pure::variants Setup Guide

Parametric Technology GmbH

Version 7.0.0.369 for pure::variants 7.0

2025

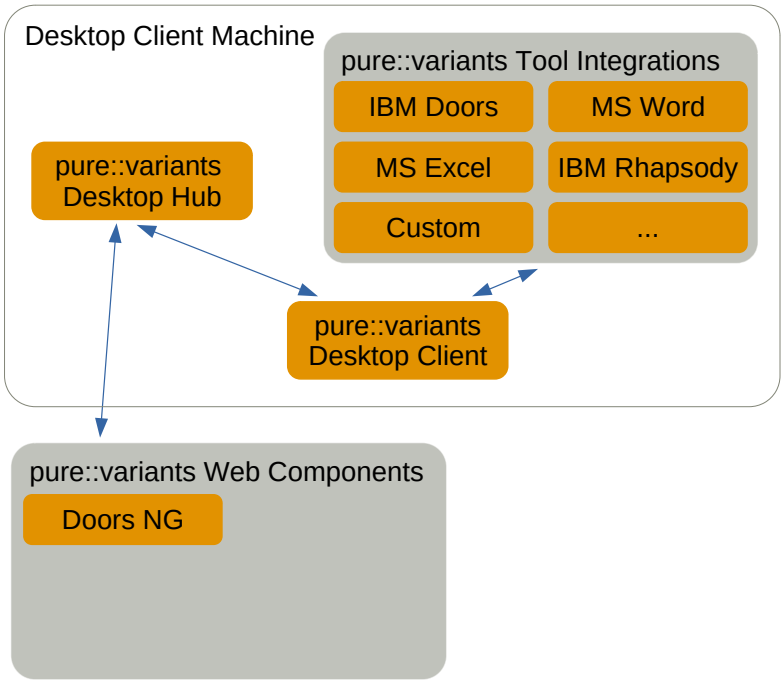
Table of Contents

1. Introduction

This setup guide describes how to install and update pure::variants Evaluation on a personal computer.

The following sections provide detailed insights in each of the individual pure::variants Deployment components from the perspective of administration and setup.

Figure 1. The Big Picture



The manual is available in online help inside the installed product as well as in printable PDF format. Get the PDF here.

2. System Requirements

pure::variants has different system requirements, depending on the part of the software going to be installed. The following lists the system requirements for all parts of the software.

2.1. pure::variants Desktop Client

- Operating System

  Only 64-bit operating running on x64 systems are supported:

  - Windows 10, 11
  - Windows Server 2016, 2019, 2022
  - Linux with X11 Window System installed
  - Mac OS
    - For Macs with Apple Silicon Rosetta 2 needs to be installed.

- Software

  - Oracle Java SE or OpenJDK

    Supported Java versions:

    - Java 17 - 23

  - The Java compatibility is tested with the official Java Standard Edition provided by Oracle (https://www.java.com/en/download/) and the OpenJDK provided by Oracle (https://jdk.java.net/archive/).

  - Eclipse 4.24 – 4.34

- Memory

  - Min: 4 GB

  - Recommended: 8 GB

- CPU

  - Min: Dual Core CPU

  - Recommended: 4 CPU cores

- HDD

  - Min: 10 GB free disk space

3. pure::variants Desktop Client

The pure::variants Desktop Client can be installed using the pure::variants installer as a stand-alone application or it can be installed into an existing Eclipse based tool chain. For both ways to install pure::variants we recommend to use the pure::variants installer.

The pure::variants installer is available for Windows only. If the operating system platform is Linux or MacOS X, pure::variants needs to be installed into an existing Eclipse instance. See Section 3.1.2, “Install into an existing Eclipse”.

In case of very strict firewalls or no network access on the installation machine either install pure::variants as a stand-alone application. (Section 3.1.1, “Install with pure::variants Installer”) or install pure::variants into an existing Eclipse instance using an update site. (the section called “Using update site”). These installation methods allow you to first download the installation packages and install pure::variants afterwards.

The installation procedures are described below. Once the initial installation has finished, installation of a license is required to use pure::variants. See following section for more information on license installation.

3.1. Install pure::variants Desktop Client

3.1.1. Install with pure::variants Installer

This installation method is available for Windows only. If you do not use Windows please see Section 3.1.2, “Install into an existing Eclipse”.

To be able to successfully install the pure::variants Desktop Client you need to following:

- pure::variants Desktop Client license

  - If a floating license is used additionally the license server URL is needed to be able to connect and obtain a license form the license server.

- pure::variants Desktop Client installer or pure::variants update site

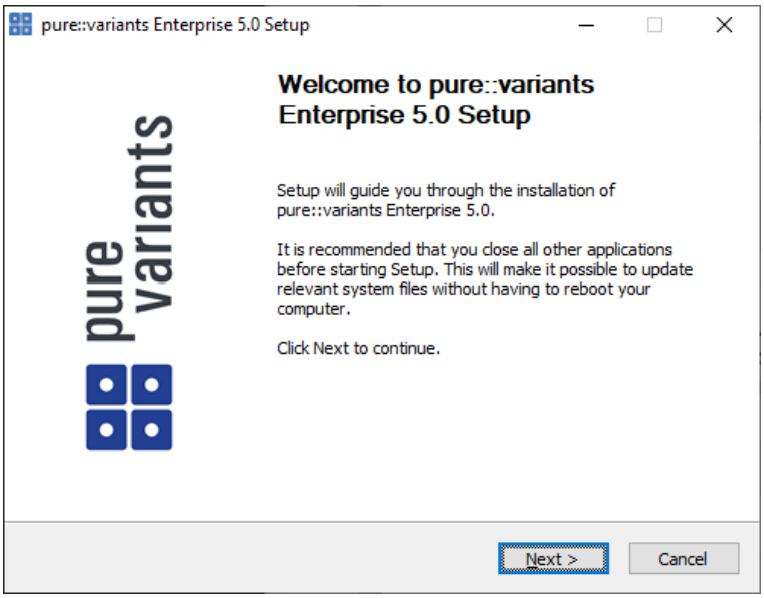
- supported 64-Bit Java version installed

The Windows Installer can be download from the pure::variants product web page. Go to https://www.pure-systems.com/pvde-update. The product download pages are protected by a password. You need to login by using the email address and the registration number from the license file.

Download the installer package ("pure::variants Windows Installer Package") and extract it. The installer will set up a fresh Eclipse with pure::variants and documentation. Start the installation by double-clicking "Setup Enterprise X.Y.ZZ.exe". Running the pure::variants enterprise installer requires Administrator privileges.

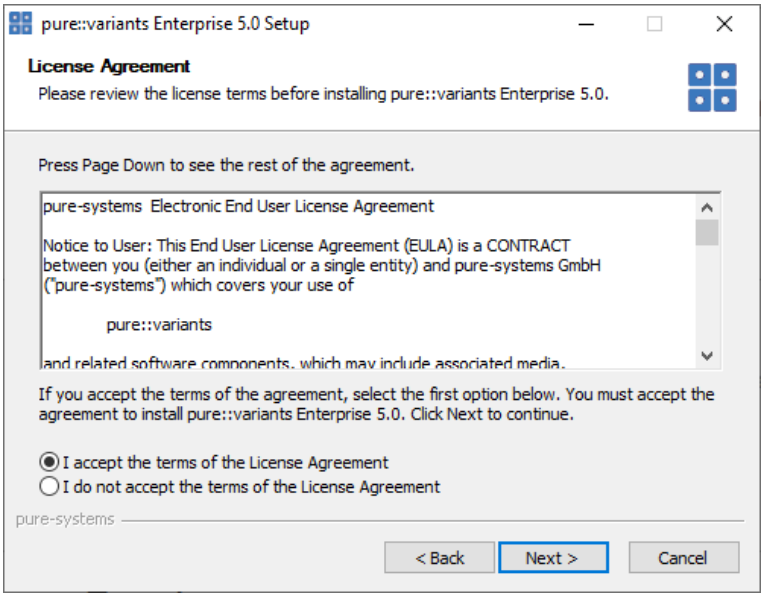
All pure::variants extensions available for the account are automatically included in the Windows Installer download. However, some may not be enabled by default in Installer. Make sure to select the desired extensions during the installation process. Later updates to the extension selection can be done either by reinstalling pure::variants or by following the alternatives described in the section called “Using update site”.

Figure 2. pure::variants Desktop Client Installer



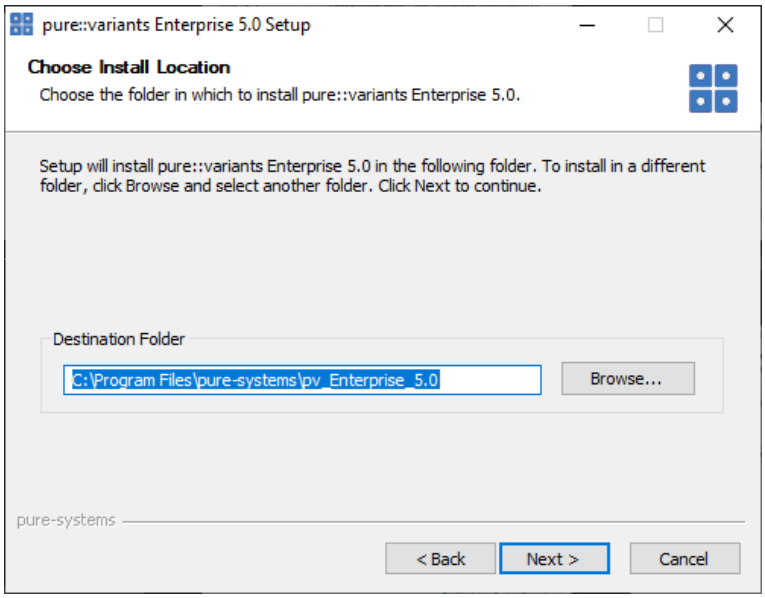
Click *Next*.

Figure 3. Setup pure::variants Desktop Client License



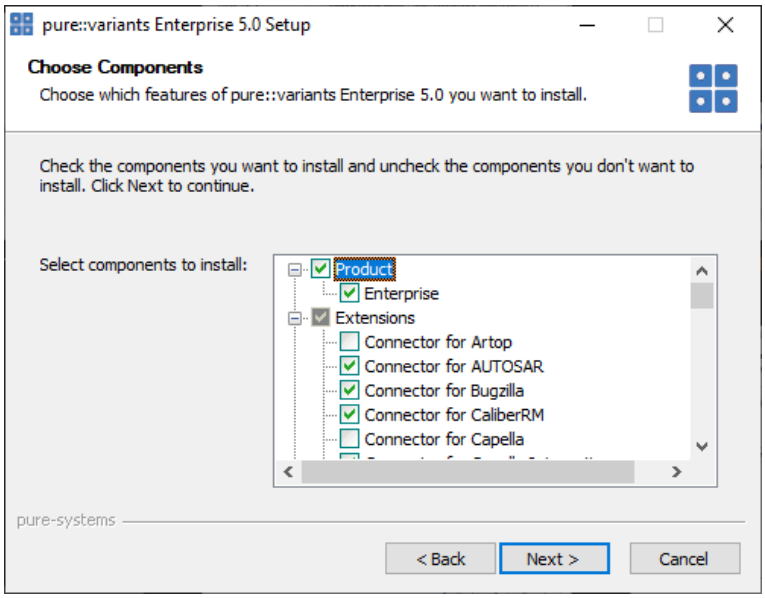
Read the license agreement and after accepting it click *Next*.

Figure 4. Setup pure::variants Desktop Client Installation Location



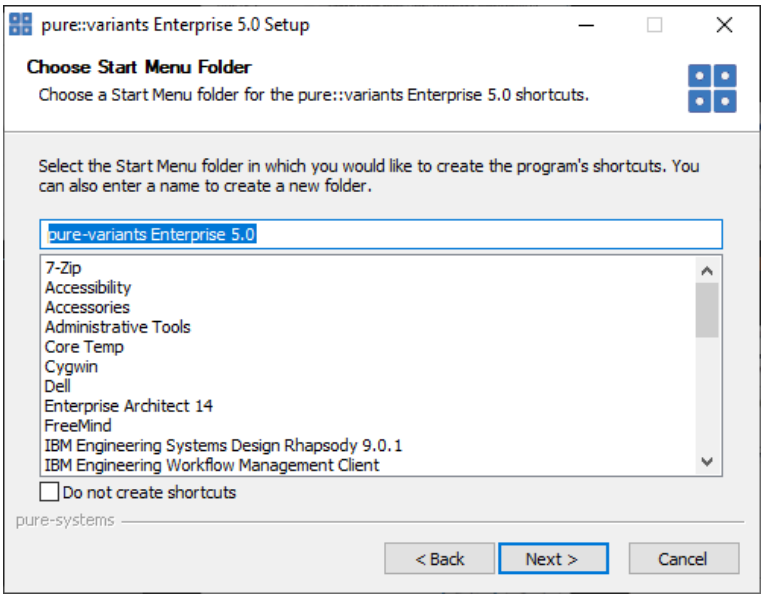
Select the folder where to install the pure::variants Desktop Client files. Click *Next*.

Figure 5. pure::variants Desktop Client Feature Selection



Select the connectors which shall be installed with the pure::variants Desktop Client. Click *Next* after the feature selection is complete.

Figure 6. Setup pure::variants Desktop Client Start Menu

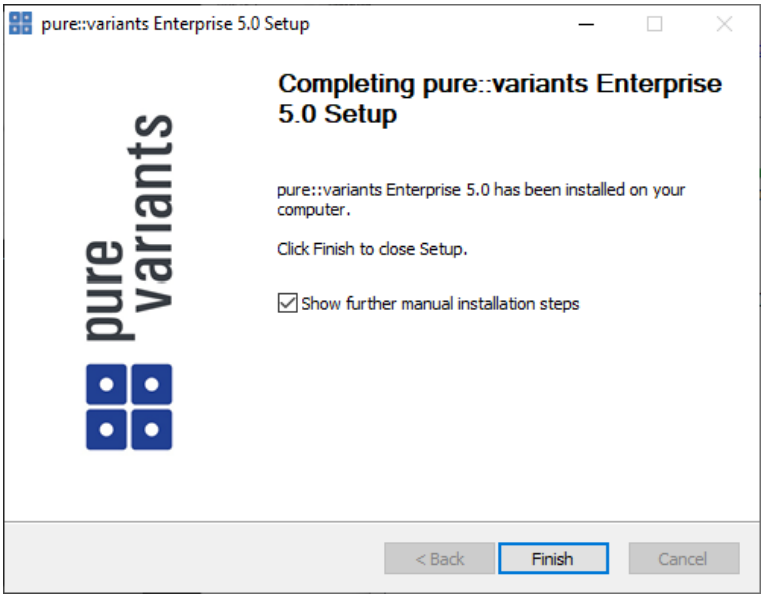


Enter the name for the Windows start menu entry, or disable the creation of the start menu entry. Click *Next*

The next pages may show information about pure::variants integrations, which are installed along with the pure::variants Desktop Client. If no connector was selected providing an integration, this page will show the *Install* button.

Click *Install* to start the installation process.

Figure 7. Setup pure::variants Desktop Client Finish Page



The Option *Show further manual installation steps* will open a text document showing more information about the installed integrations and possible manual installation steps, which have to be performed for the integrations to work properly.

pure::variants Enterprise Installer Command Line Options

The pure::variants installer provides the following command line options:

Table 1. pure::variants Installer Command Line Options

Option	Description
/S	Run the installation in silent mode. No installation dialog is opened. Automatically installs the default selected software packages, or all if used together with option /ALL.
/UPDATE /S	To update an existing installation in silent mode. Same as silent installation, no dialog is opened.
start "" /WAIT "Setup.exe" /UPDATE /S	To update an exsiting installation in silent mode. Same as silent installation, no dialog is opened, but ensures installer not running in background.
/ALL	Select all packages for installation.
/NODOTNET	Skip installation of the .NET 4 Framework.
/NOINTCOMP	Skip installation of the integration components for Java & .NET.
/JAVA	Location of the Java executable to be used for the installation. Example: /JAVA="C:\Program Files\Java\jre6\bin\java.exe"
/ECLIPSE	Path to an existing Eclipse installation into which to install pure::variants as a feature, instead of installing pure::variants as a stand-alone application. This directory must contain the file eclipsec.exe. Example: /ECLIPSE="C:\Program Files\Eclipse 3.8\eclipse"
/D	Path to the directory where to install the pure::variants stand-alone application. **Must be the last option on the command line and must not contain any quotes, even if the path contains spaces.** Example: /D=C:\Program Files\pure-variants

Example commandline with JAVA path:

```
"D:\5.x.x\pure-variants Setup 5.x.x\Setup Enterprise 5.x.x.exe" /JAVA="C:\Program Files\Java
\jre1.8.0_231\bin\java.exe"
```

Install pure::variants in silent mode

The pure::variants Desktop Client installer has a silent mode. This mode installs the pure::variants Desktop Client without user interaction by just using the standard settings of the pure::variants Desktop Client installer also considering further options on the command line.

To do this, call the installer with command line option */S*. See the section called “pure::variants Enterprise Installer Command Line Options” for all available command line options.

Update pure::variants in silent mode

It is also possible to run the update in background or silent mode to update an existing installation. Note that there will be no console output as well.

To do this, call the installer with command line option */UPDATE /S*. To run it in silent mode but not in background *start "" /WAIT "Setup.exe" /UPDATE /S* can be used. See the section called “pure::variants Enterprise Installer Command Line Options” for all available command line options.

3.1.2. Install into an existing Eclipse

pure::variant can be installed into an existing Eclipse based tool chain. To install pure::variants, the pure::variants installer package download from the pure::variants updatesite can be used. We recommend this for all Windows users.

Alternatively the pure::variants update site can be used directly with the Eclipse client. You can also download an archived update site from the pure::variants update site and use this with the Eclipse client (See the section called “Using update site”).

Installation Requirements

pure::variants needs to following features to already be installed in the target Eclipse, or the Eclipse instance has to have access to the Eclipse release update site.

- JavaScript Development Tools
  - org.eclipse.wst.jsdt.feature.feature.group
- Eclipse Business Intelligence and Reporting Tools (BIRT)
  - org.eclipse.birt.feature.group
- Graphical Modeling Framework
  - org.eclipse.gmf.feature.group

Using pure::variants Installer

The installation into an existing Eclipse instance is done the same way as installing pure::variants as stand-alone application (See Section 3.1.1, “Install with pure::variants Installer”).

There is one difference: the target Eclipse has to be defined with the */ECLIPSE* command line option.

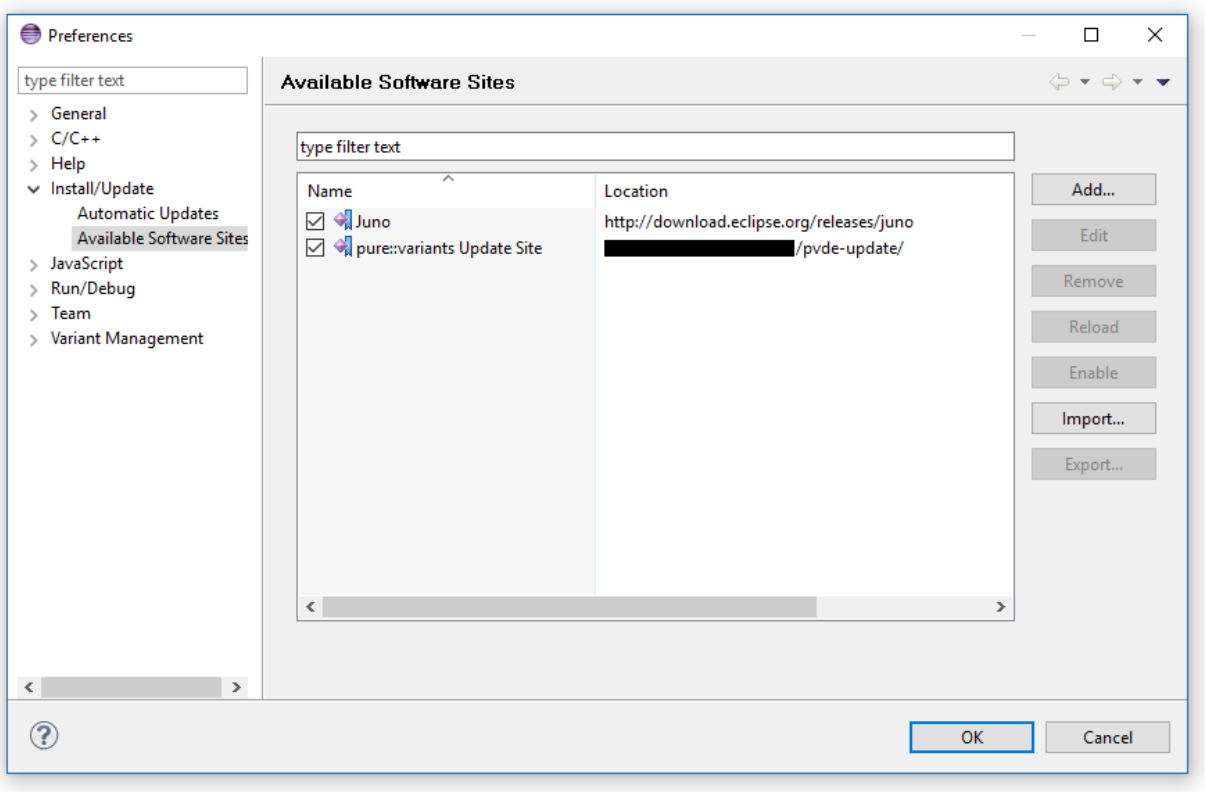
Using update site

- Start pure::variants (or the Eclipse into which pure::variants has been installed).
- Select "Help"->"Install New Software...".
- Select "pure::variants update site" from the available Software Sites.

  If location "pure::variants update site" is not present, enter your location in the edit field, or press "Add" if you have a local update site at hand.

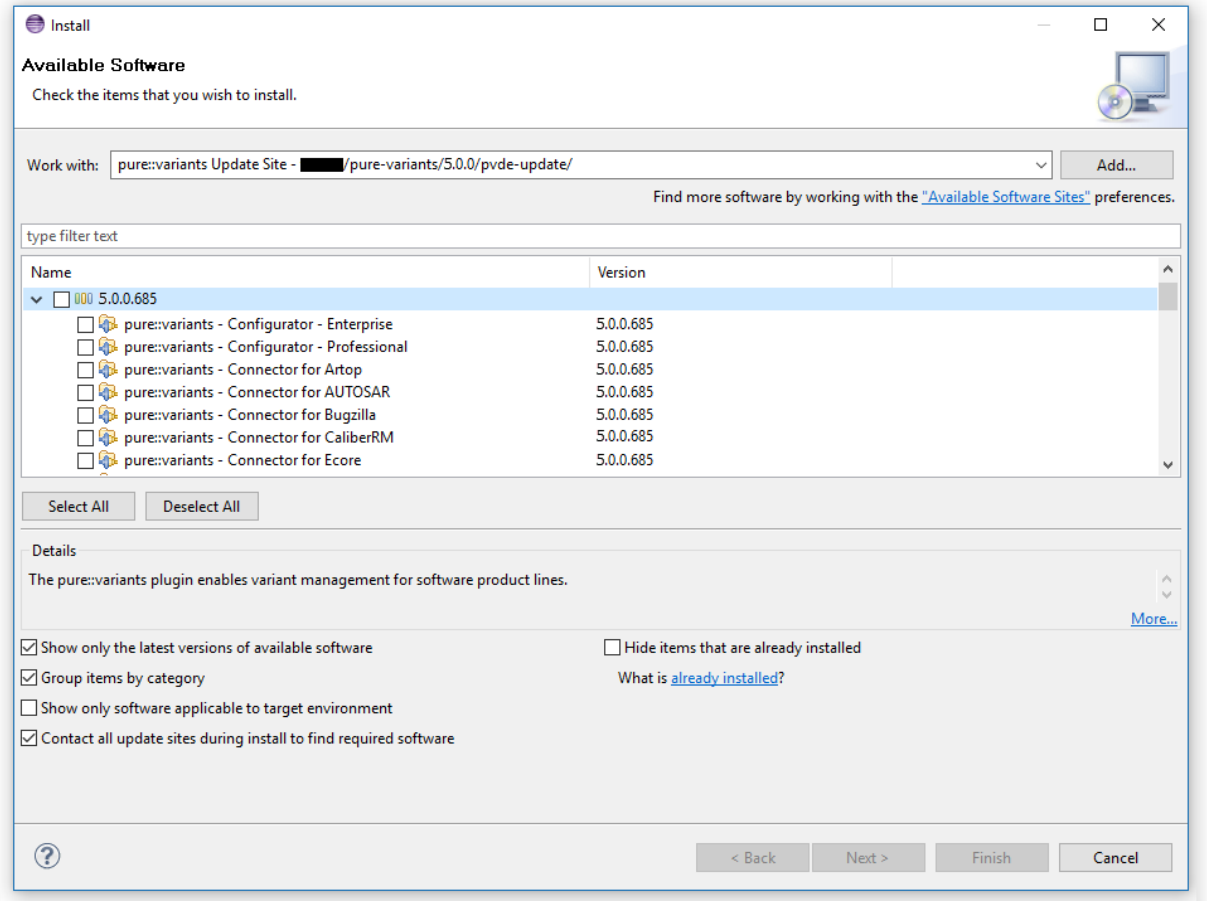
  The location of the site depends on the pure::variants product variant. Visit the Parametric Technology web site (https://www.pure-systems.com) or read your registration email to find out which site is relevant for the version of the software you are using.

