SD226856

Custom Reporting in Vault 2019 - Dress Up Your Vault Data to Meet the World

Christian Gessner
coolOrange S.r.l.

Markus Dössinger
coolOrange S.r.l.

Learning Objectives

- Learn how to programmatically query report data and pass it to a report engine
- Learn how to show rendered reports in custom Vault Explorer extensions
- Learn how to programmatically render reports and save them as PDFs
- Learn how to embed cloud-based reports to custom Vault Explorer extensions

Description

Vault 2019 software includes an updated version of the Microsoft Report Viewer 2015 Runtime for its own reporting and data visualization. This class will show you how to take advantage of this very powerful capability for your own reports and visualizations. We will show you how to generate high-quality representations of your vaulted data inside either Vault Data Standard user interface or in your Vault Client as Explorer extensions. We will show how to query data from Vault software and pass it to the reporting engine. Then we will show how to present the reports interactively to the user and how to publish them as PDFs or images programmatically. Finally, we will have a brief look at cloud-based reporting tools that you can also use to show your Vault data in its best light.

Speakers

Christian Gessner is a co-founder and a software engineer at coolOrange. He is involved in the design, specification and implementation of data management projects and customizations, in particular in the Autodesk data management environment. Besides that, he teaches programming and visits customers for on-site trainings and workshops. Previously, Christian worked as a SQA engineer for data management products at Autodesk

Markus Dössinger is a co-founder and a consultant at coolOrange. He guides teams through Autodesk data management projects with emphasis on migration to Vault. As part of this Markus is working with Autodesk Support on migrations from Autodesk Productstream Professional to Autodesk Vault Professional or Workgroup. Previously, Markus worked as a product designer for data management products at Autodesk.

AUTODESK. UNIVERSITY

Contents

Introduction

After a brief summary of Vault's built-in reporting functionality, this document describes how reports can be customized and how the Microsoft technologies can be used to extend Autodesk Vault's reporting functionality. These extended capabilities are more flexible and sophisticated than the pre-configured reporting and will fit seamlessly into the existing Vault user experience.

Target Audience

This class primarily addresses programmers but is also suitable for administrators, consultants and applications engineers with some experience in .NET with basic knowledge of the Vault API.

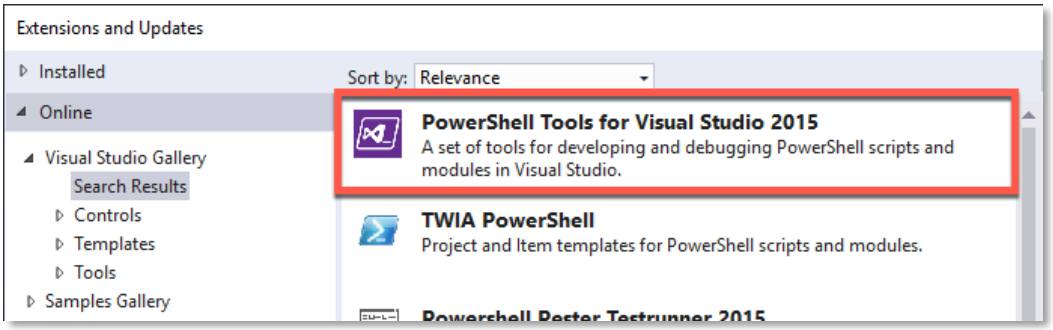
Sample Materials

We have published a Visual Studio solution with all examples and use cases used in this class plus many more examples of "Reporting".

Please visit the GitHub repository:

https://github.com/christiangessner/AU2018_SD226856

To open the solution in Visual Studio, the extension *PowerShell Tools for Visual Studio 2015* needs to be installed from the *Visual Studio Gallery* (Tools → Extensions and Updates).



All the projects are written in C# or PowerShell. To compile the solution, the Vault SDK for Vault 2019 must be installed to the default location:

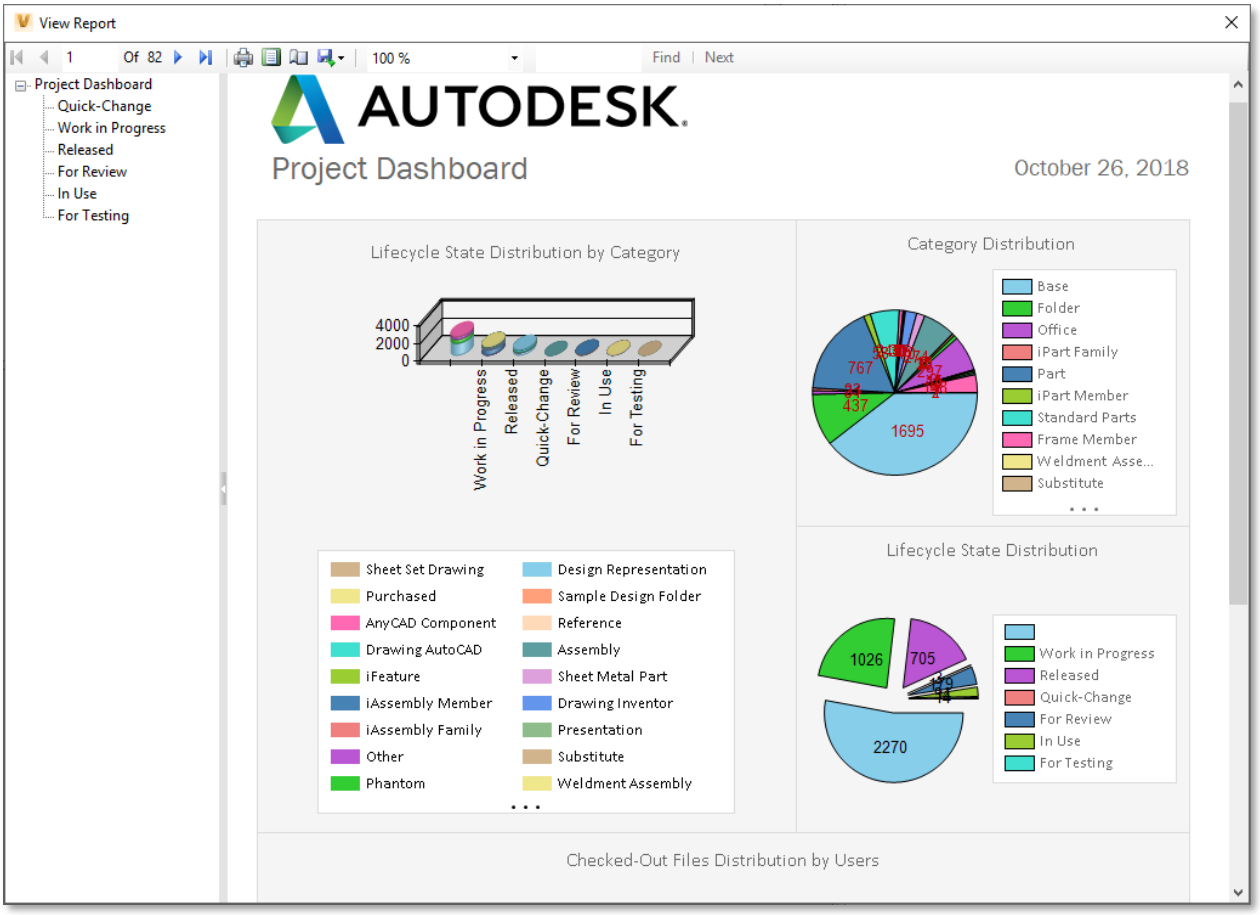
C:\Program Files (x86)\Autodesk\Autodesk Vault 2019 SDK.

If the SDK is installed to a different location, the references to the Vault assemblies must be changed in all the projects.

Vault Report Basics

Vault Workgroup and Vault Professional both provide the ability to generate nicely formatted reports representing data contained in a vault. This data includes files, items, change orders and all the metadata associated with them including properties, categories, lifecycle states and version numbers.

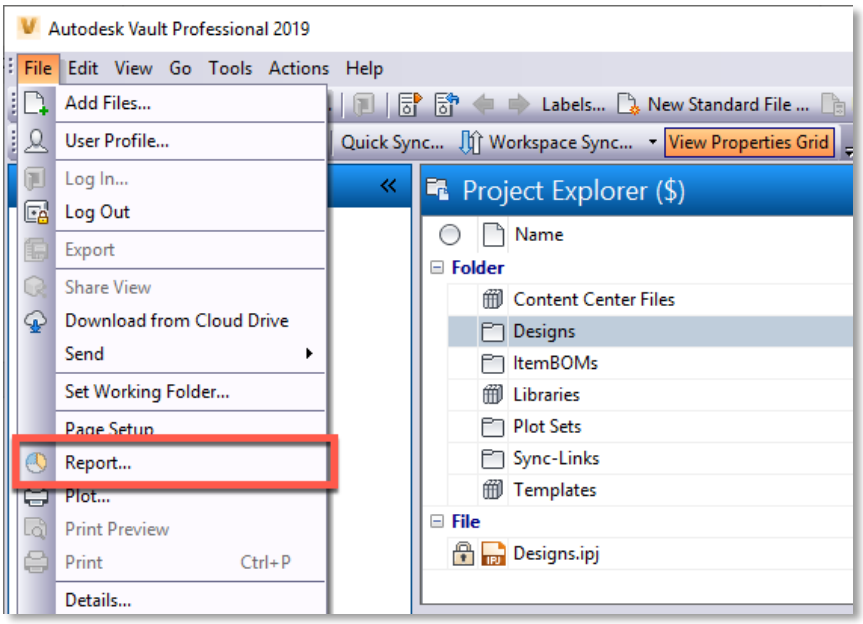
For example, a report can display files grouped by a category, summarizing currently open change orders, or show the distribution of lifecycle states across a complex model. Reports can display this data in a variety of ways, including charts, tables, and data sheets. You also have access to dozens of pre-defined operators to help format all that data.



