How to customize VAULT Data Standard
Implementing National Standards

Robert Ostermann
Developer Methods Factory Design

Irvin Hayes
Product Manager
About the speaker

Robert Ostermann

Factory Designer at Magna Steyr Fahrzeugtechnik AG & Co KG since 2000.
Mechanical engineer for factory design and expert for factory models in different OEM projects
About the speaker

Irvin Hayes Jr

- 13+ years at Autodesk
- 20+ years of IT Support and technical leadership
- Work with key customers or prospects to identify unsolved problems
- Assess and investigate different technologies to solve given market problems
- Lead user experience and development teams
Integrated Factory Model – I.F.M. @ Magna

Part of Magna’s “Digital Factory” initiative

- Methods for DMS – Data Management Systems
- Methods for CAD
- Methods for “Factory Model Integration”
Key learning objectives

- Why Magna uses VAULT and Data Standard
- What are the benefits of VAULT Data Standard
- How Magna implemented and customized VAULT Data Standard
Why MAGNA uses VAULT and Data Standard

- Making models, plans and documents transparent and verifiable
- Reducing the time needed to search and find the correct data much more efficiently
- Closing the gap between the data and properties of a classic document management system
Why Magna uses Vault and Data Standard

Making models, plans and documents transparent and verifiable

Common properties in different systems, an example (3) of a flat foundation
Why Magna uses Vault and Data Standard

Making models, plans and documents transparent and verifiable

Common properties in different systems, an example (3) of jigs and fixtures, and tooling (robotics)
Why Magna uses Vault and Data Standard

Reducing the time needed to search and find the correct data much more efficiently

Properties (1) used to structure data in different ways (2)
Reducing the time needed to search and find the correct data much more efficiently

Example of linked Vault Data (1) of an old detailed drawing

“Factory Properties” (3)
Why Magna uses Vault and Data Standard

Closing the gap between the data and properties of a classic document management system

Example of different naming conventions (1) and inflexible structure (2)
What are the benefits of Vault Data Standard

• Increasing acceptance of data management usage
• Better user experience due to structured and flexible user interfaces
• Connecting to different external data property sources
• Implementing selection dependencies for properties
Increasing acceptance of data management usage

• Guiding users on how to get their data to Vault using the correct and required properties
• Making it easier and more efficient to enter data coming from different sources
• Helping users avoid mistakes while entering data or making wrong data combinations

• How to make peoples’ lives easier?
• How to increase efficiency in managing data?
Increasing acceptance of data management usage

Guiding users on how to get their data to Vault using the correct and required properties

Example of color code (1) and tooltip (2)
Increasing acceptance of data management usage

Making it easier and more efficient to enter data coming from different sources

<table>
<thead>
<tr>
<th>Building Resources (Gebäude-Ressourcen)</th>
<th>Level acc. to ÖNORM</th>
<th>Level acc. to MAGNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Structure (Bauwerk-Rohbau)</td>
<td>Level 1: 2</td>
<td>Level 2: 03</td>
</tr>
<tr>
<td>Foundations, Floor Constructions</td>
<td>Level 2: 2C</td>
<td>Level 3: 03</td>
</tr>
<tr>
<td>Shallow Foundations</td>
<td>Level 3: 2C.03</td>
<td>Level 4: 03</td>
</tr>
<tr>
<td>Single foundations, strip foundations,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foundation slabs, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of implementing a hierarchy and level based (1) (2) standard with dependent selection boxes
Increasing acceptance of data management usage

Helping users avoid mistakes while entering data or making wrong data combinations

Example of dependent selection (1) and dependent values (2)
Better user experience due to structured and flexible user interfaces

• Status quo using Vault’s common interfaces
• Giving users an interface to see all the information in an easier and more logically structured way
• Giving users an interface to edit all the information in an easier and more logically structured way

• How to make peoples’ lives easier?
• How to increase efficiency in managing data?
Better user experience due to structured and flexible user interfaces

Status quo using Vault’s common interfaces

Example (1) of long value lists

Example (2) of a “Property Edit” window (alphabetical order, no multiline text or text-wrap)
Better user experience due to structured and flexible user interfaces

Status quo using Vault’s common interfaces

Example of limited space in main pane (1), (2) and non-adjustable columns (3) with no text-wrap
Better user experience due to structured and flexible user interfaces