Figure 8. Update Site Selection



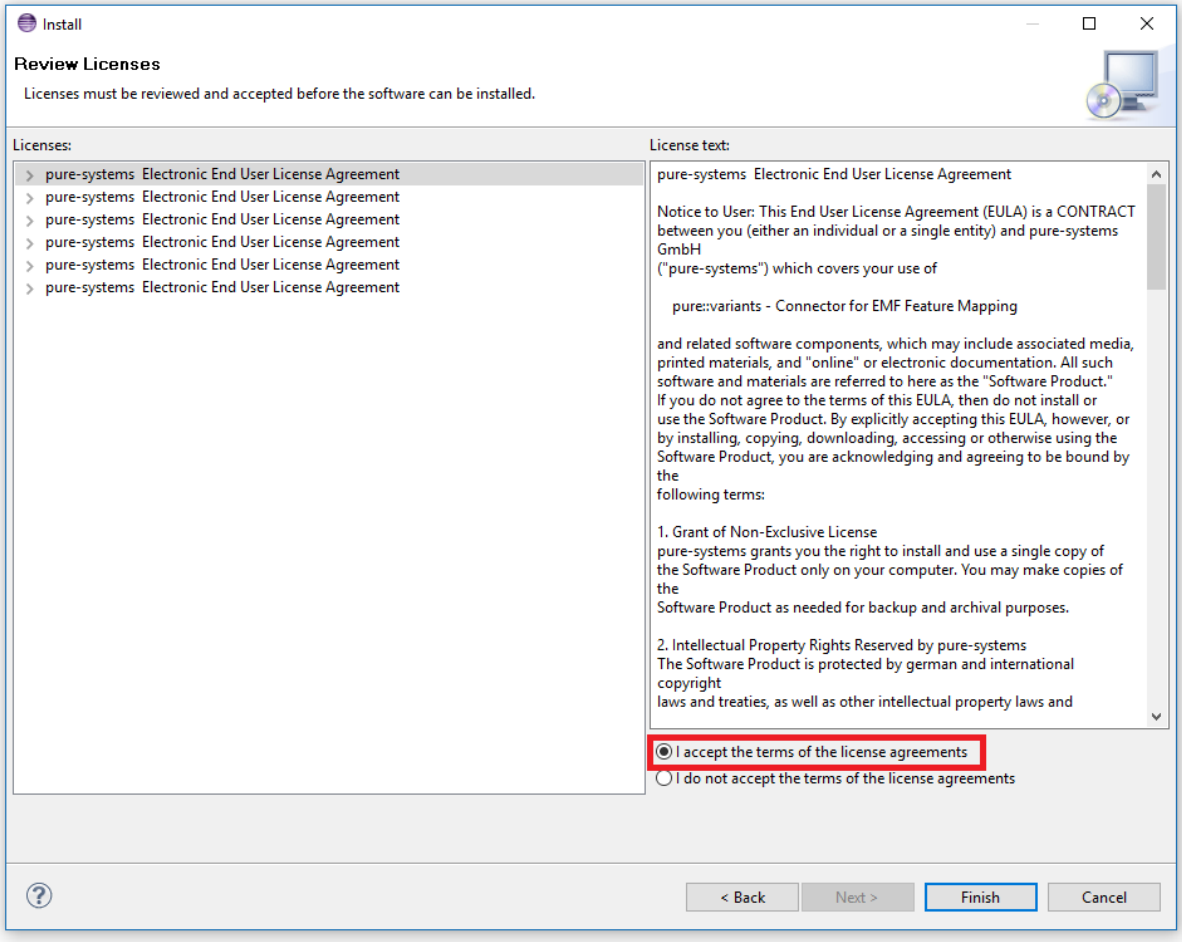
- Unfold the pure::variants update site and select all features to be updated. Select "Next".

Figure 9. Pure::variants Plugin Selection



- Accept license, hit "Next" and then "Finish".

Figure 10. Licence Agreement



- In the dialog select "Install all".
- Restart pure::variants when asked for.

If the direct remote update is not possible (often due to firewall/proxies preventing Eclipse accessing external web sites), please go to the web site using an Internet browser:

- For pure::variants Evaluation use https://www.pure-systems.com/pv-update
- For pure::variants Enterprise use https://www.pure-systems.com/pvde-update

and download the "Complete Updatesite" archive:

- Start pure::variants (or the Eclipse into which pure::variants has been installed).
- Select "Help"->"Software Updates"->"Find and Install...".
- Select "Search for new features to install" and "Next".
- Click on button "Archived Update Site" or "Local Update Site".
- Use "Browse" to select the downloaded archive file.
- Press "Ok". The pure::variants update site from the archive should be selected.
- All other check boxes should be unselected to speed up the process. Press "Finish".

- Unfold everything below pure::variants update site and select all features to be updated. Select "Next".
- Accept license, hit "Next" and then "Finish".
- In the dialog, select "Install all".
- Restart pure::variants when asked for.

3.2. Update pure::variants Desktop Client

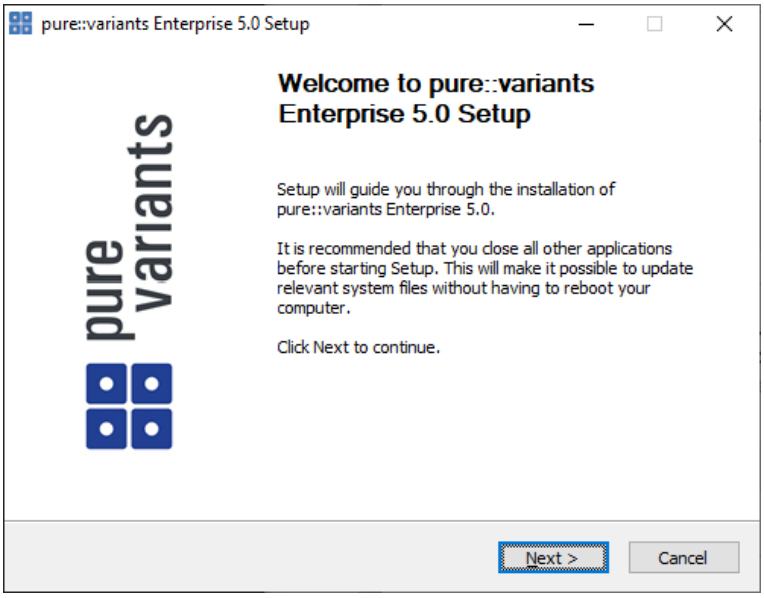
3.2.1. Update with pure::variants Installer

This update method is available for Windows only. If you do not use Windows please see Section 3.2.2, “Update with Update Action” or Section 3.2.3, “Update with Eclipse package manager”.

The Windows Installer can be downloaded from the pure::variants product web page. Go to https://www.pure-systems.com/pvde-update. The product download pages are protected by a password. You need to login using the email address and the registration number from the license file.

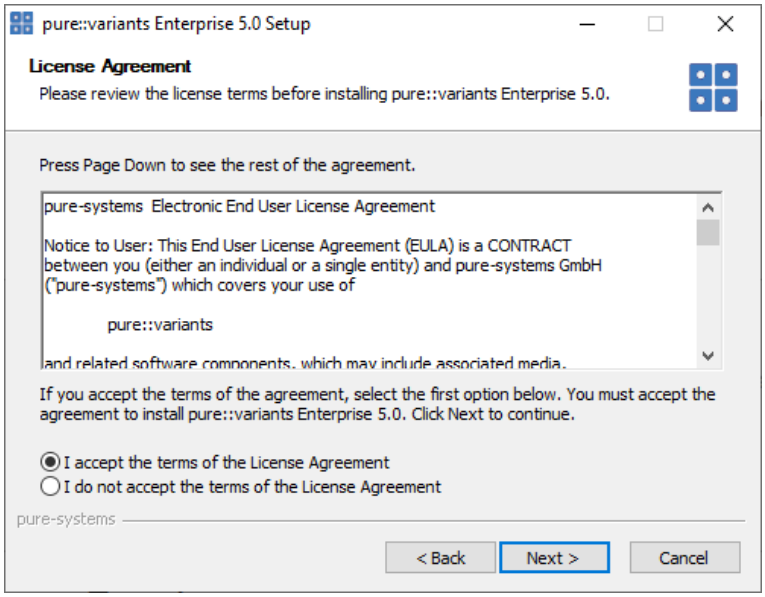
Download the installer package ("pure::variants Windows Installer Package") and extract it. The installer will check for an existing pure::variants Desktop Client installation and start in update mode if it finds one. Start the update by double-clicking "Setup Enterprise X.Y.ZZ.exe". Running the pure::variants enterprise installer requires administrator privileges.

Figure 11. pure::variants Desktop Client Installer



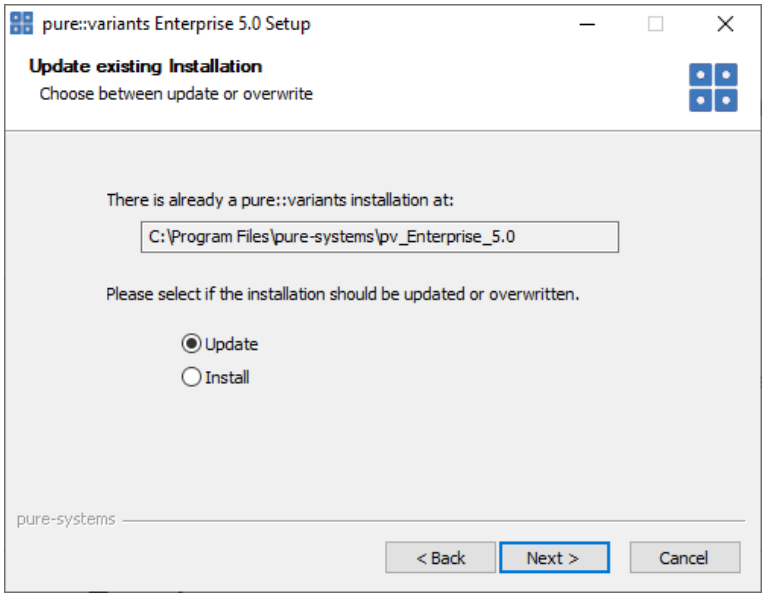
Click *Next*.

Figure 12. Setup pure::variants Desktop Client License



Read the license agreement, and after accepting it click *Next*.

Figure 13. Choose Update Mode

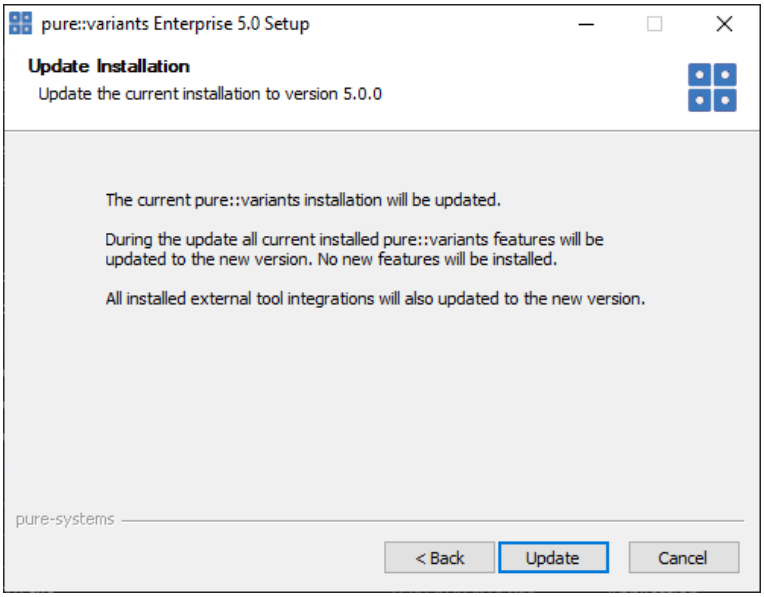


Choose *Update* if the current pure::variants Desktop Client installation shall just be updated with the same installed feature and settings. The installed pure::variants integrations will also be updated. The installed components cannot be changed. If a change of the installed components is wanted, choose *Install* mode.

Or choose *Install* if the current pure::variants installation shall be removed and a new fresh pure::variants Desktop Client shall be installed. The *Install* option runs the installer as described in Section 3.1.1, “Install with pure::variants Installer”. Please see this section for further installation steps.

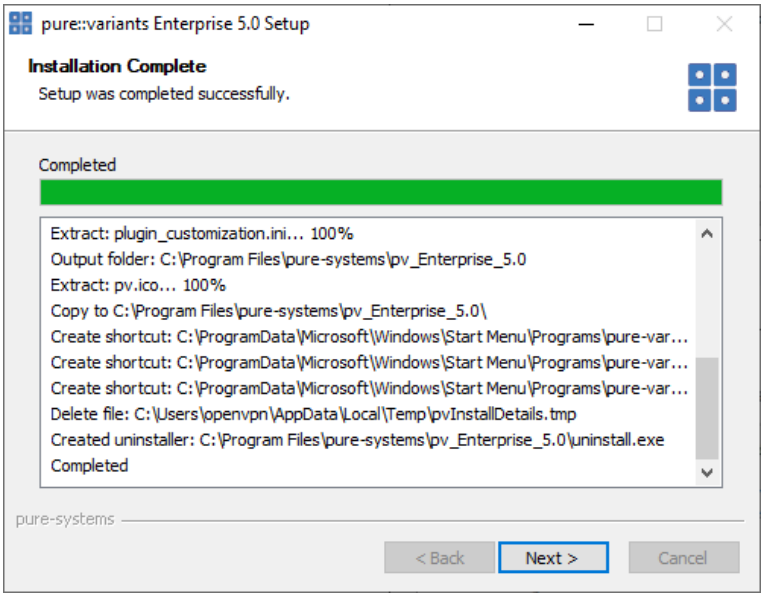
Click *Next*.

Figure 14. pure::variants Start Update



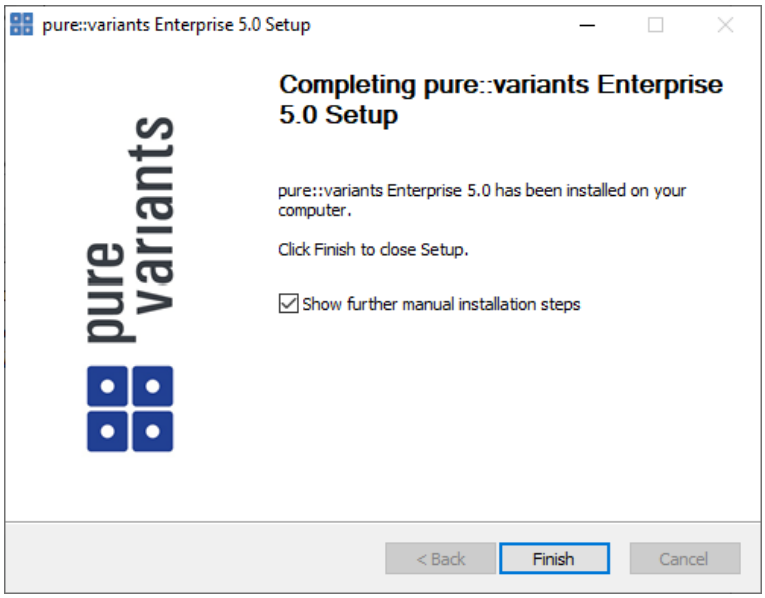
Click *Update* to start the update process.

Figure 15. pure::variants Installation Progress



This page is showing the installation details. Click *Next* after this is finished.

Figure 16. Update pure::variants Desktop Client Finish Page



The Option *Show further manual installation steps* will open a text document showing more information about the installed integrations and possible manual installation steps, which have to be performed for the integrations to work properly.

3.2.2. Update with Update Action

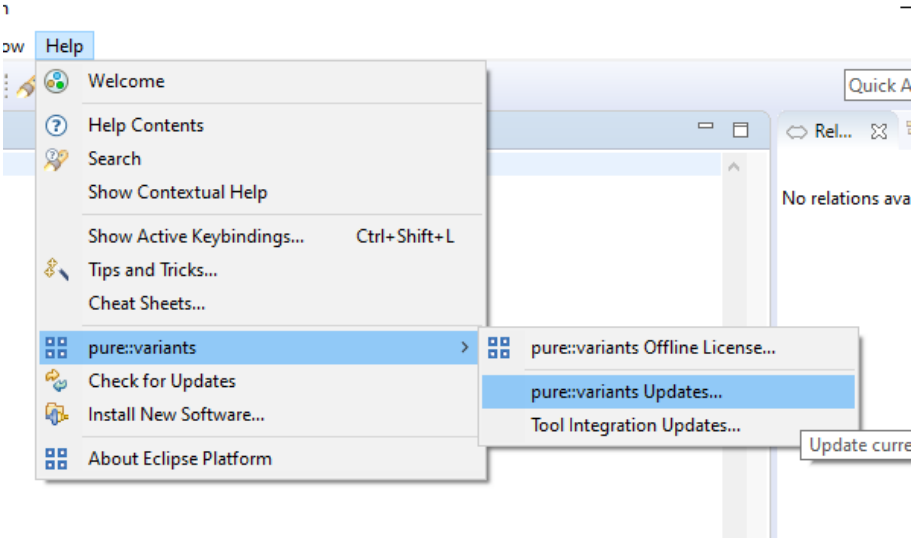
pure::variants has a built-in update action which can be used to perform an update with all the currently installed pure::variants extensions. This update action does not update the installed pure::variants integrations automatically. But they can be easily updated with the *Tool Integration Update* action. See Section 6.2, “Update pure::variants Tool Integrations” for the detailed description.

The update action requires administrator privileges.

Note

The pure::variants Desktop Client restarts automatically after the update process finished. So please make sure that all open editors are saved and closed before continuing.

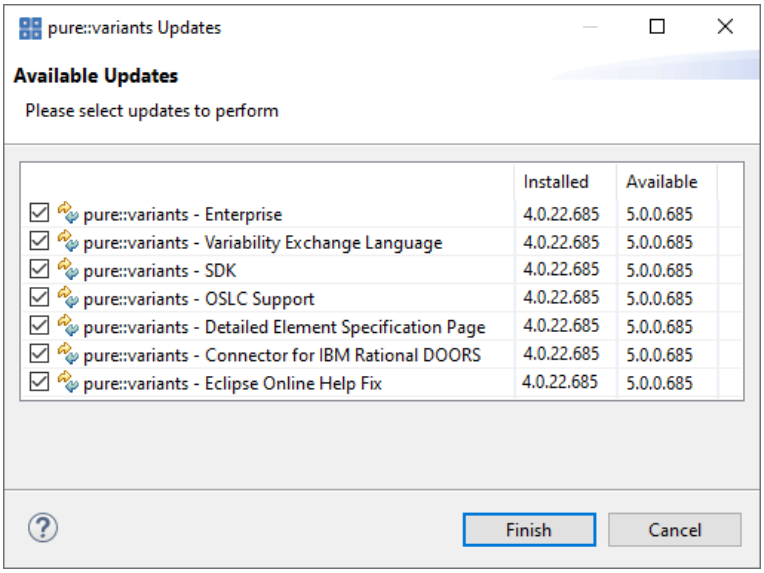
Figure 17. Start pure::variants Desktop Client Update



Start the pure::variants Desktop Client update with the *pure::variants Updates...* action from the pure::variants *Help* menu. The action can be found in the *pure::variants* sub-menu.

If the pure::variants Desktop Client is not started as Administrator, a dialog comes up to inform that pure::variants has to be started as Administrator.

Figure 18. Start pure::variants Desktop Client Update



A dialog comes up and shows all available updates. Select the features to update an click *Finish*. The update process starts and shows the progress in the same window.

Figure 19. Start pure::variants Desktop Client Update

After the update process finished, the pure::variants Desktop Client restarts automatically.

3.2.3. Update with Eclipse package manager

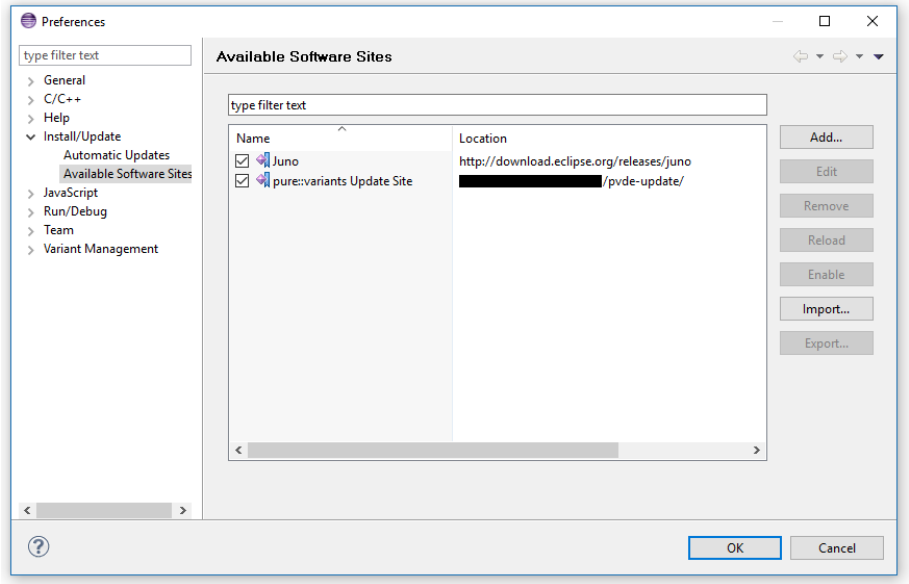
The quickest way to get a update for pure:.variants is to run the software updater inside pure::variants:

- Start pure::variants (or the Eclipse into which pure::variants has been installed).
- Select "Help"->"Install New Software...".
- Select "pure::variants update site" from the available Software Sites.

  If location "pure::variants update site" is not present, enter your location in the edit field, or press "Add" if you have a local update site at hand.

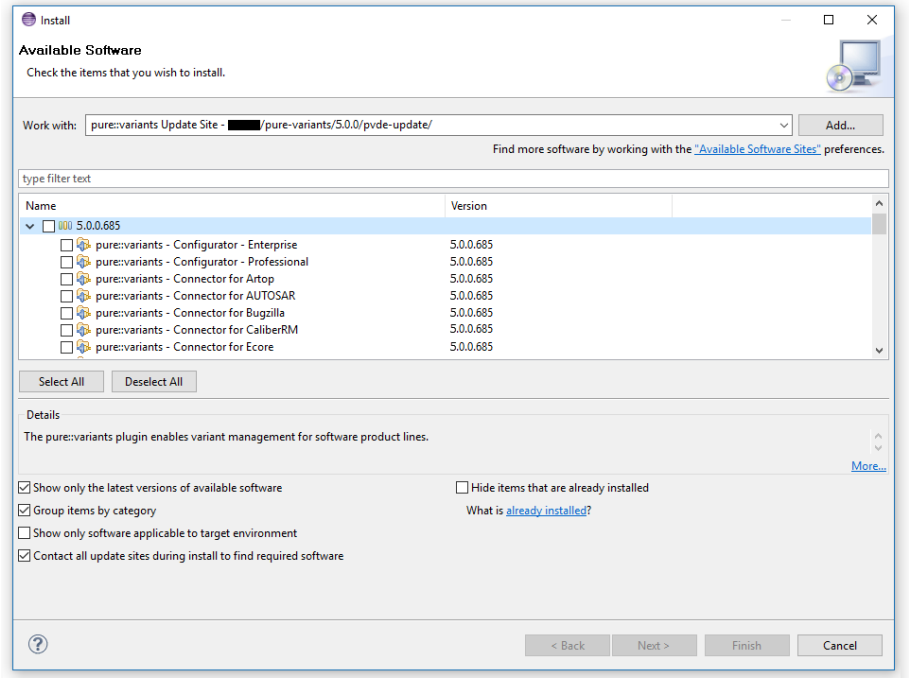
  The location of the site depends on the pure::variants product variant. Visit the Parametric Technology web site (https://www.pure-systems.com) or read your registration email to find out which site is relevant for the version of the software you are using.