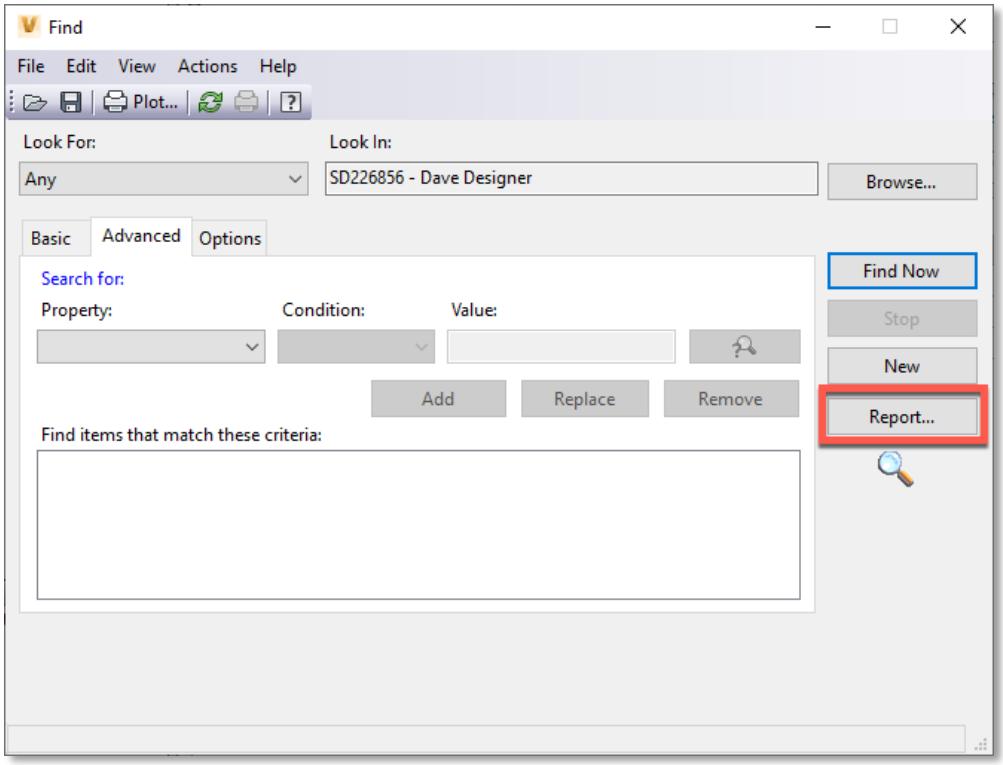
A Project Dashboard Report in Vault

Executing Vault Reports

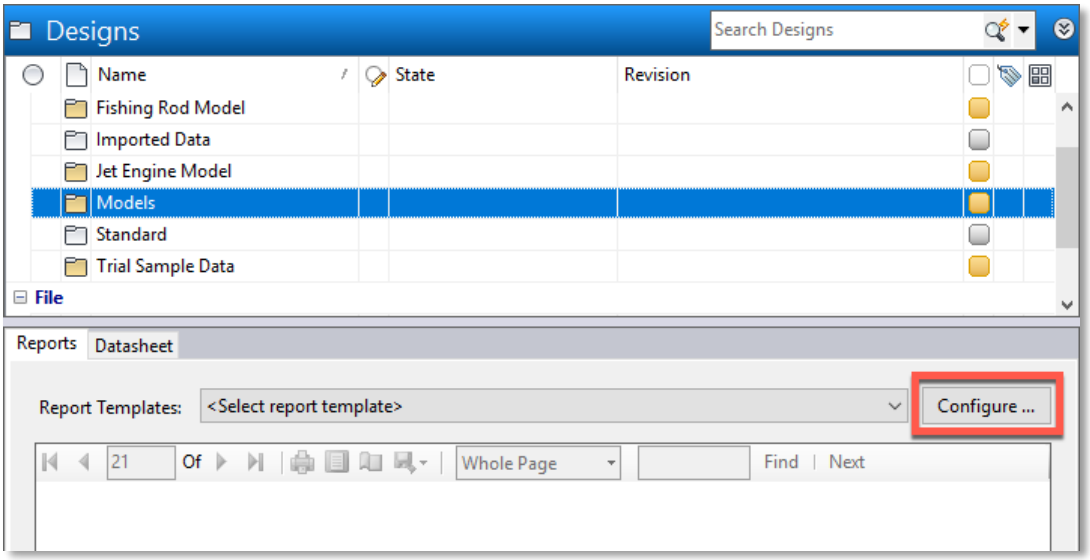
The built-in reports can be invoked from different places in Vault Explorer.



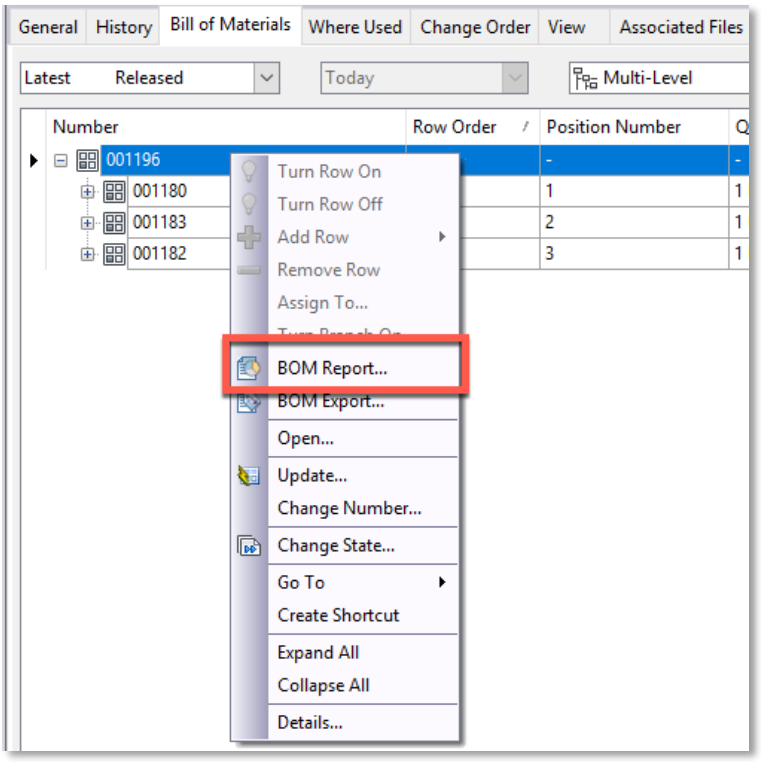
PROJECT EXPLORER MAIN MENU (FILE)



ADVANCED SEARCH DIALOG

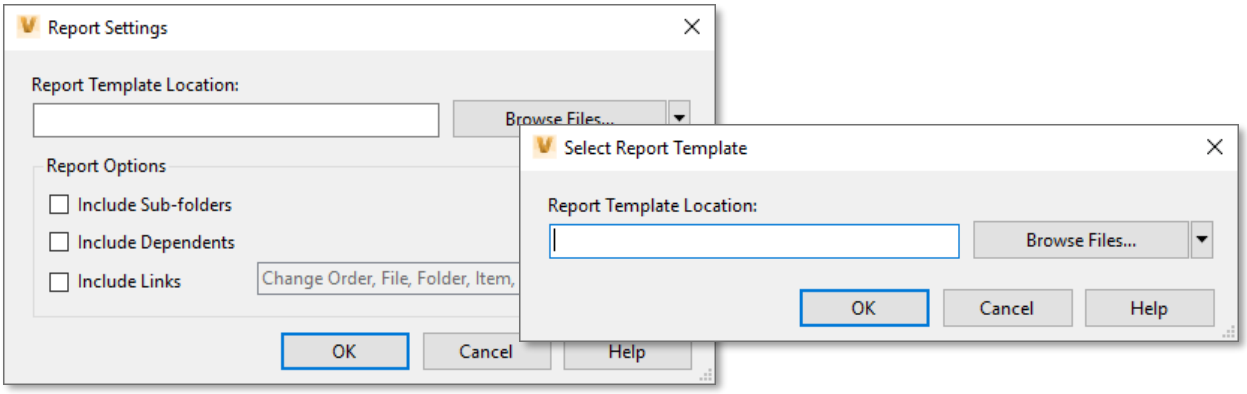


FOLDER "REPORTS" TAB



ITEM BOM CONTEXT MENU

Once you choose a report command you will be presented with a dialog where you can choose a context appropriate Report Template (stored in *.rdlc files):



VAULT REPORT SETTINGS

After selecting “OK” the report is rendered and either shown in the context (a tab) of the folder/project or in a new window.

The properties displayed in the report, as well as the report layout, are specified in a Report Template file that is selected during the report generation. Through the Report Template, you have complete control of the report content, layout, and format.

Anatomy of Vault Reports

Technology

Autodesk Vault uses Microsoft *Report Definition Language* (*.rdlc) templates files to render database query results. The template file format is *.RDLC; the "c" is added to distinguish server report definitions for server service (SQL Reporting Service) and locally executed reports on clients.

To render the report locally a Report Viewer is required. Vault comes with an embedded copy the *Microsoft Report Viewer Redistributable 2015*. This is used to render the reports both inside a Vault client tab or in a pop-up window.

The reports are fed vault database queries that are either a custom user search, saved user search, or the built-in searches in Vault Explorer or Inventor Vault Client Add-In. The Report Viewer engine renders the search result into the template interactively selected by the user.

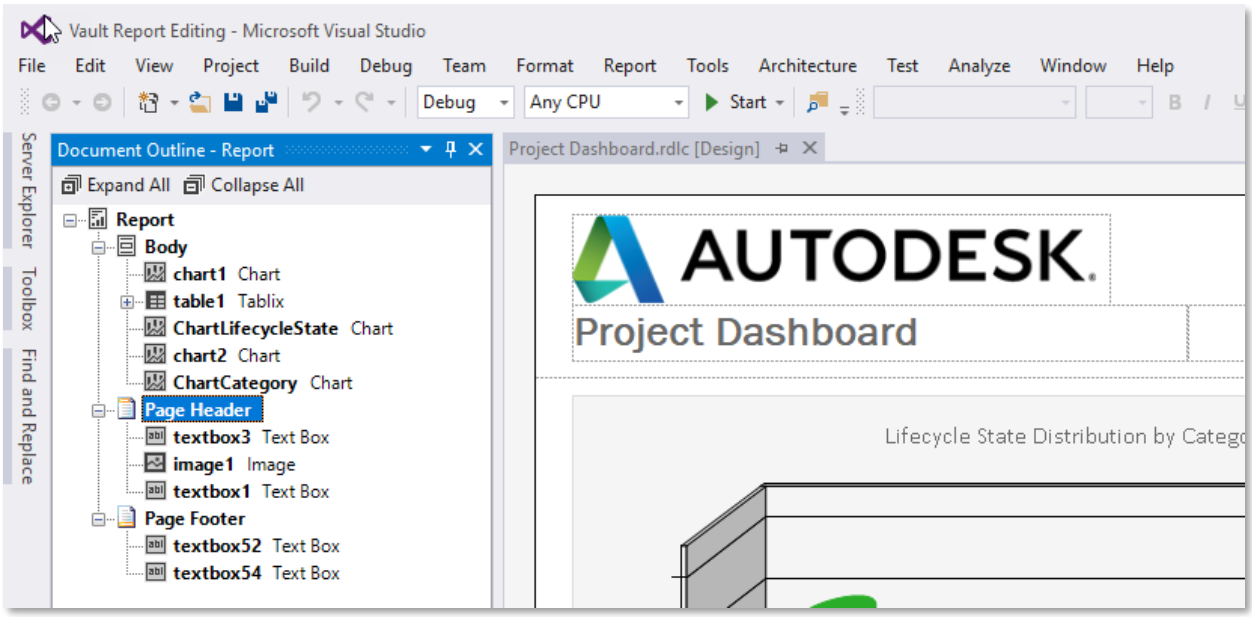
Note - The Report Viewer that is integrated to the Vault Explorer uses 2010 Report definition format. The format relates to the default SQL Server 2014 version of Vault 2019. Visual Studio 2017 (or later) may apply definition format 2016 as a default. Templates migrated to this format will result in an error message about obsolete viewer format selection.