Giving users an interface to see all the information in an easier and more logically structured way

Magna’s solution for viewing all “Factory Properties” in a nice and tidy structured view (2) combined with Vault’s standard main pane (1)
Better user experience due to structured and flexible user interfaces

Giving users an interface to edit all the information in an easier and more logically structured way

Magna’s solution for editing “Factory Properties” (1) in a nice and tidy structured (2) edit user interface with value lists from external sources (3)
Connecting to different external data property sources

- Implementing a connection to external managed data, combined with an improved user experience

- How to make peoples’ lives easier?
- How to increase efficiency in managing data?
Connecting to different external data property sources

Implementing a connection to external managed data, combined with an improved user experience

- Data Management System
  - Data Properties – Values defined in Vault
    - Single Files – Upload and property modification
  - Data Properties – Values defined in different systems e.g. FM, ERP, PM,…
  - Multiple Files – Folder upload and property modification

Class learning objective - single file viewing and property modification (1) using internal and external data sources
Implementing selection dependencies for properties

- Properties update depending on users’ input to work more efficiently with fewer mistakes

- How to make peoples’ lives easier?
- How to increase efficiency in managing data?
Implementing selection dependencies for properties

Properties update depending on users’ input to work more efficiently with fewer mistakes

Value lists (2) change depending on previous selections (1)
How Magna implemented and customized Vault Data Standard

- Essential knowledge, tools and working environment
- General configuration, file structure and file purpose
- Methods for custom user interfaces – File info tab
- Methods for custom user interfaces – File edit tab
Essential knowledge, tools and working environment

- Blogs, Visual Studio and PowerShell

- Where to find the best information?
- What’s the best way to use the systems involved?
## Essential knowledge, tools and working environment

### Blogs, Visual Studio and PowerShell

1. **How to use Data Standard and Visual Studio:**

   - [coolOrange Blog: Editing Data Standard 2016 files with Visual Studio](#)
   - [AUTODESK Vault 2019 Help: Editing XAML Files with Visual Studio](#)

2. **How to use PowerShell:**

   - [Microsoft PowerShell: The Windows PowerShell ISE](#)

3. **How to use Data Standard and Visual Studio in combination with PowerShell:**

   - [AUTODESK KNOWLEDGE NETWORK: Autodesk Vault Data Standard – Configuration Editor](#)

4. **How to deploy Data Standard to teams:**

   - [AUTODESK APP STORE: Project Thunderdome for Autodesk Vault 2019](#)
General configuration, file structure and file purpose

- Overview systems and interfaces
- Data Standard installation
- Applied file structure
- Customized files, menu definition
- Customized files, user interface for file edit
- Customized files, user interface for file info

- Systems involved?
- How to setup?
- Where are the involved files?
- What’s the purpose of these files?
General configuration, file structure and file purpose

Overview systems and interfaces

- **Power Shell**
  - Scripting behavior for Data Standard

- **Data Standard**
  - Edit and review Factory Properties

- **VAULT** (VAULT-API)
  - Documentation & CAD-Data

- **XML Standard**
  - Structure and properties from other systems

- **XAML**
  - User interface for Data Standard
General configuration, file structure and file purpose

Data Standard Installation

Installation of Vault Data Standard
General configuration, file structure and file purpose

Applied file structure

Overview of the relevant customized files (1)
General configuration, file structure and file purpose

