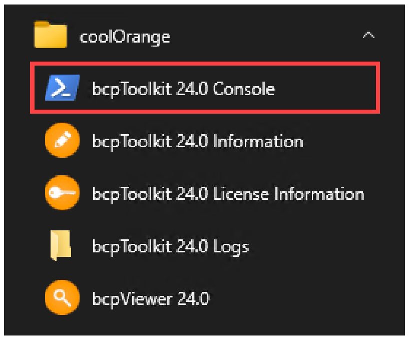
The serial number of the license and the machine code are required to generate an activation file.
The activation file for an offline activation can be generated and downloaded on the following site: bcpToolkit - Activation file generator

CHAPTER

THREE

GETTING STARTED

3.1 Using the Cmdlets



3.1.1 Start the PowerShell environment

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

3.1.2 Open the package

Before you are able to work with your BCP package you have to open it in your PowerShell environment.
You can use our `VaultBcp 2024` if you need a sample package.

Open the package by calling *Open-BcpPackage*. For now we ignore all the bomBlob*.xml files in the package.

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage' -IgnoreBomBlobs
```

3.1.3 Export the package

After the BCP package is loaded, we can export it to a new directory without links to the real files.
This allows importing the package on test environments, without the need for the source files at all.

Execute *Export-BcpPackage* with the arguments To and NoSourceFiles.

```
Export-BcpPackage -To 'C:\Temp\bcp_samplepackage_no_files\' -NoSourceFiles
```

3.1.4 Close the package

When we are done with the export, we can close the previously opened BCP package.

Running the cmdlet *Close-BcpPackage* will release all the memory resources.

```
Close-BcpPackage
```

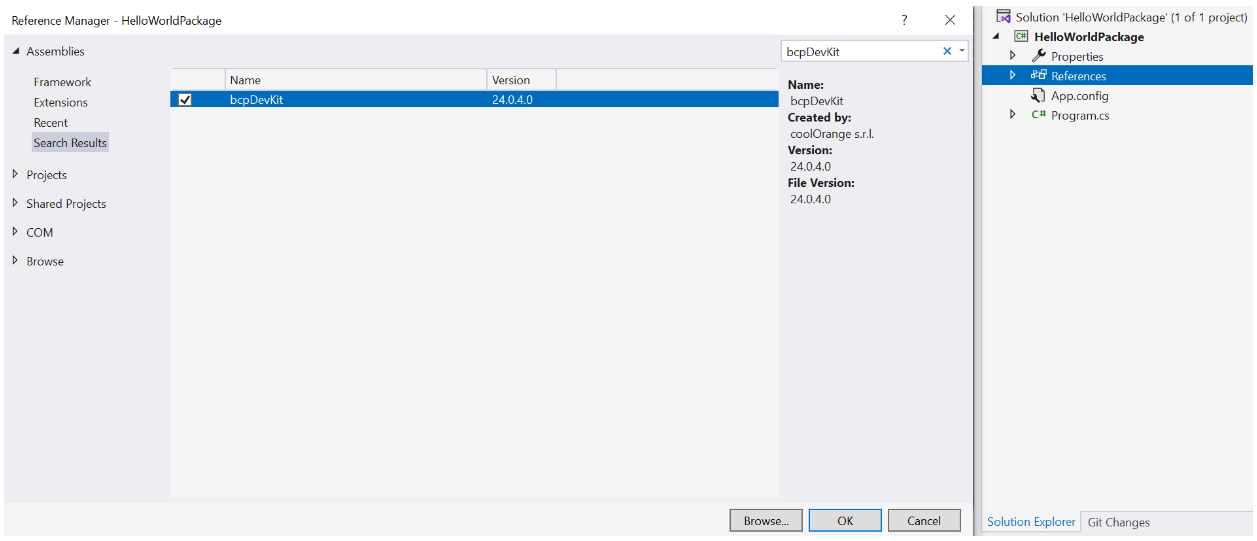
3.2 Using the .NET library

To use the .NET library you first need to *install* powerLoad (bcpToolkit) on your development machine.
The library contains all the *API's* to create your own BCP-package.

It requires your project targeting at least **.NET framework 4.7** !

1. Reference the bcpDevKit assembly

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



	Name	Version
✓	bcpDevKit	24.0.4.0

Name:
bcpDevKit
Created by:
coolOrange s.r.l.
Version:
24.0.4.0
File Version:
24.0.4.0

The assembly will be referenced from the **GAC**, therefore set *"Copy Local"* to *"false"*.

2. Create the BcpServiceBuilder and set the PackageLocation

In order to gain access to the bcpDevKit API's, following **namespaces** should be imported:

```
using bcpDevKit;
using bcpDevKit.Entities;
```

Root entry point is the class *BcpServiceBuilder*:

```
var bcpSvcBuilder = new BcpServiceBuilder();
```

First we set the target BCP *Version* for which the package should be created:

```
bcpSvcBuilder.Version = BcpVersion._2024;
```

Then we set the *PackageLocation*, where the Package should be exported to:

```
bcpSvcBuilder.SetPackageLocation("C:\\Temp\\HelloWorldPackage");
```

3. Build the BcpService and start working

After preparing the BcpServiceBuilder with the desired settings we can now *build* the BcpService:

```
var bcpService = bcpServiceBuilder.Build();
```

The *BcpService* gives us access to all the functionality for creating a importable BCP package.
We can start to add Files and Items to our package:

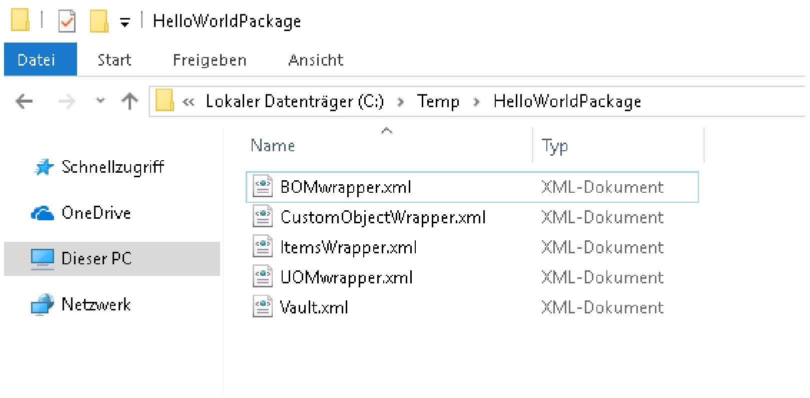
```
var file = bcpService.FileService.AddFileWithIteration("$/HelloWorldFiles/Hello.iam", @
↪"C:\HelloWorldFiles\Hello.iam");
var item = bcpService.ItemService.AddItem("999", "World", "Title 999", "Desc 999");
```

4. Create the BCP package

Finally we can create and export our BCP package to our package location by calling *Flush* on the BcpService:

```
bcpService.Flush();
```

After calling the function you can find the created package in your packageDirectory:



Name	Typ
BOMwrapper.xml	XML-Dokument
CustomObjectWrapper.xml	XML-Dokument
ItemsWrapper.xml	XML-Dokument
UOMwrapper.xml	XML-Dokument
Vault.xml	XML-Dokument

The package can now be used to import the data into Vault.

Install powerLoad (bcpToolkit) on customer machine

When shipping the binaries of your project to the customer, **also the customers machine requires a powerLoad (bcpToolkit) installation**.
Therefore delivering the *bcpDevKit* assembly within your project should be avoided, so that new bcpDevKit versions can continue to be easily *updated* at the customer's site.

See the **complete example**:

```csharp
using System;
using bcpDevKit;
using bcpDevKit.Entities;

namespace HelloWorldPackage
{
        class Program
        {
                static void Main(string[] args)
                {
                        var bcpSvcBuilder = new BcpServiceBuilder();
                        bcpSvcBuilder.Version = BcpVersion._2024;
                        bcpSvcBuilder.SetPackageLocation("C:\\Temp\\HelloWorldPackage");

                        var bcpService = bcpSvcBuilder.Build();

                        var file = bcpService.FileService.AddFileWithIteration("$/HelloWorldFiles/Hello.iam", @"C:\HelloWorldFiles\Hello.iam");
                        var item = bcpService.ItemService.AddItem("999", "World", "Title 999", "Desc 999");