Figure 20. Update Site Selection



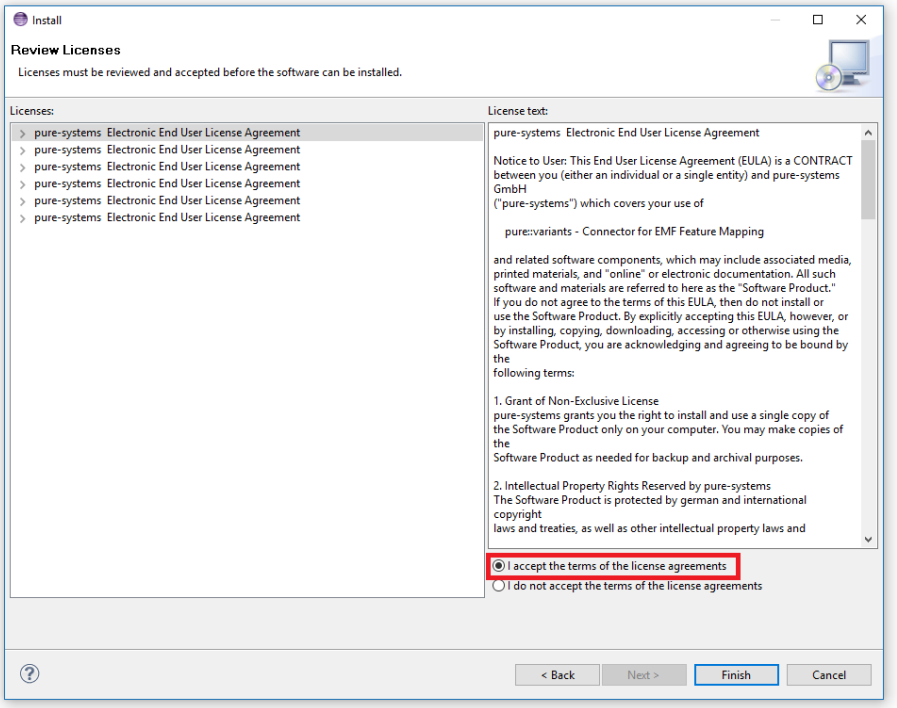
- Unfold the pure::variants update site and select all features to be updated. Select "Next".

Figure 21. pure::variants Plugin Selection



- Accept the license, hit "Next" and then "Finish".

Figure 22. Licence Agreement



- In the dialog select "Install all".
- Restart pure::variants when asked for.

If the online update is not possible (often due to firewall/proxies preventing Eclipse accessing external web sites), please go to the web site using an Internet browser:

- For pure::variants Evaluation use https://www.pure-systems.com/pv-update
- For pure::variants Enterprise/Professional use https://www.pure-systems.com/pvde-update

and download the "Complete Updatesite" archive:

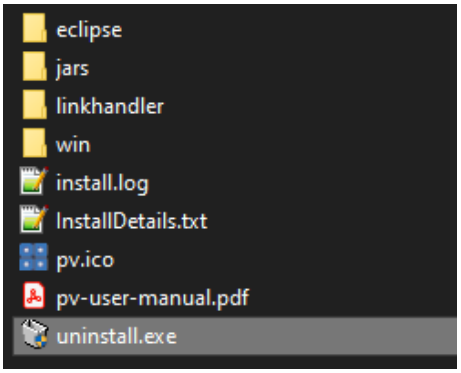
- Start pure::variants (or the Eclipse into which pure::variants has been installed).
- Select "Help"->"Software Updates"->"Find and Install...".
- Select "Search for new features to install" and "Next".
- Click on button "Archived Update Site" or "Local Update Site".
- Use "Browse" to select the downloaded archive file.
- Press "Ok". The pure::variants update site from the archive should be selected.
- All other check boxes should be unselected to speed up the process. Press "Finish".
- Unfold everything below pure::variants update site and select the features to be updated. Select "Next".
- Accept the license, hit "Next" and then "Finish".
- In the dialog select "Install all".
- Restart pure::variants when asked for.

3.3. Uninstall pure::variants Desktop Client

3.3.1. Uninstall using pure::variants Uninstaller

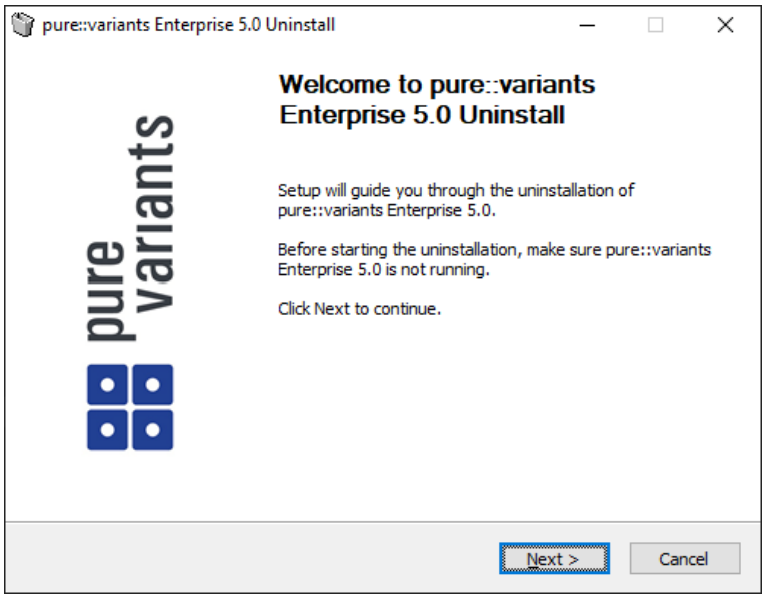
The uninstaller for the pure::variants Desktop Client can be started in two different ways. The first is to go to the Windows *Add or remove programs* application and search for *pure::variants Enterprise* and start the uninstaller by using the *Uninstall* action. The uninstaller requires Administrator privileges.

Figure 23. pure::variants Desktop Client Uninstaller



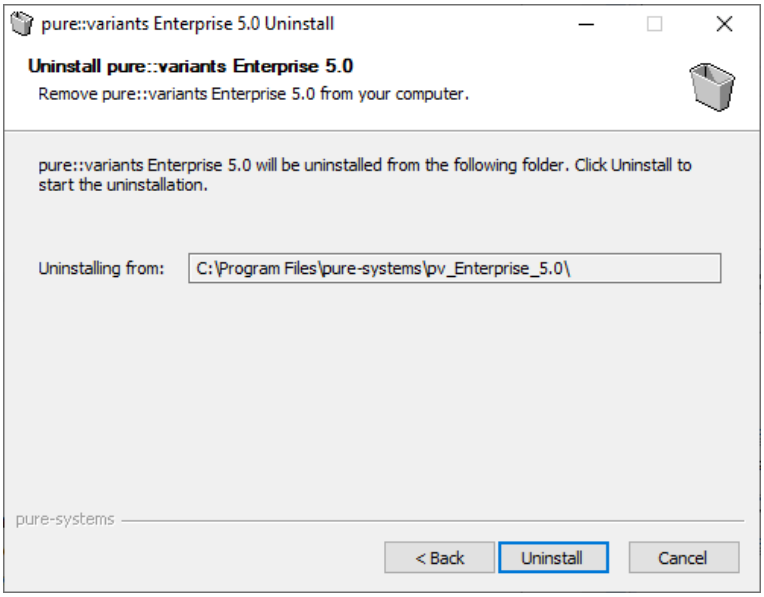
The second way is to navigate to the pure::variants Desktop Client installation folder and start the uninstaller by double clicking it.

Figure 24. pure::variants Desktop Client Uninstaller



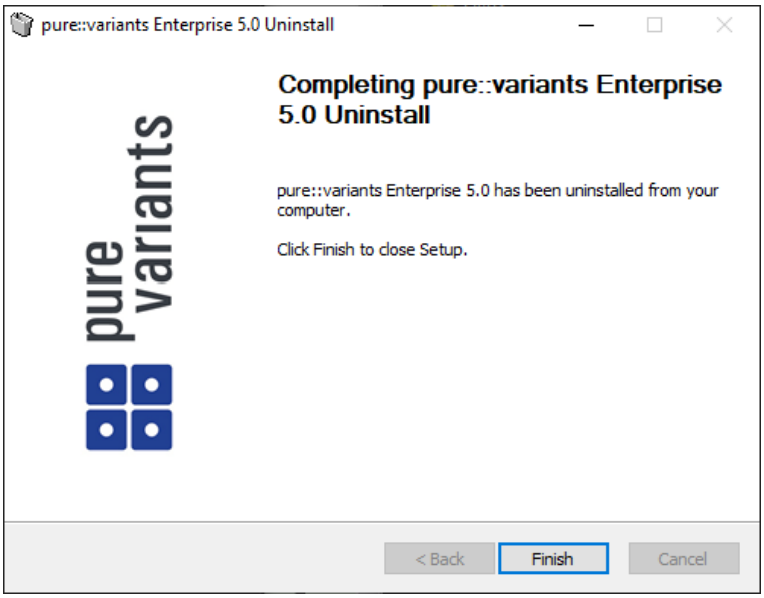
Click *Next*.

Figure 25. Uninstall from



Click *Uninstall* to start the uninstall process.

Figure 26. Completing Uninstall



Click *Finish* to close the uninstaller.

3.3.2. Uninstall pure::variants from existing Eclipse instance

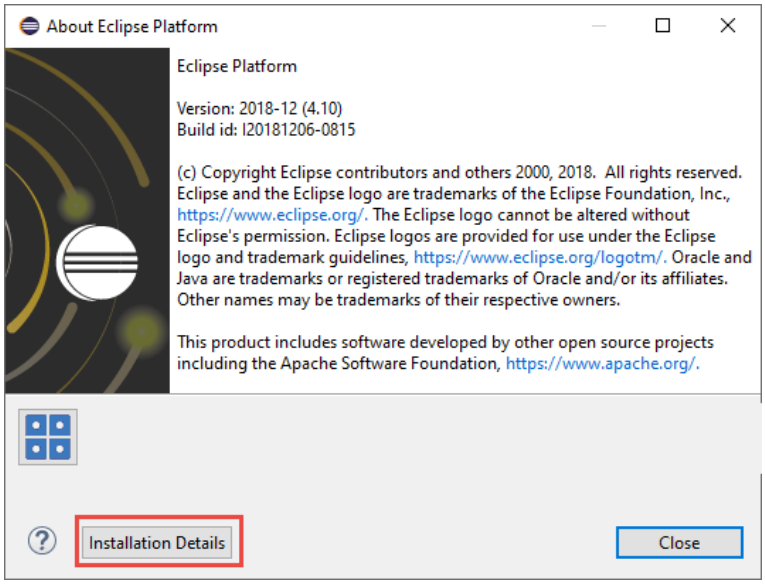
There are two ways to remove pure::variants from an Eclipse instance. You can use the Eclipse command line or remove the pure::variants features one by one in the running Eclipse Instance. Either way a cleanup of the Eclipse instance has to be performed afterwards.

If the Eclipse instance is not needed anymore you can just remove the whole Eclipse installation from the file system. If the Eclipse is of further use, follow one of the installation methods.

Uninstall pure::variants in running Eclipse Instance

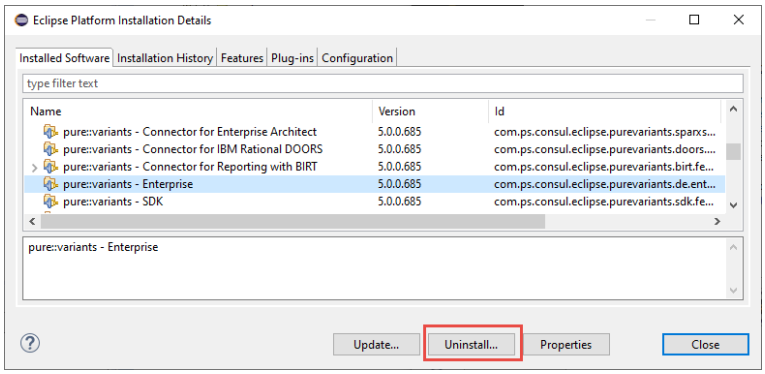
To remove an installed feature from Eclipse using the Eclipse client, open the *About Eclipse Platform* dialog with the *About Eclipse Platform* action in the *Help* menu.

Figure 27. Eclipse About Dialog



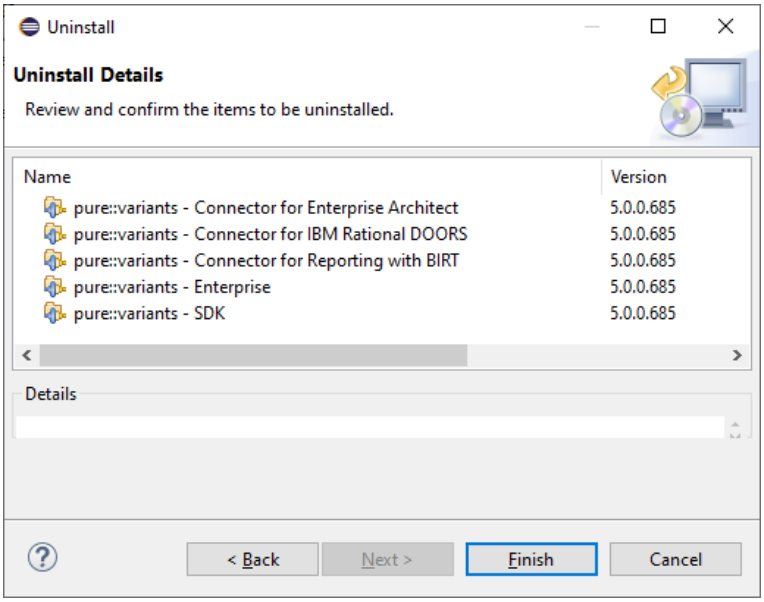
Use the *Installation Details* button to access the installation details.

Figure 28. Eclipse About Dialog



Use the *Uninstall* button to start the uninstallation of the selected features. Selecting multiple features at once is possible.

Figure 29. Eclipse About Dialog

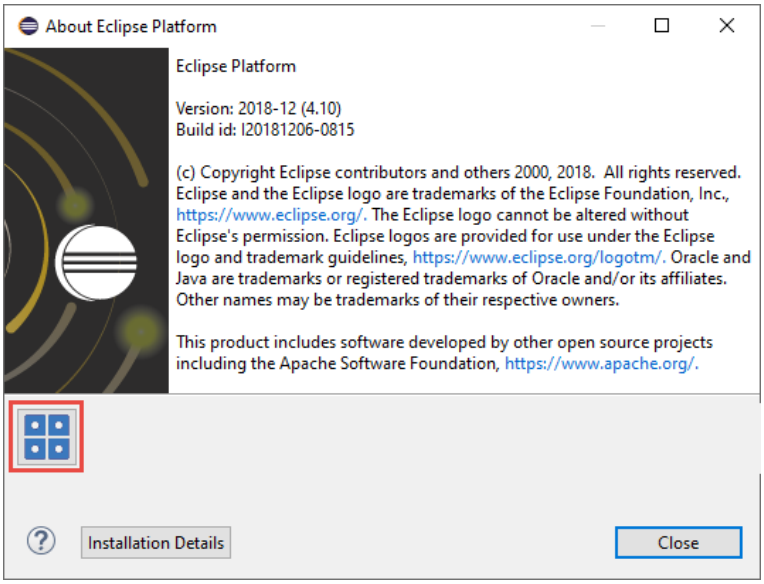


Click *Finish* to start the uninstall process. After it finished, Eclipse will prompt you to restart the application. Click *Restart* to finish the uninstallation.

Uninstall pure::variants using Eclipse uninstall application

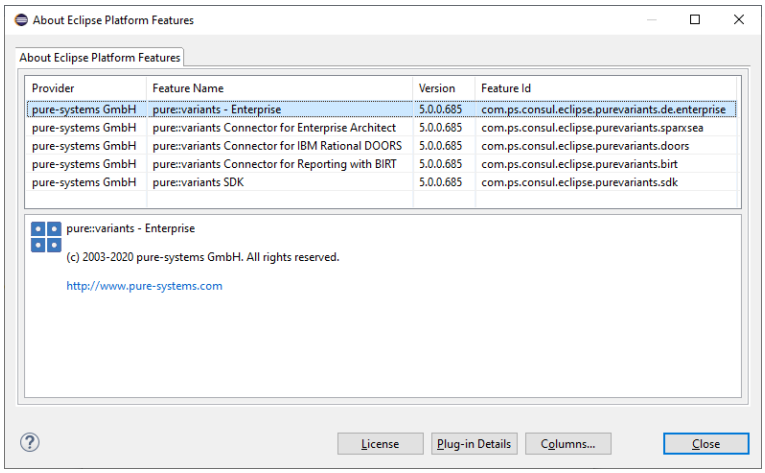
To use the uninstall application you need the feature ids of the features you want to uninstall. The feature ids can be found in the *About Eclipse Plattform* dialog. Open the dialog with the *About Eclipse Platform* action in the *Help* menu.

Figure 30. Eclipse About Dialog



Click on the pure::variants icon.

Figure 31. Installed pure::variants features



The feature ids are listed in the *Feature Id* column of the upcoming dialog. All feature ids have to be extended by ".feature.group" and are concatenated with ",". The feature id list for the example shown in the previous figure would be:

```
com.ps.consul.eclipse.purevariants.sparxsea.feature.group,com.ps.consul.eclipse.purevariants.
birt.feature.group,com.ps.consul.eclipse.purevariants.de.enterprise.feature.group,com.ps.
consul.eclipse.purevariants.doors.feature.group,com.ps.consul.eclipse.purevariants.sdk.
feature.group
```

The resulting list of feature ids is used in the following command.

```
"<Eclipse Installation Directory>\eclipsec.exe" -nosplash --launcher.suppressErrors -
application
org.eclipse.equinox.p2.director -uninstallIU "<list of feature ids>" -data "ws" -vmargs
-Dequinox.ds.block_timeout=120000
-Dorg.eclipse.ecf.provider.filetransfer.retrieve.readTimeout=120000
-Declipse.p2.mirrors=false -Xms100m -Xmx2048m -Xmnx64m -Xgcpolicy:gencon
-XX:MaxPermSize=512M -Xcompressedrefs
```

Cleanup Eclipse after uninstallation

pure::variants stores some settings, license and log files at two locations in the file system. On Windows the first one is C:\Users\<user name>\AppData\Roaming\pure-variants-6, and the second C:\ProgramData\pure-variants-6. On Linux based systems the pure-variants-6 folders are located in the users home directory and at */usr/share*. These folders should be removed after pure::variants has been completely removed from the computer.

To clean up the Eclipse instance, run the following command.

```
"<Eclipse Installation Directory>\eclipsec.exe" -nosplash --launcher.suppressErrors -
application
org.eclipse.equinox.p2.garbagecollector.application -data "ws" -vmargs
-Dequinox.ds.block_timeout=120000
-Dorg.eclipse.ecf.provider.filetransfer.retrieve.readTimeout=120000
-Declipse.p2.mirrors=false -Xms100m -Xmx2048m -Xmnx64m -Xgcpolicy:gencon
-XX:MaxPermSize=512M -Xcompressedrefs
```

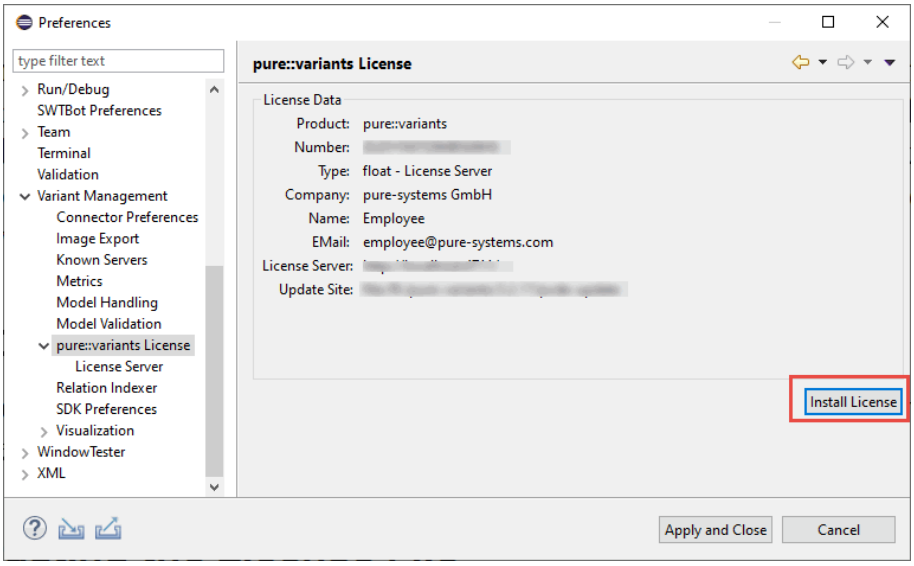
3.4. Basic Setup of the pure::variants Desktop Client

3.4.1. Setup a pure::variants Desktop Client License

A valid license file is required in order to use pure::variants. If pure::variants is started and no license is present, then the user is prompted to supply a license. Select the **Yes** button and use the file dialog to specify the license file delivered with pure::variants. The specified license will be stored in the user's application data directory. If you are using multiple workspaces then the license file has to be installed only once. The pure::variants integrations also use the installed license and thus no further setup step is needed here.

To replace an existing pure::variants license, start pure::variants and open the **Preferences** (menu Window -> Preferences). Navigate to **Variant Management -> pure::variants License** and use the **Install License** button to select the new license.

Figure 32. pure::variants License Preferences

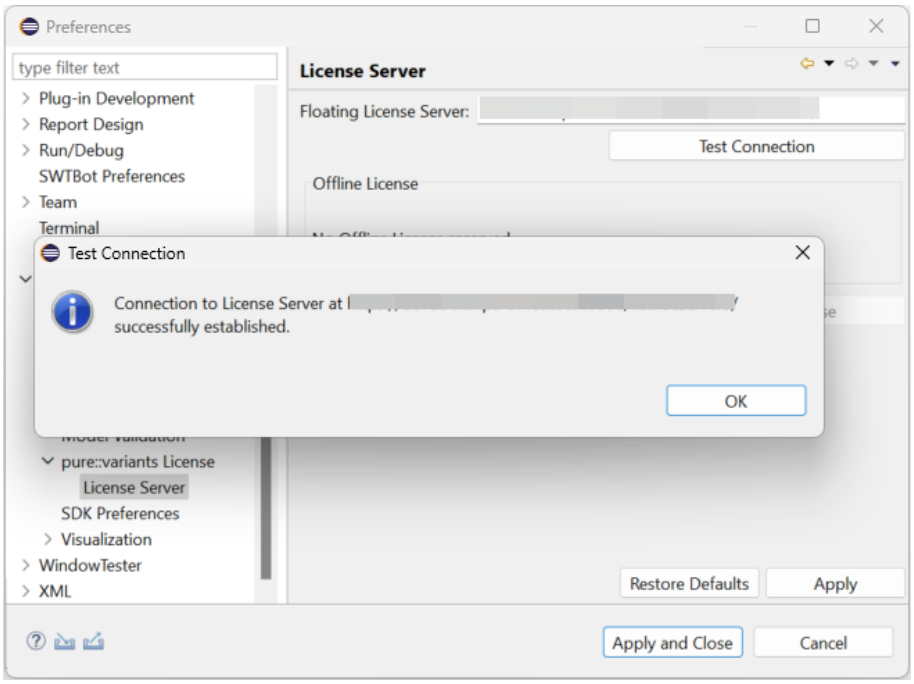


The pure::variants client license can also be defined using the *PVLICENSE* environment variable. This variable has to define the fullpath to a license file. If this variable is defined the given license file is used and the user does not need to define the license for pure::variants client instances.

The next step is necessary only if a floating license with a pure::variants license server is used. The license server URL can be provided with the floating license file, or it has to be set by the user.

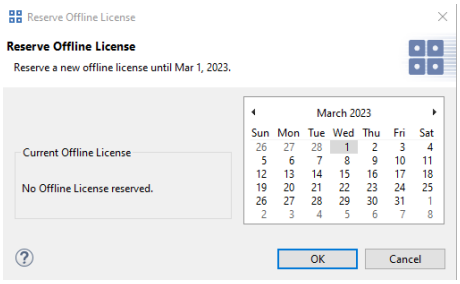
To set the license server URL, open the sub page **Variant Management -> pure::variants License -> License Server** and enter the URL of the license server into the **Floating License Server** text field, click button **Test Connection** to check that the connection is successful.

Figure 33. pure::variants License Server Preferences



Click button **Reserve Offline License** to choose how long the license shall be reserved.