Applied file structure

Overview of the relevant customized files (1)
Customized files, menu definition

1. EditFaProps item definition (2), name of the function (3) and script (4) to execute XAML definition

PowerShell script to execute XAML:

```powershell
$vaultContext.ForceRefresh = $true
$fileId=$vaultContext.CurrentSelectionSet[0].Id
$dialog = $dsCommands.GetEditDialog($fileId)
$xamlFile = New-Object CreateObject.WPF.XamlFile "C:\ProgramData\Autodesk\Vault 2019\Extensions\DataStandard\Vault\Custom\Configuration\FAProps_EditMenu.xaml"
$dialog.XamlFile = $xamlFile
$dialog.Execute()```
General configuration, file structure and file purpose

Customized files, user interface for file edit

Location (1) of XAML definition for file edit user interface
General configuration, file structure and file purpose

Customized files, user interface for file info

Location (1) of XAML definition for file info window
Methods for custom user interfaces – File info tab

• Design elements for file info tab
• Data binding and codes for file info tab

• How to design a user interface?
• How to get property values into a user interface?
Methods for custom user interfaces – File info tab

Design elements for file info tab

Dynamic grid to adjust the window to screen demands:

<GridSplitter Grid.Column="1" Width="3" HorizontalAlignment="Stretch"/>

Combining information in groups:

<ScrollView VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" MinWidth="323"/>
    <ColumnDefinition Width="5"/>
    <ColumnDefinition Width="Auto"/>
  </Grid.ColumnDefinitions>
  <StackPanel>
    <GroupBox Header="Properties" Style="{StaticResource MS_groupBox}">
      <GroupBox Header="Preview" Style="{StaticResource MS_groupBox}">
        <GroupBox Header="Information" Style="{StaticResource MS_groupBox}">
          <GridSplitter Grid.Column="1" Width="3" HorizontalAlignment="Stretch"/>
        </GroupBox>
      </GroupBox>
    </GroupBox>
    <GridSplitter Grid.Column="1" Width="3" HorizontalAlignment="Stretch"/>
    <Image Width="80" Grid.Column="2" Source="C:\magna_logo.png" Height="15" Stretch="Uniform" HorizontalAlignment="Right" VerticalAlignment="Top" Margin="2,2,10,2"/>
    <StackPanel Grid.Column="2">
      <GroupBox Header="Properties" Style="{StaticResource MS_groupBox}">
        <GroupBox Header="Preview" Style="{StaticResource MS_groupBox}">
          <GroupBox Header="Information" Style="{StaticResource MS_groupBox}">
            <GridSplitter Grid.Column="1" Width="3" HorizontalAlignment="Stretch"/>
          </GroupBox>
        </GroupBox>
      </GroupBox>
    </StackPanel>
  </StackPanel>
</ScrollView>
Methods for custom user interfaces – File info tab

**Design elements for file info tab**

Groupboxes (1), labels (2) and textboxes (3) defined with style definitions

**Code for design elements (styles):**

```xml
<UserControl.Resources>
  <Style x:Key="MS_mainColumnWidth" TargetType="{x:Type ColumnDefinition}"
    Setter Property="Width" Value="120"/>
  <Style x:Key="MS_groupBox" TargetType="{x:Type GroupBox}"
    Setter Property="Foreground" Value="Gray"/>
    Setter Property="FontStyle" Value="Oblique"/>
    Setter Property="FontSize" Value="14"/>
    Setter Property="FontWeight" Value="Medium"/>
    Setter Property="Margin" Value="10"/>
    Setter Property="Padding" Value="5"/>
    Setter Property="Border.BorderBrush" Value="Gray"/>
  </Style>
  <Style x:Key="MS_textLabel" TargetType="{x:Type Label}"
    Setter Property="HorizontalAlignment" Value="Right"/>
    Setter Property="VerticalAlignment" Value="Center"/>
    Setter Property="FontStyle" Value="Normal"/>
    Setter Property="FontSize" Value="14"/>
    Setter Property="FontWeight" Value="Normal"/>
    Setter Property="Margin" Value="5,0,5,0"/>
  </Style>
  <Style x:Key="MS_infoTextBox" TargetType="{x:Type TextBox}"
    Setter Property="FontSize" Value="14"/>
    Setter Property="FontStyle" Value="Normal"/>
    Setter Property="FontWeight" Value="Normal"/>
    Setter Property="Background" Value="LightGray"/>
    Setter Property="IsReadOnly" Value="True"/>
    Setter Property="VerticalContentAlignment" Value="Center"/>
    Setter Property="TextWrapping" Value="Wrap"/>
    Setter Property="MinWidth" Value="150"/>
    Setter Property="Margin" Value="1,1,10,1"/>
    Setter Property="Padding" Value="2,0,0,0"/>
    Setter Property="ToolTip" Value="Dieses Feld wird automatisch befüllt."/>
  </Style>
</UserControl.Resources>
```
Methods for custom user interfaces – File info tab

Data binding and codes for file info tab

Standard data binding with Data Standard in XAML:

```xml
<TextBox x:Name="MSDescription" Text="{Binding Prop[MS_Description].Value}" Grid.Column="1" Style="{StaticResource MS_infoTextBox}"/>

function FAProps_Info
{
    try
    {
        $datetime = $PROP["MS_Document_Date"].Value
        $documentdate = [datetime]::ParseExact($datetime,'dd.MM.yyyy HH:mm:ss',$null).ToShortDateString()
    }
    catch
    {
    }
    $dsWindow.FindName("MSDocumentDate").Text = $documentdate
}
```

Enhanced data binding to change date-time string from a "dd.MM.yyyy HH:mm:ss" format to a “dd.MM.yyyy” format:

```xml
<TextBox x:Name="MSDocumentDate" Grid.Column="1" Style="{StaticResource MS_infoTextBox}"/>

function FAProps_Info
{
    try
    {
        $datetime = $PROP["MS_Document_Date"].Value
        $documentdate = [datetime]::ParseExact($datetime,'dd.MM.yyyy HH:mm:ss',$null).ToShortDateString()
    }
    catch
    {
    }
    $dsWindow.FindName("MSDocumentDate").Text = $documentdate
}
Methods for custom user interfaces – File info tab

Data binding and codes for file info tab

Additional function in Default.ps1 PowerShell script to update property when file selection in main pane has changed:

```powershell
#fired when the file selection changes
function OnTabContextChanged
{
    $xamlFile = [System.IO.Path]::GetFileName($VaultContext.UserControl.XamlFile)
    if ($VaultContext.SelectedObject.TypeId.SelectionContext -eq "FileMaster" -and $xamlFile -eq "FAProps_Info.xaml")
    {
        FAProps_Info
    }
}
```

Location of PowerShell scripts for enhanced data binding:

1. 
2. 
3. 
4. 
5.
Methods for custom user interfaces – File edit tab

• Design elements for file edit tab
• Data binding and codes for file edit tab

• How to design a user interface?
• How to get property values into a user interface?
Methods for custom user interfaces – File edit tab

Design elements for file edit tab

To improve the design experience, Microsoft provides special design-time attributes which can be added to any WPF:

1. 
2. 
3. 
4.

SizeToContent and WindowStartupLocation adjust your user interface optimal by any startup:

AUTODESK KNOWLEDGE NETWORK: DesignWidth and DesignHeight in Visual Studio
Methods for custom user interfaces – File edit tab

Design elements for file edit tab

Visually separate and structure property groups:

1. `<Border Style="{StaticResource MS_border}"/>

Logically combined properties in wrap panel:

2. `<WrapPanel Orientation="Horizontal" Width="Auto">
   <Grid x:Name="GridStackPanel_9" Style="{StaticResource MS_gridStackPanel}">
     <Grid.ColumnDefinitions…>
     <Label Content="Location" Style="{StaticResource MS_textLabel}"/>
     <ComboBox IsTextSearchEnabled="True" IsTextSearchCaseSensitive="True" Grid.Column="1" ItemsSource="{Binding Prop[MS_Location].ListValues}" Text="{Binding Prop[MS_Location].Value, UpdateSourceTrigger=PropertyChanged}" Style="{StaticResource MS_valueComboBox}"/>
   </Grid>
   <Grid x:Name="GridStackPanel_7" Style="{StaticResource MS_gridStackPanel}"…>
   <Grid x:Name="GridStackPanel_8" Style="{StaticResource MS_gridStackPanel}"…>
   </WrapPanel>

“Border” (1) to structure and “WrapPanel” (2) to enable optimal user interface resizing
Methods for custom user interfaces – File edit tab

Data binding and codes for file edit tab

User interface with standard data binding (1), binding for value lists (2) and binding for value lists with user search and input (3)

Single value typed in by the user:

```xml
<TextBox Text="{Binding Prop[MS_Description].Value}" Grid.Column="1" Style="{StaticResource MS_valueTextBox}"/>
```

Value lists for properties predefined in Vault:

```xml
<Grid x:Name="GridStackPanel">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Style="{StaticResource MS_mainColumnWidth}"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Label Content="Type" Style="{StaticResource MS_textLabel}"/>
    <ComboBox Grid.Column="1" ItemsSource="{Binding Prop[MS_Type].ListValues}" Text="{Binding Prop[MS_Type].Value, UpdateSourceTrigger=PropertyChanged}" Style="{StaticResource MS_valueComboBox}"/>
</Grid>
```