                        bcpService.Flush();
                }
        }
}
```

3.3 Using the bcpViewer

3.3.1 Acceptance test

The first thing you should do for a acceptance test is to define how exactly the final vault should look like. E.g.

- Folder/Project structure
- What entities get which lifecycle/category
- How should the revision schemes be?
- What information should be in the revision table?
- Which udps are needed and what will be written into them?
- Are customizations needed?

There is much more, but the above things are the core points. In fact, you should do this before you do anything else. After you have defined how the result should be you have to look in your source system for example objects.

For a successful evaluation you will need one object of every kind. How many there will be depends on the Vault you have defined. E.g. If you have two categories for IDWs these are two objects not just one. Some of the objects could be:

- IDW with category A
- IDW with category B
- IPT

- Derived part
- Content center file
- Folder with category A
- Folder with category B
- Project
- …

It would be best to write down what you need before you proceed.

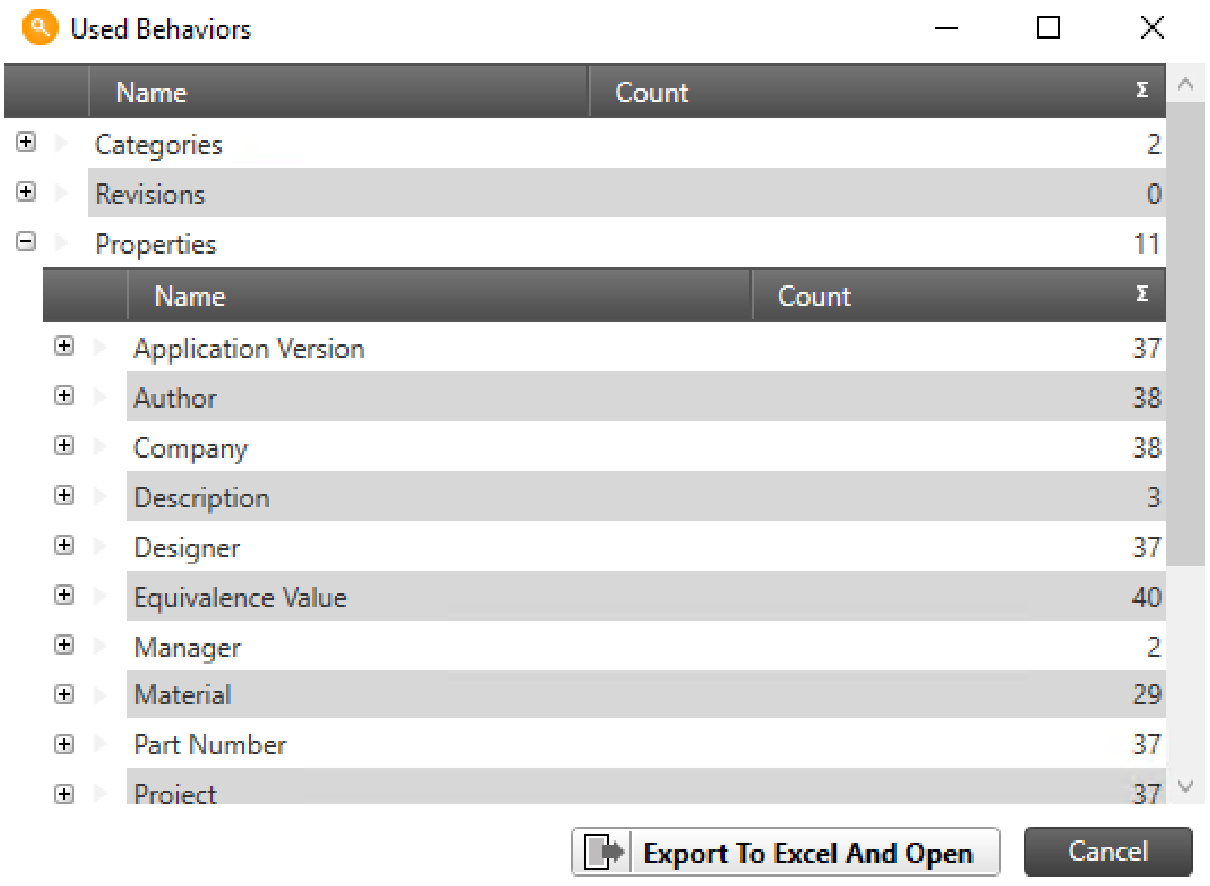
Now that you know what you need and how it should look like you can search for the sample files in your source system. Write down one example for every element you have defined above.

Now you can use bcpViewer to verify the path of every file, the category and its properties. If you are happy with the result you can import the package into your Vault with VaultBCP.

3.3.2 Calculate behaviors

The "Used Behaviors" dialog gives you a report about the used behaviors in your BCP package e.g property definitions, categories, revisions, lifecycle and users.
This gives you the opportunity to check against your Vault whether you really have defined all the properties, lifecycles, categories, etc. that are required for a correct import of the BCP package.

To open the "Used Behaviors" dialog, click on "Calculate Behaviors" in the menu.



Name	Count
Categories	2
Revisions	0
Properties	11

Name	Count
Application Version	37
Author	38
Company	38
Description	3
Designer	37
Equivalence Value	40
Manager	2
Material	29
Part Number	37
Project	37

Note: The report can be exported as excel document.

3.3.3 Files existence check

With the file existence check you can find all entries in your BCP Package which are missing and so avoid annoying errors during the import.

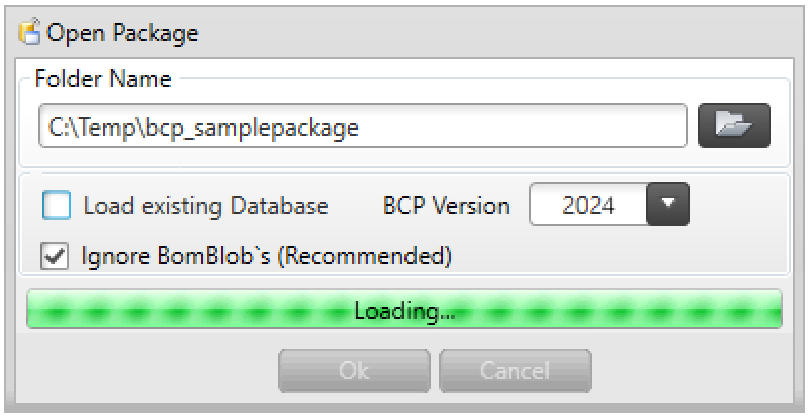
To start the file existence check, open the Checks menu and click on the Files Exist button.

With bcpViewer you can preview your BCP package before it is imported into Vault, in order to prevent errors that will be found after hours of processing.

3.3.4 Open a package

Before you can start you need a BCP package.
If you don't have one, you can use our `VaultBcp 2024` sample package.

Launch the *bcpViewer 24.0* shortcut on your Desktop and click on "File" → "Open" → "Folder Icon" for choosing your BCP package:



Open Package

Folder Name: C:\Temp\bcp_samplepackage

☐ Load existing Database   BCP Version: 2024

☑ Ignore BomBlob`s (Recommended)

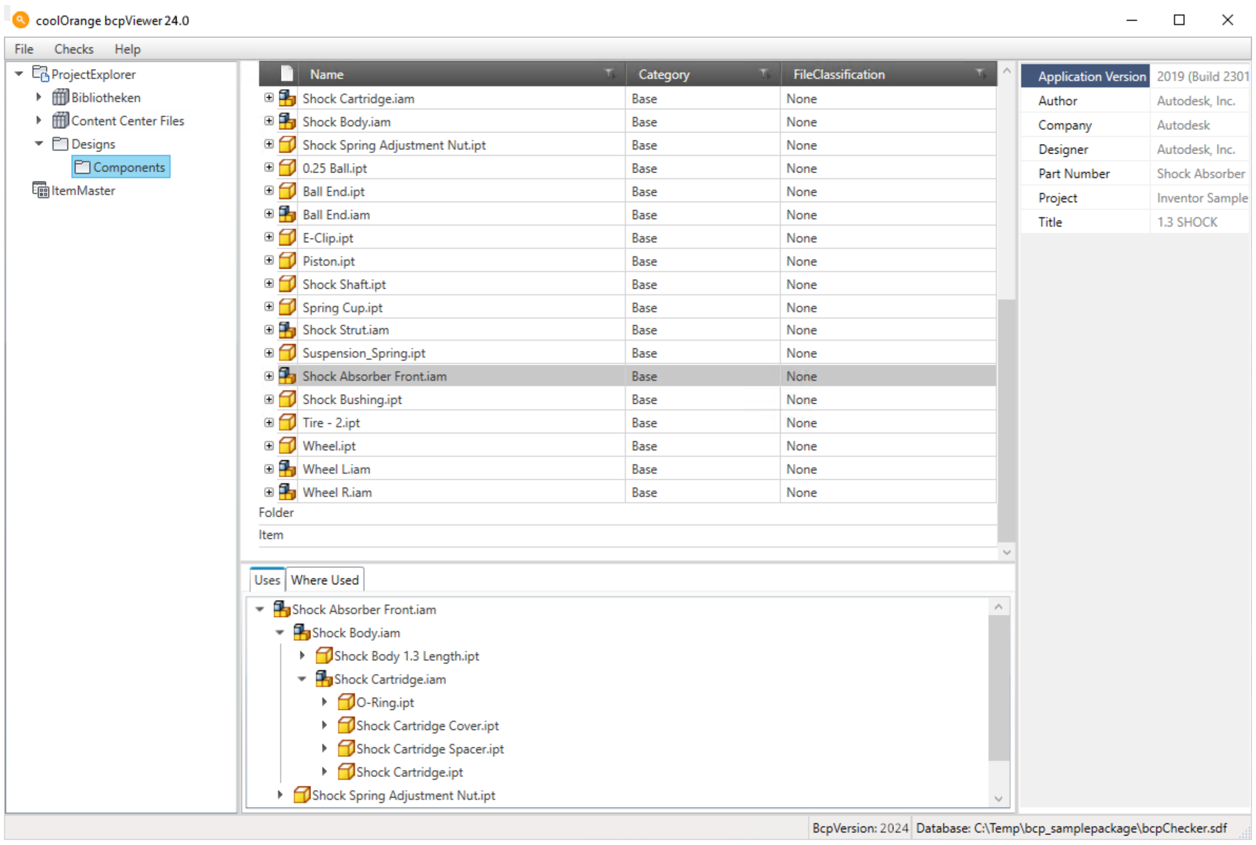
Loading...

Note: Choose the directory that contains the *Vault.xml* file, not a subfolder or the parent directory!

After opening the BCP package once, you can use the "Load existing Database" checkbox. For large packages, this option saves a lot of loading time.
As soon as your package is analyzed you can continue.

3.3.5 Check Files

By clicking on a *folder* the containing *files* are shown.
When selecting a file, you can see it's *dependencies* and *where* it is used.



coolOrange bcpViewer 24.0

File Checks Help

- ProjectExplorer
  - Bibliotheken
  - Content Center Files
  - Designs
    - Components
- ItemMaster

Name	Category	FileClassification
Shock Cartridge.iam	Base	None
Shock Body.iam	Base	None
Shock Spring Adjustment Nut.ipt	Base	None
0.25 Ball.ipt	Base	None
Ball End.ipt	Base	None
Ball End.iam	Base	None
E-Clip.ipt	Base	None
Piston.ipt	Base	None
Shock Shaft.ipt	Base	None
Spring Cup.ipt	Base	None
Shock Strut.iam	Base	None
Suspension_Spring.ipt	Base	None
Shock Absorber Front.iam	Base	None
Shock Bushing.ipt	Base	None
Tire - 2.ipt	Base	None
Wheel.ipt	Base	None
Wheel L.iam	Base	None
