# pure::variants Connector for PTC Integrity Manual

pure-systems GmbH

Version 5.0.8.685 for pure::variants 5.0



2021

# Table of Contents



# 1. Introduction

pure::variants Connector for PTC Integrity enables Integrity users to manage requirements variability using pure::variants. By coupling pure::variants and Integrity, knowledge about variability and variants can be formalized, shared and automatically evaluated. This enables getting answer for questions about valid combinations of requirements in product variants quickly, permits easy monitoring of planned and released product variants at the requirements level and also permits very efficient production of variant-specific requirements documents out of the requirements repository.

## 1.1. Software Requirements

The following software has to be present on the user's machine in order to support the pure::variants Connector for PTC Integrity:

| | |
|---|---|
| PTC Integrity Client: | PTC Integrity 10.2 or higher is required. Compatibility with other PTC Integrity releases is not guaranteed. |

The pure::variants Connector for PTC Integrity is an extension for pure::variants and is available on all supported platforms.

## 1.2. About this manual

The reader is expected to have basic knowledge about and experiences with both tools, PTC Integrity and pure::variants. Please consult their introductory material before reading this manual. This manual is available in online help as well as in printable PDF format here.

# 2. Installation

## 2.1. Setup of PTC Integrity

Please consult section **pure::variants Connectors** in the **pure::variants Setup Guide** for detailed information on how to install the connector (menu **Help** -> **Help Contents** and then **pure::variants Setup Guide** -> **pure::variants Connectors**).

## 2.2. Setup of pure::variants Integration for PTC Integrity

Please consult section **pure::variants Integrations** in the **pure::variants Setup Guide** for detailed information on how to install the connector (menu **Help** -> **Help Contents** and then **pure::variants Setup Guide** -> **pure::variants Integrations**).

# 3. Using the Connector

## 3.1. Starting PTC Integrity Client

Start the PTC Integrity client before using the pure::variants Connector for PTC Integrity and login with username and password. pure::variants will use the client to access Integrity data.

## 3.2. Running pure::variants

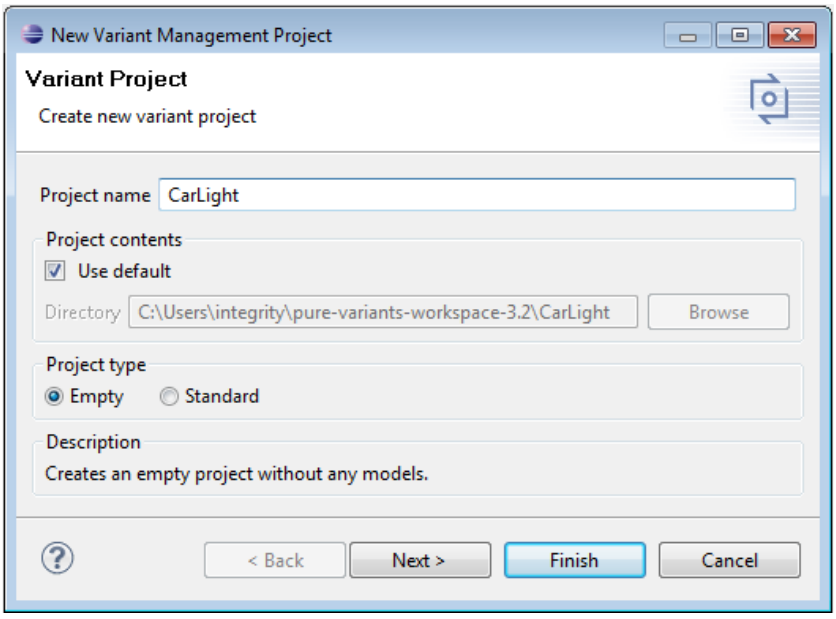
Depending on the installation method used, either start the pure::variants-enabled Eclipse or, on Windows, select the **pure::variants** item from the **Start** menu.

If the **Variant Management** perspective is not already activated, do so by selecting it from menu **Window** -> **Open Perspective** -> **Other...**.

## 3.3. Creating Feature Models

The first step is always to create corresponding feature models for each relevant Integrity document. These initial feature models serve as starting points for using existing variability information or adding variability information to the requirements. First of all a Variants project has to be created, where all models will be stored. Select **Project** in menu **File** -> **New**. Choose **Variant Projects** in section **Variant Management** on the first page of the wizard. Choose a name for the project and select **Empty** as project type.

**Figure 1. Creating a Variant Management project**



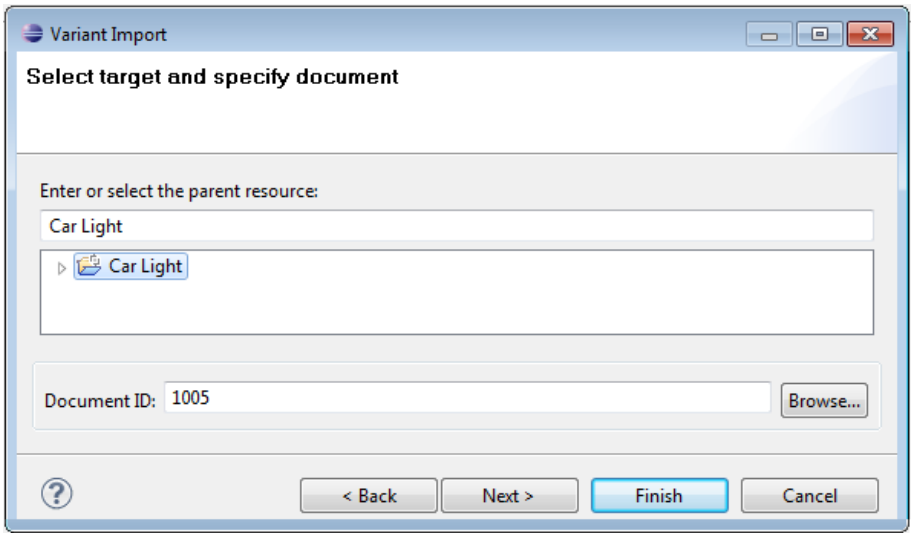
After creating the project a feature model can be created. Select **Other** in menu **File** -> **New**. Choose **Feature Model** from section **Variant Management** on the first page of the wizard. Enter a name for the model on the second page.

## 3.4. Importing Requirement Documents

The import procedure needs to be executed **only once** for each Integrity document. Each document is represented by one pure::variants family model. Import is started by selecting the **Import** action from the context menu of the **Projects** view or from menu **File**. Select **Variant Models or Projects** and click **Next**. On the following page select **Import PTC Integrity Document**.

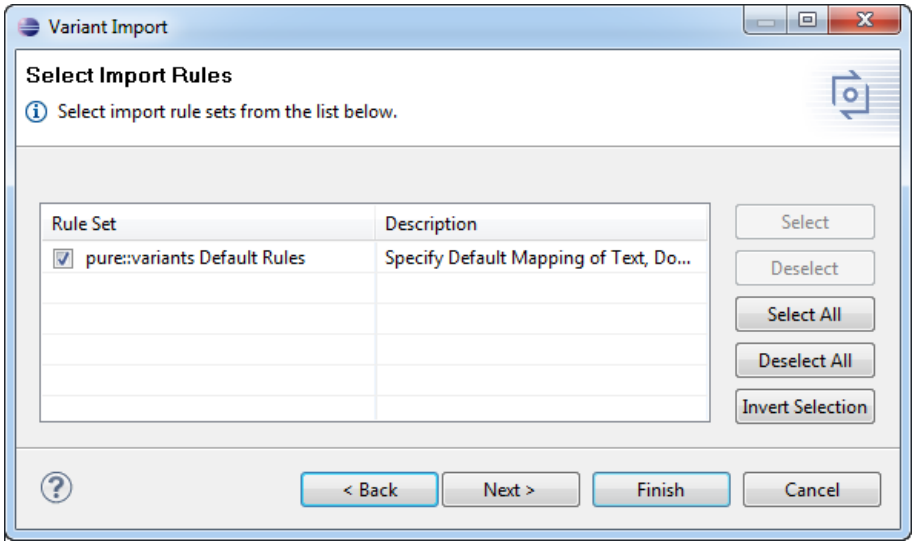
The import wizard appears. On the first page you can enter a model name and you have to choose the requirements document to be imported.