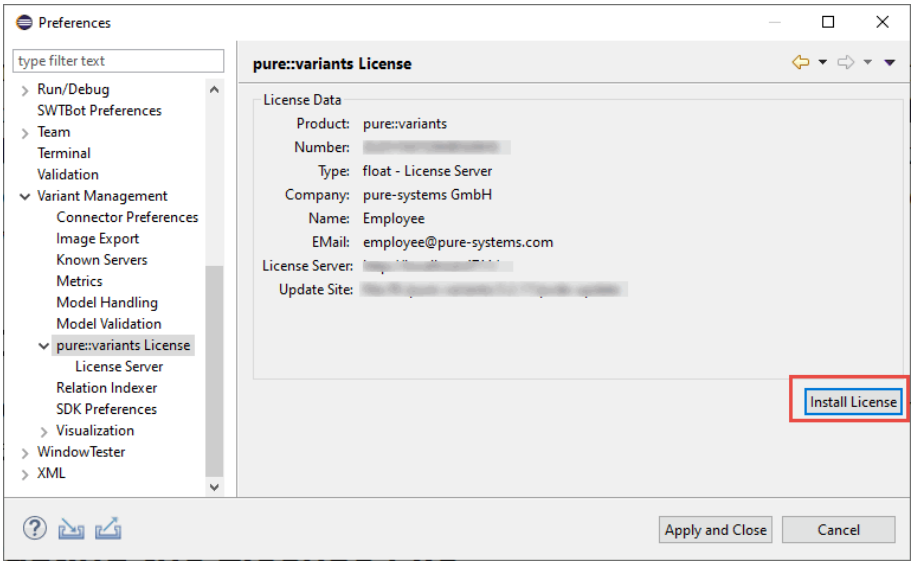
Figure 34. Reserving Offline License



3.4.2. Update a pure::variants Desktop Client License

If pure::variants is not explicitly asking for a new license, the update can be forced by starting pure::variants and opening menu **Window -> Preferences**. Select **Variant Management -> pure::variants License** and install the license using the provided **Install** button.

Figure 35. pure::variants License Preferences



3.4.3. Add pure::variants Desktop Client License using environment variable or Java property

For central or automatic deployed pure::variants Desktop Clients it may be necessary to also automatically deploy or update the pure::variants Desktop Client license. For this use case the variable **PVLICENSE** can be used. This variable can either be introduced as an environment variable or just added as a Java property to the command line starting pure::variants. If this variable is set, the given license is used instead of a possibly previously installed license.

Example for the command line parameter: *-DPVLICENSE=C:/absolute/path/to/the/license/file.lic*

3.5. Trouble Shooting

3.5.1. pure::variants is low on memory

If pure::variants is low on memory it can result in out of memory errors or causing pure::variants to run very slow since Java is trying to free up memory constantly by running the garbage collector.

To solve that problem pure::variants needs to be enabled to use more memory. This can be done by editing the eclipse.ini file, which is located in <pure::variants installation path>\eclipse\eclipse.ini.

Add the following three lines to the end of the ini file, if not existing yet. The first line tells Eclipse that there are Java Virtual Machine options following. Xms defines the minimal amount of memory Java is reserving. Xmx defines the maximum amount of memory Java is allowed to use. The default value is 1024 MB. We recommend to set the value to 6144 MB .

```
-vmargs
-Xms40m
-Xmx6144m
```

Note

If Eclipse does not start after the eclipse.ini was changed, the maximum amount of memory defined is not valid. There are multiple reasons for this, e.g. Java could not reserve enough memory. Try to decrease the defined maximum memory.

4. pure::variants Deployment for Kubernetes

The pure::variants Deployment for Kubernetes is the recommended way to install and manage the pure::variants Services in Kubernetes. This deployment reduces the complexity of installing the various components required for a successful deployment and maintenance of pure::variants. The Kubernetes deployment is based on the Helm Package Manager and includes the same services and possibilities like the pure::variants Deployment Templates for Docker. The Helm Chart can also be used for a deployment into OpenShift. The pure::variants Helm Chart for Kubernetes is included in the pure::variants Deployment Templates for Docker with the name `pv-chart-<version>.tgz`.

4.1. Deployment Architecture for Kubernetes

The Kubernetes deployment includes the pure::variants Web Client, Transform Service and Model Server. All pure::variants related services are distributed as a helm chart which can be installed with the helm package manager. In contrast to the pure::variants Deployment Templates for Docker there are no prepared template files but a configuration file for the pure::variants helm chart which is called `values.yaml`.

Internally a number of additional containers, orchestrated by Kubernetes are communicating among each other using an internal, preconfigured network structure, which is not exposed externally via ingress routes. Connections of internal services to external services not provided by the respective template, are done using encrypted access using TLS. All externally available services as per default are encrypted using TLS.

4.2. Requirements for the Kubernetes Deployment

In general all prerequisites mentioned for the pure::variants Deployment Templates for Docker are still valid

4.2.1. General Requirements

- The deployment is verified and tested with Kubernetes v1.29.4 and OpenShift 4.13.41
- Helm Package Manager with version 3.14.1 or later
- the pure::variants Docker Setup and the pure::variants Deployment for Kubernetes package from the pure::variants Updatesite
- a running pure::variants License Server and its URL which is accessible to containers running in the Kubernetes Cluster (see ???)
- Decide how many concurrent transformations shall be possible. Default in the parameters file is to have 1 Transformation Runner enabled. It is recommended to start with this and add more Runners later if needed.

- The email address and registration number from your pure::variants license. This can be retrieved from the pure::variants License Servers status page.
- A X.509 certificate and key for the hostname and port under which the pure::variant services are exposed from the docker host. The key file must not have password protection. Alternatively you can directly provide a TLS secret in the desired namespace.
- All certificates of a non-public certification authority (CA) in PEM format with `.crt` suffix which will be presented to the pure::variants services (e.g.: License Server, Model Server, Openid Connect Provider, 3rd party tools...)
- Define designated exposed hostname for the pure::variants services. The exposed port is set to 443. With this hostname users will later be able to access the pure::variants services.

4.2.2. Requirements for Single-Sign-On

- The **Web Client** registration in the Single Sign-On provider must be conform to the following specification.
  - Grant types: authorization code and refresh token
  - Redirect URL: https://pv.example.com/pvgw/auth/oidc/callback
  - Logout URL: https://pv.example.com/pvgw/ui/
  - Scopes: The default scopes are **openid profile email**.

    Note: When using the Web Client in combination with Jazz platform the additional scope **general** is needed.
  - For global configuration management introspection must be enabled and the introspecting relying parties must be added with their **Client ID** and the needed permissions to the user management of the pure::variants Model Server.
- The **Model Server** registration in the Single-Sign-On provider must be conform to the following specification.
  - Grant types: authorization code
  - Redirect URL: https://localhost/pv/openid
  - Scopes: openid

  Further instructions for the Single Sign-On Setup of the Model Server are described in ???

4.3. Building the Images

The images referenced in the pure::variants Kubernetes Deployment need to be built first. For building the images the pure::variants Deployment Templates for Docker are used. The detailed build steps are mentioned in ???.

Please provide at least the following prerequisites before running the build process:

- Set a version tag in the .env file for the images with the **PV_VERSION** variable.
- Define the target container registry in the .env file with the variable **DOCKER_REGISTRY**.
- Provide all needed root certificates within the folder `<docker-deployment-templates>/workspace/rootcer\tificates`.
- Add the parameter `PV_USERID=1000` anywhere in your .env file provided in the pure::variants Deployment Templates for Docker. This will enable the creation of an user in the transformation runner image.

Once the build is done please run `docker compose --profile build push` to push the images to your container registry.

4.4. Configuring the Kubernetes Deployment

Opening the extracted pure::variants Helm Chart the contained **`values.yaml`** file will contain all parameters to configure the pure::variants Deployment.

To keep the complexity manageable not all parameters available are directly visible in the parameters file but defaulted to a meaningful value which matches common customer needs.

The parameters file is divided into sections for each pure::variants service and one global section which applies to all services. These sections are represented in the following subchapters.

4.4.1. global

The global parameters are applied to all services installed with the pure::variants Deployment for Kubernetes.

- **PV_VERSION**

  Enter the pure::variants version label that you want to deploy. Technically this is used as tag for the images which are stored in your container registry and deployed to the Kubernetes cluster.

- **PV_NAMESPACE**

  Enter your name of your namespace to which you want to deploy the pure::variants services to.

  Note: This namespace needs to be created manually by running `kubectl create ns NameOfNamespace`

- **PV_SERVICE_ACCOUNT**

  Name of the service account used to access Kubernetes API to create/delete and inspect containers, persistent volume claims and network policies.

- **DOCKER_REGISTRY**

  Enter the address of the Container registry. This address must have a trailing slash, e.g. registry.example.com/

- **IMAGE_PULL_SECRETS**

  If you underlying container runtime cannot directly pull the images from the container registry, you can provide an array of pull secrets to access the container registry.

  Note: The mentioned pull secrets need to be created by you and are not part of the pure::variants Kubernetes deployment.

- **PV_REG_NUMBER**

  Enter your pure::variants license resp. registration number, which you for instance can find in your pure::variants license file. This number is used to access the floating licenses on the pure::variants License Server.

- **PV_LICENSE_SERVER**

  Enter the address of the pure::variants License Server, e.g. https://pvlicenses.example.com.

- **PV_EXPOSED_HOST**

  This option defines the base network address used to access the pure::variants services. Enter the full name of the host, e.g. `pv.example.com`

- **RBAC_ENABLED**

  If set to `true` automatically creates a role and role binding for the service account.

- **TZ**

Replace the default time zone `Europe/Berlin` with your time zone. This information is used to let the containers have the correct time according to your location. Please see https://en.wikipedia.org/wiki/List_of_tz_database_time_zones for the available time zones.

- **PV_INGRESS_CLASSNAME**

  Optional: You can set the name of the ingress class which shall be used for the ingress resources. If no name is provided it defaults to the `default` ingress class.

- **PV_SECRET_NAME**

  Optional: If you prefer to provide a TLS secret by yourself you can specify the name of the secret in the namespace used to encrypt the communication to the pure::variants services.

- **PV_SSL_CERTIFICATE**

  Path to the server certificate used to encrypt the communication to the pure:.variants services. Defaults to `workspace/certificates/server.crt` therefore the server certificate must be placed in this location of the helm chart.

- **PV_SSL_KEY**

  Path to the server certificate key used to encrypt the communication to the pure:.variants services. Defaults to `workspace/certificates/server.key` therefore the server certificate must be placed in this location of the helm chart.

4.4.2. webclient

- **enabled**

  Parameter to enable or turn off the Web Client in this pure::variants Deployment for Kubernetes. Defaults to `true` and therefore the Web Client is part of this deployment.

- **PV_MODEL_SERVER**

  Enter the address of the pure::variants Model Server to use, e.g. `https://pv.example.com/pv/`.

  Note: Can be kept empty if the Model Server is also deployed with this pure::variants Deployment for Kubernetes.

- **PV_WEB_CLIENT_SSO_GC_URI**

  Leave this option empty if you don't use Global Configurations nor OpenID Connect to authenticate. Otherwise enter the address of the Global Configuration service, e.g. https://jazz.example.com:9443/gc.

- **PV_WEB_CLIENT_HIDE_INACCESSIBLE_PROJECTS**

  If set to true, all projects which are not accessible for the user are completely hidden instead of greyed out in the projects overview. Defaults to `false`.

- **PV_WEB_CLIENT_LOGLEVEL**

  Change the log level of the web client to increase or lower logging information. Defaults to `INFO`. Available options are: `INFO, DEBUG, TRACE`

- **PV_TRANSFORM_STORAGE_SIZE**

  Define the size of the requested persistent volume claim which stores all transformation job information. Defaults to `25Gi`

- **PV_RAM_MAX**

  Define the maximum heap memory which Java process can acquire inside the web Web Client container. The value is defined as Megabytes and defaults to: `16389`

- **PV_WEB_CLIENT_RESOURCES**

  Please define how much CPU and Memory the POD should request and be limited to. The default values are commented out and therefore no resource definitions are set on POD level.

  Note: If you define resource specifications please make sure to align to our hardware requirements see ???

4.4.3. gateway

- **PV_GATEWAY_SSO**

  Enable OIDC as an authentication method for the pure::variants Gateway. Enabling OIDC automatically disables form based authentication method.

- **PV_GATEWAY_SSO_NAME**

  Provide a label for the configured Single-Sign-On provider. This name is used as a label of the OIDC button on the login page.

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_SSO_URL**

  Enter the well-known endpoint of the OpenID Connect provider without the trailing /.well-known/openid-configuration. Example: `https://jazz.example.com:9643/oidc/endpoint/jazzop`

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_SSO_CLIENT_ID**

  Enter the OpenID Connect client identifier for the pure::variants Web Client as defined at the OpenID Connect provider.

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_SSO_CLIENT_SECRET**

  Enter the OpenID Connect client secret for the pure::variants Web Client as defined at the OpenID Connect provider.

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_SSO_USERID_CLAIM**

  Optionally enter the claim which is used for the OpenID Connect token to identify the user id.

  Options are `idtoken:<claim>` or `userinfo:<claim>` where `<claim>` needs to be replaced with the name of the claim

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_SSO_SCOPES**

  Change this option only if a different set of token scopes are needed. The default scopes requested are: `openid profile email`).

  Note: Only needed if OpenID Connect login is enabled.

- **PV_GATEWAY_ADDITIONAL_ANNOTATIONS**

  The pure::variants Gateway is exposed via an ingress route. If your environment or ingress controller needs a specific annotation please add it below this parameter with an array annotation.

4.4.4. runner

- **enabled**

  If set to `true` the transformation runners, which are handling transformations triggered in the Web Client, are also deployed with this pure::variants Deployment for Kubernetes.

- **PV_RUNNER_LOGLEVEL**

  Define the log level of the transformation runner to increase the logging entries. Available options are `1 to 7` and defaults to `1`.

- **PV_RUNNER_RESOURCES**

  Please define how much CPU and Memory the POD should request and be limited to. The default values are commented out and therefore no resource definitions are set on POD level.

4.4.5. runnercredentials

The configured transformation runner is authenticating against the Web Client to ensure no information is passed to non authorized transformation runners.

To add more runner please add more entries to the `values.yaml` parameter file.

```
runnercredentials:
  # to create multiple runners copy the following line like the example below:
  PV_RUNNER_CREDENTIALS_01: '{"id":"runner-01","apikey":"changeit"}'
  PV_RUNNER_CREDENTIALS_02: '{"id":"runner-02","apikey":"changeit"}'
```

4.4.6. executor

The transformation executor is a container with a temporary lifespan. It is created on demand to execute the triggered transformation and is deleted once the transformation is finished.

- **PV_CPU_MIN**

  Define the requested amount of CPUs for the transformation POD.

- **PV_CPU_MAX**

  Define the limit of CPUs the transformation POD can obtain.

- **PV_RAM_MIN**

  Define the requested amount of Memory for the transformation POD.

- **PV_RAM_MAX**

  Define the limit of Memory the transformation POD can obtain.

Note: If none of the resource variables are set the transformation executor will not set any resource dependencies to its POD definition. If you choose to define resource dependencies please align them to our requirements see Section 2.1, “pure::variants Desktop Client”

4.4.7. modelserver

- **enabled**

  If set to `true` the pure::variants Model Server is deployed with this pure::variants Deployment for Kubernetes.

- **PV_MODEL_SERVER_WEB_PASSWORD**

  Enter the password with which you want to protect the status web page of the pure::variants Model Server. Defaults to `changeit`.

- **PV_MODEL_SERVER_APIKEY**

  Enter the API key with which you want to protect the API endpoints of the pure::variants Model Server.

- **PV_MODEL_SERVER_SYSTEMUSER_PASSWORD**

  Enter the password with which you want to protect the pure::variants Model Server superuser system.

  Note: This setting has no effect if an external database is connected to the Model Server.

- **PV_MODEL_SERVER_DB_PASSWORD**

  Enter the password with which you want to protect the local pure::variants Model Server PostgreSQL database or to connect to your external database.

- **PV_MODEL_SERVER_LOGLEVEL**

  Log level of the pure::variants Model Server, from 0 (just errors) to 9 (extensive logging). Defaults to level 1 for minimal logging.

- **PV_MODEL_SERVER_OPENID_CLIENT_ID**

  Client identifier issued to the pure::variants Model Server by the OpenID Connect provider.

  Note: Only relevant if the Model Server shall be connected with the Single-Sign-On provider.

- **PV_MODEL_SERVER_OPENID_CLIENT_SECRET**

  Client secret assigned to the pure::variants Model Server by the OpenID Connect provider.

  Note: Only relevant if the Model Server shall be connected with the Single-Sign-On provider.

- **PV_MODELSERVER_RESOURCES**

  Please define how much CPU and Memory the POD should request and be limited to. The default values are commented out and therefore no resource definitions are set on POD level.

  Note: If you define resource specifications please make sure to align to our hardware requirements see ???

If you want to connect the Model Server to an **external database** please remove the `#` in front of the following parameters and provide a reasonable value. The Model Server provided with the pure::variants Deployment for Kubernetes shares the same requirements for its database as our standalone Model Server please see ???. Please note when using an external database, the internal PostgreSQL database needs to be disabled by changing the value of `database.enabled` to `"false"`.

- **PV_MODEL_SERVER_DB_TYPE**

  Provide the type of database the Model Server shall connect to. Options are: `PostgreSQL`, `MSSQL` and `Oracle`.

- **PV_MODEL_SERVER_DB_HOST**

  Enter the hostname which provides access to the external database.

- **PV_MODEL_SERVER_DB_PORT**

Enter the port with which the external database can be accessed.

- **PV_MODEL_SERVER_DB_NAME**

  Enter the name of the external database which has been initialized with the init SQL script of the Model Server.

- **PV_MODEL_SERVER_DB_USER**

  Enter the name of the technical user having access to the external database.

4.4.8. database

This section provides parameters to configure the pure::variants Deployment for Kubernetes internal PostgreSQL database for the Model Server.

Note: For productive usage we highly recommend to use an external/managed database.

- **enabled**

  If set to `true` the internal PostgreSQL database is deployed with this pure::variants Deployment for Kubernetes.

- **POSTGRES_SIZE**

  Provide a size which is requested by the persistent volume claim associated with the database statefulset. Defaults to `25Gi`.

- **PV_DATABASE_RESOURCES**

  Please define how much CPU and Memory the POD should request and be limited to. The default values are commented out and therefore no resource definitions are set on POD level.

4.5. Installing the Deployment

Once the configuration is done and your images are available on the container registry you can proceed installing your pure::variants Deployment for Kubernetes.

For an installation from scratch please navigate into your helm chart and run the following command which will also create the desired namespace if it does not exist yet.

Note: Please make sure to replace `<name_of_deployment>` with the name helm should list your pure::variants Deployment for Kubernetes.

Also replace `<namespace>` with the desired namespace in Kubernetes which is defined in the parameter PV_NAMESPACE.

```
helm install <name_of_deployment> -n <namespace> --create-namespace .
```

If the command has been successful your console Output should show the following message:

```
NAME: <name_of_deployment>
LAST DEPLOYED: <Timestamp>
NAMESPACE: <namespace>
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Once the installation via helm is done you can verify the deployment by running:

```
kubectl get all -n <namespace>
```

You should see your configured pods are pulling their images and starting up. Once the Pod initialization is done all Pods should have the status `Running`.

4.6. Validate Correct Operation of Deployment

4.6.1. Model Server Operation

1. Validate Model Server Availability: Use browser to navigate to Model Server URL (make sure to have the slash at the end of the URL), e.g., to https://pv.example.com:443/pv/ . You should now see the basic status page of the Model Server. Try to log in with the password defined in `PV_MODEL_SERVER_WEB_PASSWORD`.

2. Validate Model Server Access via Desktop Client: Follow the instructions given in the *pure::variants Model Server Administration Manual* using the Model Server URL (make sure to have the slash at the end of the URL), e.g., https://pv.example.com:443/pv/ . Create at one pure::variants user with a name matching a valid Single-Sign-On user as described in the manual.

4.6.2. Web Client Operation

1. Validate Web Client Availability: Use browser to navigate to Web Client URL (make sure to have the slash at the end of the URL), e.g. to https://pv.example.com/pvweb/ . You should now see the Login page of the Single-Sign-On service. Try to log in with valid user name and credentials from the previous step. Validate Number of available Transformation Runners by switching in the Web Client to the Transformation Dashboard and check that the Idle count is equal to the number of configured Transformation Runners (Default is 1).

5. pure::variants Connectors

5.1. Installation of pure::variants Connectors

Installing a connector into an existing pure::variants installation works the exact same way like installing the pure::variants Desktop Client into an exsiting Eclipse instance. You just have to make sure the depemding pure::variants connectors are already installed or they have to be installed together with the new connector. See the section called “Using update site”.

5.2. pure::variants Connector for Capella

To install the pure::variants Connector for Capella, open Capella or Capella Studio and select **Help**->**Install New Software...**. Enter the address of your pure::variants update site. From the list of available features, select "pure::variants - Connector for Capella", "pure::variants - Connector for EMF Feature Mapping" and the pure::variants feature (e.g., "pure::variants - Enterprise").

For installation in Capella, the standard Eclipse update site needs to be set up. Otherwise the installation will fail due to missing dependencies. Depending on the Capella version, a different Eclipse update site needs to be added. To do that, open **Window**->**Preferences**->**Install/Update**->**Available Software Sites** and add the update site.

- For Capella 5.x use https://download.eclipse.org/releases/2020-06/
- For Capella 6.x use https://download.eclipse.org/releases/2021-06/
- For Capella 7.x use https://download.eclipse.org/releases/2023-03/

5.3. pure::variants Connector for Team Foundation Server

For using the pure::variants Integration for Microsoft TFS the server has to be prepared and the pure::variants Integration has to be installed.

The work item types, which should be aware of variability information, must be configured with additional attributes. These attributes can be pvRestriction, pvConstraint, pvDefaultSelected and pvName. At least, the attribute pvRestriction should be created (as shown in Figure 36, “A XML configuration for pvRestriction field.”).

Figure 36. A XML configuration for pvRestriction field.

```
<FIELD name="pvRestriction" refname="PureSystems.Restriction" type="PlainText" />
```

Administrator: For having support while defining restrictions on work items, the control type for the pvRestriction attribute must be configured with "PVRestrictionEditorControl" (see Figure 37, “A XML configuration for pvRestriction field's control type.”).

Figure 37. A XML configuration for pvRestriction field's control type.