Wheel R.iam	Base	None

Folder
Item

Application Version	2019 (Build 2301
Author	Autodesk, Inc.
Company	Autodesk
Designer	Autodesk, Inc.
Part Number	Shock Absorber
Project	Inventor Sample
Title	1.3 SHOCK

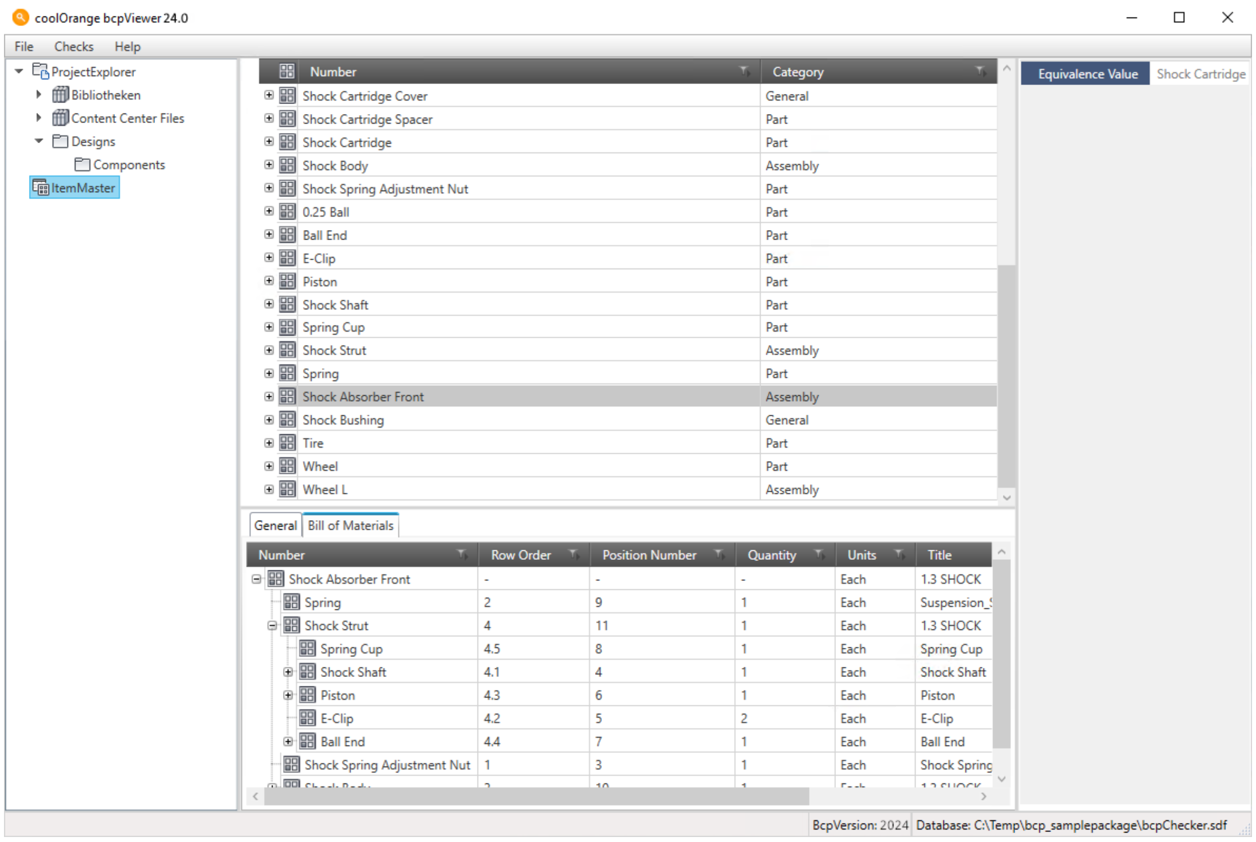
Uses | Where Used

- Shock Absorber Front.iam
  - Shock Body.iam
    - Shock Body 1.3 Length.ipt
    - Shock Cartridge.iam
      - O-Ring.ipt
      - Shock Cartridge Cover.ipt
      - Shock Cartridge Spacer.ipt
      - Shock Cartridge.ipt
  - Shock Spring Adjustment Nut.ipt

BcpVersion: 2024 Database: C:\Temp\bcp_samplepackage\bcpChecker.sdf

3.3.6 Check Items

By clicking on ItemMaster in the left panel all the *items* are shown.
After selecting an item, you get informations about the *associated files* and the *Bill of Material*.



coolOrange bcpViewer 24.0

File Checks Help

- ProjectExplorer
  - Bibliotheken
  - Content Center Files
  - Designs
    - Components
  - ItemMaster

Number	Category
Shock Cartridge Cover	General
Shock Cartridge Spacer	Part
Shock Cartridge	Part
Shock Body	Assembly
Shock Spring Adjustment Nut	Part
0.25 Ball	Part
Ball End	Part
E-Clip	Part
Piston	Part
Shock Shaft	Part
Spring Cup	Part
Shock Strut	Assembly
Spring	Part
Shock Absorber Front	Assembly
Shock Bushing	General
Tire	Part
Wheel	Part
Wheel L	Assembly

General | Bill of Materials

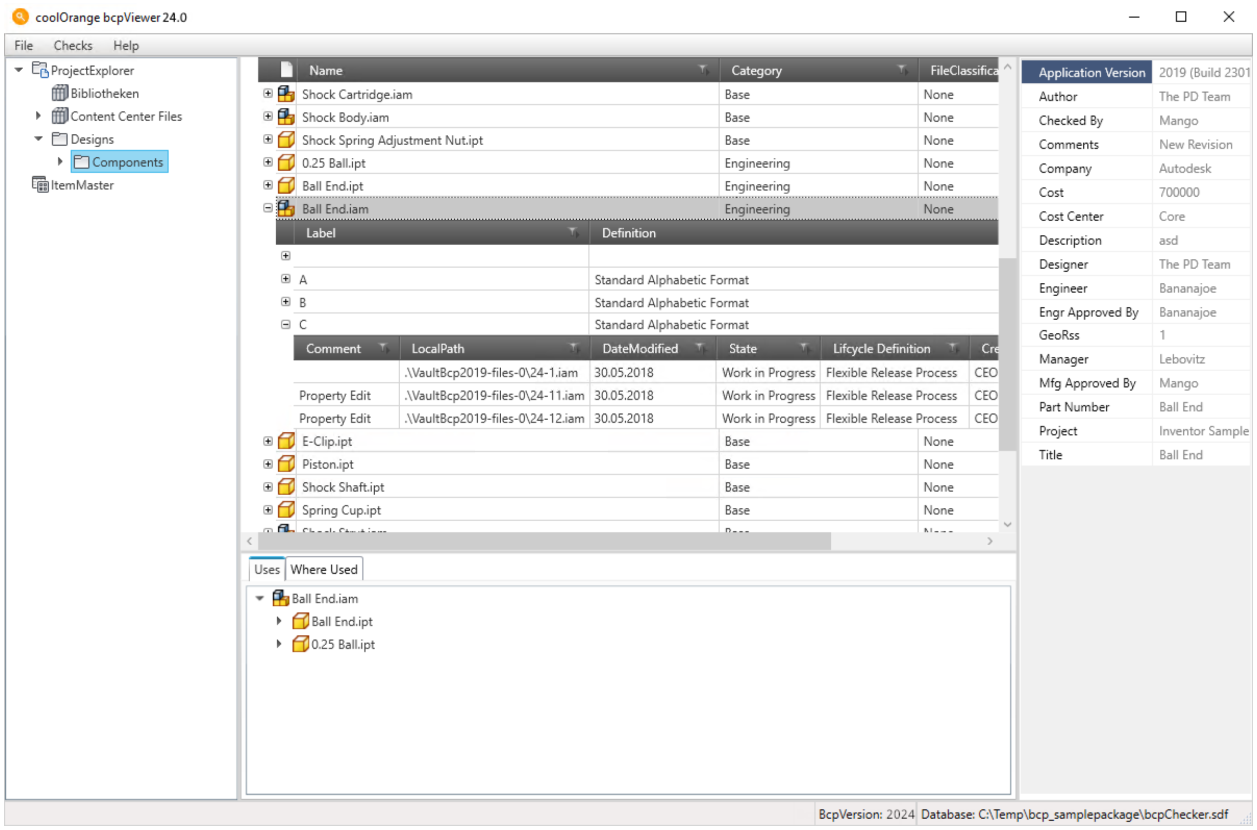
Number	Row Order	Position Number	Quantity	Units	Title
Shock Absorber Front	-	-	-	Each	1.3 SHOCK
Spring	2	9	1	Each	Suspension_S
Shock Strut	4	11	1	Each	1.3 SHOCK
Spring Cup	4.5	8	1	Each	Spring Cup
Shock Shaft	4.1	4	1	Each	Shock Shaft
Piston	4.3	6	1	Each	Piston
E-Clip	4.2	5	2	Each	E-Clip
Ball End	4.4	7	1	Each	Ball End
Shock Spring Adjustment Nut	1	3	1	Each	Shock Spring

Equivalence Value	Shock Cartridge

BcpVersion: 2024 Database: C:\Temp\bcp_samplepackage\bcpChecker.sdf

3.3.7 Check Behaviours

On files and items you can check their category, classification, lifecycle definition, state or other information.
You can view all their *revisions* and *versions* directly by expanding them.
On the right side of the Window you can see all the *user defined* properties of the selected element (system properties are not shown).



Name	Category	FileClassifica
Shock Cartridge.iam	Base	None
Shock Body.iam	Base	None
Shock Spring Adjustment Nut.ipt	Base	None
0.25 Ball.ipt	Engineering	None
Ball End.ipt	Engineering	None
Ball End.iam	Engineering	None
E-Clip.ipt	Base	None
Piston.ipt	Base	None
Shock Shaft.ipt	Base	None
Spring Cup.ipt	Base	None

Label	Definition
A	Standard Alphabetic Format
B	Standard Alphabetic Format
C	Standard Alphabetic Format

Comment	LocalPath	DateModified	State	Lifcycle Definition	Cre
	.\VaultBcp2019-files-0\24-1.iam	30.05.2018	Work in Progress	Flexible Release Process	CEO
Property Edit	.\VaultBcp2019-files-0\24-11.iam	30.05.2018	Work in Progress	Flexible Release Process	CEO
Property Edit	.\VaultBcp2019-files-0\24-12.iam	30.05.2018	Work in Progress	Flexible Release Process	CEO

Application Version	2019 (Build 2301
Author	The PD Team
Checked By	Mango
Comments	New Revision
Company	Autodesk
Cost	700000
Cost Center	Core
Description	asd
Designer	The PD Team
Engineer	Bananajoe
Engr Approved By	Bananajoe
GeoRss	1
Manager	Lebovitz
Mfg Approved By	Mango
Part Number	Ball End
Project	Inventor Sample
Title	Ball End

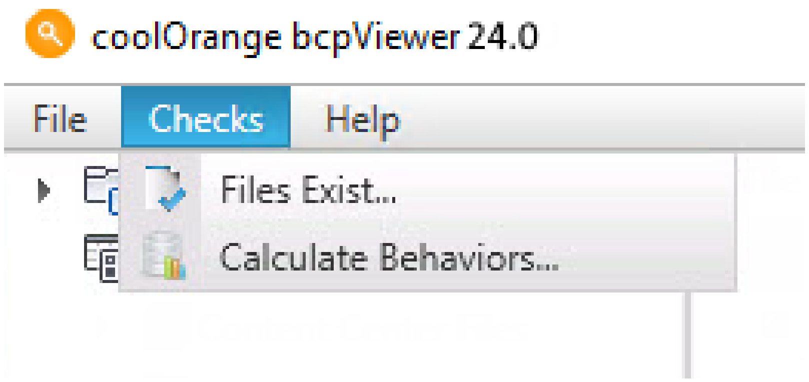
- Ball End.iam
  - Ball End.ipt
  - 0.25 Ball.ipt

BcpVersion: 2024 Database: C:\Temp\bcp_samplepackage\bcpChecker.sdf

3.3.8 Good to know

The best way to complete a migration to Vault is with an *acceptance test*.
With bcpViewer you can check the success of this test before the actual import.

Additional checks like the *Files existence check* and the *Behaviours calculation* can help you on this.
They can be accessed via Toolbar → "Checks":



Restrictions: Depending on your machine, large bcp packages (> 1 GB) can take up to 5 minutes and more to load

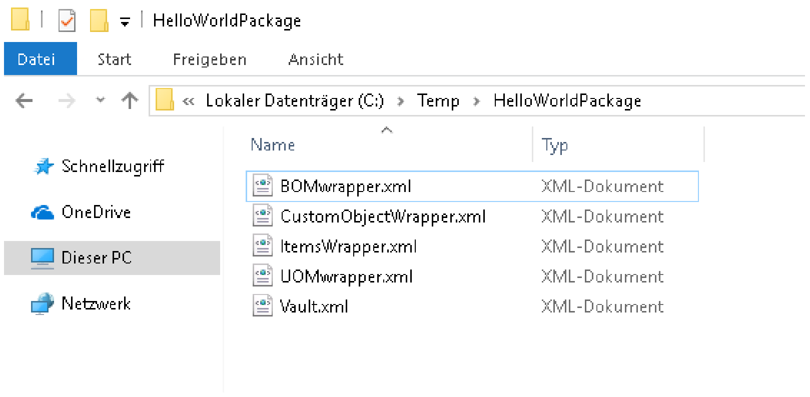
3.4 Creating a sample package

Utilities like the Autodesk Data Export Utility or Autodesk Data Transfer Utility are able to create BCP packages out of the box.
If you need a way to create your own BCP package, you could use the bcpDevKit *.NET Library* in your own Visual Studio project.

Follow *these steps* to use the *.NET library* in your C# project, and get ready to implement following example:

```csharp
using System;
using bcpDevKit;
using bcpDevKit.Entities;

namespace HelloWorldPackage
{
        class Program
        {
                static void Main(string[] args)
                {
                        var bcpSvcBuilder = new BcpServiceBuilder();
                        bcpSvcBuilder.Version = BcpVersion._2024;
                        bcpSvcBuilder.SetPackageLocation("C:\\Temp\\HelloWorldPackage");

                        var bcpService = bcpSvcBuilder.Build();

                        var file = bcpService.FileService.AddFileWithIteration("$/HelloWorldFiles/Hello.iam", @"C:\HelloWorldFiles\Hello.iam");
                        var item = bcpService.ItemService.AddItem("999", "World", "Title 999", "Desc 999");

                        bcpService.Flush();
                }
        }
}
```

Make sure following directory exists on disk *'C:\Temp\HelloWorldPackage'*.
When running your new project several xml files should be created inside there.

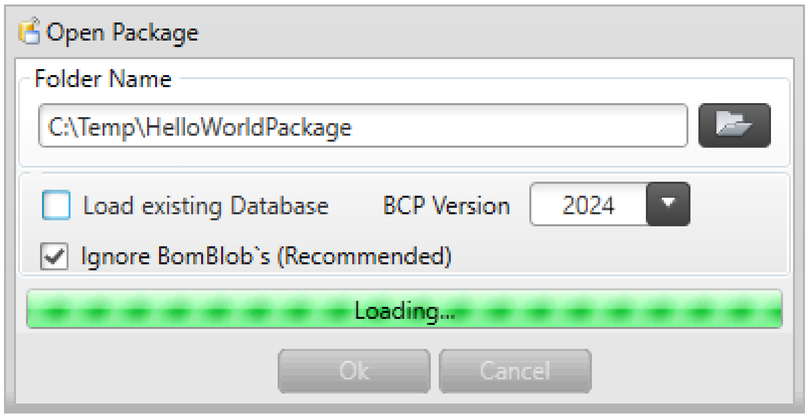


Name	Typ
BOMwrapper.xml	XML-Dokument
CustomObjectWrapper.xml	XML-Dokument
ItemsWrapper.xml	XML-Dokument
UOMwrapper.xml	XML-Dokument
Vault.xml	XML-Dokument

This newly created BCP package would now be ready for an import into Vault.

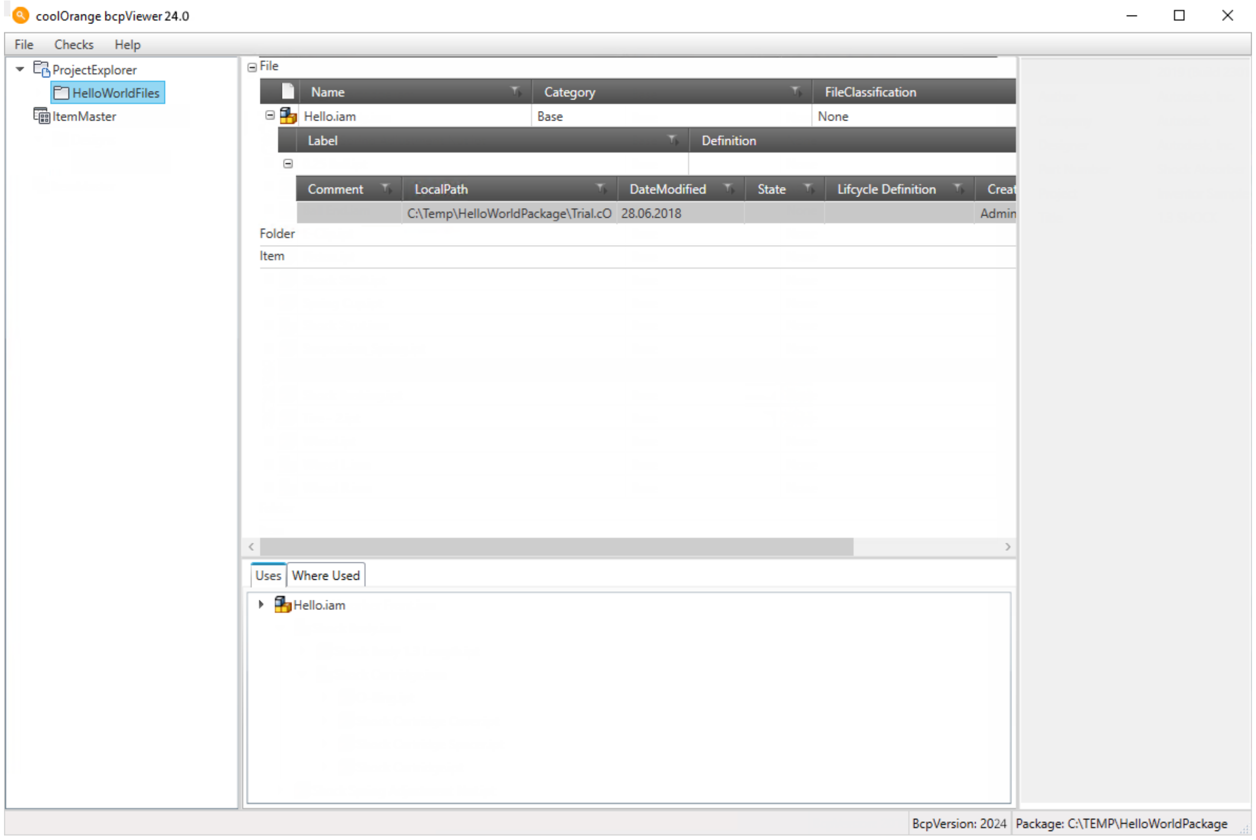
3.5 Preview the package

In order to display your BCP package in a Vault-like UI, you can use *bcpViewer*.
Launch the *bcpViewer 24.0* shortcut on your Desktop and open the package *'C:\Temp\HelloWorldPackage'* as described *here*.



Folder Name: C:\Temp\HelloWorldPackage
Load existing Database: ☐
BCP Version: 2024
Ignore BomBlob`s (Recommended): ☑

You can navigate to the folder '$/HelloWorldFiles' and see the contained file 'Hello.iam', as well as the item within the ItemMaster.



Name	Category	FileClassification
Hello.iam	Base	None

Comment	LocalPath	DateModified	State	Lifcycle Definition	Creat
	C:\Temp\HelloWorldPackage\Trial.cO	28.06.2018			Admin

BcpVersion: 2024 Package: C:\TEMP\HelloWorldPackage

3.6 Exporting a test package

Since the import of large packages takes some time, we can create a package that does not contain the real files.
The bcpToolkit *commandlets* can be used to create a new test-package out of our previous BCP package with only the metadata information.
Start a new PowerShell environment and import the `bcpToolkit` module by clicking the *bcpToolkit Console* shortcut in the start menu as described *here*.
Use *Open-BcpPackage* to open the original BCP package in your PowerShell environment.

```
Open-BcpPackage -Path 'C:\Temp\HelloWorldPackage'
```

The opening of the BCP package should be very fast, because the package got already opened in *bcpViewer* before.
Next we can directly export the package without links to the real files to a new package directory.

Execute *Export-BcpPackage* with the arguments To and NoSourceFiles.

```
Export-BcpPackage -To 'C:\Temp\HelloWorldPackage_Test' -NoSourceFiles
```

The new test package in the directory *'C:\Temp\HelloWorldPackage_Test'* can now be copied and imported on any Vault test environment without the need for the source files.

CHAPTER

FOUR

CODE REFERENCE

The Code Reference section describes:

4.1 Cmdlets

4.1.1 Close-BcpPackage

Closes an open BCP package.

Syntax

```
Close-BcpPackage [[-Path] <DirectoryInfo>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Default value	Optional
DirectoryInfo	Path	Directory of the opened BCP package that should get closed	The last opened BCP package	yes

Return type

Bool: on success the cmdlet returns $true otherwise $false.
$result.Error ← On failure with an additional property 'Error' containing the Exception
$result.Location ← The directory where the closed BCP package is located
$result.DatabaseLocation ← The location of the internal Database-file of the closed BCP package

Remarks

The cmdlet closes the previously *opened* BCP package within the current AppDomain.
When multiple BCP packages are opened simultaneously the **Path** to the desired BCP package can be specified.

After closing a BCP package ongoing operations will not use the closed package any more.
Memory resourses and locks to the internal Database-file become automatically released.

Examples

In the following examples we are using our `VaultBcp 2024` sample package for demonstration purposes:
Closing the previously opened BCP package

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage' -IgnoreBomBlobs
#...
Close-BcpPackage
```

Closing one of several opened BCP packages

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage' -IgnoreBomBlobs
Open-BcpPackage -Path '.\bcp_customerpackage'
Close-BcpPackage -Path 'C:\Temp\bcp_samplepackage'
```

Validating if the BCP package got closed correctly

```
$closePackageResult = Close-BcpPackage
if(-not $closePackageResult ) {
        throw "Failed with error: " + $closePackageResult.Error.Message
}
```

Closing a BCP package and using the close result

```
$openPackageResult = Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage' -IgnoreBomBlobs

$closePackageResult = $openPackageResult.Location | Close-BcpPackage

write-host "Closed package '$($closePackageResult.Location.Fullname)'..."
write-host "Version: '$([int]$closePackageResult.Version)'"
write-host "Internal database: '$($closePackageResult.DatabaseLocation.Name)'"
```

4.1.2 Export-BcpPackage

Exports the opened BCP package to the specified directory.

Syntax

```
Export-BcpPackage [[-Path] <DirectoryInfo>] [[-To] <DirectoryInfo>] [-NoSourceFiles] [
↪<CommonParameters>]
```

Parameters

Type	Name	Description	Default value	Optional
DirectoryInfo	Path	Directory of the opened BCP package that should get exported	The last opened BCP package	yes
DirectoryInfo	To	Destination directory that will containing the VaultBCP xml files	The directory *'.\Export'* or the current directory '.' when the package got opened by reusing an existing internal Database	yes
SwitchParameter	NoSourceFiles	The export package will be created without links to the real files	False	yes

Return type

Bool: on success the cmdlet returns $true otherwise $false.
$result.Error ← On failure with an additional property 'Error' containing the Exception
$result.Location ← The directory where the exported BCP package is located
$result.DatabaseLocation ← The location to the internal Database-file of the exported BCP package (only specified when the exported BCP package is opened)

Remarks

The cmdlet exports the previously *opened* BCP package to a destination directory.
When multiple BCP packages are opened simultaneously the **Path** to the desired BCP package can be specified.