**Figure 2. Document selection page**



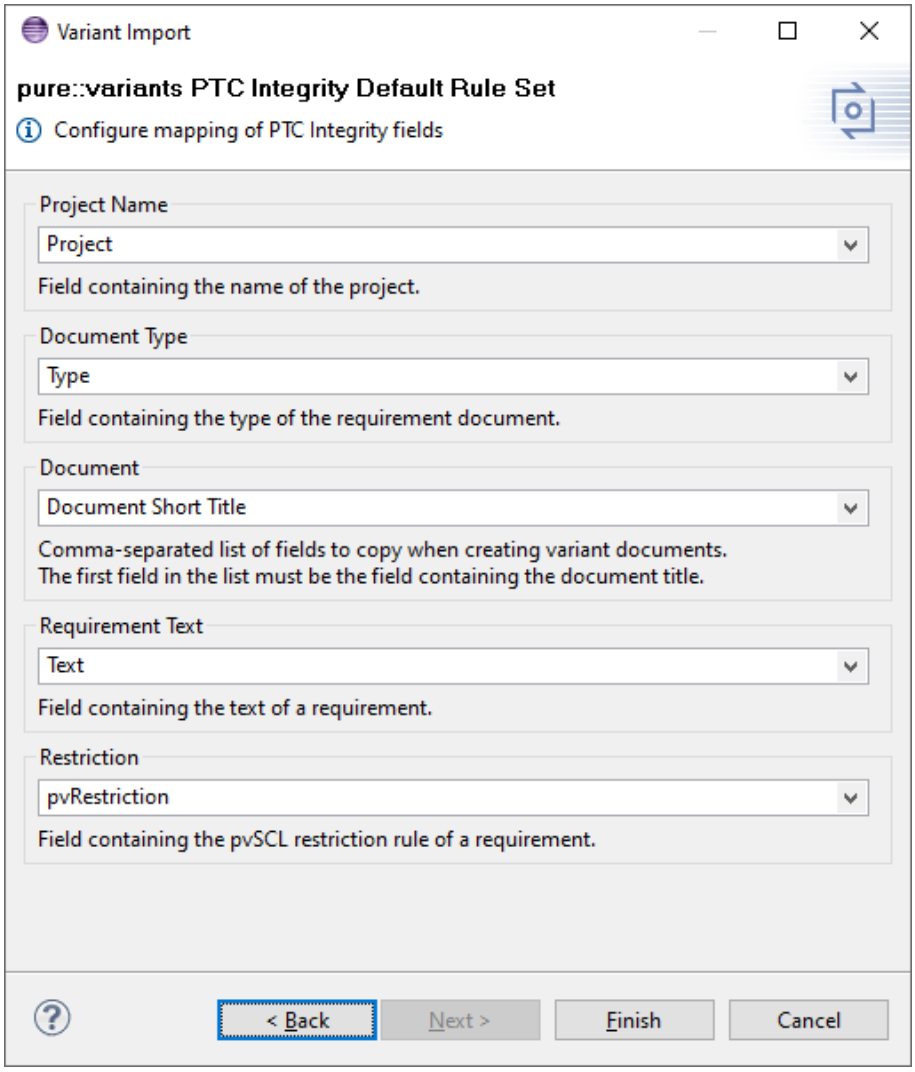

Click **Next** to bring the **Import Rules** page up. On this page you can select sets of import rules which will be used to manipulate the resulting model after import. Import rule sets can be used to create specific pure::variants model elements like restrictions from Integrity document information.

**Figure 3. Import Rule Sets page**



The last pages contain the settings of the selected Import Rule Sets. For the pure::variants Default Import Rules you can choose which attribute value will be used for creating a variant document title, element visible names and restrictions.

**Figure 4. pure::variants default rule set settings page**

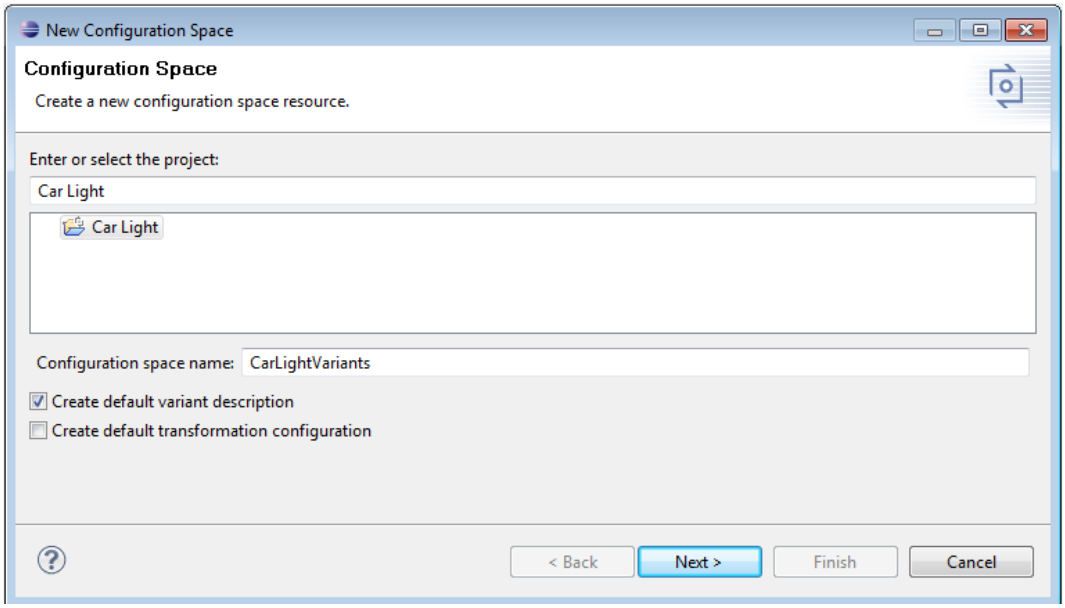



## 3.5. Defining a Variant

The next step is the definition of the actual variants of interest. Since the variability model usually permits the definition of a very large number of variants, pure::variants keeps track only of those variants which are of interest for the user. Typically this number is much lower than the number of possible variants.

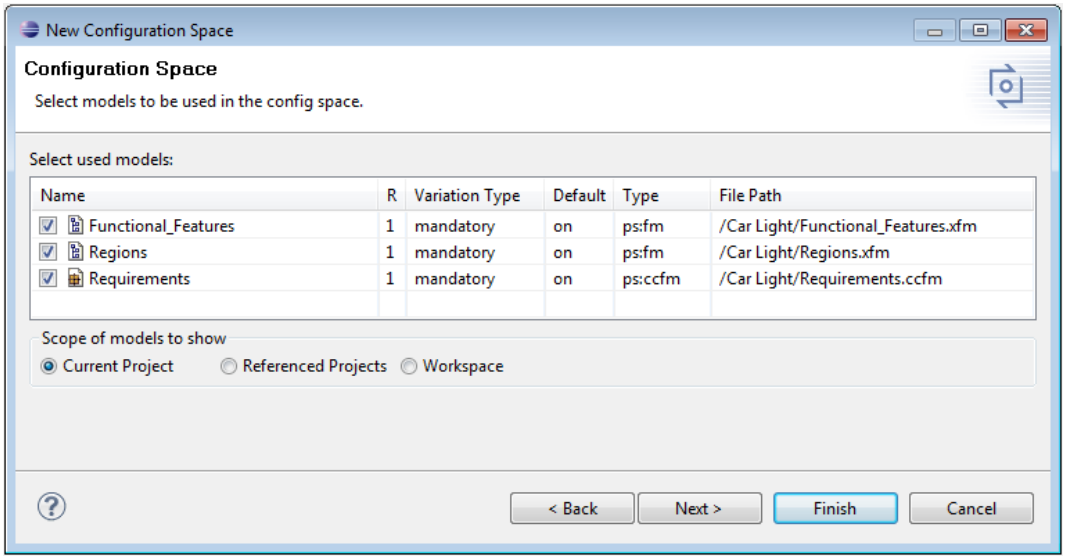
Variants are stored as separate entities called *Variant Description Models* (VDM). A VDM always belongs to a specific *Configuration Space*. Thus, before defining variants, a configuration space has to be created. Select the project containing the feature models and imported family models in the Variant Projects view and open the context menu. Click **New** -> **Configuration Space**. A wizard opens. On the first page enter a name for the configuration space. The name has to follow strict rules (no spaces, no special characters). Uncheck the box before **Create standard transformation** since for pure requirements models the standard transformation does not provide any relevant functionality (See the pure::variants User Manual for more information on transformations).