Layout

The report definition includes the visual layout as well as content elements and resources comprised of

- Report Page: including Header, Body, Footer
- Report Data: including data report fields, report parameters, data sets

Let's look inside a Report Template and its constituent parts. Microsoft Visual Studio includes a *Report Designer* that visually represents the page its data structures:

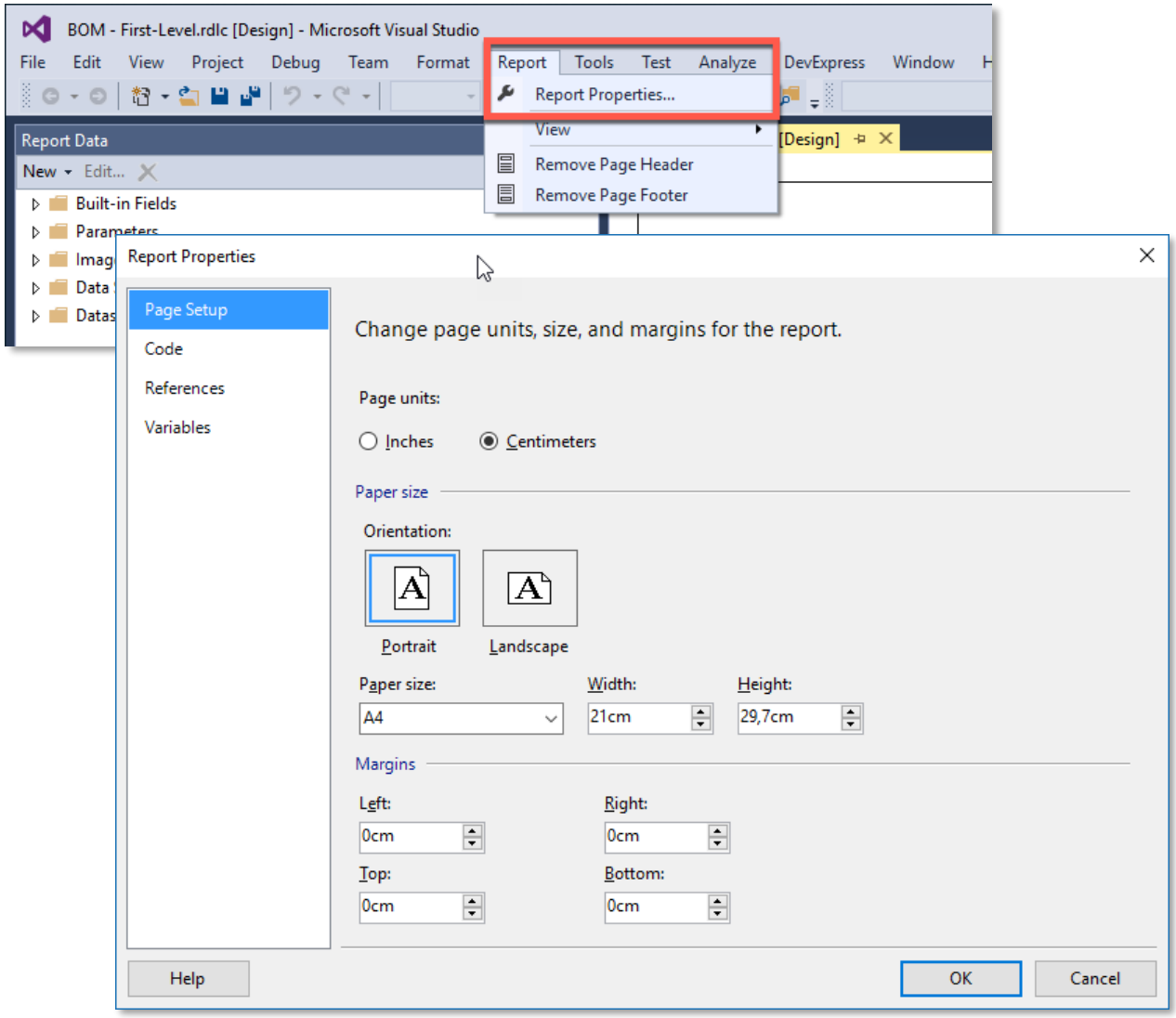


REPORT LAYOUT IN REPORT DESIGNER

Properties

We can also look at the general *Report Properties* in the designer. These properties include the page setup, code to extend expressions (Visual Basic Script) or references to external code libraries (.NET), and variables.

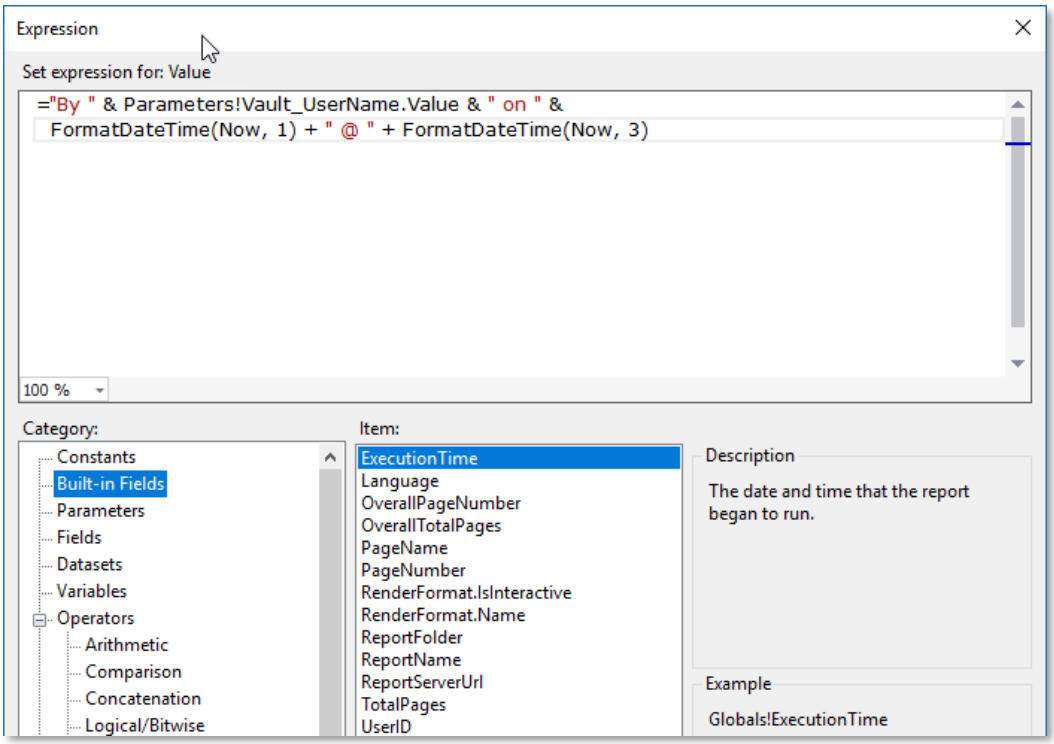
Access Report properties from the Report Menu, available in active Designer Windows:



REPORT PROPERTIES

Built-in Fields

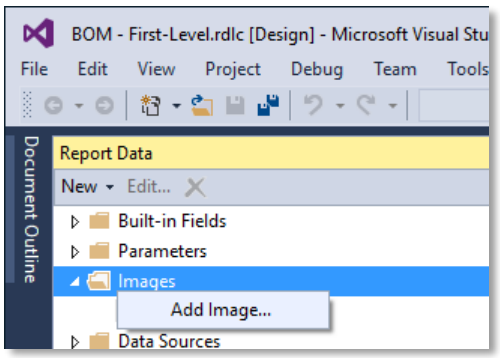
Built-in Fields are resources to be used individually in your template (drag & drop from the browser to the designer page) or combined into more complex expressions.



BUILT-IN FIELDS

Images

To use *Images* in your report they need to be imported into the template. Add / import image files right clicking on the image node:



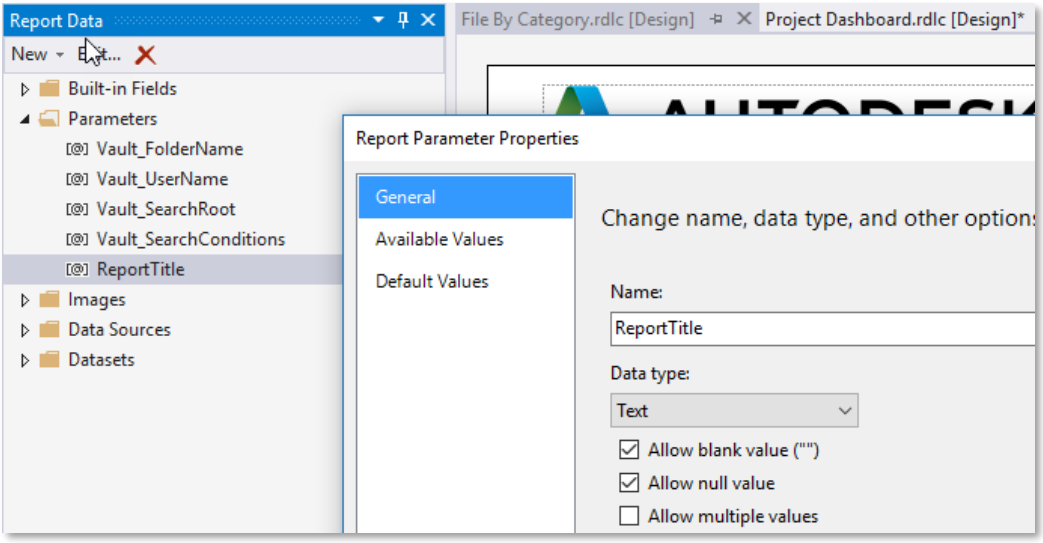
ADD IMGES TO A REPORT

Note – Image properties like size, visibility, borders and the like are editable for images once they are inserted into the report page.

Parameters

Parameters are scoped to a report's main level. They are intended to transfer determining conditions, formatting information etc. In other words, parameter values can represent the meta data for the report, such as who ran the report, “based on” search criteria, and query data/time. The parameters are filled by the calling Vault command, in context, and display the values at the time of the query.

Another use case of parameters are Vault's predefined report titles. The text displayed in the layout designer is not the title text during runtime. Vault provides the localized title string via parameter to the report. This allows to use one single Report Template file for different languages.



REPORT PARAMETER PROPERTIES

Parameters in the *Report Data Browser* reflect the context of current template open for editing only.

The following table summarizes parameters provided by Vault search results or folder views:

Parameter Name	Value
Vault_UserName	The name of the Vault user who generated the report.
Vault_VaultName	The name of the vault that provided data for the report. This is the vault that the user who generates the report is logged into.
Vault_SearchRoot	The name of the folder(s) specified in the Look in control located on the Find dialog in Vault.
Vault_LatestVersionOnly	The state of the Find latest versions only checkbox on the Options tab of the Find dialog in Vault.
Vault_SearchSubFolders	The state of the Search Subfolders" checkbox on the Options tab of the Find dialog in Vault
Vault_SearchFileContent	The state of the Search file content check box on the Basic tab of the Find dialog in Vault.
Vault_SearchConditions	A string representation of the search conditions specified on the Advanced tab of the Find dialog in Vault

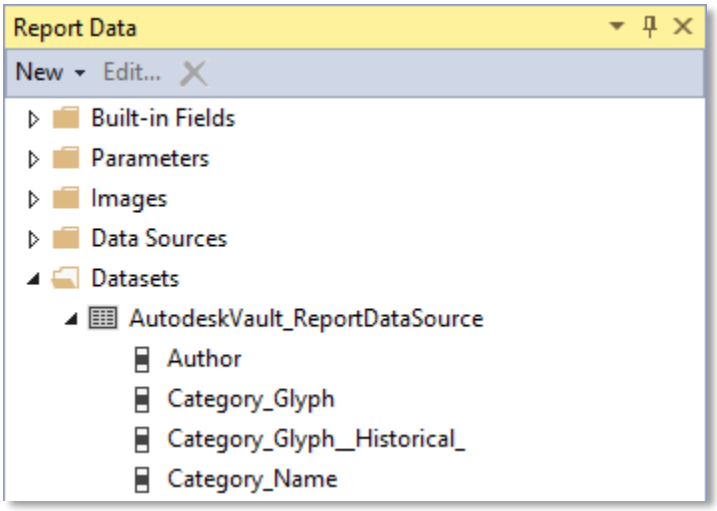
For Item BOM reports all properties of the item containing the BOM content can be used as parameter(s); the default templates contain for example (incomplete list):

Parameter Name	Value
ItemBOM_CategoryName	The name of the parent item's category.
ItemBOM_CategoryName_Ver	The parent item's historical category name.
ItemBOM_ModDate	Last date/time the item was modified.
ItemBOM_Title_Item_CO_	Title property of the parent item.
...	