The internal Database becomes exported to the destination directory requested in the **To** parameter.
When the specified directory does not exist it will be automatically generated.

For testing purpose it can be useful to export the package with the **NoSourceFiles** parameter, in order to get a package that can be imported into Vault without the need for the *real files*.

Examples

In the following examples we are using our `VaultBcp 2024` sample package for demonstration purposes:
Exporting the previously opened BCP package

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage'
#...
Export-BcpPackage -To 'C:\Temp\bcp_samplepackage\Export_01'
```

Exporting several opened BCP packages

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage'
Open-BcpPackage -Path '.\bcp_customerpackage' -Version 2024 -IgnoreBomBlobs

Export-BcpPackage -Path 'C:\Temp\bcp_samplepackage' '.\bcp_customerpackage' | Export-
↪BcpPackage
```

Validating if the BCP package got exported correctly

```
$exportPackageResult = Export-BcpPackage -To 'P:\laplapla'
if(-not $exportPackageResult ) {
        throw "Failed with error: " + $exportPackageResult.Error.Message
}
```

Exporting a BCP package to a relative path and using the export result

```
$exportPackageResult = 'C:\Temp\bcp_samplepackage' | Export-BcpPackage -To '.\Export_02'

write-host "Exported package to '$($exportPackageResult.Location.Fullname)'..."
write-host "Version: '$([int]$exportPackageResult.Version)'"
```

4.1.3 Open-BcpPackage

Opens the specified directory containing the VaultBCP xml files, for further operations.

Syntax

```
Open-BcpPackage [-Path] <DirectoryInfo> [[-Version] <BCPVersion>] [-IgnoreBomBlobs] [-
↪Force] [<CommonParameters>]
```

Parameters

Type	Name	Description	Default value	Optional
DirectoryInfo	Path	Directory containing the VaultBCP xml files		no
BcpVersion	Version	The Vault version of the BCP package	*BcpVersion._2024*	yes
SwitchParameter	IgnoreBomBlobs	All the bomBlob*.xml files get ignored	False	yes
SwitchParameter	Force	Forces the reopening of a previously opened BCP package	False	yes

Return type

Bool: on success the cmdlet returns $true otherwise $false.
$result.Error ← On failure with an additional property 'Error' containing the Exception
$result.Location ← The directory in which the opened BCP package is located
$result.DatabaseLocation ← The location of the internal Database-file containing the whole BCP-package data

Remarks

The cmdlet opens the specified BCP package in the current AppDomain.
In order to allow ongoing operations to work correctly, it is important to specify the correct **Version** of the BCP package.

When opening a BCP package for the *first time*, the whole package get's imported into an internal Database.
By doing this, the cmdlet reuses the existing Database file after BCP packages got opened once, for improving performance with large packages.

In order to bypass this optimization, the BCP package can be reopened by using the **Force** parameter.
Afterwards changes made to the original xml files will be reloaded into the internal Database.

The cmdlet supports opening *multiple packages* one after the other.
When the specified BCP package is already open, the opened package will be reused, as long as the **Force** parameter is not specified.

Note: In order to improve performance the **IgnoreBomBlobs** parameter can be used to skip the loading of all the *bomBlob._*.xml* files.
That means ongoing operations like e.g. *Export-BcpPackage* will ignore those files too, which could have an impact on the ongoing import into Vault!

Examples

In the following examples we are using our `VaultBcp 2024` sample package for demonstration purposes:
Opening a BCP package

```
Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage'
```

Opening a BCP package by settings its Version and by ignoring all the BomBlobs

```
'C:\Temp\bcp_samplepackage' | Open-BcpPackage -Version 2024 -IgnoreBomBlobs
```

Forcing the reopening of a BCP package and validating if it got opened correctly

```
$openPackageResult = Open-BcpPackage -Path 'C:\Temp\bcp_samplepackage' -Force
if(-not $openPackageResult ) {
        throw "Failed with error: " + $openPackageResult.Error.Message
}
```

Opening a BCP package by relative path and using the opening result

```
$openPackageResult = Open-BcpPackage -Path '.\Temp\bcp_samplepackage'

write-host "Opened package '$($openPackageResult.Location.Fullname)'..."
write-host "Version: '$([int]$openPackageResult.Version)'"
write-host "Internal database: '$($openPackageResult.DatabaseLocation.Name)'"
```

All Cmdlets are supporting the Vault BCP Versions from 2021 to 2024.

Feedback for long running operations

Long running cmdlets are able to provide the user with feedback about the current progress.
Depending on the used PowerShell Host a **progress bar** gets displayed in GUI applications, showing the status of the running command.

Name	Description
Open-BcpPackage	Opens the specified BCP package.
Export-BcpPackage	Exports the opened BCP package to the specified directory.
Close-BcpPackage	Closes an open BCP package.

4.2 .NET Library

4.2.1 BcpServiceBuilder Class

A factory that creates the *BcpService* with the relevant settings.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Inheritance Hierarchy

System.Object
bcpDevKit.IBcpServiceBuilder
bcpDevKit.BcpServiceBuilder

Syntax

```
public class BcpServiceBuilder : IBcpServiceBuilder
```

Properties

Type	Name	Description
BcpVersion	Version	Gets or sets the vault version for which you want to create a BCP package. The default version is *BcpVersion._2024*.
DirectoryInfo	PackageDirectory	Gets or sets the output folder where it will create the BCP package.

Methods

Type	Name	Description
void	SetPackageLocation(string packageLocation)	Assigns the specified path to the PackageDirectory.
IBcpService	Build()	Creates a new *IBcpService* with the relevant settings.

Remarks

The *PackageDirectory* is set by default to the *TEMP* directory on your computer. More details on its determination can be found here.
SetPackageLocation throws an exception of type *DirectoryNotFoundException* when an invalid path is passed to the function.

Examples

Create a new BCP package and add files

```
using bcpDevKit; using bcpDevKit.Entities;

class Program
{
        static void Main(string[] args)
        {
                var bcpSvcBuilder = new BcpServiceBuilder {Version = BcpVersion._2024};
                bcpSvcBuilder.SetPackageLocation(@"C:\Temp\Package1");
                var bcpSvc = bcpSvcBuilder.Build();

                var catchAssembly = bcpSvc.FileService.AddFileWithIteration("$/Designs/
↪Catch Assembly.iam", @"C:\Catch Assembly.iam");
                bcpSvc.Flush();
        }
}
```

4.2.2 BcpVersion Enumeration

Specifies the vault version of the BCP package.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public enum BcpVersion
```

Members

Member name	Description
_2024	Vault 2024 Server
_2023	Vault 2023 Server
_2022	Vault 2022 Server
_2021	Vault 2021 Server

4.2.3 EntitiesTable Class

FileIteration Class

Namespace: bcpDevKit.Entities.Vault
Assembly: bcpDevKit.dll

Syntax

```
public class FileIteration
```

Properties

Type	Name	Description
FileRevision	Revision	Parent Revision
string	Uid	
string	Comment	
string	Modified	Use SetModifiedDate(DateTime dt) to set this value.
string	LocalPath	
string	Id	
string	ContentSource	
CreatedObject	Created	
StateObject	State	Use Setstate(string definition, string name) to set state.
List<UDPObject>	UDp	
List<AssociationObject>	Associations	

Methods

Type	Name	Description
void	SetModifiedDate(DateTime dt)	Sets correctly formatted modified date.
void	Setstate(string definition, string name)	
UDPObject	AddProperty(string name, string value)	
Associa-tionObject	AddAssociation(*FileIteration* childFileVersion, AssocType type)	
Associa-tionObject	AddAssociation(*FileIteration* childFileVersion, AssocType type, bool bForeignRef, int refId)	
List<*FileIterati*	GetParentDesignDocuments()	
FileObject	GetFile()	

FileIterationRef Class

Namespace: bcpDevKit.Entities.Vault
Assembly: bcpDevKit.dll

Syntax

```
public class FileIterationRef
```

Properties

Type	Name	Description
FileRevision	Revision	Parent Revision
string	Id	
long	Checksum	Vault file checksum
string	CreateDate	Vault file create date

Remarks

Both `Checksum` and `CreateDate` (date and time including milliseconds) must match a file in Vault.

FileObject Class

Namespace: bcpDevKit.Entities.Vault
Assembly: bcpDevKit.dll

Syntax

```
public class FileObject
```

Properties

Type	Name	Description
FolderObject	Folder	Parent Folder
string	Name	
string	Hidden	Use SetFileHidden(bool hidden) to set this value.
string	Classification	Use SetFileClassification(FileClassification fileClass) to set this value.
string	Category	
ACLObject	AClObjects	
List<*FileRevision*>	Revisions	
FileTypeEnum	FileType	Use SetFileType() to set this value automatically.
Nullable<ACLBehavior>	AclBehavior	
ACLBehavior	AclBehaviorSerialized	
FileRevision	LatestRevision	Gets latest revision.
FileIteration	LatestIteration	Get latest iteration of latest revision.
FileIterationRef	LatestIterationRef	Get latest iterationref of latest revision.

Methods

Type	Name	Description
void	SetFileHidden(bool hidden)	
void	SetFileClassification(FileClassification fileClass)	
void	SetAsPreview(bool preview)	When true, sets Hidden to true and Classification to FileClassification.DesignVisualization.
FileClassification	GetFileClassification()	
void	SetFileType()	Sets FileType Property depending on file ending.
FileRevision	AddRevisionWithIteration(string fileLocation, string label = "")	Creates and returns a new *FileRevision* with a *FileIteration*
FileRevision	AddRevisionWithIterationRef(long checksum, DateTime createDate, string label = "")	Creates and returns a new *FileRevision* with a *FileIterationRef*
ACLObject	AddAcl()	
FolderObject	GetFolder()	Get parent Folder.
void	OrderRevisions()	Orders revisions by their label.
bool	ShouldSerializeAclBehaviorSerialized()	Returns true if ACLBehaviour is not null.

FileRevision Class

Namespace: bcpDevKit.Entities.Vault
Assembly: bcpDevKit.dll

Syntax

```
public class FileRevision
```

Properties

Type	Name	Description
FileObject	File	Parent File
string	Definition	Use SetRevisionDefinition(string revDefinition, string revLabel) to set this value.
string	Label	Used for ordering.
List<*FileIteration*>	Iterations	
List<*FileIterationRef*>	IterationRefs	

Methods

Type	Name	Description
FileIteration	AddIteration(string fileLocation)	Adds and returns an iteration
FileIterationRef	AddIterationRef(long checksum, DateTime createDate)	Adds and returns an iterationref
RevDefObject	SetRevisionDefinition(string revDefinition, string revLabel)	

FolderObject Class

Namespace: bcpDevKit.Entities.Vault
Assembly: bcpDevKit.dll

Syntax