**Figure 5. The Configuration Space Wizard, page 1**



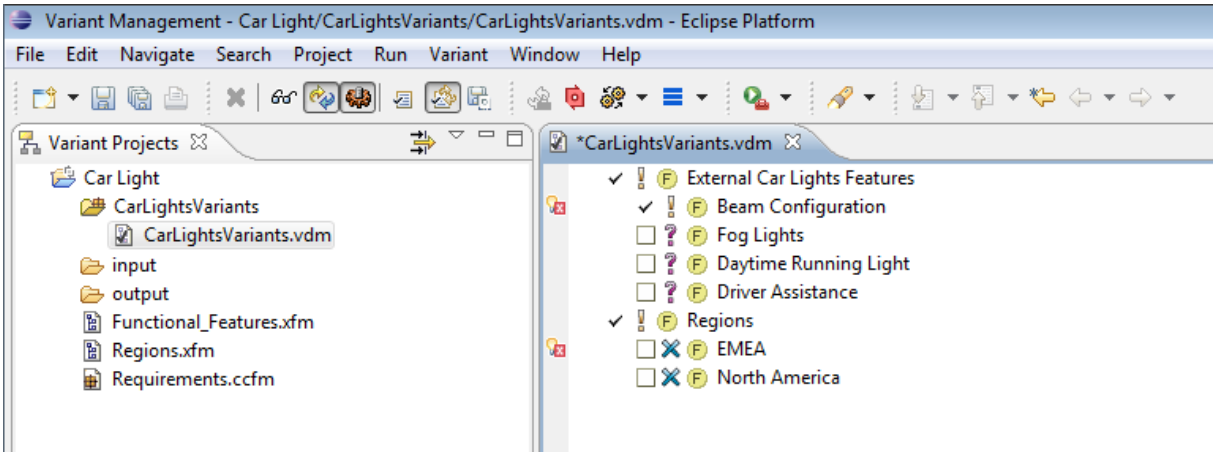

The next page is used to specify which feature and family models are to be included in this configuration space. Select here all models that represent the Integrity documents of interest. In the example below all models are selected. Now click the **Finish** button.

**Figure 6. The Configuration Space Wizard, page 2**




The resulting project structure already contains the file `CarLightsVariants.vdm` which is immediately created and opened. It resembles the structure of the previously defined models, but has a checkbox in front of each feature to permit the user to select features for this variant by clicking on it. The buttons marked in the toolbar control the evaluation of configurations. The left-most button (  ) initiates a manual check of the variant configuration. The middle button (  ) toggles between manually checking and automatic checking after changes to the VDM. Finally the right-most button (  ) toggles the auto-resolver on or off. The auto-resolver provides automatic resolution of configuration problems where possible.

**Figure 7. Initial Configuration Space Structure**




Problems are indicated by pure::variants during the selection of features. There are several places where problems are shown. The Problems view (usually located in the lower right part of the pure::variants perspective) lists all problems such as incompatible elements or a missing selection from *alternative* features. In addition, problems are shown in the model editor directly in front of the element causing the problem. A tool-tip (which can be seen by moving the mouse over the icon) explains the problem, and the context menu for the problem (right mouse button) provides possible fixes for the problem. E.g. for conflicting elements the fix is to deselect either one or the other feature.

Each variant can be represented in its own VDM. To create a new VDM, either select **Clone** from the context menu (in Variant Project view) of an existing variant or use **New** -> **Variant Model** in the context menu of the configuration space. When a valid variant is configured, it can be stored and exported to Integrity. The next section explains this in detail.

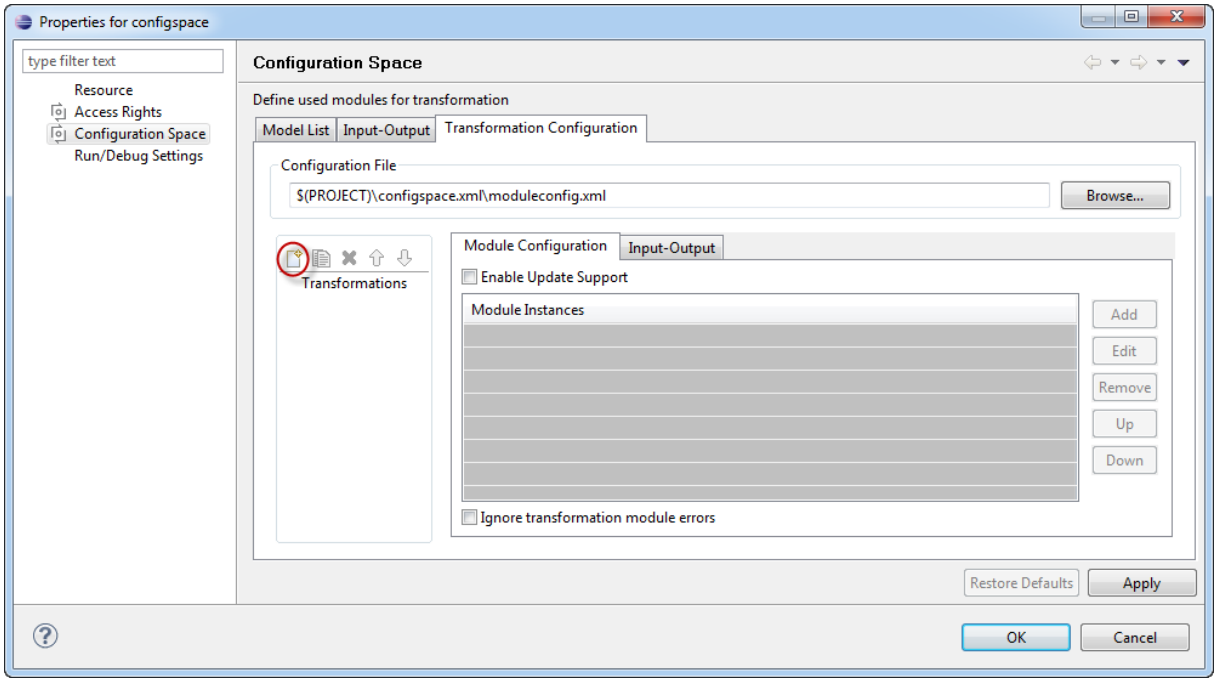
## 3.6. Exporting a Variant to Integrity

Variants stored in a variant description model can be made available in Integrity by means of an export. To export variants to Integrity first a *Transformation Configuration* has to be created. To create a configuration click on **Transformation** button in the tool bar and choose *Open Transformation Config Dialog*.

**Figure 8. Transform Model Button**

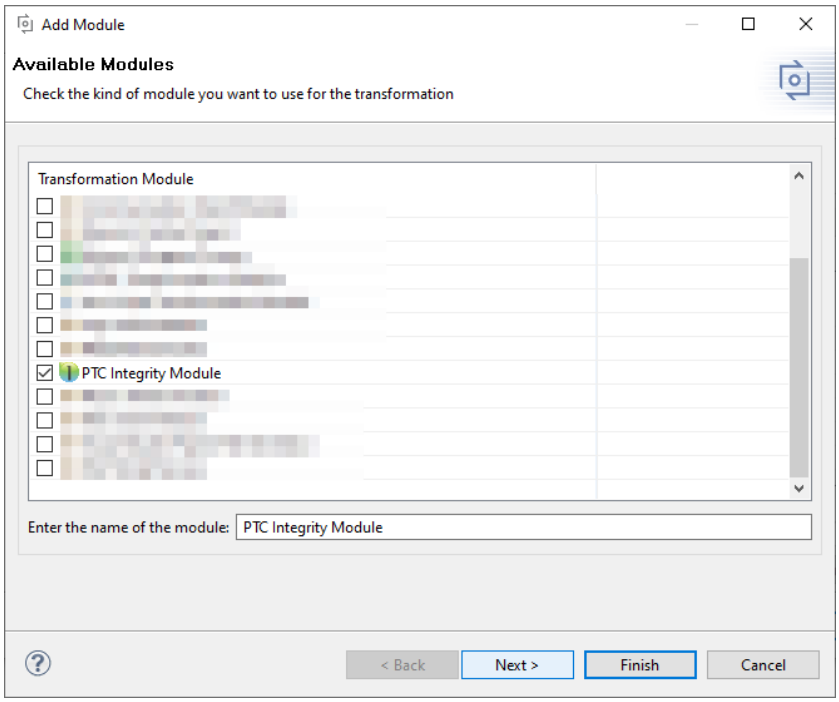
The configuration space property dialog opens and the *Transformation Configuration* tab is shown. Next step is to add a new *Module Configuration* by clicking the marked tool bar item. Now add a new Module to the Module Configuration using the **Add** button.

**Figure 9. Transformation Configuration**



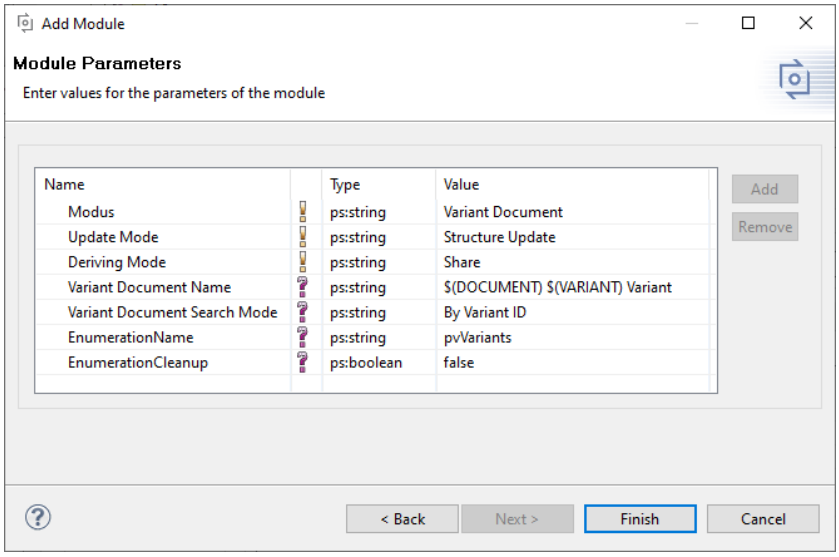

In the dialog that opens, choose **PTC Integrity Module** and enter a name.