Note - The parameter name for Item BOM reports is built as "ItemBOM_" + UDP SystemName.

Datasets

This is where we match the incoming data fields to the display elements in the report. The data source provides the fields to display queried content. The field definition is required to setup the report correctly. Therefore, the field definitions are stored in *Datasets*. To start from scratch, you need to establish a database connection and create data sets based on existing database tables and fields.



REPORT DATASETS IN VISUAL STUDIO

Each field in the Report Template consist of a name, a data field which represents the field of the data source and the data type of the field:

```
<?xml version="1.0" encoding="utf-8"?>
<Report xmlns:rd="http://schemas.microsoft.com/SQLServer/reporting/reportdesigner" ...
  <DataSets>
    <DataSet Name="AutodeskVault_ReportDataSource">
      <Fields>
        <Field Name="Author">
          <DataField>Author</DataField>
          <rd:TypeName>System.String</rd:TypeName>
        </Field>
        <Field Name="Category_Glyph">
          <DataField>CategoryGlyph</DataField>
          <rd:TypeName>System.Double</rd:TypeName>
        </Field>
```

REPORT DATASETS IN XML

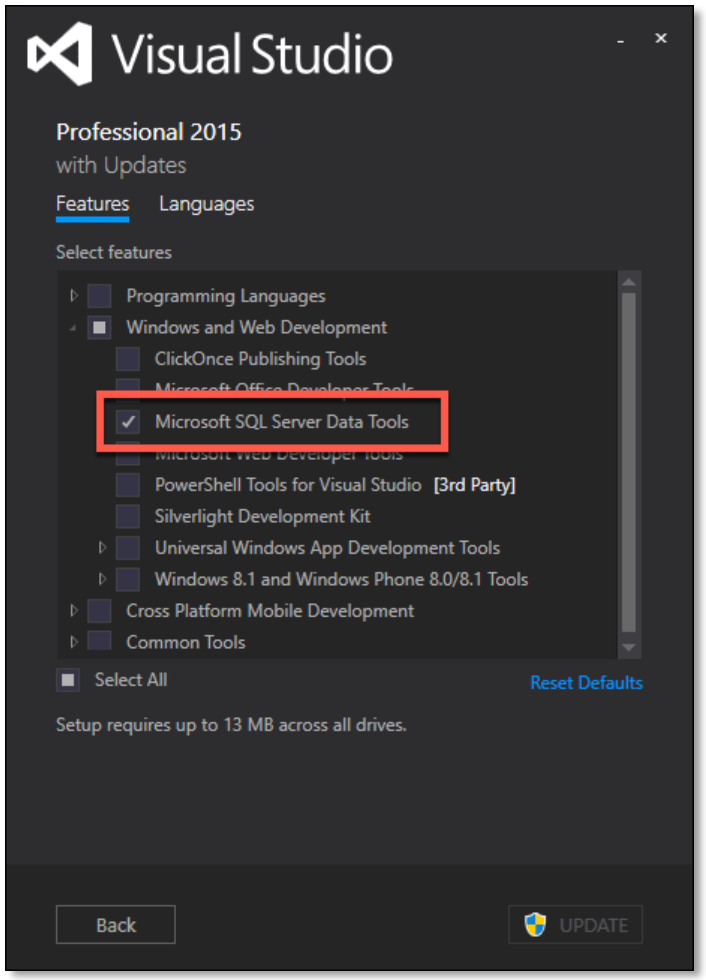
Customizing Report Templates

Requirements

To customize the built-in reports must install:

- Vault Client / Server 2019 Workgroup or Professional
- Microsoft Visual Studio Professional or Enterprise 2015

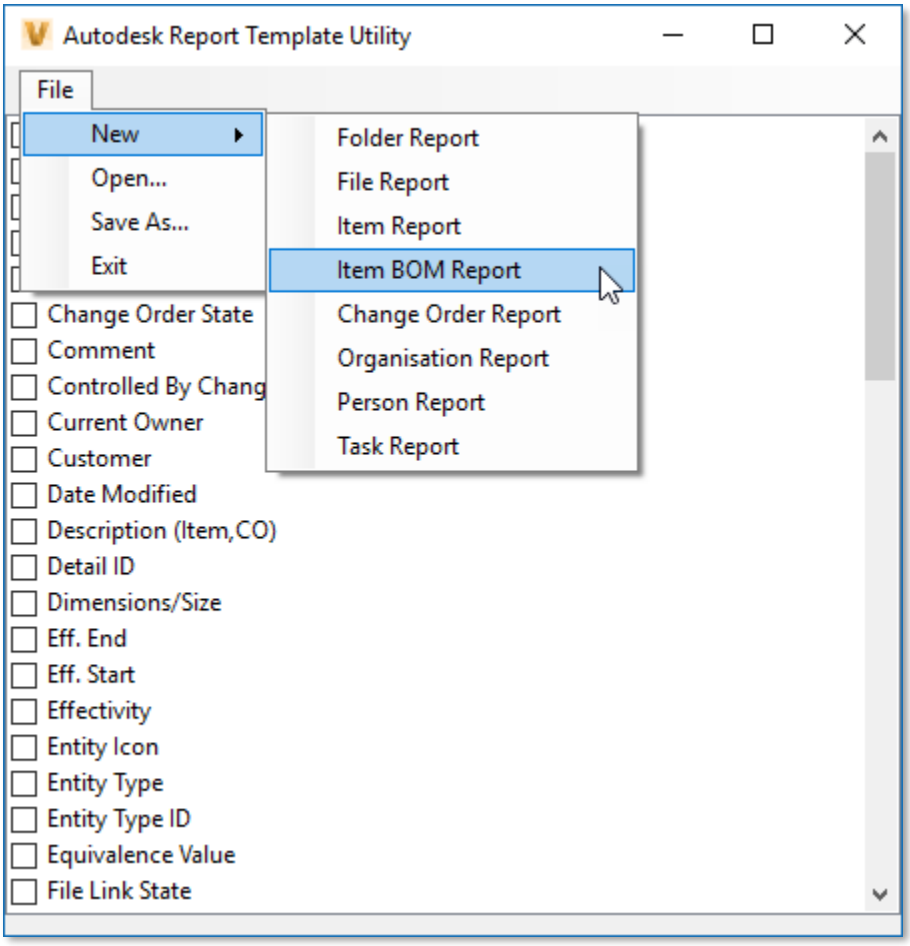
Note - Enable "Microsoft SQL Server Data Tools" during installation or add it as feature in existing Visual Studio installation:



Adding and Removing Fields

Vault provides a tool to create report definitions that already contain the data set. This reduces the steps required and know how needed drastically.

The *Autodesk Report Template Utility* can be used to create new Report Templates or to open existing templates and add or remove fields that are available in Vault.



VAULT REPORT TEMPLATE UTILITY

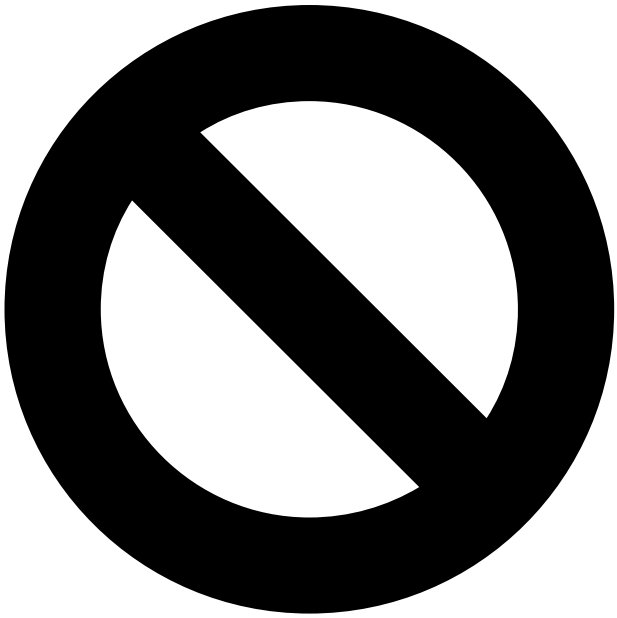
Changing Report Templates

To edit a Report Template, the RDLC file can be opened with Visual Studio 2015. Once opened, images can be added or replaced, columns added to tables, color changed and the like.

Note – Detailed descriptions on how to create and edit Report Templates are not part of this document but links to other resources can be found in the "Resources" chapter at the end of this document.

Limitations

- Fields are limited to "System Properties" and "User Defined Properties" of Vault.
- Additional parameters can be added using Visual Studio, but Vault is not able to fill these parameters since the parameters passed to the Report Templates are limited.
- The embedded report components are not exposed through the Vault API and thus, cannot be automated, extended or modified.

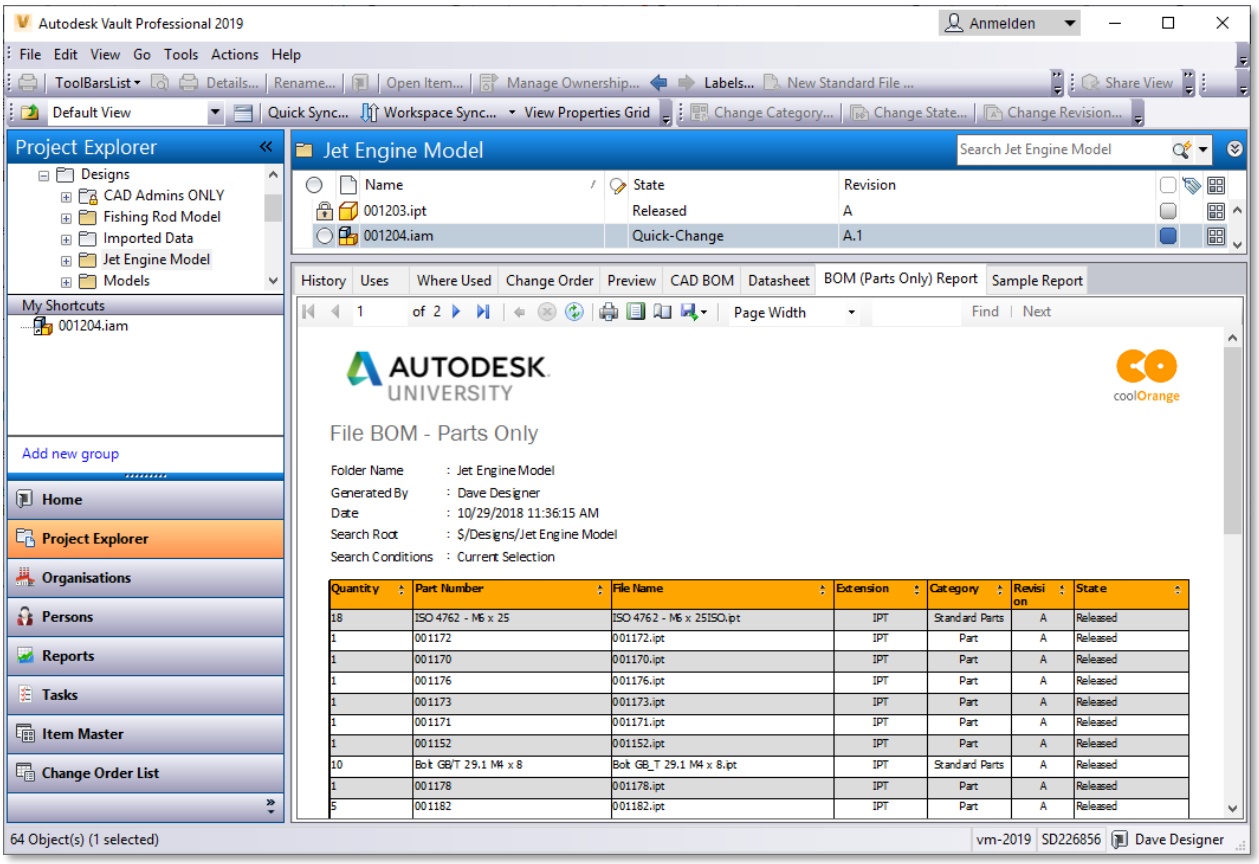


Creating your own Report functionality

We can use customizations to extend the reporting capabilities of Vault to include data that is not limited to the fields that are present as properties in Vault or standard parameters. It is also possible to use customizations to process automations (e.g. automatically creating PDF representations of the reports and add these PDFs to Vault).