Value lists for properties predefined in Vault with case sensitive search and user input enabled:

```xml
<ComboBox IsTextSearchEnabled="True" IsTextSearchCaseSensitive="True" IsEditable="True" Grid.Column="1" ItemsSource="{Binding Prop[MS_Contractor].ListValues}" Text="{Binding Prop[MS_Contractor].Value, UpdateSourceTrigger=PropertyChanged}" Style="{StaticResource MS_valueComboBox}"/>
```
Methods for custom user interfaces – File edit tab

Data binding and codes for file edit tab

Data binding from XML resource – First selection level:

```xml
<Window.Resources>
  <XmlDataProvider x:Key="Classification"
      Source="C:\ProgramData\Autodesk\Vault2019\Extensions\DataStandard\Vault.Custom\MS_Resources.xml"/>
</Window.Resources>
```

XML data binding by user value selection (1) to a text box (2)
Methods for custom user interfaces – File edit tab

Data binding and codes for file edit tab

Data binding from XML resource – Second selection level:

Second level of XML data binding
Methods for custom user interfaces – File edit tab

Data binding and codes for file edit tab

Update value in text box (2) if user has made new selection (1):

\[
\text{function InitClassWindow} \\
\{
\text{SetCode} \\
\text{\$CB01=\$dsWindow.FindName("ComboLV01")} \\
\text{\$CB01.add\_SelectionChanged\{(ComboChanged("NCode01")\})} \\
\text{\$CB02=\$dsWindow.FindName("ComboLV02")} \\
\text{\$CB02.add\_SelectionChanged\{(ComboChanged("NCode02")\})} \\
\text{\$CB03=\$dsWindow.FindName("ComboLV03")} \\
\text{\$CB03.add\_SelectionChanged\{(ComboChanged("NCode03")\})} \\
\text{\$CB04=\$dsWindow.FindName("ComboLV04")} \\
\text{\$CB04.add\_SelectionChanged\{(ComboChanged("NCode04")\})} \\
\}
\]

\[
\text{function ComboChanged($NCode)} \\
\{
\text{\$dsWindow.FindName("OENorm").Text = \$dsWindow.FindName($NCode).Text} \\
\text{\$dsWindow.FindName("ShortCode").Text = \$dsWindow.FindName("Code01").Text + \\
\text{\$dsWindow.FindName("Code02").Text + "} + \$dsWindow.FindName("Code03").Text + \\
\text{\$dsWindow.FindName("Code04").Text} \\
\text{\$Category = \$dsWindow.FindName("categoryTextBox").Text} \\
\text{\$Prop["MS\_Category"].Value = \$Category} \\
\}
\]

"SelectionChanged" binding of "TextBox" (2)
Methods for custom user interfaces – File edit tab

Data binding and codes for file edit tab

```
function InitClassWindow
{
SetCode
$CB01=$dsWindow.FindName("ComboLV01")
$CB01.add_SelectionChanged({ComboChanged("NCode01")})
$CB02=$dsWindow.FindName("ComboLV02")
$CB02.add_SelectionChanged({ComboChanged("NCode02")})
$CB03=$dsWindow.FindName("ComboLV03")
$CB03.add_SelectionChanged({ComboChanged("NCode03")})
$CB04=$dsWindow.FindName("ComboLV04")
$CB04.add_SelectionChanged({ComboChanged("NCode04")})
}

function ComboChanged($NCode)
{
$dsWindow.FindName("OENorm").Text = $dsWindow.FindName($NCode).Text
$Category = $dsWindow.FindName("categoryTextBox").Text
$Prop["MS_Category"].Value = $Category
}
```

"SelectionChanged" binding of "TextBox" (2)

Location of PowerShell script (1) for enhanced data binding:

1. 
2.
Class summary

This story comes from Magna and demonstrates how we improved our daily work.

If this journey with Vault Data Standard inspires you - and there are many more things you can do regarding customization - you might be interested in some very good Autodesk University resources:

AUTODESK University: Into the Habit Hole - Advanced Data Standard Configuration and Customization
AUTODESK University: Data Standard introduction

Also, if you are searching for somebody to customize your ideas, we can help you. I'm very interested in sharing ideas, workflows and cooperating in the field of developing the Digital Factory.