Workflow function and condition Example

See also main SDK page.

WFCE - Introduction

The workflow example is a custom workflow function and condition:

- The **Compute Total Life Time** function will count the time from creation of a work item until the action when status will be set to closed.
- The **Comments Exist** condition will check the state of a work item and if the result is positive then the action will be allowed (and available).

WFCE - what's different between function and condition

Workflow condition decides whether a certain action will be performed. If condition is satisfied then action will be available. Workflow function will do something immediately after confirmation of changes made. So condition is always tested before you start to edit a work item and function is performed when you finish.

WFCE - Work-flow function: Compute Total Life Time

The typical usage of this function is to count the time from creation of a work item until the action triggering setting of status to closed - i.e. the user marks the work item as done. (Recommended use is to set this function for transitions to the Closed state.)

This function has one parameter with name 'field' the value of which is the name of a custom field. The custom field type must be 'string'. The result will be saved this custom field only, and the field value will be the time from creation of the work item to this operation. See ~Configuration~ section to see how to set it in Polarion.

WFCE - Work-flow condition: Comments Exist

The typical usage of workflow condition is to check something before an action is performed, e.g. transition to another state. The result is always a Boolean value, and 'true' means success.

This condition checks whether any comments exist. Returns 'true' if at least one comment exists.

We can assume that we have a work item in the 'open' state and we set the condition for the action 'Resolve and Close':

- Number of comments is equal to 0 which implies that available actions are: Accept, Resolve.
- Number of comments is greater than 0 which implies that available actions are: Accept, Resolve, Resolve and Close.

WFCE - Java API Workspace preparation

See section *Workspace preparation*

WFCE - Creating project plugin

You can either create or import build project.

WFCE - Import of the example

Info: You must ensure that your plugin is compiled against your Polarion version. This example contains a precompiled jar plugin. You can remove it before you start developing your own plugin based on this example. Eclipse ensures that the new jar plugin will be created against your source code and Polarion version.

To import workflow project example to workspace, do these steps:

1. Select **File** > **Import...**
2. In the dialog that appears, select **Existing Project into Workspace** (in the **General** section) and press the Next button.
   Press the **Browse..** button, select the directory of examples (usually in

3. Press the **Browse..** button, select the directory of examples (usually in `C:\Polarion\polarion\SDK\examples\`). Submit it.
4. Select `com.polarion.example.workflow` and press Finish.

WFCE - Develop your own plug-in

- First, you have to create new plug-in project. Fill Plug-In Properties and uncheck **Generate activator..**
- Afterwards, open `MANIFEST.MF` and set `com.polarion.alm.tracker` as a Required Plug-in in the Dependencies page. You should also set, in the Build page, the `src/` folder as the source folder that should be compiled into the exported library.

  Figure 1. build.properties file content

  ```
  source.. = src/
  output.. = bin/
  source.example-plugin.jar = src/
  bin.includes = META-INF/,\
                 example-plugin.jar,\
                 .
  ```

- To let Polarion Server know that you have created new workflow function or condition, you have to create a **META-INF** directory in the **src** folder, and place the `hivemodule.xml` file there. See `hivemodule.xml` of the example for more information.

WFCE - Build and deployment

There are two basic types of plugins for Polarion. First is a 'standard' plugin that contains only source code. The second is a 'web application' plugin, or plugin with external resources. For example, web application using special directory called ~webapp~.
This directory contains external resources (web pages, images) which will be deployed by a web server to Polarion (see also Servlet Example, how to add webapp directory to web server). If you want to develop a web based application or plugin with external resources for Polarion you must separate the webapp or resource directory from the jar plugin.
This can be done with the `build.properties` file. You can add these lines to `build.properties`:

```
bin.includes = META-INF/,\
               plugin.xml,\
               webapp/,\
               example-plugin.jar
```

As we can see, the `example-plugin.jar` is our plugin for Polarion which will be deployed in the `plugin` directory. This plugin directory also contains the `webapp` directory with our pages and images.
The advantage of this deployment is that you are able to access these resource, because it's not packed in the jar plugin directly.

WFCE - Deployment to Installed Polarion

See section *Deployment to Installed Polarion*

WFCE - Execution from Workspace

See section *Execution from Workspace*

WFCE - Configuration

After successful deployment of your plug-in to Polarion, you can start using new the workflow function and condition. To check, that deployment was successful do following steps:

- Enter the Administration interface, open the project in which you would like to set new function.
- Go to Work Items > Workflow
- Go to right-most column and create a new configuration for a specific type of work item.
- In the Work Flow Designer, go to last portlet 'Actions' and click on the Edit icon (a white check mark in purple circle) on the row for any action. It will display a popup editor of the action details (conditions and functions) which are already specified. There you should see the new condition and function.