Because there is no public API for the embedded reporting components in Vault, we must reproduce its mechanism to query data from Vault. Additionally, the queried data cannot be sent to the built-in controls, so you must create your own controls to render and display the reports.

Since Vault uses the Microsoft Report Viewer Redistributable 2015 for the embedded reports, it installs these controls on each computer with a Vault client. Luckily, we can also use them for our own purposes too!



Adding a Report Viewer to Vault

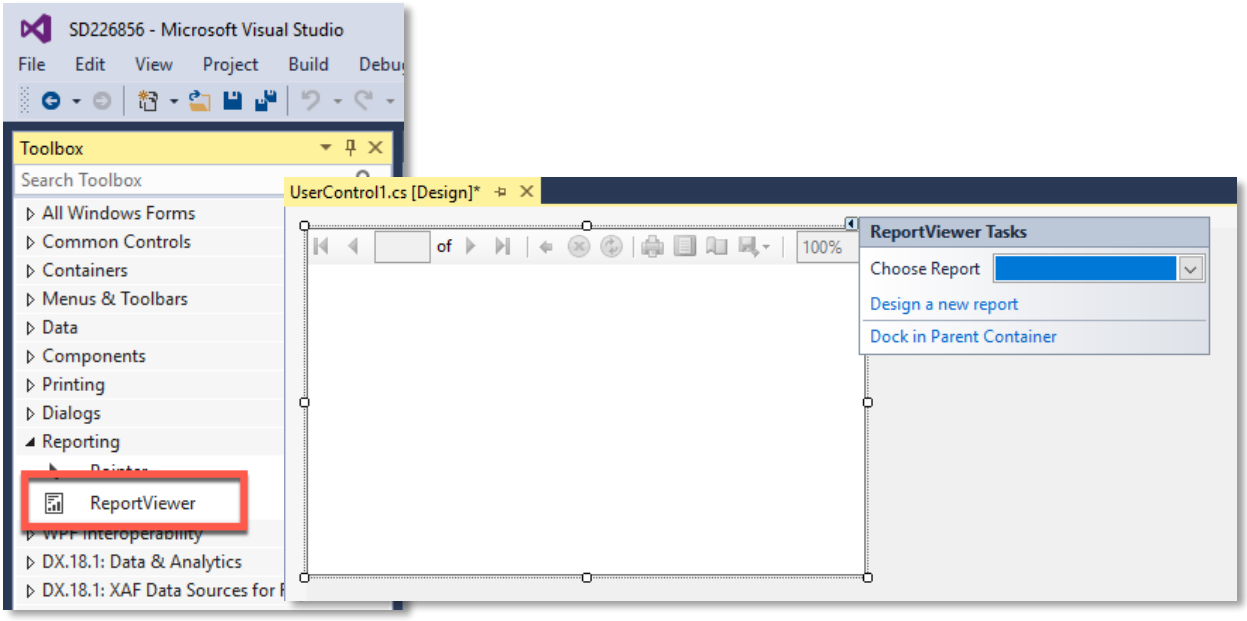
Learning Objective

Learn how to show rendered reports in custom Vault Explorer extensions

By using Vault Explorer Extensions or Vault Data Standard new custom UI elements and functionality can be added to Vault Explorer. This can be used to add our own instance of a Microsoft Report Viewer control to a Vault tab or to a new window.

Vault Explorer Extension

Vault Explorer Extensions enables the extension of the UI by using *Windows Forms*[1] (WinForms). WinForms are .NET components including container controls like *UserControl*[2] or *Form*[3]. Either of these controls can be used to house the ReportViewer control. Simply drag and drop the ReportViewer control from the Toolbox to either of these container controls.



Once the control is added to the UserControl or the Form, it can be completed with a local Report Template (RDLC) and the necessary Parameters and Datasets at runtime.

---

[1] https://docs.microsoft.com/en-us/dotnet/framework/winforms/
[2] https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.usercontrol
[3] https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.form

Note – The Report Viewer Control can also be used with Visual Studio versions other than 2015. Therefore, a reference to the assembly

C:\Windows\assembly\GAC_MSIL\Microsoft.ReportViewer.WinForms\12.0.0.0__89845dcd8080cc91\Microsoft.ReportViewer.WinForms.DLL

can be added to the project. The assembly is added to the Global Assembly Cache by the Vault client installer.

Vault Data Standard

Unlike a Vault Explorer Extensions, Vault Data Standard (VDS) uses the more modern *Windows Presentation Foundation*[4] (WPF) to add additional user interface elements to Vault.
In order to display the WinForms version of the Microsoft Report Viewer control, it has to be added to a XAML file with the help of the *WindowsFormsHost* [5] container, which allows you to host a WinForms controls in WPF elements.

```
<Window xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        xmlns:wf="clr-namespace:Microsoft.Reporting.WinForms;assembly=Microsoft.ReportViewer.WinForms"
        Title="SD226856" Height="550" Width="750" WindowStartupLocation="CenterOwner">
    <ScrollViewer VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">
        <Grid>
            <WindowsFormsHost>
                <wf:ReportViewer x:Name="ReportViewer" />
            </WindowsFormsHost>
        </Grid>
    </ScrollViewer>
</Window>
```

XAML OF AN APPLICATION THAT HOSTS A WINFORMS REPORT VIEWER CONTROL

Note – A sample of a WPF application that is implemented using PowerShell and can be used in Vault Data Standard can be found in the sample materials: SD226856.PowerShell → Scripts → Standalone.ps1

[4] https://docs.microsoft.com/en-us/dotnet/framework/wpf/
[5] https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.integration.windowsformshost

Make the Report Viewer Alive

The Report Viewer control provides a lot of different properties and methods to change its behavior and appearance. All of these settings can be taken from the official Report Viewer documentation[6].

In order to use the control in a Vault extension, the following common steps have to be performed:

1) Load a Report Template (RDLC) to create a LocalReport:
   `reportViewer1.LocalReport.LoadReportDefinition(StringReader);`
2) Add a data source to the Local Report:
   `reportViewer1.LocalReport.DataSources.Add(ReportDataSource);`
3) Set the report Parameters:
   `reportViewer1.LocalReport.SetParameters(ReportParameter[]);`
4) Adjust the Report Viewer appearance:
   `reportViewer1.ZoomMode = ZoomMode.PageWidth;`
5) Load and render the report:
   `reportViewer1.RefreshReport();`

Report Viewer Localization

The Report Viewer used the *culture* and the *UI culture* of the current thread to display the texts that are visible through the control's user interface. To change the language, the cultures can be changed:

```
var culture = new CultureInfo("de-DE");
Thread.CurrentThread.CurrentUICulture = culture;
Thread.CurrentThread.CurrentCulture = culture;
```

SETTING A THREADS CURRENT CULTURE

[6] https://msdn.microsoft.com/en-us/library/microsoft.reporting.winforms.reportviewer.aspx

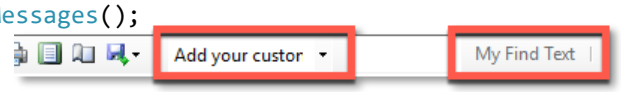
Report Viewer Custom UI Messages

If there is a need to change the UI messages and texts to use your own terminology, the interface *IReportViewerMessages*[7] can be derived to a class and added to a Report Viewer control:

```
public class CustomReportViewerMessages : IReportViewerMessages
{
    ….
    public string FindButtonText => "My Find Text";
    ….
    public string ZoomToPageWidth => "Add your custom text here.";
    public string ZoomToWholePage => null;
  ....
}
```

CLASS THE DERIVES FROM IREPORTVIEWERMESSAGES

```
var customMessages = new CustomReportViewerMessages();
reportViewer1.Messages = customMessages;
```



USAGE OF A CLASS THAT IS DERIVED FROM IREPORTVIEWERMESSAGES

Note – To keep the default text of specific text instances, even if the IReportViewerMessages are used, null can be returned in the derived class instead of a custom text.

---

[7] https://msdn.microsoft.com/en-us/library/microsoft.reporting.winforms.ireportviewermessages.aspx

Passing Data to the Report Viewer

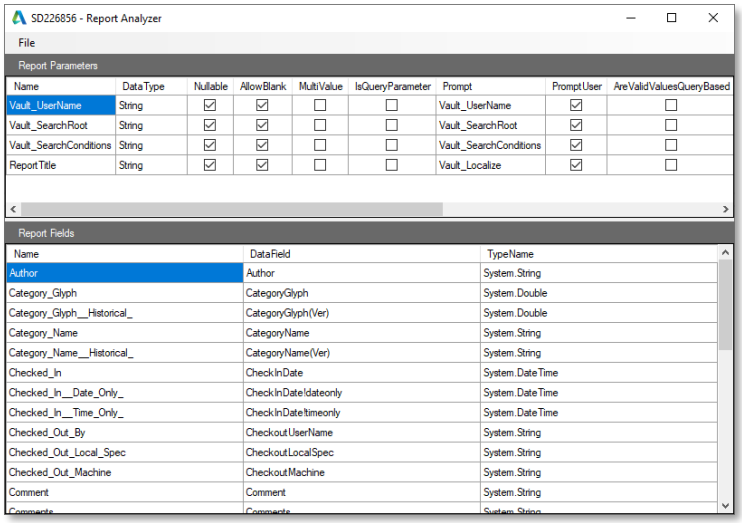
Learning Objective

Learn how to programmatically query report data and pass it to a report engine

LocalReport

A *LocalReport*[8] represents the report that is shown in the Report Viewer control and thus needs to have information about the Report Template (RDLC), the data source and the parameters.

Note – RDLC files are XML based files and thus information about Parameters and Fields can be extracted programmatically:



An example of this is the Report Analyzer application that is part of the sample materials (SD226856.RdlcAnalyzer).

---

[8] https://msdn.microsoft.com/en-us/library/microsoft.reporting.winforms.reportviewer.localreport.aspx

To load a Report Template file to a local report, a *Stream* or a *StringReader* that contains the local file content needs to be passed to the function *LoadReportDefinition()* of the LocalReport:

```
var rdlc = @"C:\Temp\ReportTemplate.rdlc";
var xmlDocument = new XmlDocument();
xmlDocument.Load(rdlc);
using (var stringReader = new StringReader(xmlDocument.OuterXml))
{
    reportViewer1.LocalReport.LoadReportDefinition(stringReader);
    stringReader.Close();
}
```

LOADING A RDLC REPORT TEMPLATE TO THE LOCALREPORT

Table and Columns

Before a LocalReport can be filled with data, this data needs to be queried from Vault or any other data source and then added to a *DataRow* objects of a *System.Data.DataTable*.