**Figure 10. Module Selection Page**




Click **Next** to open the next page listing the parameters of the transformation module.

**Figure 11. Module Parameter Page**




The transformation module supports two modes which can be switched by parameter **Modus**: Variant Document and Variant Enumeration.

## Variant Document Mode

In **Variant Document** mode, a new document is created in Integrity based on the original requirement document but containing only requirements which are part of the variant.

This mode has the following configuration parameters.

Update Mode

If set to **Structure Update** (Default), then a previously created variant requirement document is updated rather than creating a new document. Requirements are added to the variant requirements document which are part of the variant, and requirements not anymore part of the variant are removed. If there is no matching variant requirement document, a new document is created.

If set to **New Instance**, then always a new variant requirement document is created, regardless if there has been one created for this variant before.

Deriving Mode

If set to **Share** (Default), all requirements are shared copies of the branched document and can not be changed afterwards, but changes on the original requirements are reflected.

If set to **Reuse**, all requirements are physical copies and can be changed independently of the original requirements afterwards.

Variant Document Name

The name pattern for variant requirement documents. The default variant document name is a concatenation of the name of the original document followed by the name of the variant, followed by the word Variant.

The name pattern can contain any of the pure::variants predefined variables, such as `$(VARIANT)`, `$(QUALIFIER)` and `$(ENV:variable)`, and additionally the variable `$(DOCUMENT)` resolving to the name of the original document. Please refer to the pure::variants User's Guide for a complete list of the predefined variables.

Variant Document Search Mode

If set to **By Variant ID** (Default), then variant documents are searched by comparing the ID of the variant with the stored variant ID of the documents in Integrity. This is the recommended mode.

If set to **By Document Title**, then variant documents are searched by comparing the title of the documents in Integrity. This mode uses the **contains** text comparison operator with keyword **LIKE** and thus has only limited support for special characters in document titles.

### Variant Enumeration Mode

In **Variant Enumeration** mode, the requirements in the master requirement document will be annotated with the name of the variant if they are part of this variant.

This mode has the following configuration parameters.

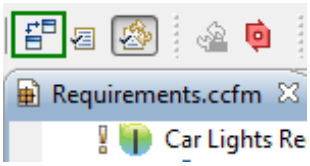
| | |
|---|---|
| EnumerationName | The name of the field used on requirements to store the variants they are part of. |
| EnumerationCleanup | If set to **true**, then all variant names which are not part of the current transformation are removed from the variant enumeration field on each requirement of the requirements document. |

After finishing the configuration, the transformation can be started by clicking on the **Transform Model** button in the tool bar, or choosing the transformation in the pull down menu of this button.

## 3.7. Updating Model from Integrity

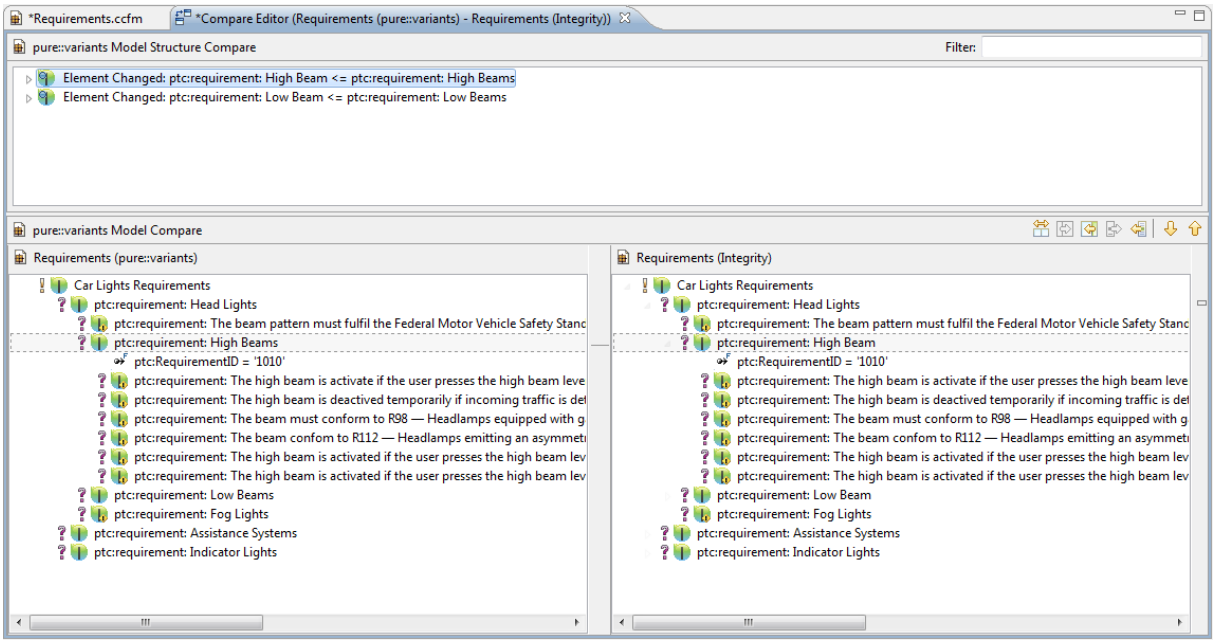
Since there is no live connection between the Integrity database and pure::variants, it is necessary to update the pure::variants family models with information from Integrity whenever relevant changes have been made. To facilitate the synchronization, pure::variants provides a **Synchronize** action. To start the update, open the model representing the Integrity document and click the **Synchronize** button in the tool bar. After answering the usual connection information dialog, pure::variants will connect to Integrity client and present the so-called **Compare Editor** for pure::variants models.

**Figure 12. Synchronize Model Button**




The compare editor is used throughout pure::variants to compare model versions. In this case it is used to compare the Integrity data (displayed in the lower right side) with the current pure::variants model (lower left side). All changes are listed as separate items in the upper part of the editor, ordered by the affected elements. Selecting an item in this list highlights the respective change in both models. In the example, the changed attribute values are marked with boxes and connected to their respective counterparts in the other model.

**Figure 13. Model update from Integrity in Compare Editor**




The merge toolbar provides tools to copy single or all (non-conflicting) changes from the Integrity document to the family model.

# 4. Using the Integration

To support users of the pure::variants Connector for PTC Integrity in adding variability information to Integrity documents, the pure::variants Integration is provided. It can be used to define element restrictions for the currently selected requirement. It provides an editor featuring auto completion, syntax highlighting and error check.

The instructions for installing the Integration can be found in Section 2.2, “Setup of pure::variants Integration for PTC Integrity”.

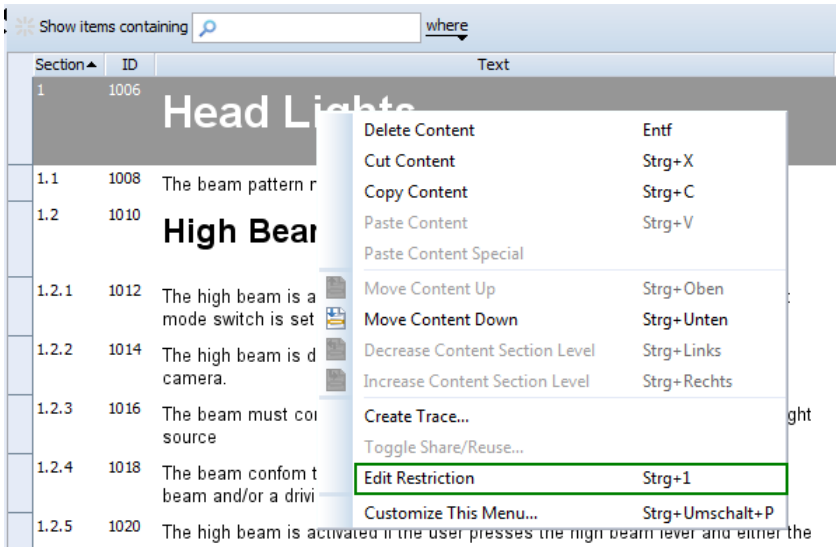
## 4.1. Starting PTC Integrity Client

Start the Integrity client with the required Integrity user name and password. The pure::variants Integration will use the client to access Integrity data, so the user must have sufficient access to the relevant data in the Integrity database.