```
public class FolderObject
```

Properties

Type	Name	Description
FolderObject	ParentFolder	Gives direct access to the parent folder of the current folder.
string	StandardFolderCat-egory	Gets or sets the standard folder category.
string	Name	Gets or sets the name of the current folder.
string	Category	Gets or sets the category of the current folder.
string	Id	Gets or sets the id of the current folder.
string	IsLibraryStr	Gets or sets whether the current folder is a library or not. (boolean value as string)
CreatedObject	Created	Gets or sets information about the creation of the current folder like the user or the date.
StateObject	State	Gets or sets information about the state of the current folder.
List<UDPObject>	UDP	Gets a list of properties about the children folders of the current folder.
List<*FolderObjec*	FolderObjects	Gets a list of all children folders of the current folder.
List<*FileObject*>	FileObjects	Gets a list of all files in the current folder.
List<LinkObject>	Link	Gets a list of all links of the current folder.

Methods

Type	Name	Description
Folder-Object	AddFolder(string folderName)	Takes a folder name to create and add a new child folder to the current folder (if it is not already a child).
Folder-Object	AddFolder(*FolderObject* folder)	Takes a folder and add it to the children of the current folder, or do nothing if its already added.
Folder-Object	GetFolder(string folderName)	Takes a folder name and return a the specified child folder.
FileObject	AddFileWithIteration(string targetFileName, string sourceFileName)	Takes a target file name and a source file name, create a file and add it to the current folder. This file will already contain a default iteration.
FileObject	AddFileWithIterationRef(string targetFileName, long checksum, DateTime createDate)	Takes a target file name, a source file name and a creation date and add it to the current folder. This file will already contain a default iteration ref.
FileObject	GetFile(string fileName)	Takes a file name and return the specified file if it is located inside the current folder.
UD-PObject	AddProperty(string name, string value)	Takes a name and a value and add it to the properties of the current folder.
void	Setstate(string definition, string name)	Takes a definition and a name and sets the state of the current folder.
LinkObject	AddLink(string id)	Takes an id, create a link and adds it to the current folder.
void	SetLibrary(bool library)	Takes a boolean value and determine, whether the current folder is a library or not.
bool	IsLibrary()	Returns whether the current folder is a library.
FileObject	AddFile(FileObject file)	Takes a file and add it to the child files of the current folder.

GroupObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Inheritance Hierarchy

System.Object
bcpDevKit.Configuration.SecurityPrincipal
bcpDevKit.Entities.Configuration.GroupObject

Syntax

```
public class GroupObject : SecurityPrincipal
```

Properties

Type	Name	Description
string	Email	Gets or sets an email address for the group.

InGroupObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public class InGroupObject
```

Properties

Type	Name	Description
string	Name	Gets or sets the name of the assigned group.

Remarks

Name is the ‘Name’ of the linked *GroupObject*.

InRoleObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public class InRoleObject
```

Properties

Type	Name	Description
string	Name	Gets or sets the name of the assigned role.

Remarks

Name is the 'Name' of the linked *RoleObject*.

PermissionObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public class PermissionObject
```

Properties

Type	Name	Description
string	Name	Gets or sets the name of the permission.

RoleObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public class RoleObject
```

Properties

Type	Name	Description
string	Name	Gets or sets the name of the role.
string	Name	Gets or sets the role description.
IEnumerable<*Permission*>	Name	Gets all the roles permissions.

Methods

Type	Name	Description
Permission	Name	Creates and adds a new permission to the role if no one with the specified name exists.
Permission	Name	Retrieves the permission with the specified name if exists.

Remarks

Examples

Assign user to specific role and add permissions:

```
var security = bcpService.EntitiesTable.Vault.Security;
var user = security.AddUser("Kim");
user.AddRole("Document Manager (Level 1)");

var role = security.GetRole("Document Manager (Level 1)") role.AddPermission(
↪"FileChangeCategory");
role.AddPermission("FileChangeRevision");
```

SecurityObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public abstract class SecurityObject
```

Properties

Type	Name	Description
IEnumerable<*UserObject*>	Users	Gets all the configured users.
IEnumerable<*GroupObject*>	Groups	Gets all the configured groups.
IEnumerable<*RoleObject*>	Roles	Gets all the configured roles (only working for Vault 2019 and above).

Methods

Type	Name	Description
UserObject	AddUser(string userName)	Creates a new user if no one with the specified name exists.
UserObject	GetUser(string userName)	Retrieves the user with the specified name if exists.
GroupObject	AddGroup(string groupName)	Creates a new group if no one with the specified name exists.
GroupObject	GetGroup(string groupName)	Retrieves the group with the specified name if exists.
RoleObject	AddRole(string roleName)	Creates a new role if no one with the specified name exists.
RoleObject	GetRole(string roleName)	Retrieves the role with the specified name if exists.
void	RemoveRole(string roleName	Removes the role if one with the specified name exists.

Examples

Disable the export of all configuration entities

```
var security = bcpService.EntitiesTable.Vault.Security;
var user = security.AddUser("Angela");
```

SecurityPrincipal Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Syntax

```
public abstract class SecurityPrincipal
```

Properties

Type	Name	Description
string	Name	Gets or sets the entity name.
bool	Active	Gets or sets whether the user is active or inactive. Default is *true*.
SecurityPrincipal.VaultAccess	Access	Possible Values: - SecurityPrincipal.VaultAccess.NoVaults- SecurityPrincipal.VaultAccess.AllVaults (**Default**)- SecurityPrincipal.VaultAccess.ThisVault
SecurityPrincipal.VaultAuthentication	Authentication	Possible Values: - SecurityPrincipal.VaultAuthentication.Vault (**Default**) - SecurityPrincipal.VaultAuthentication.Windows
IEnumerable<InRoleObject<inrc	InRoleObject	Gets all the assigned roles.
IEnumerable<InGroupObject<inç	InGroupObject	Gets all the assigned groups.

Methods

Type	Name	Description
InRoleObject	AddRole(string roleName)	Creates and adds a new role if no one with the specified name exists.
InGroupObject	AddGroup(string groupName)	Creates and adds a new group if no one with the specified name exists.

UserObject Class

Namespace: bcpDevKit.Entities.Configuration
Assembly: bcpDevKit.dll

Inheritance Hierarchy

System.Object
bcpDevKit.Configuration.SecurityPrincipal
bcpDevKit.Entities.Configuration.UserObject

Syntax

```
public class UserObject : SecurityPrincipal
```

Properties

Type	Name	Description
string	FirstName	Gets or sets the FirstName.
string	LastName	Gets or sets the LastName.
string	Password	Gets or sets the Password.
string	Email	Gets or sets the Email address.

Examples

Assign user to group:

```
var security = bcpService.EntitiesTable.Vault.Security;
var user = security.AddUser("Trump");
var group = secutity.AddGroup("Presidents");

user.AddGroup(group.Name);
```

Grants direct access to the BCP entities.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public class EntitiesTable
```

Properties

Type	Name	Description
Vault	Vault	Provides access to default Vault entities.
Itemswrapper	Itemswrapper	Provides access to Item entities.
BOMwrapper	BOMwrapper	Provides access to BOM entities.
CustomObjectWrapper	CustomObjectWrapper	Provides access to CustomObject entities.
UOMwrapper	UOMwrapper	Provides access to UOM entities.

Remarks

For more details about how this classes are structered, please give a look to the *xsd files* coming with the vaultBCP installation.

Examples

Accessing CustomObjectWrapper via EntitiesTable:

```
var customObjects = bcpService.EntitiesTable.CustomObjectWrapper;
var definition = customObjects .AddCustomObjectDefinition("Dragonball", "Dragonballs");
definition.SetIcon(@".\Dragon-Ball-icon.ico");
```

4.2.4 ExportSettings Class

Provides settings for the creation of typical Vault settings that will be created in the package.

Namespace: bcpDevKit.Entities
Assembly: bcpDevKit.dll

Syntax

```
public class ExportSettings
```

Properties

Type	Name	Description
bool	CategoryDefinitionsExport	Gets or sets whether category definitions will be exported or not.
bool	PropertyDefinitionsExport	Gets or sets whether property definitions will be exported or not.
bool	LifecycleDefinitionsExport	Gets or sets whether lifecycle definitions and states will be exported or not.
bool	RevisionDefinitionsExport	Gets or sets whether revision definitions and revision sequences will be exported or not.
bool	UsersExport	Gets or sets whether users will be exported or not.
bool	GroupsExport	Gets or sets whether groups will be exported or not.
bool	RolesExport	Gets or sets whether roles will be exported or not.
BcpVersion	PackageVersion	Gets the vault version for which the BCP package is created.

Methods

Type	Name	Description
void	DisableConfigurationExport()	Disables the export for all the configuration entities.

Remarks

By default, all the configuration entities will be exported to the vault package.
In many situations the targeting Vault is already configured correctly, therefore the export of the different configuration entities can be disabled by setting the intended property to *false*.

If you set e.g. the *CreateUser* of a file to „Hans Peter“, the API will create automatically the User definition in the vault package, and the user will be later imported into Vault.
To disable this behaviour, set the property 'UsersExport' to *false*.

Examples

Disable the export of all configuration entities

```
bcpService.Settings.DisableConfigurationExport();
```

4.2.5 IBcpService Interface

Provides all the functionality to create a BCP package.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public interface IBcpService
```

Properties

Type	Name	Description
EntitiesTable	EntitiesTable	Gives direct access to the entities layer, for direct manipulation of the BCP entities.
FileService	FileService	Contains functionality to add, search and manipulate files and folders in your package.
ItemService	ItemService	Provides functionality to easily add, search and manipulate items and BOMs in your package and create links between Items and Files in the complex vault-based way.
CustomObjectService	CustomObjectService	Provides functionality to easily add, search and manipulate custom objects (e.g. Persons or Groups) and its definitions in your package.
ExportSettings	Settings	Here you can do export specific settings (e.g. if you want to not export Users or PropertyDefinitions).

Methods

Type	Name	Description
void	Flush()	This function will take all the data that you have created in your package and write it to the filesystem.

Remarks

IBcpService grants access to the service layer, which is built on top of the entities layer. The service layer contains functionality for creating a BCP package, without having to understand the entities layer in detail.

If you are more experienced and familiar with the BCP entities you can access the entities layer directly via the EntitiesTable and manipulate the objects to suite your needs.

After calling the **Flush()** method, you can look in the *PackageDirectory* where you should find all the XML files created from this function. It is also possible to flush multiple times, i.e. if you have a long running process and you want to flush in specific time intervals to not lose data if your process crashes.