In order to create rows, the table must know about the columns:

```
var table = new DataTable("AutodeskVault_ReportDataSource");

table.BeginInit();
table.Columns.Add(new DataColumn("Author", typeof(string)));
table.Columns.Add(new DataColumn("Description", typeof(string)));
table.Columns.Add(new DataColumn("CheckInDate", typeof(DateTime)));
...
table.EndInit();
```

CREATING A SYSTEM.DATA.DATATABLE AND ADD DATACOLUMNS

Field Types

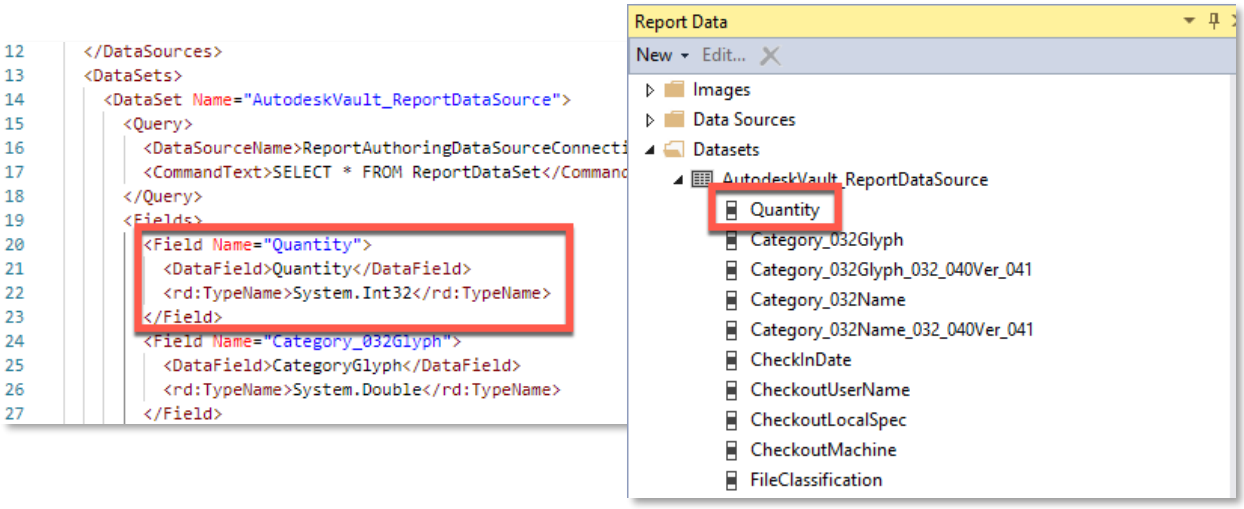
If the data is queried from Vault, the following mapping applies to use the correct data types:

Vault Data Type	.NET Type
DataType.String	System.String
DataType.Numeric	System.Double
DataType.Bool	System.Byte
DataType.DateTime	System.DateTime
DataType.Image	System.String (Base64)

DATA TYPE MAPPING BETWEEN VAULT- AND .NET-TYPES

External Fields

Fields that are not filled with Vault data but with data from external data sources must be introduces manually to the Report Template file. Therefore, the XML based RDLC file has to be opened with a Text- or XML-Editor and the field has to be added manually:

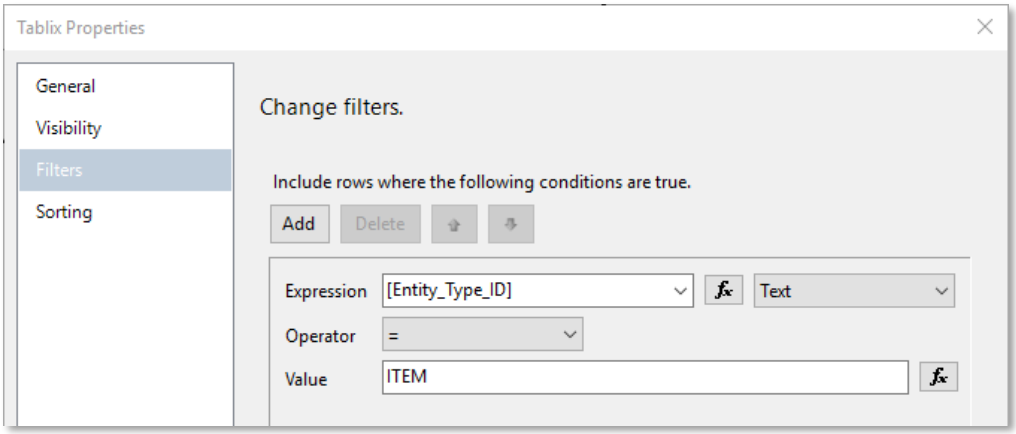


ADDING A FIELD MANUALLY

After reopening the RDLC file in Visual Studio, the new field will be visible in the Report Data window.

Obligatory Fields

If a Report Template is related to a specific Vault entity (like most of the templates shipped with Vault), two columns must be added to the table because the Report Template filters out all rows that doesn't have a specific value.



FILTER IN A VAULT REPORT TEMPLATE

```
<Field Name="Entity_Type">
    <DataField>EntityType</DataField>
    <rd:TypeName>System.String</rd:TypeName>
</Field>
<Field Name="Entity_Type_ID">
    <DataField>EntityTypeID</DataField>
    <rd:TypeName>System.String</rd:TypeName>
</Field>
```

ENTITY TYPE DEFINITIONS IN A VAULT REPORT TEMPLATE

To automatically fill the cells of each row with that specific value, a *DefaultValue* can be set when creating the *DataColumn*:

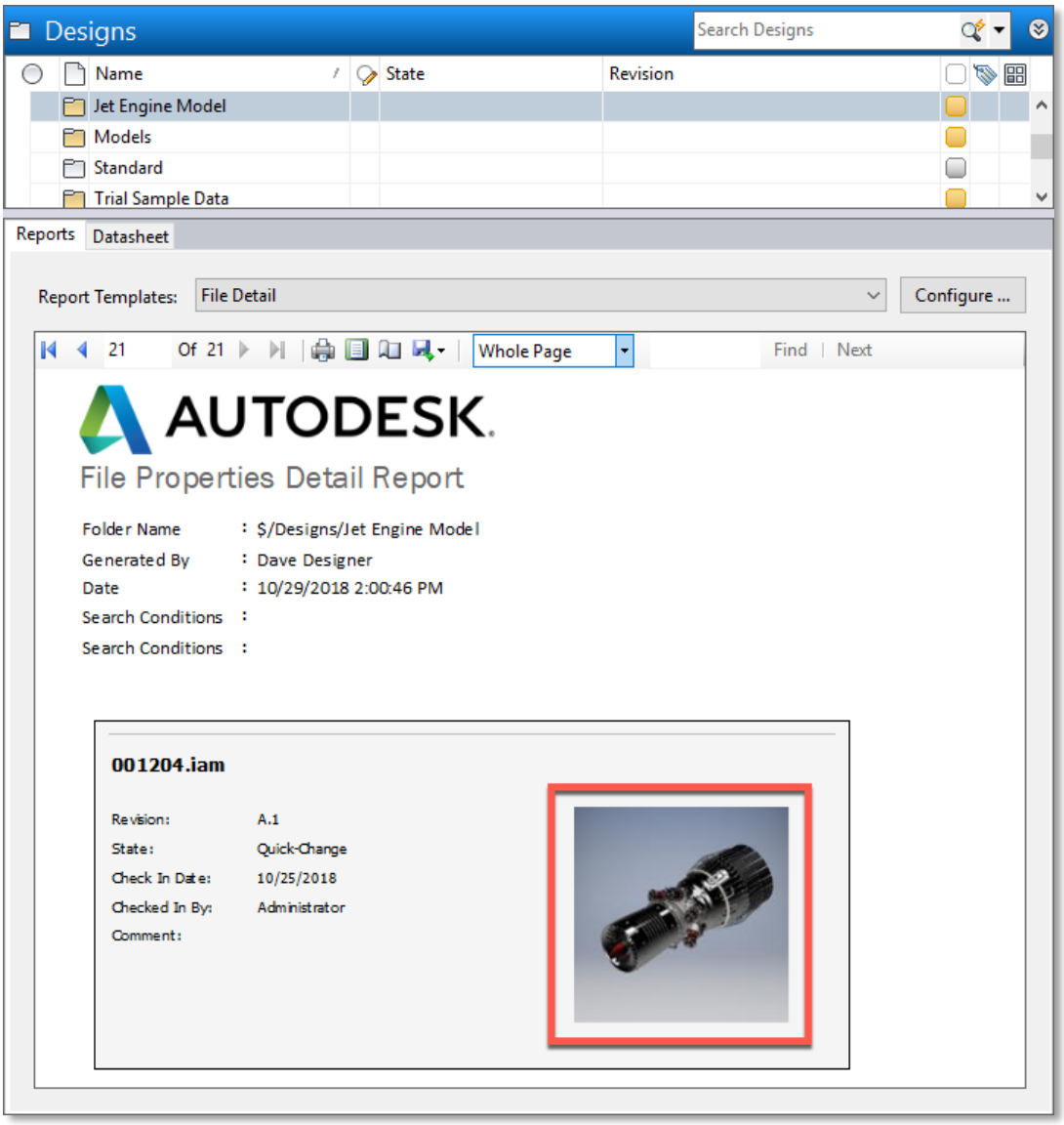
```
var colEntityType = new DataColumn("EntityType", typeof(string))
    { DefaultValue = "File" };
table.Columns.Add(colEntityType);

var colEntityTypeId = new DataColumn("EntityTypeID", typeof(string))
    { DefaultValue = "FILE" };
table.Columns.Add(colEntityTypeId);
```

SETTING DEFAULTVAULE OF A SYSTEM.DATA.DATACOLUMN

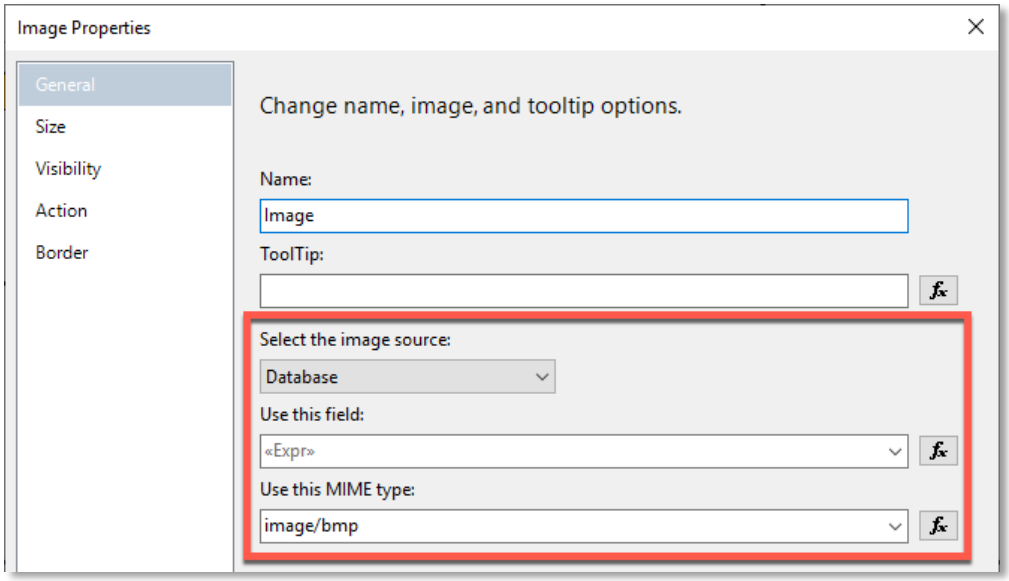
Images

Each row of a report not only can show texts, numbers and dates but also images. One example is a thumbnail of a Vault entity.