## 4.2. Opening pure::variants Integration

To start the Integration, first open an Integrity document and select the requirement for which you want to define a restriction. Select the context menu item **Edit Restriction**, or use menu item **Edit Restriction** in menu **Custom**, or click the appropriate custom button (see ???). Now the pure::variants integration opens and the restriction editor is displayed.

**Figure 14. Edit Restriction**




## 4.3. First Use

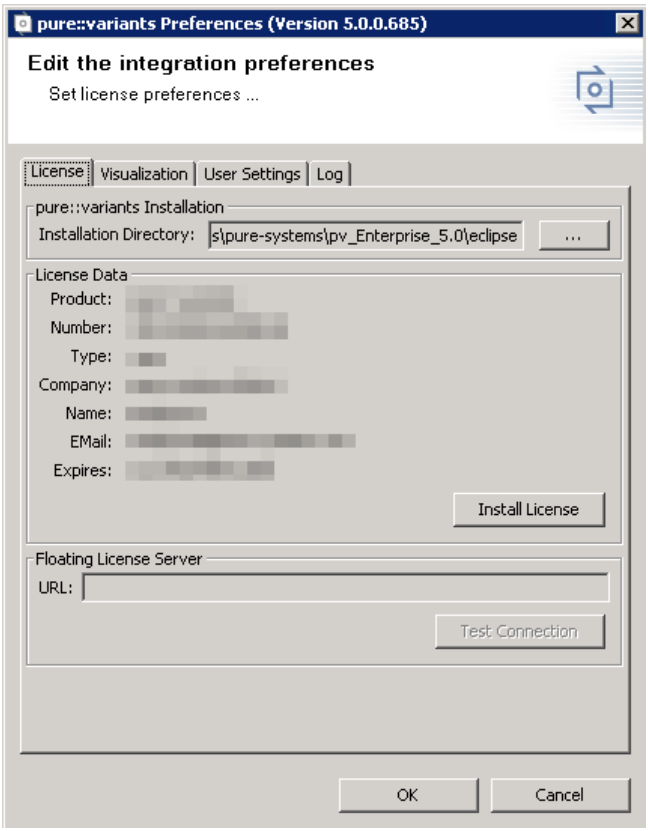
When you first use the Integration after installation, it is necessary to check whether the license preferences are correct. For this purpose open the Integration preferences dialog via the button in the Integration window.

A dialog opens that shows the path to your pure::variants installation and your license information (see Figure 15, “Preferences Dialog”). If any of the information is missing, you need to enter it. Use the **...** button in the **pure::variants Installation** group to enter the installation directory, and the **Install License** button to specify your license.

If you are using a floating license and the URL in the **Floating License Server** group is not set already, you need to enter the URL. To test if the connection to the floating license server can be established, click the button **Test Connection**.

Now you can use the Integration.

**Figure 15. Preferences Dialog**




## 4.4. Editing Variability Information

### Connecting with pure::variants Models

For editing variability information and viewing visualizations, it is necessary to connect your Integrity project with one or more pure::variants models. The following types of pure::variants models can be loaded:

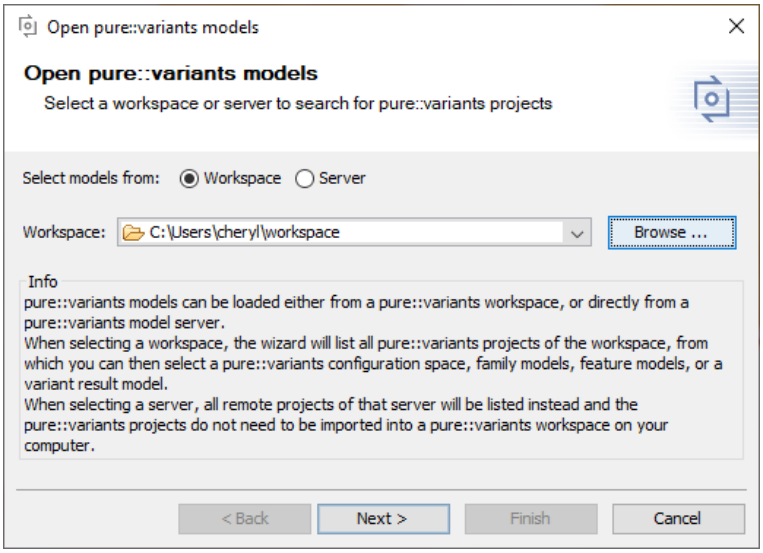
- Recommended: pure::variants configuration spaces, which enable selection of contained variant description models (.vdm)
- pure::variants variant result models (.vrm)
- pure::variants feature models (.xfm)
- pure::variants family models (.ccfm)

pure::variants models can be opened from two different sources: Either from a *pure::variants/Eclipse workspace* or from a *pure::variants model server*.

#### Opening models from a workspace

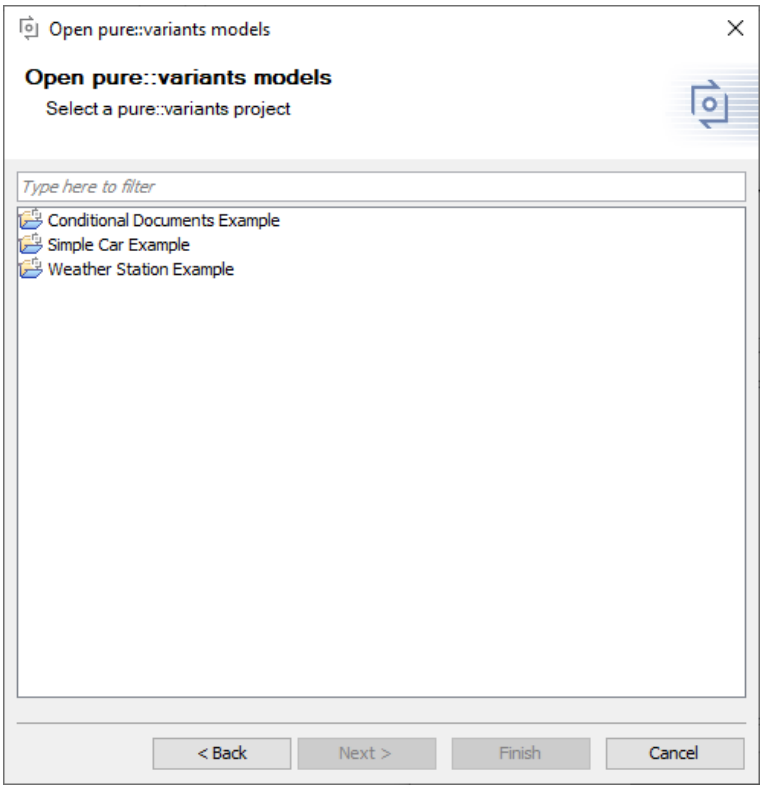
To open a model or configuration space, press on the Integration window. This will open a wizard, which first allows choosing the source (workspace or server). Choose *workspace* and then browse to find your pure::variants workspace folder. Already known workspaces are listed in the workspace dropdown box. If you later need to add or remove a workspace from the list, you can go to tab *User Settings* of the Integration preferences (accessible via ).

**Figure 16. Mode selection page**



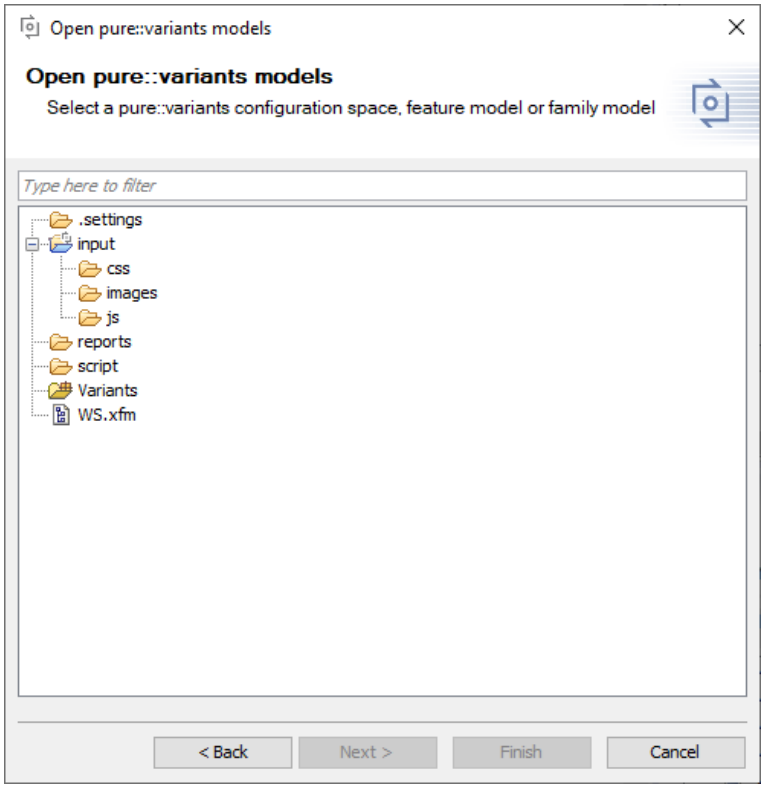

On the next page, all projects are listed that are located in the selected workspace folder or that are linked into the pure::variants/Eclipse workspace.