4.2.6 ICustomObjectService Interface

Provides functionality to add, search and manipulate custom objects and their definition Syntax.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public interface ICustomObjectService
```

Methods

Type	Name	Description
CustomObject	AddCustomObject(string definitionName, string name)	Adds custom entities to the package and creates their definitions if required. definitionName is the singular name of the custom object (e.g. Person). Name should be unique.
CustomObject	GetCustomObject(string definitionName, string name)	Returns an entity of the specified definition if it exists.

Remarks

AddCustomObject handles the creation of the custom object definition and of the custom object itself. It handles the situation when the definition has to be created in the package or when an entity with the same name is already there.

Examples

Adds a CustomObject "Dragonball"

```
var definition = bcpService.EntitiesTable.CustomObjectWrapper.AddCustomObjectDefinition(
↪"Dragonball", "Dragonballs");
definition.SetIcon(@".\Dragon-Ball-icon.ico");
var customObject = bcpService.CustomObjectService.AddCustomObject("Dragonball", "4");
```

4.2.7 IFileService Interface

Contains functionality to add, search and manipulate files and folders in your package.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public interface IFileService
```

Methods

Type	Name	Description
RootObject	GetRootFolder()	Returns the Vault root folder.
FolderObject	AddFolder(string folderName)	Creates the passed folder structure. *folderName* is a valid vault folder path.
FileObject	AddFileWithIteration(string targetFileName, string sourceFileName, bool isLibrary=false)	Creates the passed folder structure and file. *targetFileName* is the path to the file destination, *sourceFileName* is the path to the sourceFile, if *isLibrary* is true, the file gets handled as content center file.
FileObject	AddFileWithIterationRef(string targetFileName, long checksum, DateTime createDate, bool isLibrary=false)	Creates folder structure and a file with an IterationRef referencing a file that already exists in the Vault. *targetFileName* is the path to the file destination, *checksum* is the checksum of the file in Vault, *createDate* is the creation date of the file in Vault **(requires milliseconds)**, if *isLibrary* is true, the file gets handled as content center file.
IEnumerable<*Fi*	SearchFilesByName(*FolderObject* rootFolderOfSearch, string fileName, bool searchRecursive=true)	Looks for files by their name in a specific folder and its subdirectories. *rootFolderOfSearch* is the first folder where the search is performed, *fileName* is the name the function searches for.
FileObject	SearchFileByLocation(string fileLocation)	This function searches for a specific file in the export package if *fileLocation* exists.
FolderObject	SearchFolderByPath(string folderPath)	Searches the export package for a specific folder if *folderPath* exists.

Remarks

The functions `AddFileWithIteration` and `AddFileWithIterationRef` can handle situations where a file or folder already exists. If a library file is added to a folder that is not marked as „Library“, the folder will not be marked as a library. `SetLibrary(true)` can by used to mark the folder as a library. Only if the folder has to be created automatically or is already a library, the file will be added as a content center.

Examples

Search Files by Name

```
var searchRecursive = true;
var foundFiles = bcpService.FileService.SearchFilesByName(GetMyFolder(), "Pad Lock.iam",␣
↪searchRecursive);
```

4.2.8 IItemService Interface

Contains functionality to deal with items and BOM.

Namespace: bcpDevKit
Assembly: bcpDevKit.dll

Syntax

```
public interface IItemService
```

Methods

Type	Name	Description
ItemMaster	AddItem(string itemNumber, string cat, string titel, string desc)	Adds an Item to the package. The item number will be an unique identifier in vault and it should be unique, for instance "100001".
ItemMaster	GetItem(string itemNumber)	Returns the item, which has the passed itemNumber if it exists.
BOMComponent	AddFileLinkToItem(ItemIteration itemVersion, FileIteration fileVersion)	Links a file version to a specific item version.

Remarks

AddFileLinkToItem ensures that all files are correctly linked to the item. It handles if the link should be primary, secondary or tertiary.

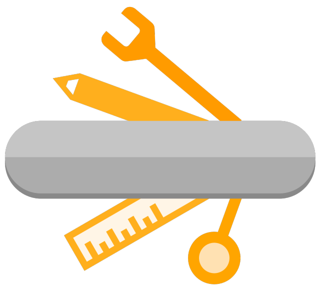
If the file is a content center file, a StandardComponent link will be created. If the file has design files, they will be linked as tertiary links to the item. Files marked as ConfigurationFactory will only be attached to the item.

You can get the latest iteration of an item/file with itemMaster.LatestIteration or file.LatestIteration.

Examples

You have to set the creation date otherwise, the Vault client brings the error message "1417 GetBOMFailed-NothingEffective" when accessing that item.

```
var item = bcpService.ItemService.AddItem("9992", "Document", "Title 9992", "Desc 9992");
item.LatestIteration.SetCreateDateFormatted(DateTime.Today);
```



The bcpDevKit .NET library contains a set of classes, interfaces, and value types that makes it very easy to create your own BCP-package which can be imported into Vault.

The libary *supports*:

- .Net Framework 4.7 or higher.

4.2.9 Classes

Class	Description
BcpService-Builder	This factory creates the BcpService with the relevant settings for you.
Export-Settings	This class provides settings for the creation of typically Vault settings that will be created in the package.
Entities-Table	Grants direct access to the BCP entities.

4.2.10 Interfaces

Interface	Description
IBcpService	Provides all the functionality to create a BCP package.
ICustomObjectSer-vice	Provides functionality to add, search and manipulate custom objects and their definition syntax.
IFileService	Contains functionality to add, search and manipulate files and folders in your package.
IItemService	Contains functionality to deal with items and BOM.

4.2.11 Enumerations

Enumeration	Description
BcpVersion	Specifies the Vault version of the BCP package.

CHAPTER FIVE

LOGGING

powerLoad (bcpToolkit) tools use 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\bcpToolkit\Logs\bcpToolkit.log'* and it contains Infos, 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\bcpToolkit\bcpToolkit.log4net*.
Further information about log4Net Configurations can be found here.

You can change the logging behaviour of:

- *bcpViewer*, it's engine, Window and database
- the *PowerShell IDE*
- projects using the *.NET Library*

5.1 When to change the logging behavior?

When you have issues or when you want to get a more detailed knowledge about what went wrong, you can increase the loglevel.

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

Additionally you can change the logfile location or integrate the logging mechanism into your administrative environment by using build in EventLogMessages etc.

Following section is used to control the logging behaviour for all tools:

```
<root>
...
</root>
```

For the moment only following *LogAppender* is used:

5.2 LogFile

This is the main *LogAppender* used in all the loggers. If you want to change the logging level in the logfile, please visit following appender:

```
<appender name="FileAppender" type="log4net.Appender.RollingFileAppender">
```

In the line

```
<param name="File" value="${LOCALAPPDATA}\coolOrange\bcpToolkit\Logs\bcpToolkit.log" />
```

you can configure the outputpath and name of the logfile.

Since this appender has no configured *LevelRangeFilter*, its loggingLevel has to be configured on the loggers.
In the lines

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

you can configure the logging level. You could set the level to "DEBUG", then all the levels between the range Debug and Fatal will be logged.

5.3 bcpViewer

Following section is used to control the logging behaviour for the *bcpViewer.exe* and it's *engine* logic:

```
<logger name="bcpViewer"></logger>
```

In order to configure sub-functionalities like the *UserInterface*, the following configuration section could be used:

```
<logger name="bcpViewer.UI"></logger>
```

This is the place where you want to increase the logging level when you need more detailed informations about whats happening in the bcpViewer *Database*.

```
<logger name="bcpViewer.Database"></logger>
```

For all this sections, only the *LogFile* appender is used.

5.4 PowerShell IDE

When using bcpToolkit cmdlets in PowerShell environments, logs are written to the PowerShell *Console Window*. Following section is used to control the logging behaviour for the *Cmdlets* of this module:

```
<logger name="bcpToolkit.Cmdlets"> ... </logger>
```

In order to customize the logging level in the console window, visit following appender that is used in addition to the *LogFile*.

5.4.1 ColoredConsoleAppender

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

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

In the lines

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

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

We are using a ColoredConsoleAppender, therefore you could also change the colors of the messages, depending on their log level:

```
<mapping>
        <level value="DEBUG" />
        <backColor value="White" />
</mapping>
<mapping>
        <level value="INFO" />
        <backColor value="Green" />
</mapping>
<mapping>
        <level value="WARN" />
        <backColor value="Yellow" />
</mapping>
<mapping>
        <level value="ERROR" />
        <backColor value="Red" />
</mapping>
<mapping>
        <level value="FATAL" />
        <backColor value="Red, HighIntensity" />
</mapping>
```

Troubleshooting

The **PowerShell ISE** currently does not support *console logs* at all.

In addition to the earlier described logging section *'bcpToolkit.Cmdlets'*, following section can be adjusted for PowerShell environments as well, since they make use of the *.NET library* internally.

5.5 Projects using .NET Library

When using the bcpDevKit .NET library in *custom projects*, logging can be controlled in following section:

```
<logger name="bcpDevKit"> ... </logger>
```

In addition to the *LogFile*, all projects that support console windows will make use of the configuration for the *ColoredConsoleAppender* too.
For debugging purpose, following additional appender can be adjusted in order to meet your projects requirements:

5.5.1 OutputDebugStringAppender

The *OutputDebugStringAppender* writes to the OutputDebugString system.

```
<appender name="OutputDebugStringAppender" type="log4net.Appender.
↪OutputDebugStringAppender" >
        <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%date [%thread] %-5level %logger - %message
↪%newline" />
        </layout>
</appender>
```

When developing new projects this can be useful because the logs are directly shown in the Visual Studio Output Window.

CHAPTER

SIX

CHANGE LOGS

6.1 powerLoad (bcpToolkit) v24

6.1.1 v24.0.4

`29-09-2023`

General

- Updated Licensing to version: 18.3.1
- End User License Agreement (EULA) has changed
- Updated minimum required .NET Framework version to 4.7
- Removed support for VaultBCP 2017, 2018, 2019 and 2020

Features

- Added support for VaultBCP 2021, 2022, 2023 and 2024
- Added support for *IterationRef* s:
  - *FileService*: New method `AddFileWithIterationRef`
  - *FileObject*: New method `AddFileWithIterationRef`
  - *FileRevision*: New method `AddIterationRef`
  - *FolderObject*: New method `AddFileWithIterationRef`

Fixed

- *FileIteration* not being added to *FileRevision* when an Iteration with the identical LocalPath already exists

Breaking Changes

Updated BcpVersion-enum
Removed *BcpVersion* Enum values:

- Removed *BcpVersion._2020*
- Removed *BcpVersion._2019*
- Removed *BcpVersion._2018*
- Removed *BcpVersion._2017*

Renamed IFileService, FolderObject and FileObject methods

- *IFileService*:
  - Renamed `AddFile` to `AddFileWithIteration`
- *FileObject*
  - Renamed `AddRevision` to `AddRevisionWithIteration`
- *FolderObject*
  - Renamed `AddFile` to `AddFileWithIteration`

6.2 powerLoad (bcpToolkit) v20

6.2.1 v20.0.5

`20-01-2021`

General

- Updated Licensing to version: 18.1.22
- End User License Agreement (EULA) has changed
- Added *bcpToolkit Information* shortcut to startmenu
- Removed *bcpToolkit Help* shortcut from startmenu as it can be accessed via bcpToolkit Information shortcut
- Removed Splashscreen
- Copyright notices have changed

Fixed

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

6.2.2 v20.0.2

`05-11-2019`

General

- Updated Licensing to version: 18.0.10

6.2.3 v20.0.1

`10-07-2019`

Features

- Added support for VaultBCP 2020
- Added support for *Stand-Alone Licensing*

General

- End User License Agreement (EULA) has changed
- Updated Licensing to version: 18.0.7
- Removed support for VaultBCP 2016