```
<Control FieldName="PureSystems.Restriction" Type="PVRestrictionEditorControl" Label="pvRestriction" LabelPosition="Left" />
```

5.4. pure::variants Connector for PTC Integrity

The pure::variants Connector for PTC Integrity as well as the pure::variants Integrity Integration expect a variant-related preparation of the solution item. For this several changes on the solution and settings are necessary, which are described in the following.

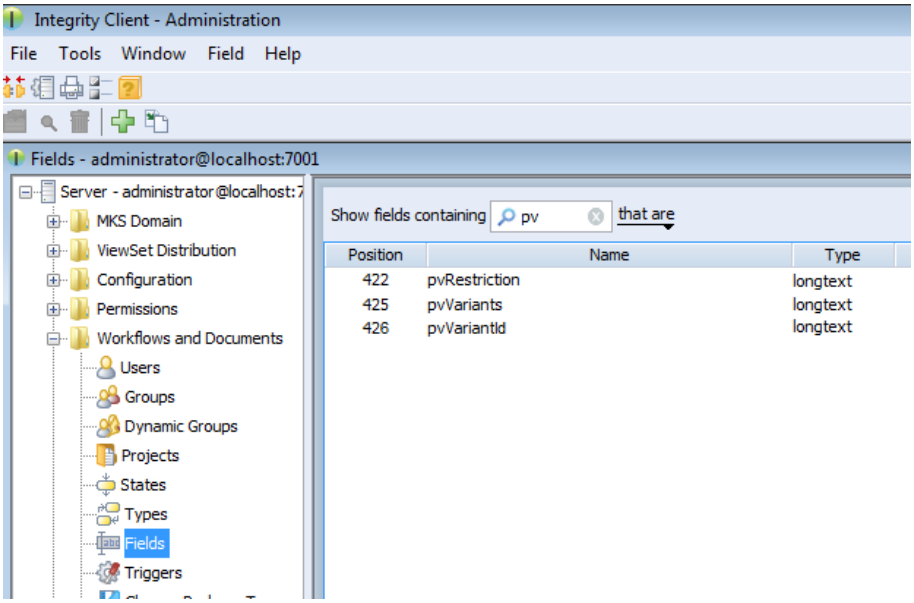
5.4.1. Add additional Fields for pure::variants

The following fields have to be created for the solution item, e.g. **MKS Solution**. In the PTC Integrity Administration open **Workflows and Documents** -> **Fields**. Add the fields by choosing **Create Field...** from the context menu.

Table 2. Additional fields

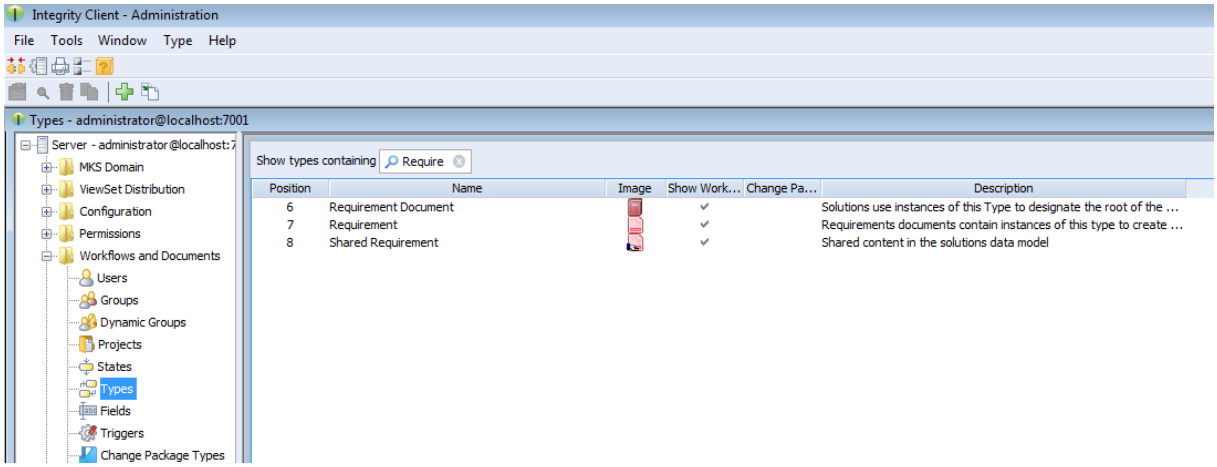
Field	Description
pvRestriction	This field is needed to store the restriction rule on a requirement. Set **longtext** as the type of the field, and ensure that the field is editable.
pvVariants	This field is needed to store the names of variants a requirement is part of. Set **longtext** as the type of the field. Ensure that the field is editable.
pvVariantId	This field is needed to store the hexadecimal encoded ID of the pure::variants variant description model which was used to create a requirement document variant in PTC Integrity. Set **longtext** as the type of the field. Ensure that the field is editable.

Figure 38. Added fields



Open **Workflows and Documents** -> **Types** in the PTC Integrity Administration and filter for all **Requirement** types.

Figure 39. Solution Types

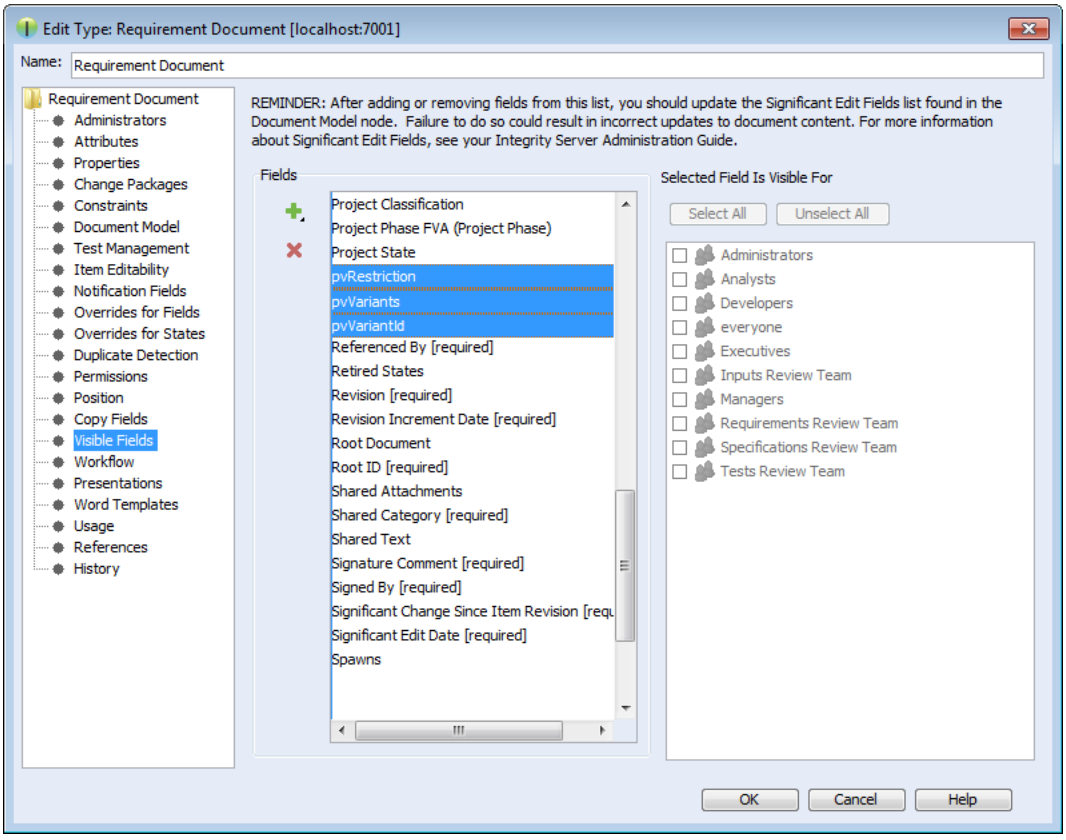


Add the new fields as **Visible Fields** to the following types.

Table 3. Types to add the fields for

Type	Fields to add
Requirements Document	pvRestriction, pvVariants, pvVariantId
Requirement	pvRestriction, pvVariants
Shared Requirement	pvRestriction

Figure 40. Fields added to type Requirement Document



5.4.2. Change Connector and In-Tool Integration Settings

pure::variants uses following settings in order to connect to PTC Integrity. This includes settings for the pure::variants Connector for PTC Integrity, which allows importing and exporting documents, as well as the settings for the In-tool Integration, which allows adding and changing restriction rules in PTC Integrity.

Table 4. Settings

Setting	Default Value	Description
Solution Item	MKS Solution	Used to get configuration properties
Document Type Field	Type	Used for the export of variant documents
Project Field	Project	Used for the export of variant documents
Document Fields	Document Short Title	Used to get several information from imported requirement documents, and to copy fields when exporting variant documents. The first field must always be the document title field
Document Title Field	Document Short Title	Used to get the document title from imported requirement documents, and to calculate the title of exported variant documents
Restriction Rule Field	pvRestriction	Used to get restriction rules from imported requirement documents, and to read and write restriction rules
Variant Enumeration Field	pvVariants	Used to store enumerated variant names in requirement documents
Variant Document ID Field	pvVariantId	Used to store the ID of pure::variants variant description models in variant requirement documents
Requirement Text Field	Text	Used while import to get the text of requirements from requirement documents

Some of these settings can be directly changed before importing and exporting requirement documents. Others can only be changed in the connector configuration file and in the solution type. The following table shows the property names used to change these settings in the configuration file and the solution type.

Table 5. Properties

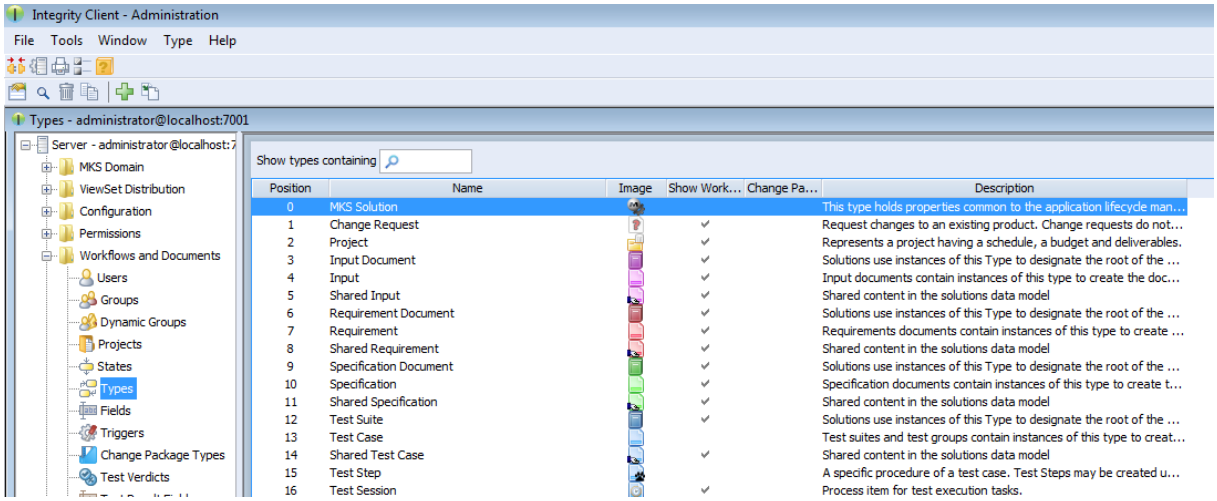
Setting	Config File Property	Solution Type Property
Solution Item	solutionType	
Document Type Field	fieldname.documenttype	PUREVARIANTS.TYPE.FIELD
Project Field	fieldname.documentproject	PUREVARIANTS.PROJECT.FIELD
Document Fields		PUREVARIANTS.VARIANT.FIELDS
Document Title Field	fieldname.documenttitle	PUREVARIANTS.TITLE.FIELD
Restriction Rule Field	attrname.restrictions	PUREVARIANTS.RESTRICTION.FIELD
Variant Enumeration Field	attrname.variants	PUREVARIANTS.VARIANTS.FIELD
Variant Document ID Field	fieldname.variantid	PUREVARIANTS.VARIANTID.FIELD
Requirement Text Field	fieldname.requirementtext	PUREVARIANTS.TEXT.FIELD

To change these settings in the connector configuration file, create the file **pvIntegrity.properties** in directory **%APPDATA%\pure-variants-6**. For each setting to change, add a line with the config file property of the setting assigned to the new value. To change for instance the default field used for storing restriction rules to **MyRestriction** and the default field used for storing the enumerated variants to **MyEnumeratedVariants**, you would add the following two lines to the configuration file (the comments are optional):

```
# Field used to store restriction rules
attrname.restrictions=MyRestriction
# Field used to store enumerated variant names
attrname.variants=MyEnumeratedVariants
```

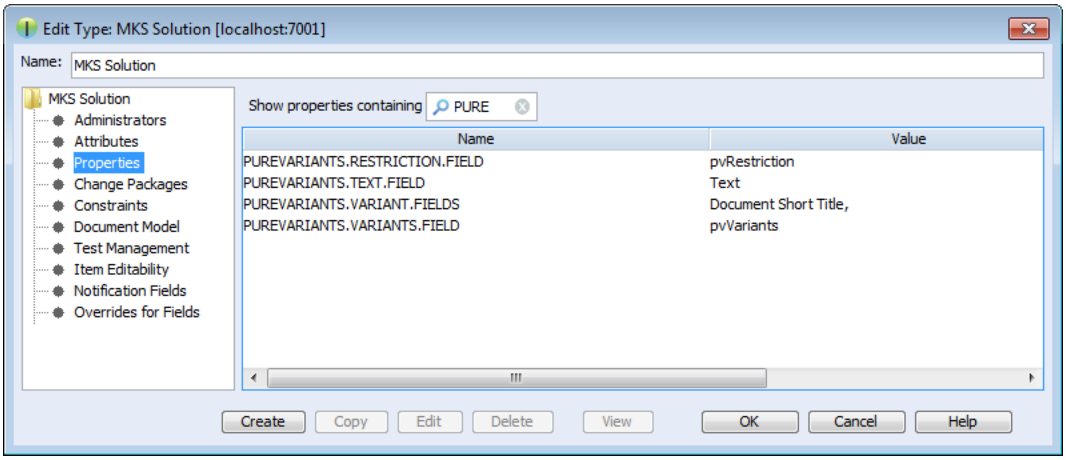
To change these settings on the solution item, open the Administration of PTC Integrity. Switch to **Workflows and Documents** -> **Types** and select the solution type (e.g. **MKS Solution**).

Figure 41. Type MKS Solution



Right-click the solution type and choose **Edit Type** from the context menu. Then switch to **Properties**, click the **Create** button and enter the solution type property name of the setting you want to change as name. Add the new default value as value and click **OK**.

Figure 42. Added Properties

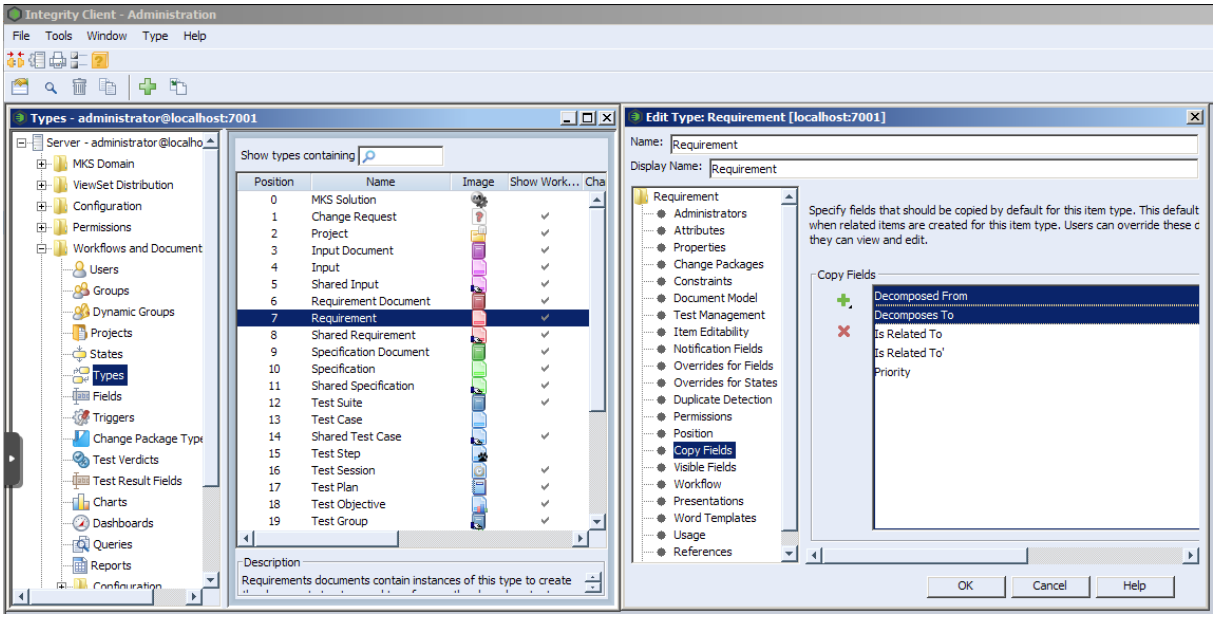


5.4.3. Change Fields Copied for Variant Creation

Several fields of the original document are copied while creating a document variant. This includes the fields mentioned in section Section 5.4.1, “Add additional Fields for pure::variants” but also some fields that are copied by default. The list of fields that are to be copied by default can be configured by the Integrity administrator.

To additionally copy for instance decompose relationships, the administrator has to open the Administration of PTC Integrity. Then switch to **Workflows and Documents** -> **Types** and edit type **Requirement**. Open the **Copy Fields** list and add the fields **Decomposes To** and **Decomposed From**. This way fields could be added for the types **Requirements Document**, **Requirement**, and **Shared Requirement**.

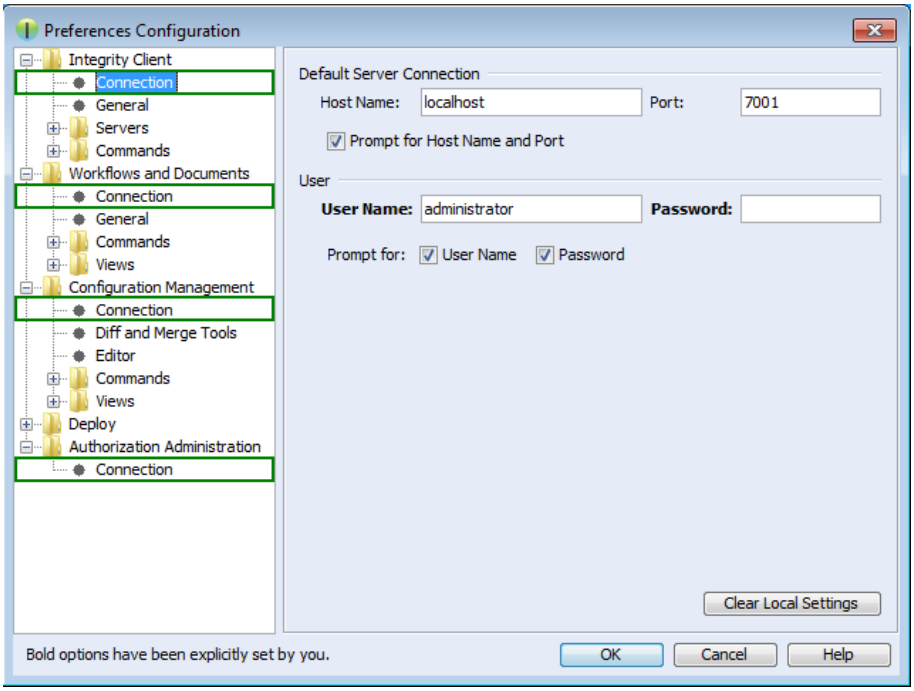
Figure 43. Decompose Fields added



5.4.4. Enable PTC Integrity Client Access

The connector and the integration for PTC Integrity require access to the Integrity client in order to work. Start the PTC Integrity client and open menu **File** -> **Preferences**. For the entries **Integrity Client**, **Workflow and Documents**, **Configuration Management**, and **Authorization Administration** click the **Connection** section and enable **Prompt for Host Name and Port**, prompt for **User Name**, and prompt for **Password**.

Figure 44. pure::variants Credentials



Note

These connection settings always are used even if you have the option "Prompt for Host Name and Port" enabled and change the connection settings in the corresponding dialog when importing documents from Integrity into pure::variants or exporting variants to Integrity.

5.5. Connector for IBM Rational Rhapsody

If you are working with Rhapsody Model Manager (RMM) projects, additional setup steps are needed.

First, you need to make sure that all required software is installed. That includes RMM (Architecture Management) on the Jazz server and Rational Team Concert (RTC) on your client machine. Also in Rhapsody, the Rhapsody Model Manager add-on needs to be installed.

5.5.1. Preparing IBM Rational Team Concert

In RTC two integrations need to be installed: The *IBM Rational Rhapsody integration for Rational Team Concert* and the *pure::variants Integration for RTC transformation*. The IBM Rational Rhapsody integration for RTC is needed for RTC to work with RMM projects. Please consult the RTC documentation for installation instructions.

The *pure::variants Integration for RTC transformation* is needed during transformation of RMM projects. Without it, the transformation will fail. To install the pure::variants Integration for RTC transformation, please open RTC and use **Help** > **Install New Software** to install all contents of archived update site `com.ps.consul.eclipse.rtc.integration.feature_[version].zip`. You can find the archived update site zip in your Rhapsody integration installation folder (default is `C:\Program Files\pure-sys\tems\pv_Enterprise_6.0\com.ps.consul.eclipse.ui.rhapsody.integration`).

5.5.2. Preparing pure::variants

For the transformation of RMM projects to work, you still need to set the RTC executable location. You can do this in the pure::variants preferences at **Window** > **Preferences** > **Variant Management** > **Connector Preferences** > **Connector for IBM Rational Rhapsody**. Alternatively, you can define an environment or Java system variable that is named **PV_RTC_EXEC_PATH** and whose value points to the RTC executable location.

When using Rhapsody 9.0 or above, it is possible to run a transformation of RMM projects in offline mode. This means that the transformatin is carried out without starting RTC/EWM client. At the above mentioned preference page, you can select the checkbox to enable this feature. Alternatively, you can set an environment or Java system variable that is named **PV_RHAPSODY_RMM_OFFLINE_MODE** to true to set the offline mode.

Additionally, you can set a custom location of RTC/EWM's eclipse.ini and lscm.bat files. You can set an environment or Java system variable that is named **PV_RTC_INI_PATH** to set the location of the eclipse.ini file and **PV_RTC_LSCM_BAT** to set the location of the lscm.bat file. If this is not set, the trasformation will look for it in the default location.

5.6. Connector for Codebeamer

This chapter describes installation instructions specific to the Codebeamer connector.

5.6.1. Installation of pure::variants Desktop Client

Follow the steps as described in ‘3.1. Install pure::variants Desktop Client’, in case installing pure::variants into an eclipse client, see chapter ‘3.1.2. Install into an existing Eclipse’.

5.6.2. Installation of Server Component and pure::variants Widget to Codebeamer

Following components that are delivered as part of the pure::variants Enterprise installation package need to be installed on the Codebeamer server:

1. The pure::variants server component for Codebeamer to be deployed on the Codebeamer server.

   The Jar files are packed in a zip file **com.ps.consul.codebeamer.vel.jar-<*version*>.zip** indicating the compatible pure::variants version for identification.

2. The *'pure::variants Widget to Codebeamer'* that needs to be deployed on the Codebeamer server is packed in a zip archive **com.ps.consul.codebeamer.pvwidget-<*version*>.zip**.

The server component is a Spring based custom component running in the application context as defined for Codebeamer (for details see https://codebeamer.com/cb/wiki/18830).

Before deploying, unzip the jar files 'com.ps.consul.codebeamer.vel.jar' and 'pvcore.jar' from the zip archive of the server component. Also make sure the server certificate that is used by the Codebeamer server is trusted on the pure::variants Desktop Client side.

Similarly, the folder 'pv_integration' contained in the zip for the to the pure::variants Widget needs to be unzipped.

> **Note**
> The above steps are required only for Codebeamer versions up to 2.2.0.0. More details on how to enable the pure::variants widget in Codebeamer, starting from version 2.2.0.0, can be found in the following link: Enabling pure::variants Widget in Codebeamer.

5.6.3. Installation without running in a docker container

Follow the steps to install:

1. Stop the Codebeamer server
2. Copy 'com.ps.consul.codebeamer.vel.jar' and 'pvcore.jar' found in the zip archive to <codebeamer>/tomcat/webapps/cb/WEB-INF/lib
3. Copy following files included in 'pv_integration/widget' to following locations:

   'pv_integration/widget' to <codebeamer>/tomcat/webapps/pv-widget/, e.g. /home/appuser/codebeamer/tomcat/webapps/pv-widget/
4. Restart the Codebeamer server
5. In Codebeamer, add following the "externalWidgetExtensions" section to the Application Configuration (https://<codebeamer>/sysadmin/configConfiguration.spr) as System Administrator:

```
    ...},
    "externalWidgetExtensions" : {
        "uri" : "https://<codebeamer>/pv-widget/extension.json"
    }
}
```

> **Note**
> The above steps are required only for Codebeamer versions up to 2.2.0.0. More details on how to enable the pure::variants widget in Codebeamer, starting from version 2.2.0.0, can be found in the following link: Enabling pure::variants Widget in Codebeamer.

5.6.4. Installation in a docker image

When running Codebeamer server in a docker container (https://codebeamer.com/cb/wiki/5562876), following additional information needs to be defined in the docker compose configuration file:

```
volumes:
      -./com.ps.consul.codebeamer.vel.jar:<codebeamer>/tomcat/webapps/ROOT/WEB-INF/lib/
com.ps.consul.codebeamer.vel.jar
      -./pvcore.jar:<codebeamer>/tomcat/webapps/ROOT/WEB-INF/lib/pvcore.jar
e.g.
- ./libs/com.ps.consul.codebeamer.vel.jar:/home/appuser/codebeamer/tomcat/webapps/ROOT/WEB-
INF/lib/com.ps.consul.codebeamer.vel.jar
- ./libs/pvcore.jar:/home/appuser/codebeamer/tomcat/webapps/ROOT/WEB-INF/lib/pvcore.jar
```

To deploy the 'pure::variant Widget to Codebeamer' following additional information needs to be added to the docker compose configuration file:

```
volumes:
```

```
      - ./pv_integration/widget:<codebeamer>/tomcat/webapps/pv-widget/

e.g.
- ./pv_integration/widget:/home/appuser/codebeamer/tomcat/webapps/pv-widget/
```

Then follow the steps to install:

1. Shut down the docker container first
2. Copy 'com.ps.consul.codebeamer.vel.jar' and 'pvcore.jar' found in the zip archive to a location accessible by docker, and as defined in the volumes mapping (see above)
3. Copy the folder 'pv_integration' including all content to a location accessible by docker, and as defined in the volumes mapping (see above). When updating, please make sure to remove old content of the complete directory first.
4. Restart the docker container
5. In Codebeamer, add following the "externalWidgetExtensions" section to the Application Configuration (https://<codebeamer>/sysadmin/configConfiguration.spr) as System Administrator:

```
    ...},
    "externalWidgetExtensions" : {
        "uri" : "https://<codebeamer>/pv-widget/extension.json"
    }
}
```

> **Note**
>
> The above steps are required only for Codebeamer versions up to 2.2.0.0. More details on how to enable the pure::variants widget in Codebeamer, starting from version 2.2.0.0, can be found in the following link: Enabling pure::variants Widget in Codebeamer.

5.6.5. Pre-defined settings for Web Integration in Codebeamer

The Web Integration for Codebeamer allows you to pre-define specific settings to ease up or limitate the setup for end-user.

Please see chapter Section 7.2, “Pre-defined settings for Web Integration” for detailed explanations, which settings are available and how they are configured.

To use these pre-defined settings for Codebeamer Web Integration, you need to write the configuration (in JSON format) into a file called *settings.json* and put this into the deployment directory.

This settings.json must be placed into the existing deployment directory of the Integration.

E.g. <codebeamer>/tomcat/webapps/pv-widget/

```
|-index.html
|-extension.json
|-settings.json
|-... (other files)
```

Configure Widget for SSO setup

The Widget for Codebeamer must be at least configured, if Single-Sign On (SSO) is used in the company's environment. If SSO is used, the pure::variants OIDC proxy has to be used (as described in section Section 5.6.8, “Configuration To Enable Open ID Connect (OIDC) Authentication”). Additionally, the widget must know this proxy's public URL, so the WIdget properly works in all use-cases. Therefore, please add the *settings.json* with the following content:

```
{
```

```
  "SSO_SERVER_URL: "https://codebeamer.server:9443/"
}
```

Please adapt the SSO server url accordingly to your environment setup.

5.6.6. Permissions

The communication of pure::variants Desktop Client and Widget to Codebeamer application uses the Codebeamer REST API. In order to use the REST API end points, the user needs to have *Rest / Remote API - Access* permissions.

To do this, make sure the user group that the users are assigned to in Codebeamer have this permission set via *System Admin->User Groups* menu.

5.6.7. Getting Version Information of the Server Component

Use following REST call to query the version information of the server component, this way it can be checked if the server component is running correctly:

https://<path_to_codebeamer>/rest/v3/ps/vel/version

Note: Use the credentials (basic authentication) of a Codebeamer user with 'api_permission'.

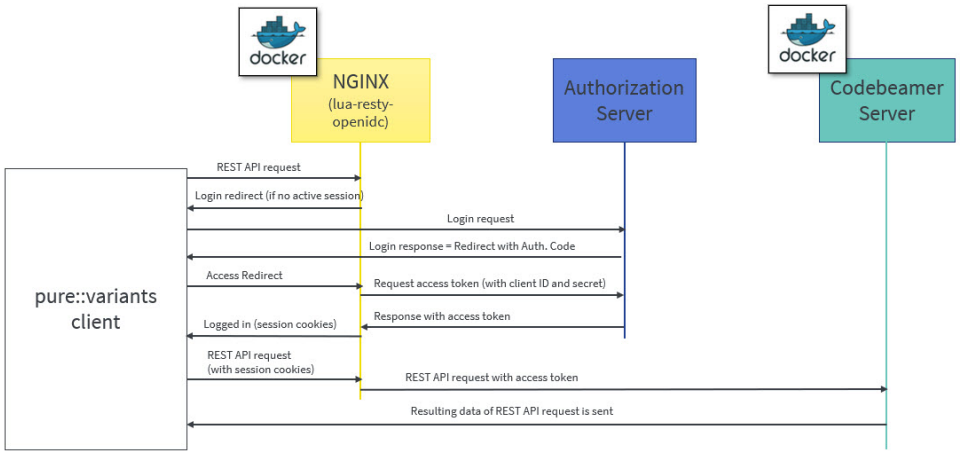
5.6.8. Configuration To Enable Open ID Connect (OIDC) Authentication

OpenID Connect (OIDC) is an authentication protocol that is an extension of OAuth 2.0.

According to this, a dedicated system (Authorization server/ Identity provider) takes care of authenticating a user and issuing access and id token if authentication was successful. This token can be used by clients to obtain data from the Resource Server, in this case the Codebeamer server. The REST API of Codebeamer requires such access token to enable this way of authentication.

To obtain the access token an Authentication Proxy needs to be deployed between the client and the server. All REST API calls are redirected through it, while the authentication process including the refreshing of the tokens is also managed by this proxy in the background.

Figure 45. Authorization process using a proxy



Client registration steps:

1. During the client registration process, both Codebeamer and the Authentication Proxy needs to be registered.

2. The provided configuration files use the 'lua openresty' library for NGINX, implementing OIDC.

3. The registered client's Client ID and Client secret should be added in the configuration files of docker-compose and NGINX (see later).

Following chapter describe the docker configuration files and their parameters.

5.6.9. docker-compose.yml for the NGINX Proxy

This file configures the auth-proxy service that is required by the pure::varaints client. The docker container image named “oidc-auth-proxy” will be created and started, on which NGINX service will be available, which in-turn will be used by pure::variants Desktop Client to make the REST calls.

Following parameters are to be set:

- **build:** Builds a docker image from a dockerfile. The path is a directory of the host system.
- **ports:** Specifies the port to which NGINX is listening to. 9943:9943 shows the mapping between host port and container port (host port: docker container port).

  Note: The port used in oidc-auth-proxy-nginx.conf should be used as docker container port.
- **volumes**: Contains the data which will be used by docker container. It is of the format source:target [:mode] where, source are the host files and target are container path where volumes are mounted.

Any dependent container(s) that will be used by oidc-auth-proxy or any additional container that needs to be built together can be deployed on the same docker machine by adding in the new container configuration under services.

Following code listing shows an example:

```
version: '3.1'
services:
  oidc-auth-proxy:
    build:
      context: .
      dockerfile: oidc-auth-proxy.dockerfile
    ##Specify the port to which nginx is listening to. (host port:docker port)
    ports:
      - 9943:9943
    volumes:
      - ./oidc-auth-proxy-nginx.conf:/usr/local/openresty/nginx/conf/nginx.conf:ro
      - ./server.crt:/usr/local/openresty/nginx/server.cert:ro
      - ./server.key:/usr/local/openresty/nginx/server.key:ro
      - ./cacerts.crt:/usr/local/openresty/nginx/cacerts.crt:ro
    restart: always
```

5.6.10. oidc-auth-proxy.dockerfile for the NGINX Proxy

This file contains the set of commands that has to be executed to build a docker image. lua-resty-openidc library for NGINX is used to authenticate user against Open ID Connect provider. Hence this file contains the command to load the base image of openresty from docker hub and then Install the required packages on the current docker image, followed by command to start the NGINX.

Following code listing shows an example:

```
FROM openresty/openresty:alpine-fat
RUN apk add --update openssl-dev git && luarocks install lua-resty-openidc
CMD ["/usr/local/openresty/bin/openresty", "-g", "daemon off;"]
```

5.6.11. oidc-auth-proxy-nginx.conf for the NGINX Proxy

The directives that need to be adapted are as follows:

- Set **listen** port to which NGINX should listen to.

- The **server name** can be domain name or ip address of the host machine on which docker is running.
- The **redirect_uri**_path should match the uri pre-registered in Authorization server during client registration.
- OpenID Connect defines a **discovery** mechanism where OpenID Server publishes its metadata at a well known url of the format: https://server.com/.well-known/openid-configuration
- The **client_id** and **client_secret** are obtained from the Authorization Server after the client registration.
- **proxy_pass** value can be a docker container on which Codebeamer application is running, e.g. http://container-name:8090; or it can be a Codebeamer application server url to which a request should be forwarded, e.g. http or https://server-name:port(optional);

Following code listing shows an example:

```
events {
  worker_connections 128;
}

http {
  resolver 127.0.0.11 ipv6=off;
  lua_package_path '~/lua/?.lua;;';
  lua_ssl_trusted_certificate /usr/local/openresty/nginx/cacerts.crt;
  lua_ssl_verify_depth 5;
  lua_shared_dict discovery 5m;
  lua_shared_dict jwks 5m;

  server {
    listen 9943 ssl; ##mention the port to which nginx should listen to
    server_name codebeamer.example.com; ##domain name or ip address of the host on which
 docker is running
    ssl_certificate /usr/local/openresty/nginx/server.cert;
    ssl_certificate_key /usr/local/openresty/nginx/server.key;
    ssl_protocols TLSv1.2 TLSv1.3;
    client_max_body_size 16m;

    location / {
      access_by_lua_block {
        local opts = {
   ##redirect_uri should match the uri pre-registered in Authorization server during client
 registration.
          redirect_uri_path = "/login/oauth/authenticate.spr",
  ##OpenID Connect defines a discovery mechanism where OpenID Server publishes its metadata at
 a well known url of the format: https://server.com/.well-known/openid-configuration
          discovery = "https://jas.example.com:9643/oidc/endpoint/jazzop/.well-known/openid-
configuration",
  ##Client_id and client_secret obtained from the authorization server after the client
 registration.
          client_id = "<set here the client ID>",
          client_secret = "<set here the client secret>",
          scope = "openid profile email",
          access_token_expires_leeway = 30,
          accept_none_alg = false,
          accept_unsupported_alg = false,
          renew_access_token_on_expiry = true,
          access_token_expires_in=3600,
          session_contents = {access_token=true, id_token=true}
        }
        if ngx.req.get_headers()["Authorization"] == nil or (not
 string.match(ngx.req.get_headers()["Authorization"], "Bearer")) then
          local res, err = require("resty.openidc").authenticate(opts)
          if err then
            ngx.status = 500
            ngx.say(err)
            ngx.exit(ngx.HTTP_INTERNAL_SERVER_ERROR)
          end
          ngx.req.set_header("Authorization", "Bearer " .. res.access_token)
          ngx.req.set_header("X-User", res.id_token.email)
        end
      }
```

```
   ##proxy_pass value can be a docker container on which Codebeamer application is running.
 For ex., http://container-name:8090;
   ##or it can be a Codebeamer application server url to which a request should be forwarded.
 For ex., http or https://server-name:port(optional);
      proxy_pass http://codebeamer-app:8090;
    }
  }
}
```

5.6.12. Steps to Setup a Docker-container the NGINX Proxy

Following are the steps to create a docker container image named “oidc-auth-proxy”. NGINX will available on this docker container on the specified port.

1. Place the files provided (docker-compose.yml, oidc-auth-proxy.dockerfile, oidc-auth-proxy-nginx.conf) in a folder.

2. Place certificates to be used within the same folder. This will be used by NGINX for SSL handshake.

3. Modify the NGINX configuration file oidc-auth-proxy-nginx.conf as explained in the previous section (oidc-auth-proxy-nginx.conf).

4. Run docker-compose up to create/start a container.

5.7. pure::variants Connector for Siemens Polarion

This chapter describes how to setup the components in order to work properly together with Polarion

5.7.1. Installation of pure::variants Desktop Client

Follow the steps as described in ‘3.1. Install pure::variants Desktop Client’, in case installing pure::variants into an eclipse client, see chapter ‘3.1.2. Install into an existing Eclipse’.

5.7.2. Installation of pure::variants server component for Polarion

The pure::variants enterprise ("pure::variants Windows Installer Package") contains the package **com.ps.consul.polarion.api-<*version*>.zip** which needs to be deployed to the Polarion server. Therefore please follow the following steps:

1. Unzip the archive **com.ps.consul.polarion.api-<*version*>.zip** which contains a folder

2. Place the extracted folder in the Polarion server at **<*polarion home*> /polarion/extensions/pure-systems/eclipse/plugins/**

3. Delete the cached Polarion configuration to ensure the pure::variants integration will be loaded properly on startup by deleting the folder **<*polarion home*> /data/workspace/.config**.

   #### Note

   *Please make sure that you only delete the **.config** folder*

4. Restart the Polarion service

The previous steps are only for installing the integration. To configure the the pure::variants plugin please follow the steps in Section 5.7.3, “Configuration of pure::variants server component for Polarion”

5.7.3. Configuration of pure::variants server component for Polarion

There are several steps necessary to prepare the Polarion server for pure::variants.

Configuration steps in Polarion

In order to show the in tool integration in the project's sidebar the pure::variants topic needs to be added in Polarion which can be done in the **Global Administration** or in the **Project Administration**

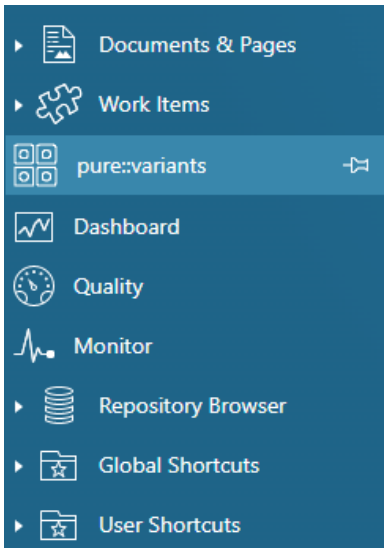
1. Navigate to the **Administration area** -> **Poartal** -> **Topics**
2. Create a new topics configuration or add the pure::variants topic to an existing one
3. Add **<topic id="purevariants" />** to the topic xml

   an example topic definition is given here:

```
<?xml version="1.0" encoding="UTF-8"?>
<topics xmlns="http://polarion.com/schema/Portal/Topics" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://polarion.com/schema/Portal/Topics">
    <topic id="wiki"/>
    <topic id="workitems"/>
    <topic id="plans"/>
    <topic id="testruns"/>
    <topic id="collections"/>
    <topic id="purevariants"/>
    <topic id="baselines"/>
    <topic id="builds"/>
    <topic id="dashboard"/>
    <topic id="quality"/>
    <topic id="reports"/>
    <topic id="monitor"/>
    <topic id="repository_browser"/>
    <topic id="global_shortcuts"/>
    <topic id="project_shortcuts"/>
    <topic id="user_shortcuts"/>
</topics>
```

4. If all steps from Section 5.7.2, “Installation of pure::variants server component for Polarion” and the above mentioned ones were successful the pure::variants menu should be visible on the sidebar of the project with the configured topics.

Figure 46. Polarion menu on the project's sidebar including pure::variants.



Connection configuration of the pure::variants integration

The connection settings of the pure::variants in tool integration for Polarion can be preconfigured with proper settings in order to ease the process of configuration for the user. Therefore a simple javascript file with the

name **pv.settings.js** is placed in the subfolder **webapp** of the Polarion server extension. The configuration and possibilities of the javascript file are documented here: Section 7.2, “Pre-defined settings for Web Integration”. The default configuration looks lik this:

```
            pvWidgetConnectionSettings = {
  // PV_HUB: Defines either 'webhub' or 'desktophub' as tool to connect with
  "PV_HUB": undefined,
  // PV_HUB_EDITABLE: true or false. Defines if the user is able to change the connection
  "PV_HUB_EDITABLE": true,
  // PV_HUB_URL: provide the url if webhub is defined at PV_HUB
  "PV_HUB_URL": undefined,
  // PV_HUB_URL_EDITABLE: true or false. Defines if the webhub url is modifieable
  "PV_HUB_URL_EDITABLE": true
 };
```

Note

Please make sure not to modify the variable name **pvWidgetConnectionSettings** during the modification of the settings

5.7.4. Preparation of the Polarion project to store variability information

Adding general settings information to Polarion

For managing varaiblity information in Polarion assets pure::variants needs to know where restrictions are stored and which characters are starting a calculation. These settings can be stored either **globally**, on **project level** or even on **document level**. Therefore the settings priority follows this rule: **document settings > project settings > global settings**

The settings are stored in the key **pvSettings** in json format e.g.:

```
            pvSettings={"beginMarker":"[","endMarker":"]","escapeMarker":"%",
"performPartialTextSubstitution":true,"pvRestrictionFieldName":"pvRestriction"}
```

- To store the settings globally navigate to the **Global Administration** and add the pvSettings property to the **Configuration Properties**
- To store the settings for a specific project navigate to the **Project Administration** and add the pvSettings property to the **Configuration Properties**
- To store the settings in a document of a project you need to add a custom field to the document. Therefore please navigate to the **Project Administration** -> **Document & Pages** -> **Document Custom Fields** and add a custom field to the document typs you want to support. The custom field's ID must be **pvSettings** and the type of the custom field must be **String (single line plain text)**

Preparing restrictions for work items

In order to store the restriction of a work item. The work item needs to have a custom field with the type **String (single line plain text)** the ID of the custom field must match the ID which is stored in the pvSettings object from above. Our default ID for the custom field is **pvRestriction**.

To add a custom field to a work item type please navigate to the **Project Settings** -> **Work Items** -> **Custom Fields**. There you need to select the work item type or all types and add a new field with the limitations mentioned.

Preparing the enumeration transformation

During the enumeration transformation of pure::variants the work items will be tagged with their respective variant in which they are contained. To store that information another custom field is necessary. The ID of the custom field must match the ID which is provided in the transformation configuration of the variant which is transformed

via pure::variants. Our default ID for the custom field is **pvVariants**. This custom field must be of type **Rich Text (multi-line)**.

The procedure to add a custom field to a work item is described in the section called “Preparing restrictions for work items”

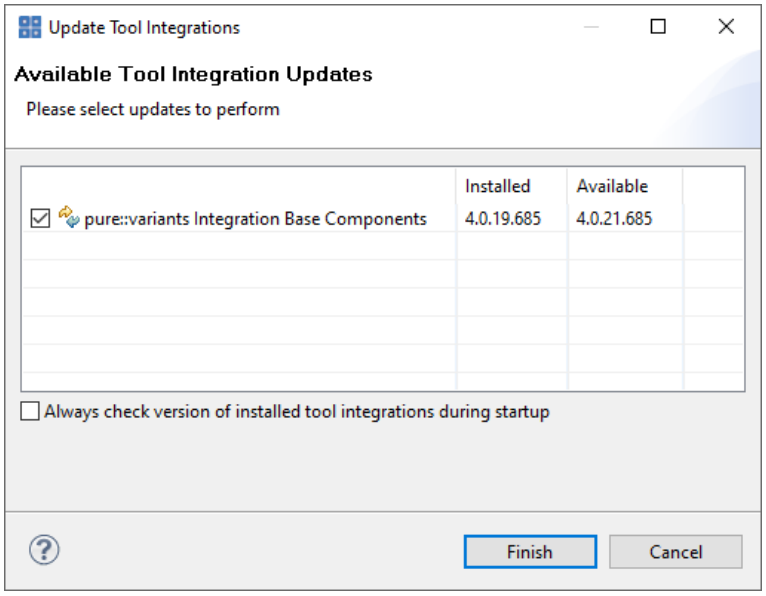
6. pure::variants Tool Integrations

6.1. Install pure::variants Tool Integrations

For using the pure::variants Integrations for several tools the integrations have to be installed. If pure::variants is installed using the Installer, the tool integrations are installed along with the corresponding pure::variants connector. If the installer was not used or the integration was not installed along with the corresponding connector then follow the next steps. Usually a dialog comes up after pure::variants starts informing about not installed or not up-to-date integrations.

If this dialog does not come up automatically it can be opened using the follwing menu entry in the pure::variants **Help** menu. Go to **Help**->**pure::variants** the item **Tool Integration Updates**.

Figure 47. Install Tool Integrations



A dialog comes up listing all available Tool Integrations. Just select the integrations you want to install or update and finish the dialog. pure::variants will guide you through the installation process. If an automatic update is possible, pure::variants will just perform the update, without showing an installer.

The option **Always check version of installed tool integrations during startup** enables a check if all available tool integrations are already installed and up to date. This check is performed each time pure::variants starts.

6.1.1. Install pure::variants Tool Integrations in silent mode

The pure::variants Tool Integration installers have a silent mode. This mode installs the pure::variants Tool Integrations without user interaction. The different Tool Integration installers can be found in this path <PV_INSTALL_DIR>\<ECLIPSE_DIR>\plugins.

To do this, call the installer with command line option */S*. Note that the target directory must not contain any quotation marks. Here's the generic example commandline for silent install of a Tool Integration:

```
<PV_INSTALL_DIR>\<ECLIPSE_DIR>\plugins\<integration_plugin_folder>\<tool_name>AddIn\setup.exe"
```

```
            /S /D=<PV_INSTALL_DIR>\com.ps.consul.eclipse.ui.<tool_name>.integration
```

For example, to silently install the Office Integration the following commandline can be performed:

```
"C:\Program Files\pure-systems\pv_Enterprise_6.0\eclipse\plugins
\com.ps.consul.eclipse.ui.ms.office.integration_5.x.x.xxx\msOfficeAddIn\setup.exe"
            /S /D=C:\Program Files\pure-systems
\pv_Enterprise_6.0\com.ps.consul.eclipse.ui.ms.office.integration
```

6.1.2. Update pure::variants Tool Integrations in silent mode

It is also possible to run the update in the background or silent mode to update an existing installation. Note that there will be no console output as well.

To do this, call the installer with command line option */UPDATE /S*. To run it in silent mode but not in background *start "" /WAIT "Setup.exe" /UPDATE /S* can be used.

6.1.3. pure::variants Desktop Hub

If the pure::variants Desktop Client was installed with the installer the pure::variants Desktop Hub is already installed along with the pure::variants Desktop Client. If the installer was not used or the pure::variants Desktop Hub is missing for another reason the installation can be triggered from within the pure::variants Desktop Client.

Please see Section 6.1, “Install pure::variants Tool Integrations” for installing the **pure::variants Integration Base Components**.

6.1.4. pure::variants Integration for Doors

Please see Section 6.1, “Install pure::variants Tool Integrations” for installing the integration executable.

After finishing the installation successfully, the Doors client has to be started with a command line option, which enables the Integration menu within Doors.

The command line should look similar to this: `doors -a "<Menu installation path>\pure-variants"`. Without this command line option the Integration cannot be triggered and so not be used.

On Windows platforms it is also possible to add the directory to the registry key `HKLM\Software\Telelogic\Doors\Doors version\Addins`.

1. Open the Registry editor
2. Browse to Doors installation in HKEY_LOCAL_MACHINE\SOFTWARE\Telelogic\DOORS\<DOORS version number>\Config
3. Right click config Key to add a new string value
   - Value Name set to **Addins**
   - Value Data set to the path of the pure::variants menu directory

With administrative access to the Doors installation the Add-In can also be installed for all users of this installation using the shared DXL library. See the Doors Help topic "Configuring Doors" for more information.

Note

This requires adaptions of the pure::variants Integration menu DXL scripts.

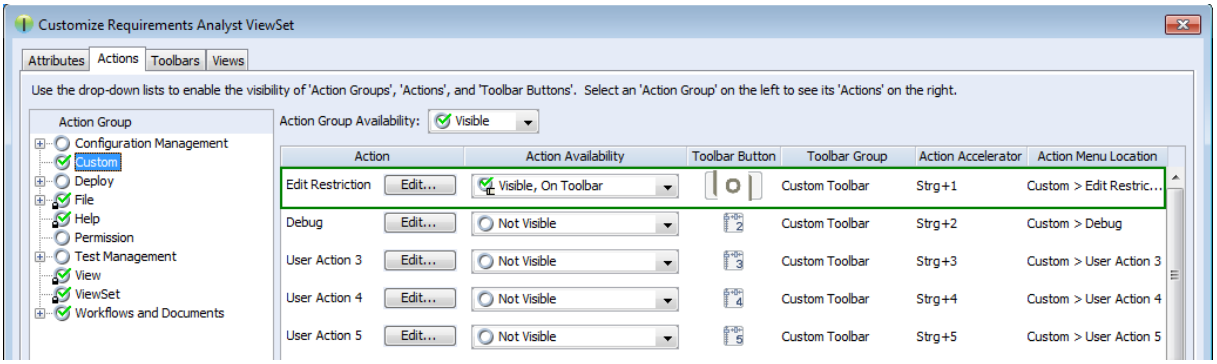
Now the Integration should be available in Doors. To verify if the installation was successfull, open a Doors module and select from the menu **pure::variants** the item **Open pure::variants Integration**. If the pure::variants Integration window opens, the Integration was installed correctly.

6.1.5. pure::variants Integration for PTC Integrity

Please see Section 6.1, “Install pure::variants Tool Integrations” for installing the integration executable.

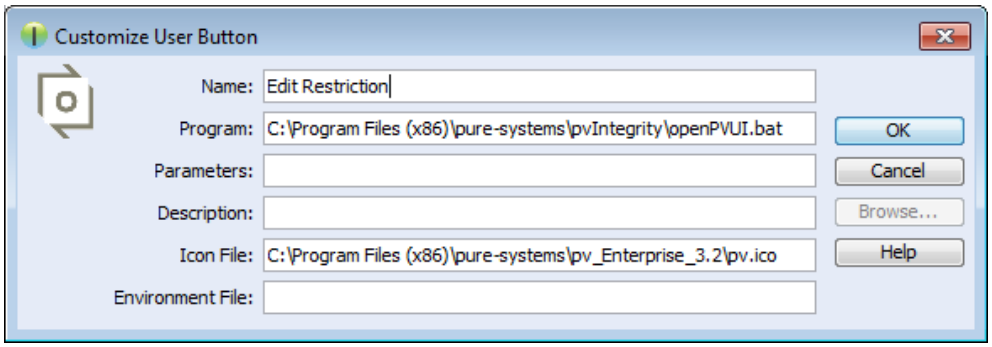
For starting the integration within PTC Integrity some additional steps have to be performed. Start the PTC Integrity client application. Open menu **ViewSet** -> **Customize** and switch to page **Actions**. Click on action group **Custom** to view the custom user actions.

Figure 48. Custom Actions



Click the button **Edit** of a user action to customize it. Name the action **Edit Restriction**. As the program to execute enter or browse to the path of file **openPVUI.bat** which is located in the installation path of the pure::variants Integration for PTC Integrity. There is also an icon file **pv.ico** in this directory you can use.

Figure 49. Custom Button "Edit Restriction"



6.1.6. pure::variants Integration for IBM Rational Rhapsody

Please see Section 6.1, “Install pure::variants Tool Integrations” for installing the integration executable.

To use the Integration, it still needs to be added to your Rhapsody project:

1. Open a Rhapsody project
2. Select **File** > **Add Profile to Model...**
3. Select the file "pvRhapsody.sbs" from the Integration installation directory

Now the Integration should be available for the given project. You can open the Integration window at **Tools** > **pure::variants**.

Per default the Integration is loaded when opening the Rhapsody project. If you want to only load the Integration when clicking **Tools** > **pure::variants**, you can edit file `pvRhapsody.prp` in the pure::variants Integration for Rhapsody installation folder. Open the file with a text editor and set property `showonstart` to `False`. After restarting Rhapsody, the Integration window should only open after clicking **Tools** > **pure::variants**.

6.1.7. pure::variants Integration for Enterprise Architect

Please see Section 6.1, “Install pure::variants Tool Integrations” for installing the integration executable.

Now the Integration should be available in Enterprise Architect. Select **Extensions**->**Add-In Windows** to open the Add-In window, which shows the pure::variants Integration user interface. Furthermore, you can enable or disable the Integration at **Extensions** -> **Manage Add-Ins...** (Since Enterprise Architect 14, you can find both entries in tab **Specialize**).

6.1.8. pure::variants Integration for Microsoft Office

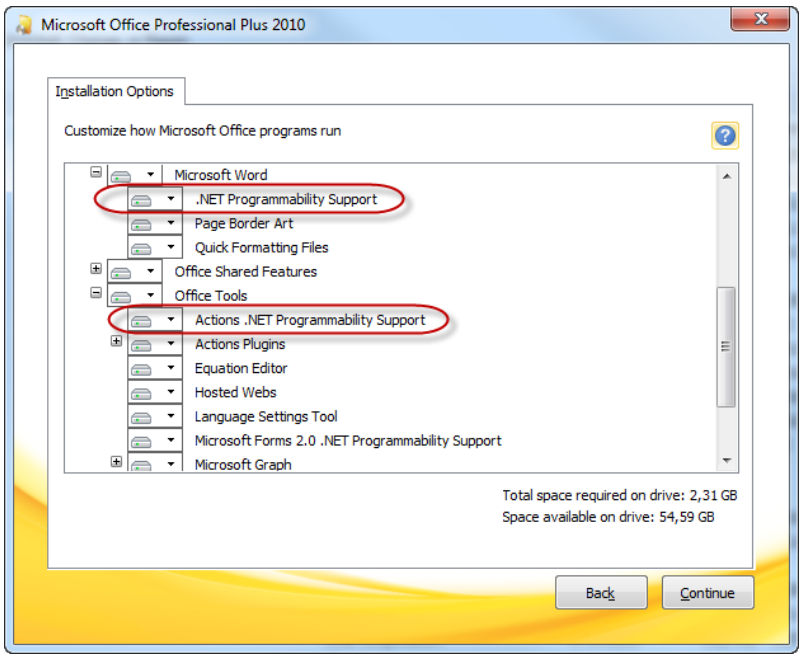
The Integration will not work if the following features are not installed with Microsoft Office:

- Microsoft Word / .NET Programmability Support (if using Microsoft Office Word Integration)
- Microsoft Excel / .NET Programmability Support (if using Microsoft Office Excel Integration)
- Office Tools / Actions .NET Programmability Support

They can be added to the Microsoft Office installation as follows:

1. Open the Windows Control Panel and navigate to **Programs and Features**
2. Right-click on your Microsoft Office Installation and select **Change**
3. Select **Add or Remove Features**
4. Add the features marked in Figure 50, “Adding Missing Features to the Office Installation” to your Office Installation.
5. Press **Continue** and close the Dialog.

Figure 50. Adding Missing Features to the Office Installation



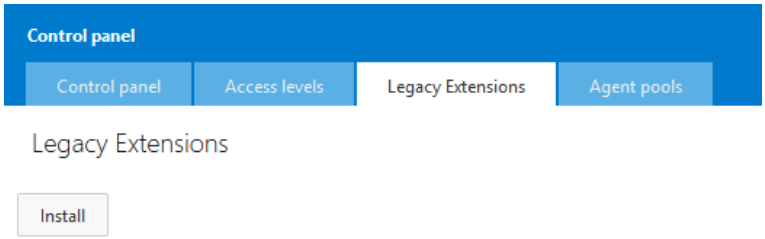
Now the installation of the pure::variants Integration for Microsoft Office should run successfully.

6.1.9. pure::variants Integration for Team Foundation Server

Administrator: At first, please download the pure::variants windows installer package from your pure::variants update site and extract the pure::variants TFS integration zip archive with name *com.ps.consul.web.ui.tfs2015-*

x.x.x.zip. To install the integration navigate to the **Control panel**->**Legacy Extensions** and press **Install**. Thereby browse to your local copy of pure::variants TFS Integration zip archive.

Figure 51. Install pure::variants TFS Integration



Please ensure that the pure::variants TFS Integration is enabled, after installation.

6.1.10. Advanced Integration Setup

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

The steps needed to do in these cases differ, depending on which type of integration you are using. For java-based integrations, such as the Integration for IBM Rational Rhapsody and the Integration for PTC Integrity, it is necessary to specify proxy and certificate settings manually in pv.properties files. For .NET-based integrations (all other integrations), the Windows proxy and certificate settings are used automatically, so no additional setup is necessary.

Advanced Setup of Java-based Integrations

The following integrations are Java-based:

- pure::variants Integration for IBM Rational Rhapsody
- pure::variants Integration for PTC Integrity

All extra configurations in java-based integrations 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-6/pv.properties
```

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

```
%APPDATA%/pure-variants-6/pv.properties
```

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 6. 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. 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 7. 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 could append the following lines:

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

However, it is also possible to permanently accept a self-signed certificate when trying to first connect to your model or license server. pure::variants or the integrations will open an certificate acceptance dialog on the first connection attempt.

Advanced Setup of .NET-based Integrations

The following integrations are .NET-based:

- pure::variants Desktop Hub
- pure::variants Integration for Doors
- pure::variants Integration for Microsoft Excel and Microsoft Word
- pure::variants Integration for Enterprise Architect
- pure::variants Integration for Zuken CR-8000

Since pure::variants 4.0.19, the way the connection to pure::variants model or license servers is done has changed. Therefore, no advanced setup as for java integrations is necessary anymore. The proxy and certificate settings configured in Windows are used for the connection. So if the connection works in a browser that uses the Windows certificate and proxy settings (e.g. Chrome or Edge), the connection should work in all .NET-based integrations, too. The only exception from that rule are the supported security protocols:

Per default, the following security protocols are supported in .NET-based integrations: SSL3, TLS 1.0 - 1.3. To instead let Windows decide which protocols to support, you need to add registry entry "SystemDefaultTlsVersions" as documented here: https://docs.microsoft.com/en-us/mem/configmgr/core/plan-design/security/enable-tls-1-2-client#configure-for-strong-cryptography.

If for some reason you want to switch back to the old server connection behavior as used in all java integrations, you can do that as described in the following section.

Switching Back to Previous Server Connection Behaviour

There are two ways to switch back to the previous server connection behaviour. Either you add the Windows environment variable `PV_FORCE_JAVA_SOAP_SERVER_CONNECTION` with value `true`, or you add line `forceJavaSoapConnection=true` to file `pv.properties` (see the section called “Advanced Setup of Java-based Integrations” for instructions how to edit `pv.properties`).

After switching back to the previous server connection behaviour, you may need to configure connection settings (e.g., proxy settings) in the same way as for java integrations.

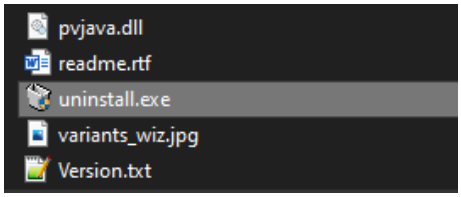
6.2. Update pure::variants Tool Integrations

To update an pure::variants tool integration the same mechanism as for installing a pure::variants tool integration is used. Please consult section Section 6.1, “Install pure::variants Tool Integrations” for a detailed description.

6.3. Uninstall pure::variants Tool Integrations

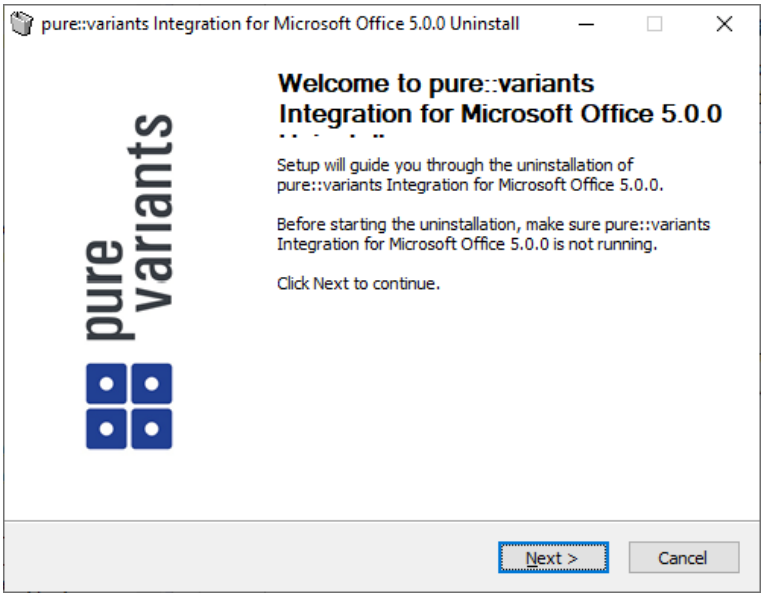
The uninstaller for the pure::variants integration can be started in two different ways. The first one is to go to the Windows *Add or remove programs* application and search for the pure::variants integration and start the uninstaller by using the *Uninstall* action. The uninstaller requires Administrator privileges.

Figure 52. pure::variants Integration Uninstaller



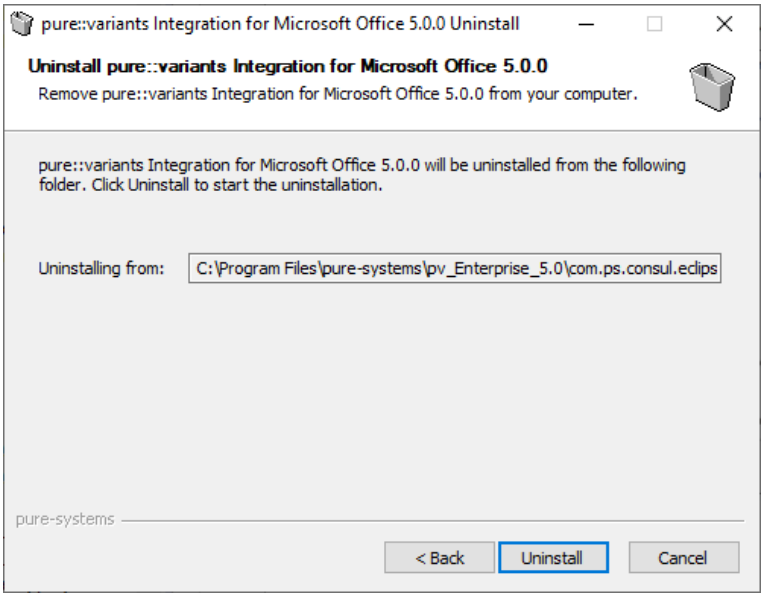
The second possibility is to navigate to the pure::variants Desktop Client installation folder and start the uninstaller by double clicking it.

Figure 53. pure::variants Integration Uninstaller



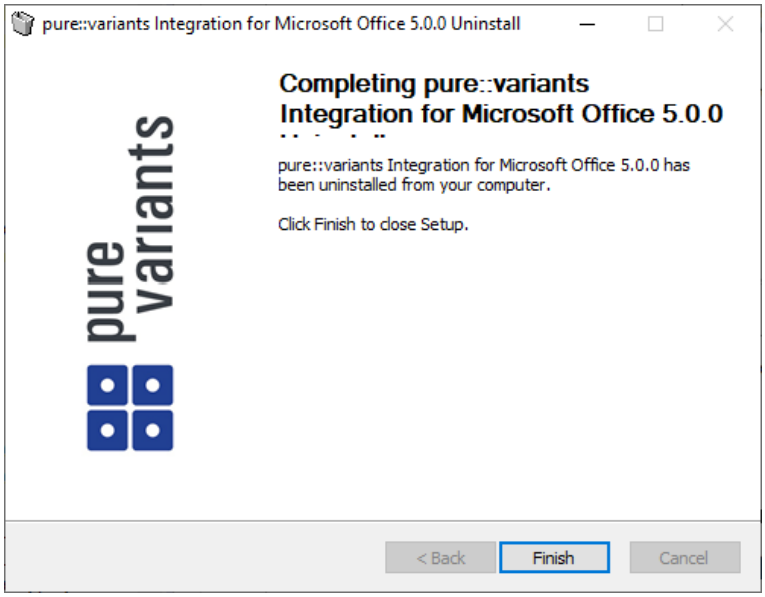
Click *Next*.

Figure 54. Uninstall from



Click *Uninstall* to start the uninstall process.

Figure 55. Completing Uninstall



The installation is succesfully finished. Click *Finish* to close the uninstaller.

6.4. Basic Setup of pure::variants Tool Integrations

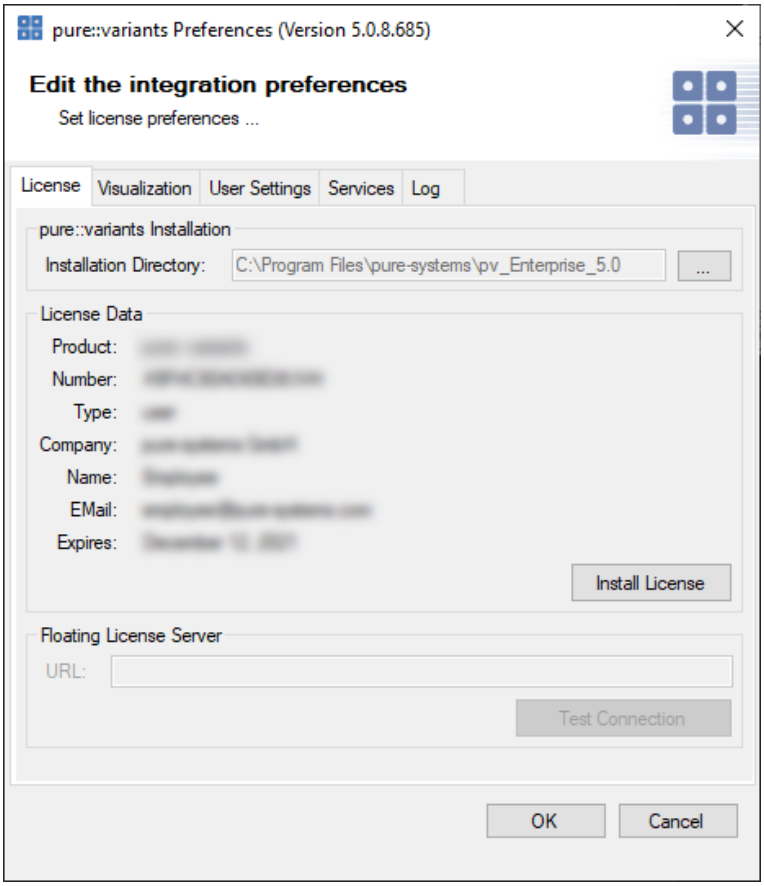
When you first use the Desktop Hub after installation, it is necessary to check whether the license preferences are correct. To this end, open the preferences dialog via the button in the Desktop Hub window or by selecting **Hub Configuration** from the pure::variants tray menu.

A dialog opens that shows the path to your pure::variants installation and your license information (see Figure 56, “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 is established, press the button **Test Connection**.

Now you can use the Desktop Hub.

Figure 56. Preferences Dialog



6.4.1. Server Connection 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.

Since pure::variants 4.0.19, the way the connection to pure::variants model or license servers is done has changed. Therefore, no advanced setup should be necessary anymore, since the settings configured in Windows are used (e.g., proxy settings, certificates). However, it is still possible to switch back to the previous server connection behavior.

Switching Back to Previous Server Connection Behaviour

This sections describe the switch back to the Java based Soap connection method. It is not recommended to use this but may help to fix connection problems. This mechanism is available for .net based integrations only.

.net based integrations are

- Section 6.1.4, “pure::variants Integration for Doors”
- pure::variants Integration for Microsoft Office
- Section 6.1.7, “pure::variants Integration for Enterprise Architect”
- Section 6.1.9, “pure::variants Integration for Team Foundation Server”

There are two ways to switch back to the previous server connection behaviour. Either you add the Windows environment variable `PV_FORCE_JAVA_SOAP_SERVER_CONNECTION` with value `true`, or you add line `forceJavaSoapConnection=true` to file `pv.properties` (see below for instructions how to edit `pv.properties`).

Once you have switched back, you may again need extra configuration to connect to a license or model server. See the section called “Advanced Integration Setup” for extra configuration setps.

Advanced Integration Setup

This steps are necessary only, if you have switched the Soap server connection method to the ols behavior. See the section called “Switching Back to Previous Server Connection Behaviour”.

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-6/pv.properties
```

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

```
%APPDATA%/pure-variants-6/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 8. 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. 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 9. 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
```

HTTPS Connection with Model Access Service (pure::variants Desktop Hub only)

Per default, the self-signed certificate that is used for securing the model access service connection is only generated for the current user. Thus, when there are multiple users working on the same machine, each user would use a different self-signed certificate and each user would get a security exception the first time he uses the model access service (e.g., when working with DoorsNG).

To prevent that, you as an administrator, can manually configure which certificate is used for all users and register it in the Windows *Trusted Root Certification Authorities* store to make sure no security exception is shown. To achieve that, you can set the following properties in

```
%PROGRAMDATA%/pure-variants-6/pv.properties
```

Table 10. Model Access Service Settings

Property Name	Comments
enableHttpService	true if the model access service should be enabled
httpServicePort	Port that the model access service should run on
enableHTTPS	true if the connection should be secured
keystore	Path to your key store
keystorePassword	Password of the given key store

For example, if you wanted to enable the model access service for all users, use a secure connection, and use your own keystore that is located at `C:/ProgramData/pure-variants-6/selfsigned.jks`, you would need to append the following lines to `pv.properties`:

```
# enable the model access service
enableHttpService=true
# secure the connection
enableHTTPS=true
# the port on which the service listens
httpServicePort=9443
# the path to your keystore, which must be a java keystore and which contains your certificate
keystore=C:/ProgramData/pure-variants-6/selfsigned.jks
# the password to the keystore
keystorePassword=password
```

7. pure::variants Web Integration

7.1. IBM Rational DOORS NG Web Integration

The pure::variants Integration for DOORS NG is distributed in a WAR archive (`com.ps.consul.web.ui.doorsng-x.x.x.war`) and can be found in the **pure::variants Windows Installer package** on the pure::variants update site.

Note

For brevity we have renamed the war-archive to **pvwidget.war**. At least for Apache Tomcat, the war-archive name implies the context-path, so the pure::variants Integration's Catalog will be reachable at **https://[pv-server-FQDN]:[port]/pvwidget/catalog.xml**.

Remember FQDN is the fully qualified domain name. The port number might be optional, if configured with standard SSL port (443)

7.1.1. Requirements for pure::variants Integration Deployment

The deployment of pure::variants Integration for DOORS NG has the following requirements:

- Extenstion must be hosted on a web server application that can be configured to run with Oracle JDK/JRE or OpenJDK.
- Extension must be accessible from a web server via HTTPS.
- The web server must not require any form of authentication to read the extension files.
- The certificate that is installed in the web server must be a valid certificate and must match the server's domain.
- Java Runtime Envritonment (JRE) or Java Development Kit (JDK) version 1.6 or later.

7.1.2. Installation on Apache Tomcat

Software Requirements

Apache Tomcat 8.5.x is recommended for deployment.

Installation of the pure::variants Integration for DOORS NG

The pure::variants Integration for DOORS NG must be deployed on your application server. This has to be done once by the system administrator. It entails copying of ***pvwidget.war*** into the ***webapps*** directory, which is located in the Apache Tomcat installation directory.

Note

In a Tomcat deployment, name of the war file becomes the context path of the web application. In order to configure a nested context path for a web application, the name of the war should contain the indi-

vidual context path parts separated by '#'. For example, to make the pure::variants Integration's Catalog accessible by a url like https://localhost:8443/**pv/widget/dng**/catalog.xml, the name of the war file needs to be named as **pv#widget#dng.war**. And the war file should be placed into the **webapps** directory. For further details please see the Official Tomcat documentation.

Configuration of Trust Store

As pure::variants Integration does client requests to DOORS NG application server, the Integration's client must be configured with the trusted SSL-certificates. One way to do this is to create a keystore, add the DOORS NG SSL-certificate to it and configure Tomcat's Java-Runtime with this keystore as a trust store.

In case of **Windows**, please create and open *[TOMCAT_INSTALL_DIR]/bin/setenv.bat* file to add the following lines:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustAnchors=[TRUST_STORE_PATH]"
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustStore=[TRUST_STORE_PATH]"
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustStorePassword=[TRUST_STORE_PASSWORD]"
```

An example configuration could look like this:

```
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustAnchors=c:\keystore.jks"
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustStore=c:\keystore.jks"
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustStorePassword=password"
```

In case of **Linux**, please create and open *[TOMCAT_INSTALL_DIR]/bin/setenv.sh* file to add the following lines:

```
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustAnchors=[TRUST_STORE_PATH]"
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustStore=[TRUST_STORE_PATH]"
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustStorePassword=[TRUST_STORE_PASSWORD]"
```

An example configuration could look like this:

```
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustAnchors=c:\keystore.jks"
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustStore=c:\keystore.jks"
export JAVA_OPTS=$JAVA_OPTS" -Djavax.net.ssl.trustStorePassword=password"
```

Update or reinstallation of pure::variants Integration for DOORS NG

Stop the Apache tomcat. Go to the Apache Tomcat installation directory, go in to **work** directory then go in to **Catalina** directory and delete the **localhost** directory. Go back to Apache Tomcat installation directory, go in to **webapps** directory and **delete** the previously installed **pvwidget.war** and **pvwidget** directory. Now **copy** the new **pvwidget.war** in the **webapps** directory and start the Apache Tomcat again. No configuration change in Apache Tomcat is required in case of an update.

7.1.3. Installation on WebSphere Liberty

Software Requirements

WebSphere Liberty Kernel v19.0.0.6+ is recommended for deployment.

Server Setup

Please follow the following steps for WebSphere Liberty setup:

1. Install the WebSphere Liberty according to the official documentation.
2. Create a server by running the following command:

```
bin\server create [SERVER_NAME]
```

3. Please add the following lines into the **server.config.dir/server.xml** file:

```
<featureManager>
  <feature>jsp-2.3</feature>
```

```
  <feature>ssl-1.0</feature>
</featureManager>
```

Please run the following command:

```
bin\installUtility install [SERVER_NAME]
```

SSL Configuration

Please ensure that the server is configured with SSL. Please refer to the **Securing communications with Liberty** section of the official documentation.