**Figure 17. Project selection page**




Select one and on the next page choose the model(s) or configuration space you want to open.

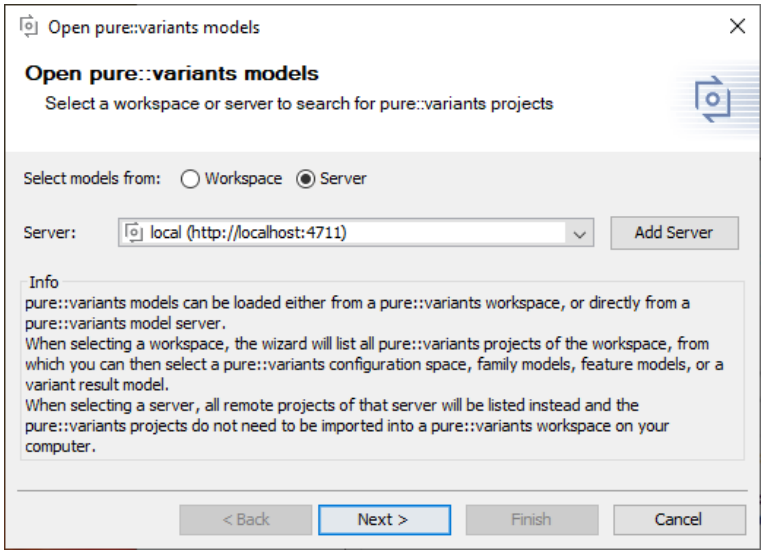
**Figure 18. Model selection page**




## Opening models from a model server

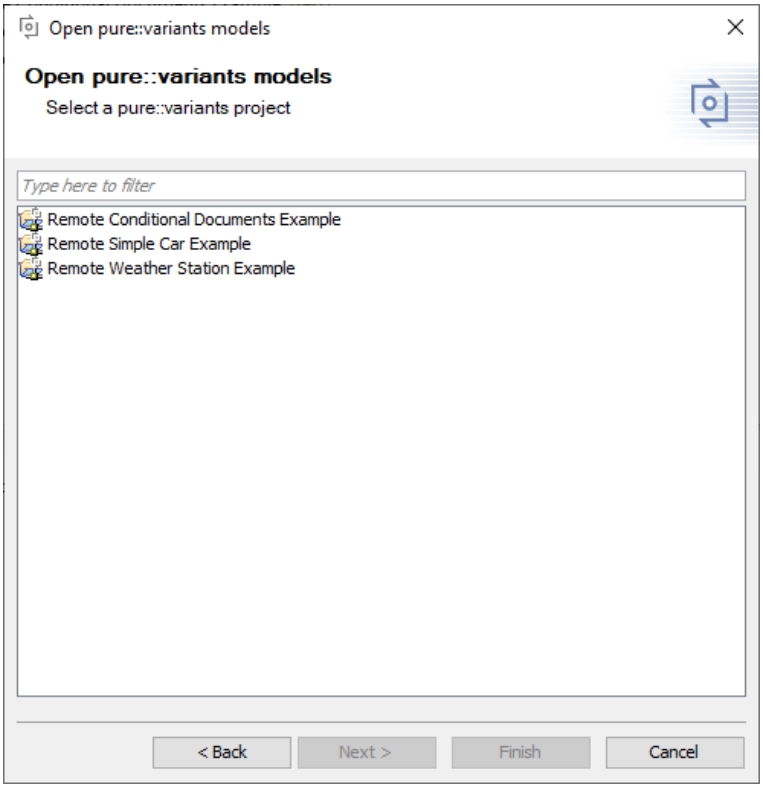
To open a model or configuration space directly from a pure::variants model server, also press . On the first page of the wizard, choose *server* and add the server address via button **Add Server**. Like in pure::variants, new servers need a name and the server address (e.g., https://yourserveraddress:443). Any known servers are listed in the server dropdown box. If you later need to add or remove a server from the list, open the Integration preferences by pressing . On tab *User Settings*, you can add or remove servers.

**Figure 19. Mode selection page**



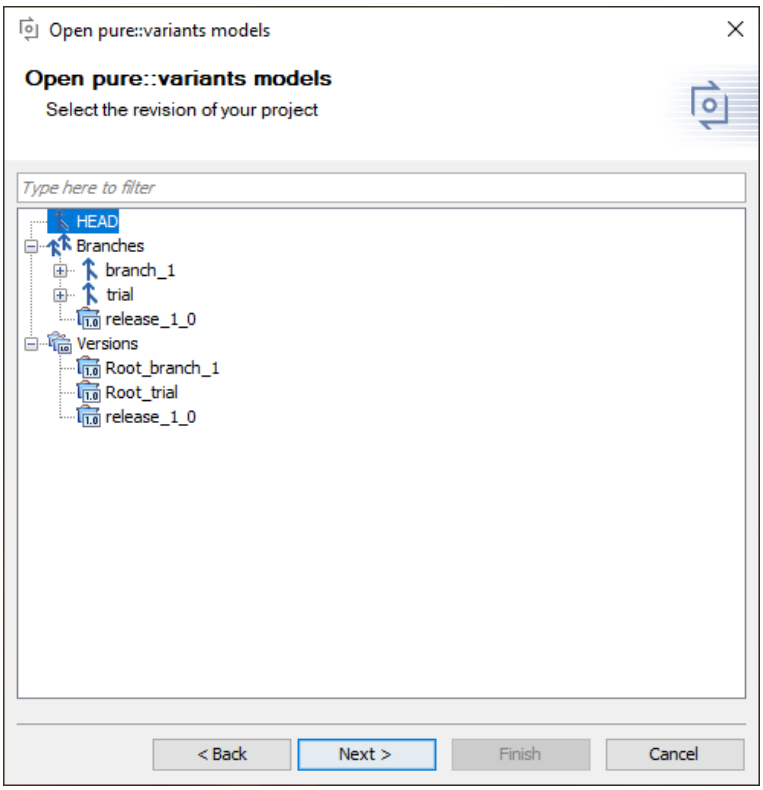

On the next page of the wizard, all projects of the server that the current user has read access to are listed.

**Figure 20. Project selection page**



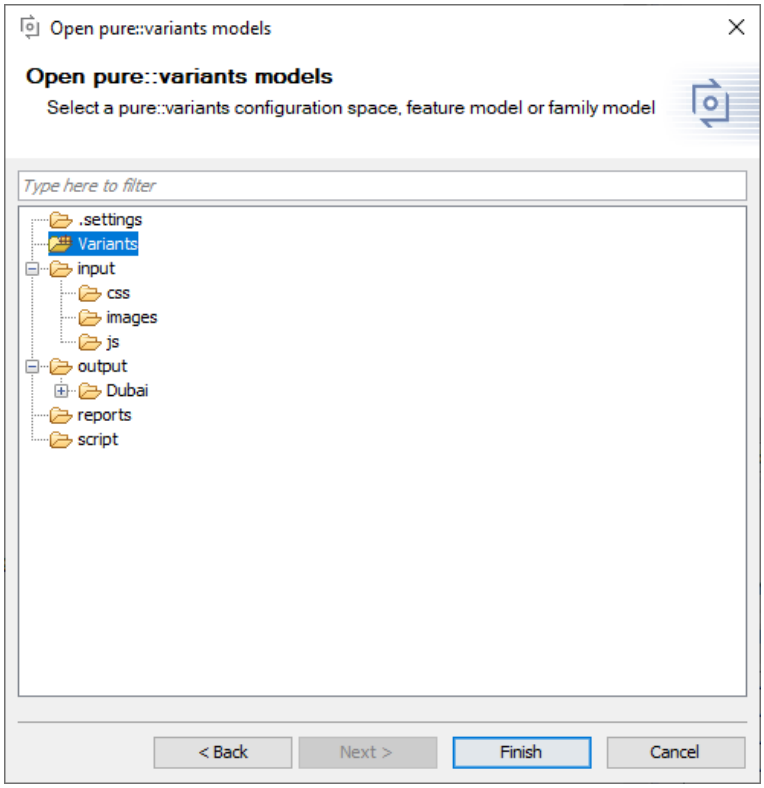

Select one and on the next page choose the project's revision (branch or tag) from which you want to load a model.