REPORT WITH AN IMAGE IN THE DATASOURCE

Because Vault passes the image as base64 string to the report, the settings of the images in the report must be as follows: The image source must be set to “Database”, the field must contain an specific expression (**`=Convert.FromBase64String(Fields!<FieldName>.Value)`**) and the MIME-Type must be set accordingly:



RDLC IMAGE PROPERTIES

When images are passed programmatically to a report, they must be converted to base64 strings:

```
string base64string;
using (var stream = new MemoryStream())
{
    image.Save(stream, ImageFormat.Bmp);
    base64string = Convert.ToBase64String(stream.ToArray());
}
```

IMAGE TO BASE64-STRING CONVERSION

Thumbnails in Vault are stored in UDPs as *byte[]* and can also be used:

```
var val = propDef.Typ == DataType.Image ? Convert.ToBase64String((byte[])propInst.Val)
    : propInst.Val;
```

VAULT-THUMBNAIL TO BASE64-STRING CONVERSION

Parameters

Parameters can be added, modified or removed from the RDLC file using Visual Studio.

To programmatically fill a report with parameters, an array of type ReportParameter must be passed to the *SetParameters()* method of the LocalReport:

```
var reportParameters = new List<ReportParameter>
{
    new ReportParameter("Vault_FolderName", folder.Name),
    new ReportParameter("Vault_UserName", connection.UserName),
    new ReportParameter("Vault_SearchRoot", folder.FullName),
    new ReportParameter("Vault_SearchConditions", "Current Selection"),
    new ReportParameter("ReportTitle", "File BOM – Parts Only")
};

reportManager.LocalReport.SetParameters(reportParameters.ToArray());
```

REPORT PARAMETERS

Custom Code

In the Report Properties, Custom Code can be added as VB (Visual Basic) functions. These functions are only available in the context of a single report (stored in the RDLC file) and are available in expressions with the prefix “Code”:

CUSTOM CODE IN REPORTS

ReportDataSource

Once filled with data, the DataTable can be converted to a *ReportDataSource*[9] that is finally passed to the LocalReport object:

```
var reportDataSource = new ReportDataSource("AutodeskVault_ReportDataSource", table);
reportViewer1.LocalReport.DataSources.Add(reportDataSource);
```

Note – The first argument of the ReportDataSource constructor must match the name of the DataSet of the Report Template file!

[9] https://msdn.microsoft.com/en-us/library/microsoft.reporting.winforms.reportdatasource.aspx

Automatically render reports to PDF

Learning Objective

Learn how to programmatically render reports and save them as PDFs

Instead of having a ReportViewer control hosting the LocalReport object, it can also be run independently. This way it can be used to programmatically create reports. These reports can be saved to neutral formats such as PDF or send to a printing device.

Independent LocalReports

A LocalReport object can be created using its constructor:

```
LocalReport localReport = new LocalReport();
```

CONSTRUCTION OF A LOCALREPORT

From then on, the local report can be filled with data just like the LocalReport that is hosted by a Report Viewer control.

Render Reports

Reports can programmatically be rendered to the following formats using different *Rendering Extensions*:

Rendering Extension	Format	File Extension
PDF	Portable Document Format Version 1.3 (Acrobat 4.x)	pdf
WORD	Microsoft Word (Legacy format)	doc
WORDOPENXML	Microsoft Word (Office Open XML format)	docx
EXCEL	Microsoft Excel (Legacy format)	xls
EXCELOPENXML	Microsoft Excel (Office Open XML format)	xlsx
IMAGE	Image (MIME-Type undocumented)	tif

RENDERING EXTENSIONS

All available Rendering Extensions can be listed using the method *ListRenderingExtensions()*:

```
localReport.ListRenderingExtensions();
```

LIST ALL RENDERING EXTENSIONS

Reports can be rendered to *byte[]* with the method *Render()*[10] of the LocalReport. This method has different overloads that can be used to pass different parameters:

```
var bytes = localReport.Render("PDF");
```

RENDER TO PDF WITH MINIMAL PARAMETERS

```
string mimeType, encoding, extension;
string[] streamids;
Warning[] warnings;

bytes = localReport.Render("EXCELOPENXML", null, out mimeType, out encoding,
    out extension, out streamids, out warnings);
```

RENDER TO XLSX WITH ADVANCED PARAMETERS

Note – In the sample material an example can be found how to send the different pages of a report to a printing device.

Export Reports

Once a report is rendered to a *byte[]*, it can be easily exported/save to the local drive:

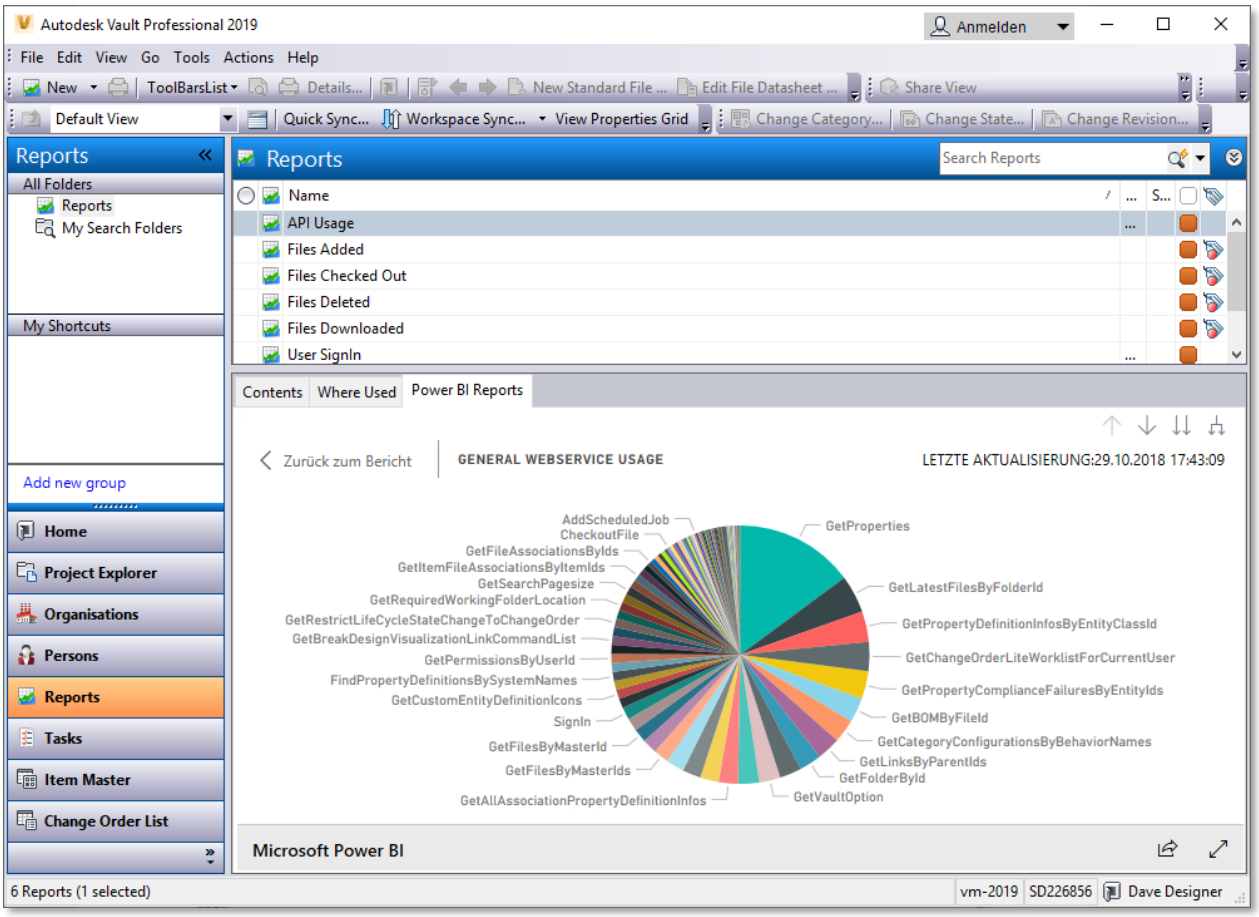
```
System.IO.File.WriteAllBytes(@"C:\Temp\Report.pdf", bytes);
```

WRITE BYTES TO FILE

[10] https://docs.microsoft.com/en-us/dotnet/api/microsoft.reporting.webforms.localreport.render

Alternatives

Instead of using Microsoft Report Viewer components, any other reporting components can be used in Vault Explorer extensions. One interesting alternative is *Microsoft Power BI*.



Microsoft Report Viewer vs. Microsoft Power BI

Learning Objective

Learn how to embed cloud-based reports to custom Vault Explorer extensions

Unlike RDLC which uses local data sources as input to reports, Power BI renders and provides reports from data that is only available in the cloud. This can be either an Azure SQL database, an Excel spreadsheet that was manually uploaded or a Power BI dataset that can be created and filled using the Power BI REST API.

Capture Vault Traffic

Because Power BI cannot directly use the data that is stored in Vault's SQL database, a mechanism must be established that pushes the necessary data to the Power BI database.

One example would be to capture every single Vault API call that arrives the Vault server and send a subset of this data to Power BI:

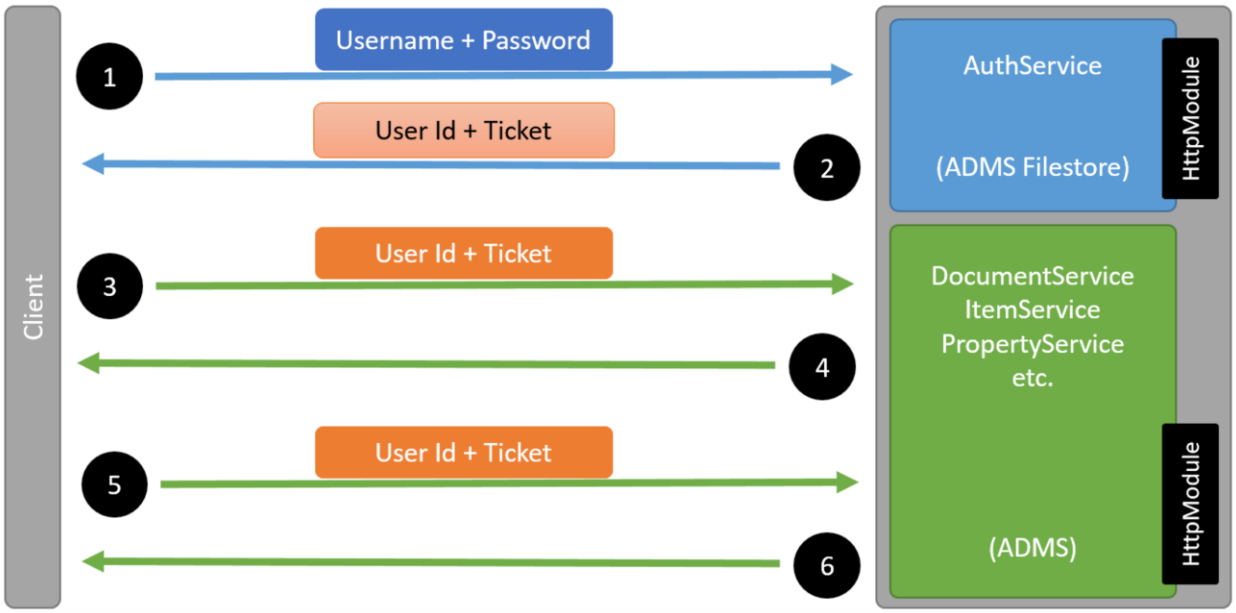
With this information a lot of statistical data is present that allows the creation of interesting reports about the usage of Vault.