Breaking Changes

Updated BcpVersion-enum

Removed *BcpVersion*-Enum value: *BcpVersion._2016*

6.3 bcpToolkit v19

6.3.1 v19.0.5

`17-07-2018`

Features

- Added support for VaultBCP 2019
- *.NET Library* *bcpDevKit* gets installed in the GAC
- Added *PowerShell module* with new Cmdlets: *Open-BcpPackage*, *Export-BcpPackage* and *Close-BcpPackage*

General

- Renamed *bcpChecker* to *bcpViewer*
- Standardized Logging same as for other products
- added PostSharp Diagnostics for extended Debug logging (replaces Log4PostSharp in bcpDevKit)
- Changed registry keys to "HKLM\Software\coolOrange s.r.l.\bcpViewer": Location and Version
- Removed support for VaultBCP 2012, 2013, 2014, 2015 & 2015R2
- Signed bcpDevKit library with a Strong Name
- Extended *SecurityObject* in bcpDevKit with new *Roles* functionality
- Removed 'Save Package' functionality from bcpViewer (use *Export-BcpPackage* instead)

Fixed

- removed underscore from available "BCP Versions" in bcpViewer "Open Package" dialog
- *BomComponent.AddInstance* method in bcpDevKit automatically assigns 'FileIteration' (and 'FileIterationId') to the file associated with the new childComponent
- performance improvements (approximately 10%) with opening large BCP packages via *Open-BcpPackage* and *bcpViewer*

Breaking Changes

All projects that are using bcpDevKit have to be **recompiled** using the *strong named* version from GAC.
References to *bcpDevKit.Entities.dll* should be **removed**, since all *entity types* got moved into *bcpDevKit.dll*

Updated BcpVersion-enum

Removed *BcpVersion*-Enum values: 'BcpVersion._2012', 'BcpVersion._2013', 'BcpVersion._2014', 'BcpVersion._2015' and 'BcpVersion._2015R2'.
Moved from namespace 'bcpDevKit.Entities' to 'bcpDevKit'

Added Enumerations for Access and Authentication

Changed type of ‘UserObject.Access’ and ‘GroupObject.Access’ from *string* to *SecurityPrincipal.VaultAccess*.
Changed type of ‘UserObject.Authentication’ and ‘GroupObject.Authentication’ from *string* to *SecurityPrincipal.VaultAuthentication*.

Renamed FileClassificationEnum to FileClassification

Moved ‘bcpDevKit.Entities.Vault.FileObject.FileClassificationEnum’ to ‘bcpDevKit.Entities.FileClassification’.
Changed types of methods ‘FileObject.FileClassificationEnum FileObject.GetFileClassification()’ and ‘FileObject.SetFileClassification(FileObject.FileClassificationEnum fileClass)’ to ‘bcpDevKit.Entities.FileClassification’.

Renamed ClassTypEnum to ClassType

Moved ‘bcpDevKit.Entities.Configuration.BehaviorsObject.ClassTypEnum’ to ‘bcpDevKit.Entities.ClassType’.
Changed types of several class-properties and methods in namespace ‘bcpDevKit.Entities.Configuration’ from ‘bcpDevKit.Entities.Configuration.BehaviorsObject.ClassTypEnum’ to ‘bcpDevKit.Entities.ClassType’.

Renamed BOM-types

Renamed following BOM-types in namespace ‘bcpDevKit.Entities.Items’:

- renamed BOMComponent to BomComponent
- renamed BOMComponentItemToComp to BomComponentItemToComp
- renamed ItemsBOMComponentProperty to BomComponentProperty
- renamed instance to BomInstance

Removed Ser_Quantity from BomLink

Removed obsolete property ‘Ser_Quantity’ from BomLink.
Use typed property ‘Quantity’ instead.

Removed SchemeId from BomDetail

Removed obsolete property ‘SchemeId’ from BomDetail.
Use alternative property ‘IsStructured’ instead.

Replaced Get/SetLinkType on BOMComponentItemToComp

Removed methods ‘LinkTypeEnum BOMComponentItemToComp.GetLinkType()’ and ‘BOMComponentItemToComp.SetLinkType(LinkTypeEnum linkType)’.
Use alternative property ‘BomComponentItemToComp.LinkType’ instead, by using new Enum-values ‘PrimarySubcomponent’ or ‘SecondarySubcomponent’.
Moved Enum ‘bcpDevKit.Entities.Items.BOMComponentItemToComp.LinkTypeEnum’ to ‘bcpDevKit.Entities.LinkTypeEnum’

Renamed BomInstanceStructure to BomStructureType

Moved ‘bcpDevKit.Entities.Items.instance.BomInstanceStructure’ to ‘bcpDevKit.Entities.BomStructureType’.
Changed type of ‘instance.Structure’ from ‘instance.BomInstanceStructure’ to ‘bcpDevKit.Entities.BomStructureType’.

Moved Enums from BOMComponent to Entities namespace

Moved following Enums out from type ‘bcpDevKit.Entities.Items.BOMComponent’:
* Instead of ‘BOMComponent.BomComponentType’ use ‘bcpDevKit.Entities.BomComponentType’

- Instead of ‘BOMComponent.BomStructureType’ use ‘bcpDevKit.Entities.BomStructureType’
- Instead of ‘BOMComponent.ContentSource’ use ‘bcpDevKit.Entities.ContentSource’

Moved Enums from BomLink to Entities namespace

- Instead of ‘BomLink.BomLinkType’ use ‘bcpDevKit.Entities.BomLinkType’

Moved Enums from AssociationsObject to Entities namespace

- Instead of 'AssociationObject.AssocType' use 'bcpDevKit.Entities.AssocType'

Fixed property namings in following classes

- CategoryAssignObject
- CategoryDefObject
- GroupObject
- InGroupObject
- InRoleObject
- PropertyDefObject
- Attachment

6.4 bcpDevKit v18

6.4.1 v18.0.3

`12-03-2018`

General

- Added new version of Licensing assemblies
- Increased .NET Framework to v4.5

Fixed

- Some elements in *Vault.xml* file have *'d7p1'* namespace attributes (or similar) in certain situations

6.4.2 v18.0.1

`08-09-2017`

- Official Release

General

- Added support for BCP 2017
- Added support for BCP 2018

Breaking Changes

Replaced *Instance.ChildCompId* with *ChildComponent* property (still directly accessible on *BomInstanceOld* type if required)
Replaced *BomLink.SetInstanceCount* with InstanceCount
Replaced *BomLink.SetIsCAD* with IsCAD
Replaced *BomLink.SetUnitSize* with UnitSize
Replaced *Occurrence.ChildCompId* with *ChildComponent* property (still directly accessible on *BomOccurrenceOld* type if required)
Changed type of *ItemMaster.ItemMasterID* from Integer to string
Changed type of *ItemMaster.ItemMasterID* from Integer to string

Changed type of *Occurrence.SchemeType* to enum *BomSchemeType*
Changed type of *ItemIteration.AddBomLink()* to return type *BomLink*
Renamed *CustomObject.Setstate* to *SetState*
Renamed *Instance.StructureName* to *Structure* and **changed type** to enum *BomInstanceStructure*
Added parameter classification to *BehaviorsObject.AddRevisionDef*
Added function *ItemIteration.AddBomDetail*

Fixed

- Function: ItemRevision.SetRevision automatically creates RevisionDefinition with correct AssignmentType

6.5 bcpDevKit v16

6.5.1 v16.0.29

`29-08-2016`

Fixed

- File revisions do not become ordered automatically. Similiar as for Items, the class File has a new function *OrderRevisions()* for automatically ordering the revisions.

6.5.2 v16.0.28

`28-06-2016`

Features

- Updated Licensing to latest Version.

Fixed

- Issue where bcpDevKit runs in Trial after successful license activation

6.5.3 v16.0.19

`02-09-2015`

- First release of bcpDevKit 2016
- Introduced new licensing
- supports BCP 2016
- supports BCP 2015 R2

Features

- ExportSettings are able to be changed also after a Flush happened
- UserExport flag in ExportSettings for disabling automatic generation of users
- DisableConfigurationExport function in exportSettings, for disabling export of all configuration object types

Fixed

- Changing ExportSettings like CategoryDefinitionsExport / LifecycleDefinitionsExport was ignored and had no affect at all

6.6 bcpChecker v18

6.6.1 v18.0.6

`12-03-2018`

General

- Assemblies *coolorange.licensing* and *coolorange.Utils.UI* now gets installed in the GAC

Fixed

- Issue which caused the Dialog to crash when showing properties for Files/Folder/Items

6.6.2 v18.0.4

`08-09-2017`

- Official Release

General

- Added support for BCP 2017
- Added support for BCP 2018
- Installed "bcpChecker 18.0 Logs" shortcut in start-menu section of bcpChecker
- Moved LogFile to %LOCALAPPDATA%/coolOrange/bcpChecker/Logs in order that Non-Admin users have write-access to the files.

Fixed

- Issue which caused the Dialog to crash when using the folder browser
- Issue with the Online Help button in the Help menu

6.7 bcpChecker v16

6.7.1 v16.0.25

`02-09-2015`

- support for BCP 2016
- added latest version of licensing
- Add Activator.exe to Setup

Features

- Add filters for each column

Fixed

- Icons of the General Tab in ItemMaster are not shown correctly

6.8 bcpChecker v15

6.8.1 v15.0.297

`10-10-2014`

Features

- support for Vault BCP 2015 R2

6.8.2 v15.0.289

`17-09-2014`

Features

- support for items
- progress bar while exporting, importing or reloading package
- Open-Package and export can be cancelled
- reload database function
- Switch to exclude BomBlobs when importing\exporting bcp package
- better user experience when reading data from database takes too long
- more logging (log4Postsharp)

Fixed

- uses structure is no longer shown
- whereUsed shows dependency when is attached
- opening BCP-packages from network path
- ItemVersion was showing wrong PartNumber
- fixed different errors in 2014 xsd for export
- trial mode behaviour on contentView
- wiki-link was not working

The Data Transfer Utility (aka VaultBCP) allows to import data of any size into Vault in a reliable and complete way. It allows importing files, items, bill of materials and custom objects, including historical versions, and all the related links.
powerLoad (bcpToolkit) provides several tools for creating, manipulating and viewing such BCP packages easily, in order to be ready to import them into Vault!

CHAPTER

SEVEN

BCPVIEWER

An import of several thousand files might take some hours and if there are some erroneous configurations in the BCP package you may figure it out very late.
The *bcpViewer* lets you **preview** the BCP package in an UI that looks similar to Vault.
You can expand the folders, see the files in each folder, their properties, links, history, etc.

CHAPTER

EIGHT

.NET LIBRARY

The *bcpDevkit* is a .Net library which can be used in your C# or VB projects when you need to **develop** a custom migration with BCP.
You can start building your BCP-package right away, without having to know the XML format from BCP.
The library creates all the right settings for you and saves you a lot of time and nerves! Documentation, samples and tutorials will help you getting started with minimal effort.

CHAPTER

NINE

CMDLETS

The bcpToolkit module provides *commandlets* that allow creating scripts for **automating** the work with BCP packages. These Cmdlets allow handling large BCP-packages in a very simple, flexible and performant way.