**Figure 21. Project selection page**




Finally, on the last page select the model(s) or configuration space you want to open.

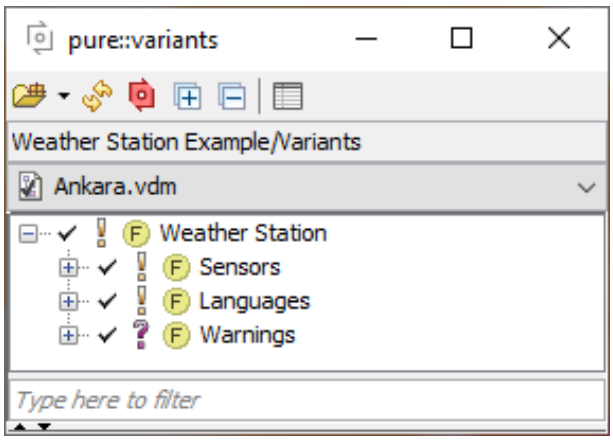
**Figure 22. Model selection page**




## Opening pure::variants Configuration Spaces

To open a pure::variants configuration space, use the wizard as described above. On the last page, select a configuration space folder. Now the Integration window should show all used models of your configuration space. Please note that family models (.ccfm) are not opened per default. You can enable loading family models in the Integration preferences on the **Visualization** tab. After selecting a variant from the dropdown list, selections should be shown in front of features. To ease usage of configuration spaces with many variants, the latest opened variants are shown at the top of the list.

**Figure 23. Configuration Space with Selected Variant**




[1]You can create a variant result model in pure::variants by clicking the **Save Result to File** button that is shown in the toolbar of a variant description model.

## Opening Other pure::variants Models

Other pure::variants models, such as variant result models[1] (.vrm), feature models (.xfm), and family models (.ccfm) can also be opened via  . Please note that family models (.ccfm) are not listed per default. You can enable loading family models in the Integration preferences on the **Visualization** tab.

## Live Connection with pure::variants

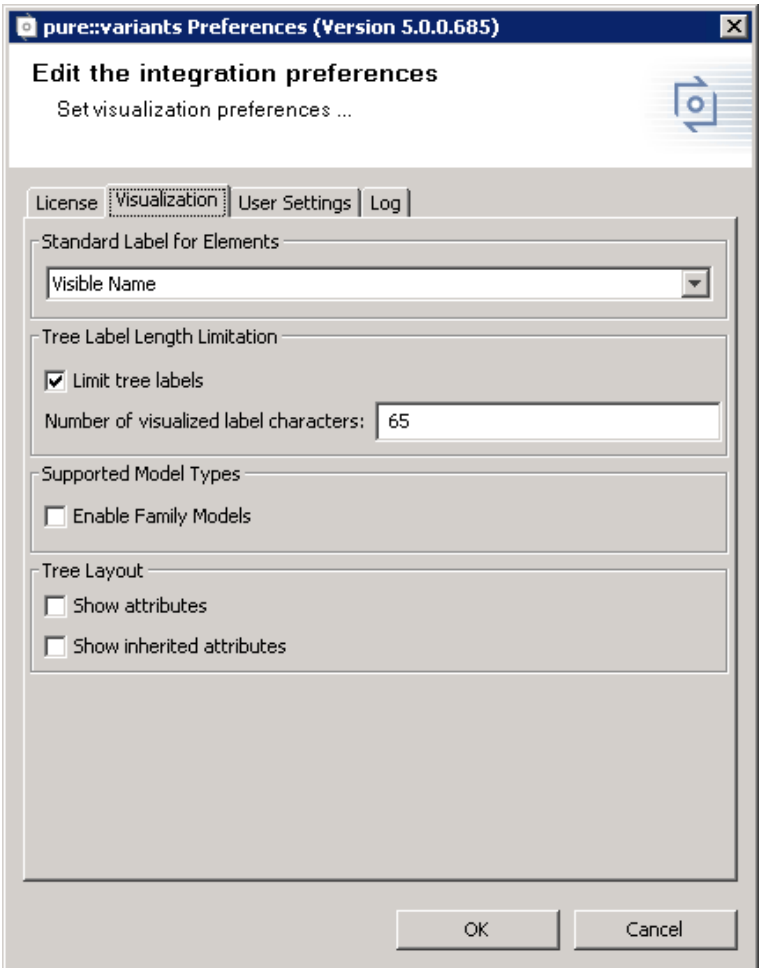
Since pure::variants 4.x, changes of the loaded pure::variants models are propagated live to the Integration. For example, directly after editing the name or changing the selection of a feature the loaded models are updated in the Integration window. To enable this live update, the following prerequisites need to be fulfilled:

- the opened model needs to be located in an Eclipse workspace
- the changes have to be done on the same Eclipse workspace using pure::variants 4.x or later
- either a configuration space, feature model or family model needs to be loaded (Variant result models can only be updated automatically when the .vrm file is saved)

## Model Visualization Preferences

In the Integration preferences, you can set how pure::variants models will be displayed and which model types are supported. To do that, open the Integration preferences by pressing  and go to the **Visualization** tab (see Figure 24, “Preferences Dialog Visualization Tab”). The first dropdown box enables you to set how elements in the pure::variants model view are labeled. Furthermore, you can limit how many characters are shown for each element in the tree, enable or disable the loading of family models, and set whether attributes are shown in the model tree. To also show attributes inherited from parent elements, select **Show inherited attributes**.

**Figure 24. Preferences Dialog Visualization Tab**

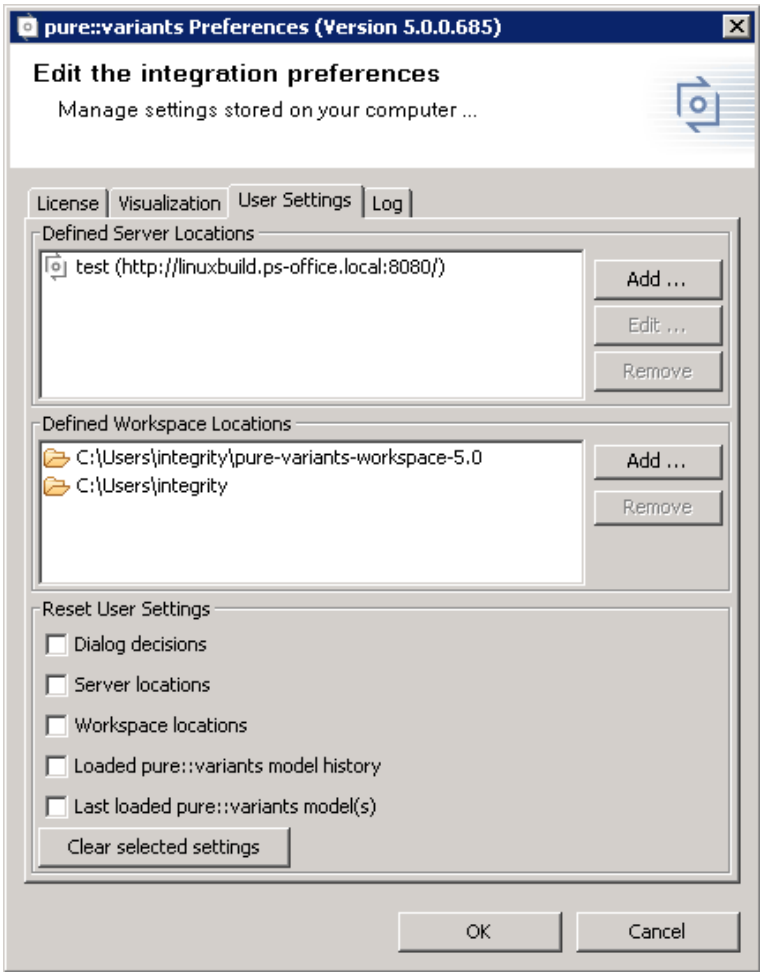



## Manage Settings

In the Integration preferences, you can also manage settings stored by the Integration. You can, for example, add or remove known workspace and server locations or clear certain settings. To do that, open the Integration preferences by pressing .

On tab *User Settings*, you can manage settings that are stored only on your local machine, specifically for your user. (see Figure 25, “Preferences Dialog User Settings Tab”). This includes server and workspace locations, dialog decisions, the history of previously loaded pure::variants models, and so on. The first section, **Defined Server Locations** enables you to add, edit and remove the server locations that are not locked. The second section, **Defined Workspace Locations** enables you to add and remove the workspace locations. The last section **Reset User Settings** enables you to clear the selected settings. For example, you can select checkbox *Last loaded pure::variants model(s)* and press the clear button, to make sure no pure::variants model is loaded from the user settings at startup of the Integration.