Installation of the pure::variants Integration for DOORS NG

Copy the war archive file **pvwidget.war** to **server.config.dir/apps**. Add following line into the **server.config.dir/server.xml** file:

```
<webApplication contextRoot="pvwidget" location="${server.config.dir}/apps/pvwidget.war"/>
```

Note

Nested context path can be configured by specifying the whole path in contextRoot attribute of the webApplication. For example, to make the pure::variants Integration's Catalog accessible by a url like https://localhost:8443/**pv/widget/dng**/catalog.xml, the contextRoot attribute should be configured as **contextRoot="pv/widget/dng"**.

The final **server.config.dir/server.xml** file should look similar to this:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">
  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.3</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  <httpEndpoint host="*" httpPort="-1" httpsPort="9443" id="defaultHttpEndpoint"/>
  <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="SSL"/>
  <keyStore id="defaultKeyStore" location="keystore.jks" password="{xor}LykrOiwr"
 type="JCEKS"/>

  <webApplication contextRoot="pvwidget" location="${server.config.dir}/apps/pvwidget.war"/>
</server>
```

Configuration of Trust Store

As pure::variants Integration does client requests to DOORS NG application server, the Integration's client must be configured with the trusted SSL-certificates. One way to do this is to create a keystore, add the DOORS NG SSL-certificate to it and configure Liberty's Java-Runtime with this keystore as a trust store. Please create and open **server.config.dir/jvm.options** (see Directory locations and properties) to add the following lines:

```
-Djavax.net.ssl.trustAnchors=[TRUST_STORE_PATH]
-Djavax.net.ssl.trustStore=[TRUST_STORE_PATH]
-Djavax.net.ssl.trustStorePassword=[TRUST_STORE_PASSWORD]
```

An example configuration could look like this:

```
-Djavax.net.ssl.trustAnchors=c:\keystore.jks
-Djavax.net.ssl.trustStore=c:\keystore.jks
-Djavax.net.ssl.trustStorePassword=password
```

Update or reinstallation of the pure::variants Integration for DOORS NG

Stop your Web Application Server (like Tomcat or WebSphere Liberty. Replace the **pvwidget.war** in the **server.config.dir/apps** directory. **Start** the web server again.

If you have used the pre-defined settings, as described in the section Section 7.1.8, “Pre-defined settings for Web Integration in DOORS NG”, please re-add the settings.json file accordingly, as done before.

Once finished, please start your Web Application Server again.

7.1.4. Uninstall the pure::variants Integration for DOORS NG

Uninstall on Apache Tomcat

In order to uninstall pure::variants Integration for DOORS NG, stop the Apache Tomcat. Please go to Apache Tomcat installation directory and then go to **webapps** directory. Delete the **pvwidget.war** file and the **pvwidget** directory from the webapps folder.

Go back to Apache Tomcat installation directory, go in to **work** directory and then go in to **Catalina** directory and delete the **localhost** directory from it. Start the Apache Tomcat.

Uninstall on WebSphere Liberty

In order to uninstall the pure::variants Integration for DOORS NG, please stop the WebSphere Liberty server. Go to **server.config.dir/apps** folder and remove the**pvwidget.war**. Edit the **server.config.dir/server.xml** and remove the following line from the file.

```
<webApplication contextRoot="pvwidget" location="${server.config.dir}/apps/pvwidget.war"/>
```

Save the file and start the WebSphere Liberty server again.

7.1.5. Administrative Setup of the pure::variants Integration for DOORS NG

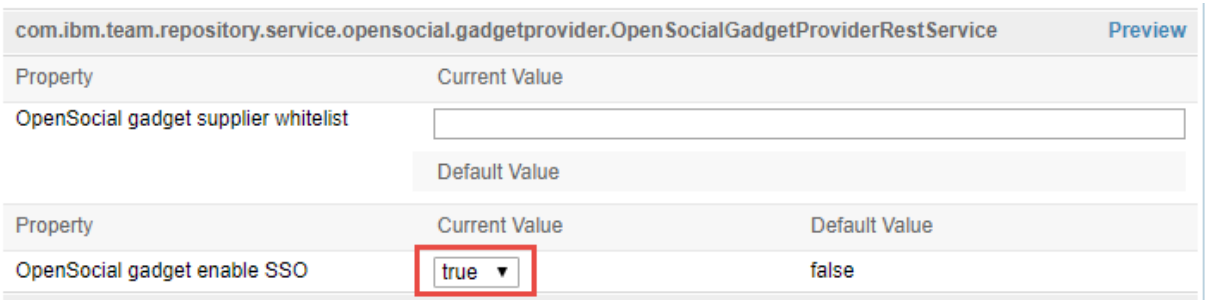
In order to visualize the variability, folloing settings need to be configured on the Jazz side.

JTS Advanced Settings

Browse to JTS home page. Click **Manager Server** and then click **Advanced Properties** in the left menu.

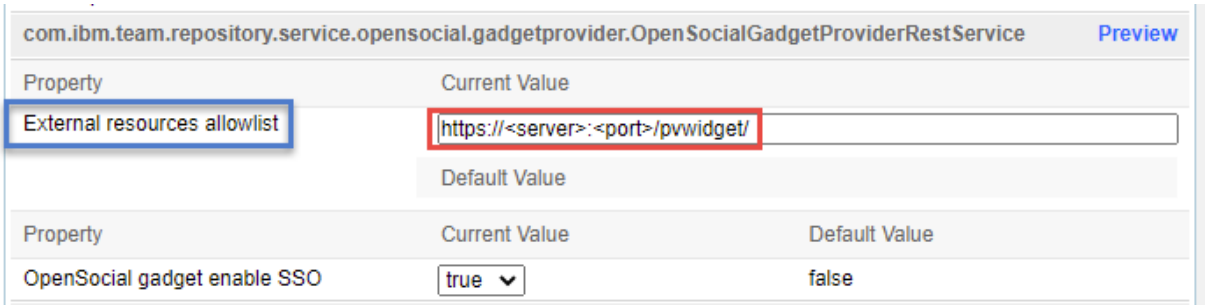
Scroll down to **OpenSocial gadget enable SSO** and set its value to **true** as shown in figure Figure 57, “OpenSocial gadget enable SSO Setting”.

Figure 57. OpenSocial gadget enable SSO Setting



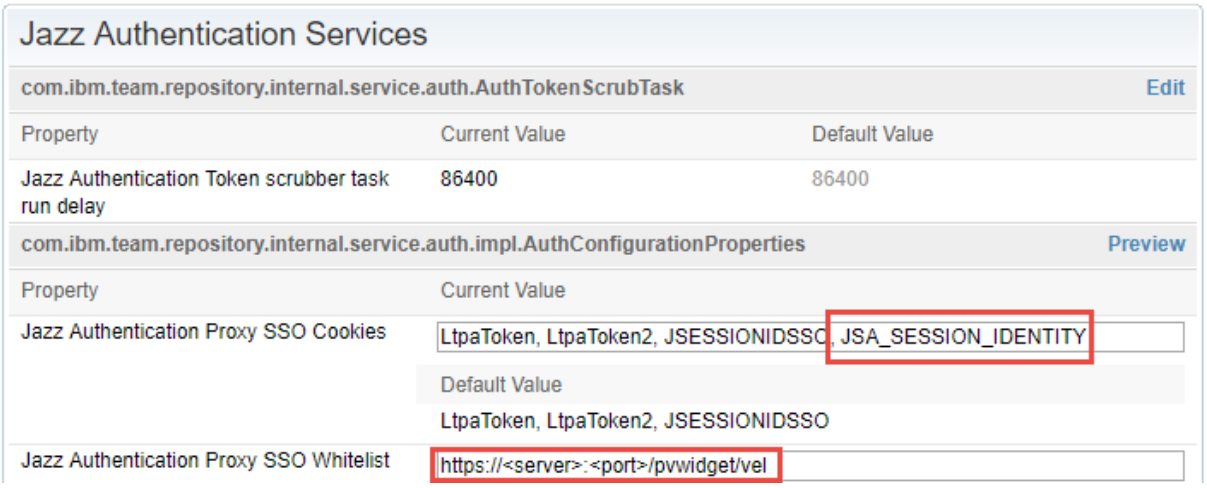
For the versions ELM 7.0.2 iFix004, ELM 7.0.1 iFix009, CLM 6.0.6.1 iFix018, CLM 6.0.6 iFix022 and onwards, an extra configuration is required. In **External resources allowlist**, please enter the URL of the widget. For example if pure::variants Integration for DOORS NG is accessible at https://jazz.server.net:8443/pvwidget/catalog.xml then enter https://jazz.server.net:8443/pvwidget/ in the allowlist as shown in the figure Figure 58, “External resources allowlist”.

Figure 58. External resources allowlist



Further, scroll down to **Jazz Authentication Services**, in **Jazz Authentication Proxy SSO Cookies** field and replace its value with `LtpaToken, LtpaToken2, JSESSIONIDSSO, JSA_SESSION_IDENTITY`. In the next field in **Jazz Authentication Proxy SSO Whitelist**, write the URL of the server hosting the widget. Assuming that pure::variants Integration DOORS NG `war` was renamed as `pvwidget.war`, the URL must end with `/pvwidget/vel` as shown in the figure Figure 59, “Jazz Authentication Proxy SSO Settings”.

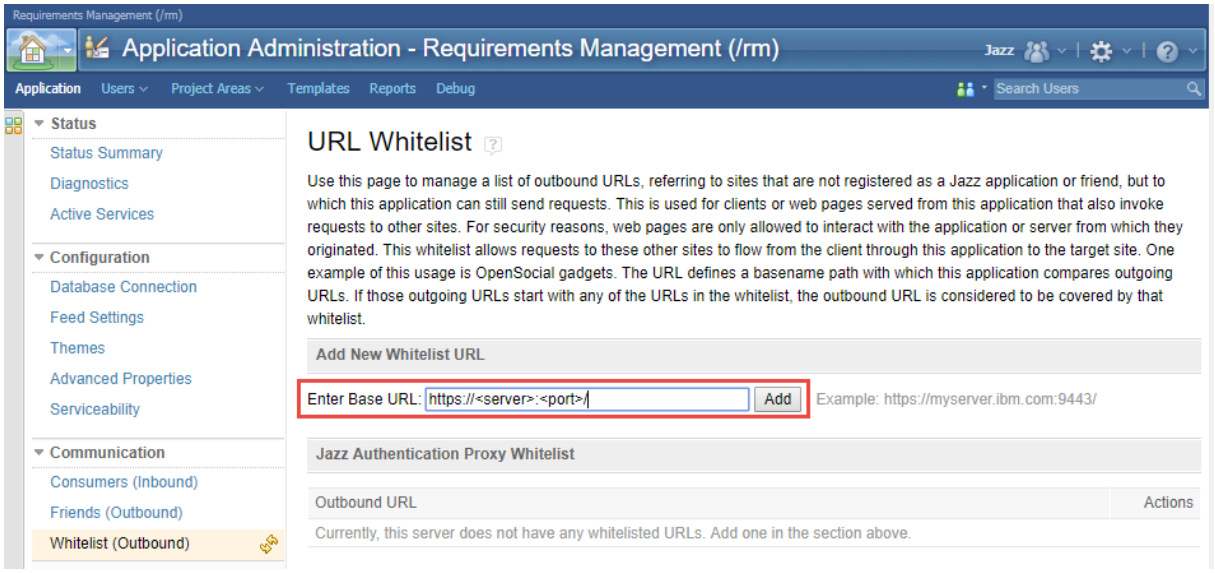
Figure 59. Jazz Authentication Proxy SSO Settings



Jazz RM Whitelist

As shown in Figure 60, “Adding A URL In RM Whitelist”, go to Jazz RM application administration page. Click the **Whitelist (Outbound)** on the left menu. In the **Enter Base URL** field of the **Add New Whitelist URL** section, enter the **Base URL** of the server where the pure::variants Integration for DOORS NG is installed. Click **Add** button and the newly added URL should be added to the **Outbound URL** list on the same page.

Figure 60. Adding A URL In RM Whitelist



Note

On clicking the **Add** button, Jazz tries to access the URL being added. And if the URL is accessible only then the URL is added to the list of **Outbound URL**.

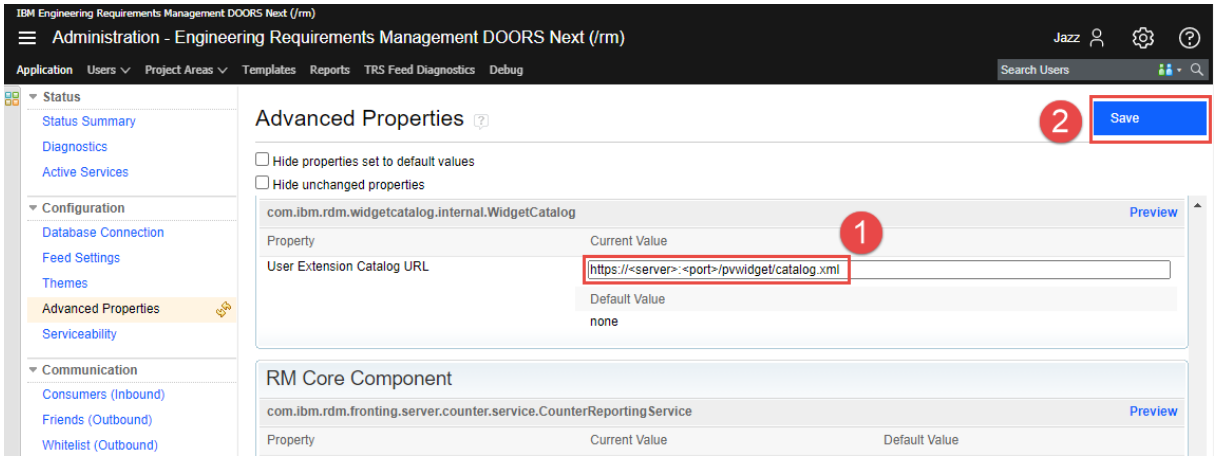
User Extension Catalog URL

Browse to Jazz RM Advanced Settings page. RM Advanced Settings page can be accessed by pointing the web browser to following URL:

```
https://<server>:<port>/rm/admin#action=com.ibm.team.repository.admin.configureAdvanced
```

Scroll to **User Extension Catalog URL** as shown in figure Figure 61, “RM User Catalog Configuration”.

Figure 61. RM User Catalog Configuration



Considering that the context path of the pure::variants Integration is **pvwidget**, enter the Catalog URL as `https://<server>:<port>/pvwidget/catalog.xml` and **Save** the settings.

Adding DOORS NG Integration in an already existing Catalog

Jazz RM allows configuration of only one Catalog URL. If you already have a Catalog URL configured in RM Advanced Properties. Then the pure::variants Integration for DOORS NG can be added to the existing catalog

document. Lets consider that pure::variants Integration for DOORS NG is deployed on a web server and accessbile via a URL like this:

```
https://server.local.com:8443/pvwidget/catalog.xml
```

Open the catalog xml document that is already configured in the RM Advanced Properties. Add the following ju:catalog-entry in the list of existing ju:catalog-entry nodes.

```
<ju:catalog-entry>
 <dc:title>pure::variants Integration</dc:title>
 <dc:description>Enables adding variability information to requirements in a DOORS NG module.
 Enables editing of Restrictions and Calculations. Also provides the functionality of creating
 a Variability Preview.</dc:description>
 <ju:gadget rdf:resource="https://server.local.com:8443/pvwidget/pvscl.xml"/>
 <ju:icon rdf:resource="https://server.local.com:8443/pvwidget/pv.png"/>
 <ju:preview rdf:resource="https://server.local.com:8443/pvwidget/preview.png"/>
 <ju:thumbnail rdf:resource="https://server.local.com:8443/pvwidget/thumbnail.png"/>
 <ju:category>pure::variants</ju:category>
 <ju:category>Requirements</ju:category>
</ju:catalog-entry>
```

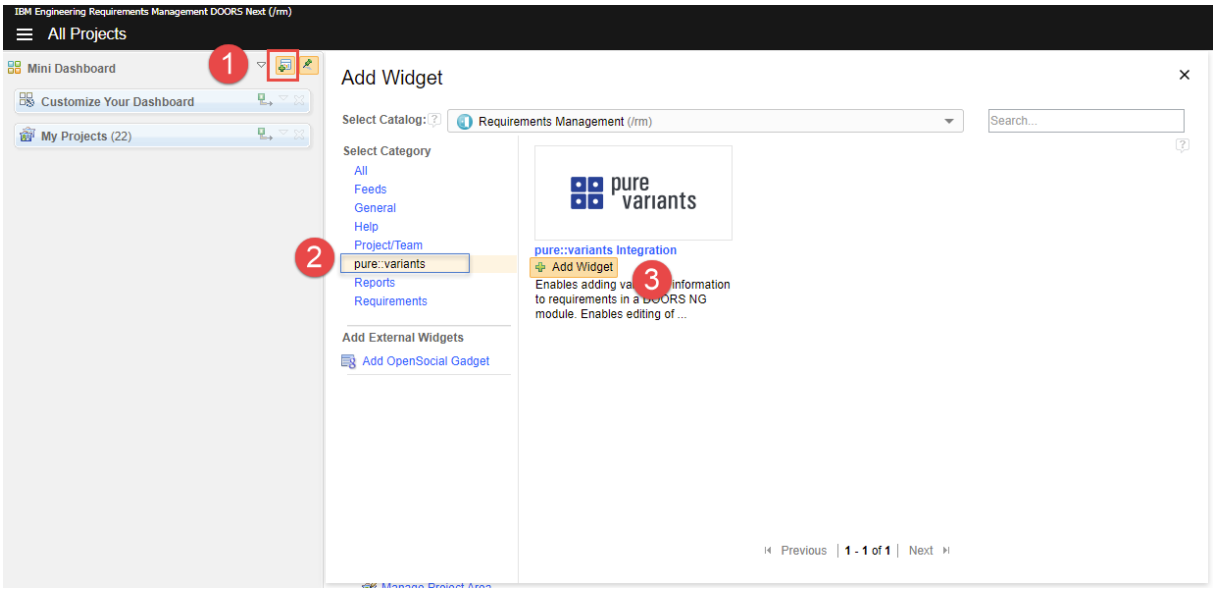
Note

Please note that the starting URL in the ju:gadeget, ju:icon, ju:preview and ju:thumbnail depends on the deployement of the pure::variants Integration. Try to access the ju:gadget URL in browser and see if the pvscl xml document is shown or not. If the xml document is shown in the browser then the pure::variants widget should be available in the 'Add Widget' dialog.

7.1.6. Add pure::variants Integration to DOORS NG

In order to interact with the Integration, it needs to be added inside Jazz' *Mini Dashboard*. Hence, open the *Mini Dashboard* on the left side panel inside DOORS NG. At the top left of Mini Dashboard, click the **Add Widget** button. A dialog opens as shown in the figure Figure 62, “Adding Integration inside Mini Dashboard”.

Figure 62. Adding Integration inside Mini Dashboard



On the left side of the dialog, in the widget category, click **pure::variants**. Then under the pure::variants Integration click the **Add Widget**.

For further instructions see section *Adding an OpenSocial gadget* in the documentation of IBM's *Rational Collaborative Lifecycle Management*

Note

If **pure::variants Integration for DOORS NG** is going to be used in **Web Hub** mode with **Jazz V7.0** then it is required that the pure::variants Web Components' relative domain-name must be same as DOORS NG's relative domain-name (e.g. *.**local.net**). For example, if DOORS NG is accessible via https://**server.local.net**:9443/rm then pure::variants Web Components must be accessible via URL like https://**otherserver.local.net**:8443/pv.

If a common domain-name is not possible, the WebComponents should be configured regarding Same-Site attributation for Cookies. For this, please see the optional step in our Tomcat resp. WebSphere configuration guide.

7.1.7. Check-up list for a successful deployment

The following list contains all necessary steps to be have taken to make successful deployment of the DOORS NG integration along with the DOORS NG application:

- Verify the web application server is started with a Java/JDK/JRE, which is not provided by IBM

  Hint: Choose any other Java vendor, like Oracle, Amazon, Adoptium, OpenJDK instead.

- Verify the web application server is configured to be accessed via HTTPS/SSL endpoint.

  Hint: Try to access via browser, using the HTTPS scheme, e.g. https://<server:port>/<widget>/pvscl.xml

- Verify the web application server is configured to have the SSL certificate of the DOORS NG application trusted.

  Hint: See section *Configuration of Trust Store* (Apache Tomcat or WebSphere Liberty) for defining a truststore, which contains the DOORS NG application's SSL certificate.

- Verify the web application server can resolve the hostname of the DOORS NG application's URL.

  Hint: Log-in to the web application server's machine, and try to follow the link of DOORS NG application URL, with help of browser application, or any other command-line tool, like *wget* or *curl*.

- Verify the web application server has deployed the DOORS NG integration functional, by accessing the the following URL: *https://<server:port>/<widget>/vel* (Please substitute with proper server and port, if other than 443, and use the appropriate context-path).

  Hint: If successful, you should see a webpage showing the following line:

```
{"errorClass":"","errorMessage":"Module URI not received","errorCode":400}
```

7.1.8. Pre-defined settings for Web Integration in DOORS NG

The Web Integration for DOORS NG allows you to pre-define specific settings to simplify or limit the setup for end-user.

Please see chapter Section 7.2, “Pre-defined settings for Web Integration” for detailed explanations, which settings are available and how they are configured.

To use these pre-defined settings for DOORS NG Integration, you need to write the configuration (in JSON format) into a file called *settings.json* and put this into the deployment directory.

If your Web Application Server (like Tomcat or WebSphere Libterty) extracts the war-archive automatically, please add the settings.json file into the war-archive (called "com.ps.consul.web.ui.doorsng.*.war"

If your Web Application Server (like Tomcat or WebSphere Libterty) does not extract the war-archive automatically and you extract it manually, please add the *settings.json* into the deployment directory of the Integration.

Finally, the deployment directory should have located the *settings.json* next to the *pvscl.xml*, as follows:

```
|-META-INF
|-WEB-INF
|-pvscl.xml
|-settings.json
|-... (other files)
```

7.2. Pre-defined settings for Web Integration

As administrator, one can pre-define the settings for the end-user, e.g. to connect to the WebHub immediately. Thus, the user does not need to be aware of the connection configuration and can use the Widget out-of-the-box.

Therefore, a ***JSON*** structure must be created as described in the following for WebHub and DesktopHub, respectively, and deployed:

```
{
     "PV_HUB": "webhub",
     "PV_HUB_URL": "https://webhub.server/pv",
     "PV_HUB_EDITABLE": true,
     "PV_HUB_URL_EDITABLE": true
}
```

```
{
     "PV_HUB": "desktophub",
     "PV_HUB_PORT": "4773",
     "PV_HUB_EDITABLE": true
}
```

The *PV_HUB* property defines the hub to be used, which can be either the WebHub (*webhub*) or DesktopHub (*desktophub*).

The *PV_HUB_PORT* property defines the port number for the DesktopHub. It can be configured as either a string or a number

The *PV_HUB_URL* property defines the URL for the WebHub.

The *PV_HUB_EDITABLE* property defines, if the hub should be editable by the user. If it is undefined, the hub is editable by the user in the widget's settings.

The *PV_HUB_URL_EDITABLE* property defines, if the webhub URL should be editable by the user. If it is undefined, the webhub URL is editable by the user in the widget's settings.

Note

Regarding the deployment of the JSON structure, see the previous sections.

7.3. Friend Setup

You need to add **pure::variants Web Components** as a **Friend** to **Jazz Team Server (JTS)** in order to enable JTS to access the pure::variants web components data. Jazz Official Documentation can be visited for details.

Note

After finishing the pure::variants web components setup and before starting the friend setup in JTS, it is a must that the web server hosting the pure::variants web components must be restarted.

7.3.1. SSO mode

Procedure

1. Browse to the **JTS Admin page**. Then click **Manage Server** and then click **Friends (Outbound)**. This page can also be directly accessed by pointing your web browser to **https://[jazz-server-FQDN]:9443/jts/admin#action=com.ibm.team.repository.admin.friends**.

2. On the Friends Page, in the **Friends List** section, click **Add**.

3. In the **Add Friend** dialog, provide the following information.

   - Provide the URI for root services of the pure::variants Web Components in this form: **https://[pv-server-FQDN]:[port]/pv/rootservices**.

   - Enter a name to identify the pure::variants web components e.g. VM.

   - Click Next to continue.

4. As both jazz and pure::variants web components are configured in SSO mode that's why this dialog does not ask for any credentials. Click **Create Friend** and click **Finish** to close the wizard.

Note

On step 4 of the above procedure, if you are asked to enter OAuth credentials then it is a problem. It means that JTS is unable to recognize that the pure::variants web components is also SSO enabled and registered with the same Jazz Authorization Server as JTS. In this case you have to close the friend setup dialog and wait for 10 to 15 minutes and start the friend setup again. If the problem persists then you have to restart the jazz server and then do the friend setup again.