HttpModule

To capture Vault server traffic, a *HttpModule*[11] can be used that is added to the IIS that hosts Vault Server. Using HttpModules, it's possible to extract data like

- Username
- Ticket
- Vault
- Service + Method
- Time

from every single request that arrives the Vault server. This can be realized by temporary saving and sharing requests and responses between two instances of the same HttpModule in the Vault Filestore and in ADMS:



AUTHENTICATION AND FURTHER VAULT REQUEST

[11] https://docs.microsoft.com/en-us/dotnet/api/system.web.ihttpmodule

The sample project *SD226856.PowerBIHttpModule* in the sample materials shows in an exemplary way how tracking the data and sending it to Microsoft Power BI can be realized.

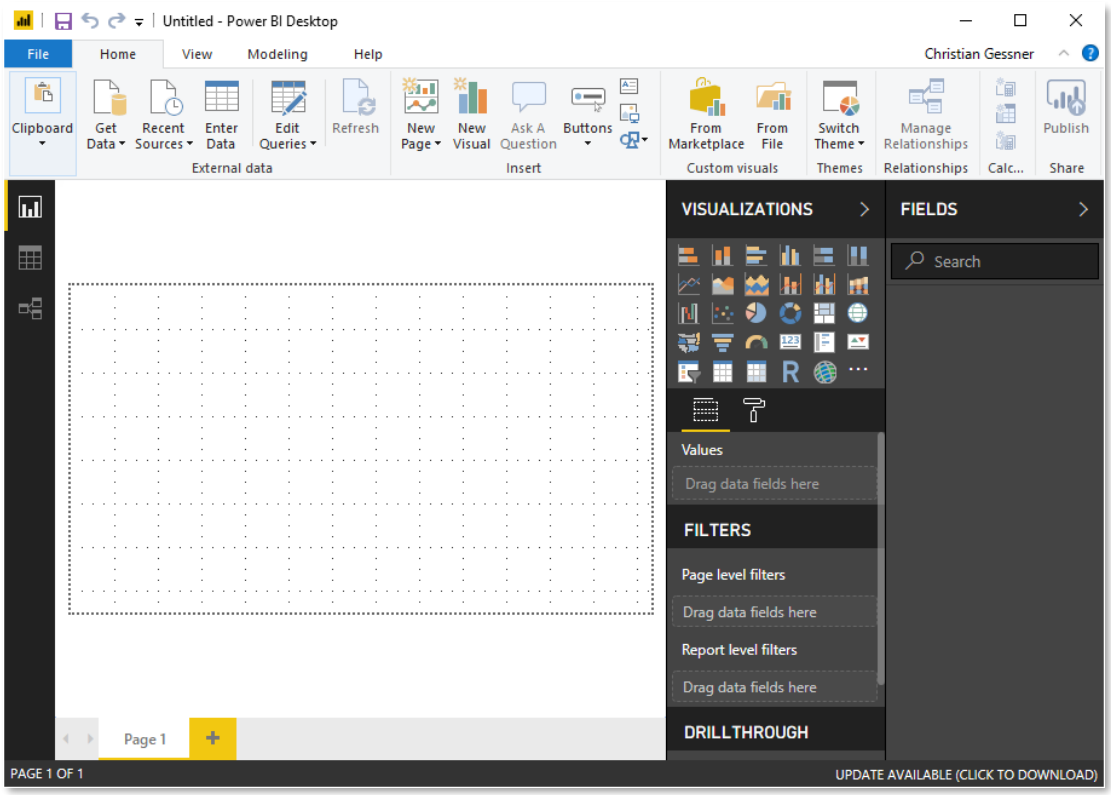
Note – More Information about IIS HttpModules can be taken from Microsoft:
https://msdn.microsoft.com/en-us/library/bb398986.aspx

Custom Event Handlers and Explorer Extensions

Instead of collecting all the traffic of the Vault server, the gathering of data can also be done on the client side using Vault Custom Event Handlers and Vault Explorer Extensions.

Creating Power BI Reports

The create reports using Power BI either the Power BI website can be used or the application Power BI Desktop:



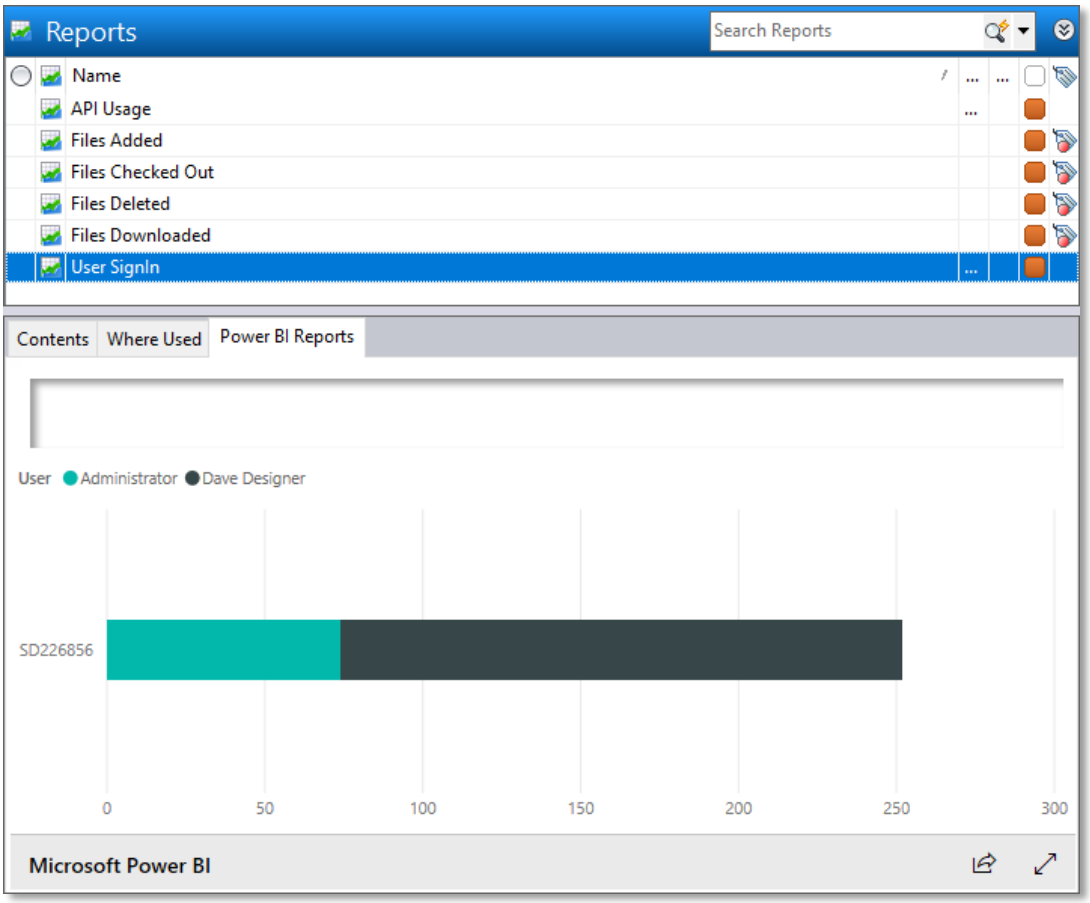
POWER BI DESKTOP

Showing Power BI Reports in Vault

Like with RDLC, Power BI Reports can be shown in a custom Vault tab with the only difference that the Power BI reports are hosted in a web browser control.

Note – Exporting Power BI reports is only possible with a valid Power BI license or within the trial period.

The example project *SD226856.PowerBIReportsExplorerExtension* uses different custom objects, each having an URL that stores the link to the exported Power BI Report. When the custom object is selected, the Power BI report is shown.



CUSTOM OBJECTS REPRESENTING DIFFERENT POWER BI REPORTS

Resources

AU Online Classes

- PL-29011: Dashboard, Smashboard – Vault Reporting on Steroids
  http://au.autodesk.com/au-online/classes-on-demand/class-catalog/classes/year-2016/vault-professional/pl20911#chapter=0

- PL6164: Analyze That: Vault Reporting and Monitoring
  http://au.autodesk.com/au-online/classes-on-demand/class-catalog/classes/year-2014/vault/pl6164

- DM5693: Creating Custom Autodesk Vault Report
  http://aucache.autodesk.com/au2011/sessions/5693/nov29_virtual_handouts/v1_DM5693%20-%20Creating%20Custom%20Autodesk%20Vault%20Report%20Templates.pdf

- SD124422: Vault Extensions Snorkeling—First Touch to Vault Extension and Automation Programming
  http://au.autodesk.com/au-online/classes-on-demand/class-catalog/classes/year-2017/vault-professional/sd124422#chapter=0

- MFG124959: Vault Extensions Deep Dive—Explore Vault Extension and Automation Programming
  http://au.autodesk.com/au-online/classes-on-demand/class-catalog/classes/year-2017/vault-professional/mfg124959#chapter=0

- SD11344: Programming the Vault with Vault Developer Framework
  http://au.autodesk.com/au-online/classes-on-demand/class-catalog/2015/vault/sd11344#chapter=0

Reporting and RDLC Links

- Autodesk Knowledge Network: Vault 2019 Custom Reporting - Training Material
  https://knowledge.autodesk.com/community/article/196571

- Autodesk Knowledge Network: Reports and Templates Administration
  https://knowledge.autodesk.com/support/vault-products/learn-explore/caas/CloudHelp/cloudhelp/Help/ENU/Vault/files/GUID-53ACB5F6-B6AA-4EFC-8223-1D0F9E1F249C-htm.html

- Plan for report design and report deployment | Microsoft Docs
  https://docs.microsoft.com/en-us/sql/reporting-services/plan-for-report-design-and-report-deployment-reporting-services?view=sql-server-2017

- ReportViewer Controls (Visual Studio)
  https://msdn.microsoft.com/library/ms251671.aspx

- Brian Schanen (Autodesk): Intro to Visual Data Management Configuration
  https://www.youtube.com/watch?v=7hSiIiyQD3w&list=PL6CBC15F0011C9C44

- Hagerman & Company: Reporting in Autodesk Vault
  https://www.youtube.com/watch?v=iIYa4oXDjAo

- It's All Just Onces And Zeros: BOM Report Job
  https://justonesandzeros.typepad.com/blog/2014/09/bom-report-job.html

- Markus Koechl (Autodesk): Item BOM Report Job
  https://github.com/koechlm/Vault-Sample---Item-BOM-Report-Job

- coolOrange Blog
  https://blog.coolorange.com/

Microsoft Power BI Links

- Power BI REST API documentation:
  https://docs.microsoft.com/en-us/rest/api/power-bi/

- Creating Power BI reports
  https://docs.microsoft.com/en-us/power-bi/service-report-create-new