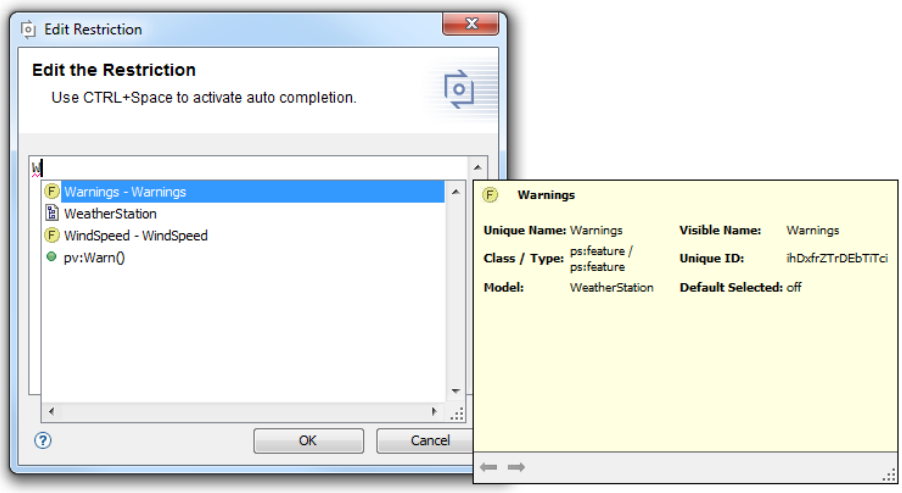
**Figure 25. Preferences Dialog User Settings Tab**




## Working with the pvSCL Rule Editor

The pvSCL rule editor features auto completion and syntax highlighting. You can automatically complete words by pressing CTRL + space. If more than one word is possible, a list containing possible keywords and features including all pvSCL functions is displayed. For auto completion the loaded pure::variants models are evaluated.

**Figure 26. Using the Rule Editor**



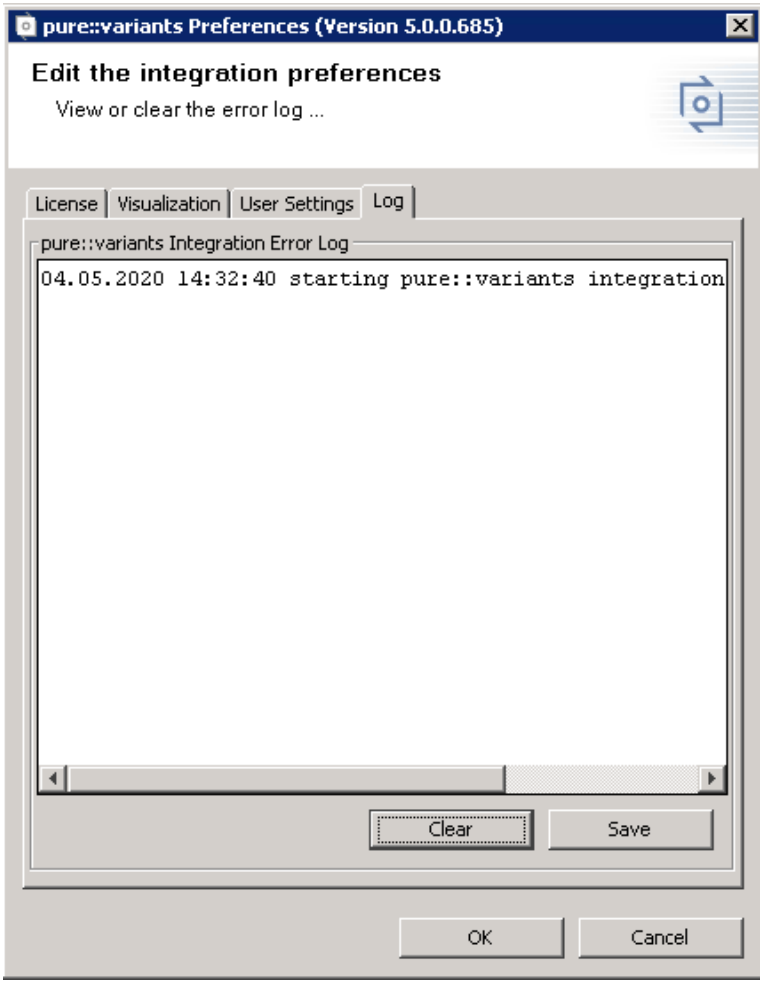
When you are done editing the pvSCL script and press the OK button, the entered expression is checked for errors. If no errors were found, the pure::variants integration closes and the entered pvSCL expression is added to the selected requirement of the opened Integrity document. If an error is found, it is reported, so you can correct the pvSCL expression before writing it to the Integrity document.

Errors in pvSCL expressions are reported if the expression's syntax is not pvSCL compliant, or if an element is unknown based on the loaded pure::variants models. Unknown elements are highlighted in red.

## Troubleshooting

If the Integration does not behave as expected, it may be useful to check its log file. You can find it in the Integration preferences ( ) on the **Log** tab (see Figure 27, “Preferences Dialog Log Tab”). It also enables you to save it to your disk or clear the contents of the log file.

**Figure 27. Preferences Dialog Log Tab**




## Advanced Integration Setup

If you are using a pure::variants floating license, establishing a connection with the pure::variants license server may need extra configuration. This may be the case, for example, if the license server is located behind a proxy server or the communication with the server is encrypted and a self-signed certificate is used.

All extra configurations can be done by manually editing file `pv.properties`. You can find it in two different locations:

- If you want to configure the settings for all users on the machine, please edit

```
%PROGRAMDATA%/pure-variants-5/pv.properties
```

- If you want to configure the settings only for the current user, please edit

```
%APPDATA%/pure-variants-5/pv.properties
```

When editing `pv.properties`, please note that:

- In case the file does not exist, you need to create it and any necessary folders first.
- Before editing `pv.properties`, make sure that no pure::variants Integration is running (e.g. Integration for Word, Excel, Doors, or the p::v Desktop Hub), otherwise your changes may be overwritten when closing the integration.
- Path delimiters in any paths you enter must be forward slashes or escaped backward slashes (`/` or `\\`). Otherwise the path cannot be read.
- All property names are case-sensitive.

## Proxy Settings

The following properties can be set to configure your proxy settings.

**Table 1. Proxy Settings**

| Property Name | Comments based on Java system property documentation |
|---|---|
| http.proxyHost | The hostname, or address, of the proxy server |
| http.proxyPort | The port number of the proxy server |
| https.proxyHost | The hostname, or address, of the proxy server in case HTTPS is used |
| https.proxyPort | The port number of the proxy server in case HTTPS is used |
| http.nonProxyHosts | Indicates the hosts that should be accessed without going through the proxy. Typically this defines internal hosts. The value of this property is a list of hosts, separated by the '\|' character. In addition the wildcard character '*' can be used for pattern matching. For example http.nonProxyHosts=*.foo.com\|localhost will indicate that every hosts in the foo.com domain and the localhost should be accessed directly even if a proxy server is specified.<br><br>The default value excludes all common variations of the loopback address. |
| java.net.useSystemProxies | Set this to "true" to use Windows' global proxy settings (default: false), which are set in the Internet Explorer or in the Windows system settings. If one of the above properties is set, it overrides the respective Windows system property. |

For example to use Windows' proxy settings, you would need to append this line to `pv.properties`:

```
java.net.useSystemProxies=true
```

Or to set all properties manually, you would need to append something like this:

```
http.proxyHost=YourHTTPProxyHost
http.proxyPort=80
https.proxyHost=YourHTTPSProxyHost
https.proxyPort=443
http.nonProxyHosts=*.foo.com|localhost
```

## HTTPS Connection with License Server

The following HTTPS-related properties can be set. For more details, please refer to the respective Java system property documentation.

**Table 2. HTTPS Settings**

| Property Name | Comments |
|---|---|
| javax.net.ssl.trustStore | Path to your trust store |
| javax.net.ssl.trustStorePassword | Password of your trust store |
| javax.net.ssl.trustStoreType | Trust store type (e.g. JKS) |
| javax.net.ssl.keyStore | Path to your key store |
| javax.net.ssl.keyStorePassword | Password of your key store |
| javax.net.ssl.keyStoreType | Key store type (e.g. JKS) |
| javax.net.debug | Activation of debug mode (e.g. "all" to write all possible debug logs) |
| com.sun.net.ssl.checkRevocation | Enable certificate revocation checking |

For example when using a self-signed certificate that is stored in trust store `D:/sandbox/servercert/cert-trusted.jks` you would need to append the following lines:

```
javax.net.ssl.trustStore=D\:/sandbox/servercert/cert-trusted.jks
javax.net.ssl.trustStorePassword=password
```