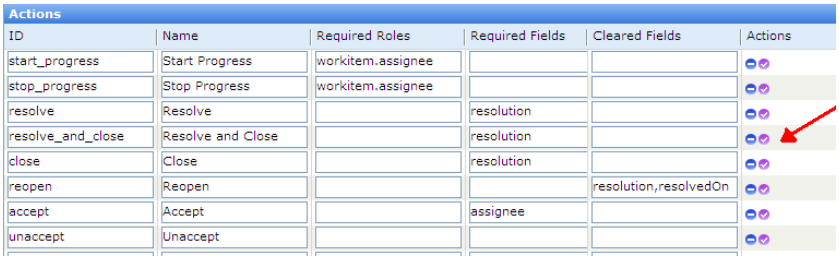

Figure WFCE-1: Choose the action, where you would like to set condition or function

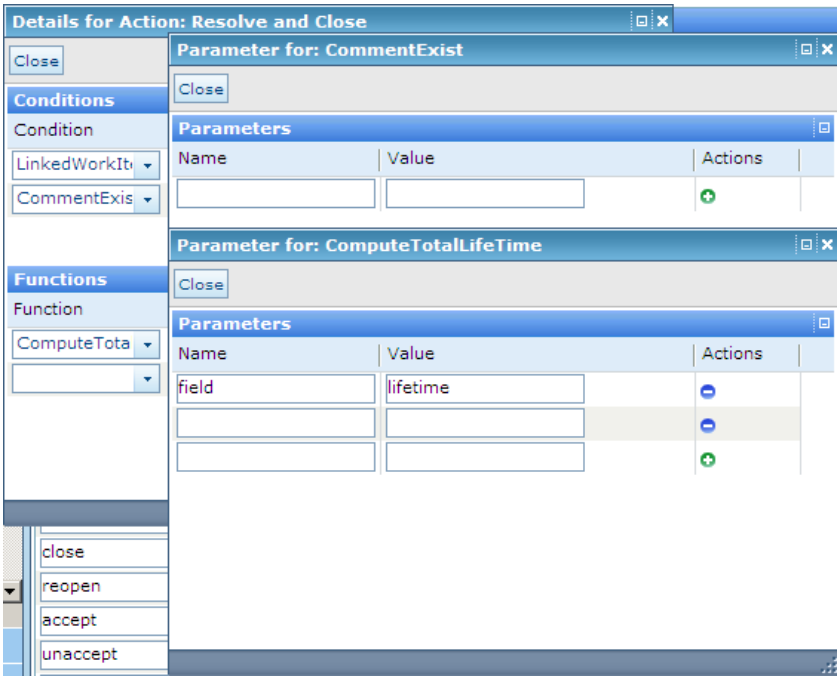

Figure WFCE-2: Set properties according the image

Requirements

Development Environments

- Eclipse IDE for Enterprise Java Developers or any other Eclipse IDE with The Eclipse Plug-in Development Environment. (Go to Help > Install New Software… > Install Eclipse Plug-in Development Environment > Restart Eclipse)
- AdoptOpenJDK 11 for building and running your code.

Workspace Preparation

To start developing a Polarion Java API plug-in, you first need to perform following steps:

1. Start Eclipse, then select **Window** > **Preferences...**
2. In the dialog that appears, select **Plug-In Development** > **Target Platform**.
3. Click the **Add** button on the right.
4. Keep the **Nothing: Start with an empty target definition** option selected and click **Next**.

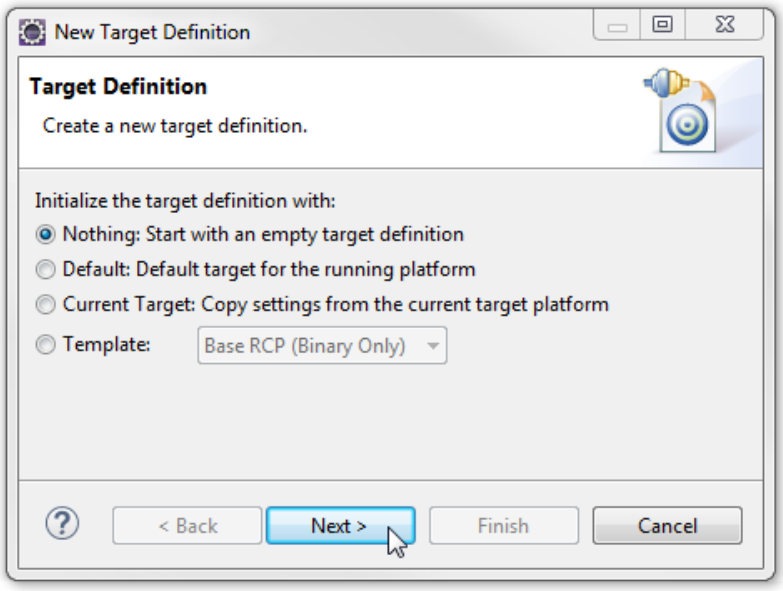

Figure WP-1: Starting with an Empty Target Definition

5. Enter a **Name** and click **Add**.
6. Select **Directory** and click **Next**.
7. Click **Browse** and select the `C:\Polarion\polarion` folder *(Windows)* or `/opt/polarion/polarion` *(Linux)*. (One level above the *plugins* folder.)
8. Click **Next**.

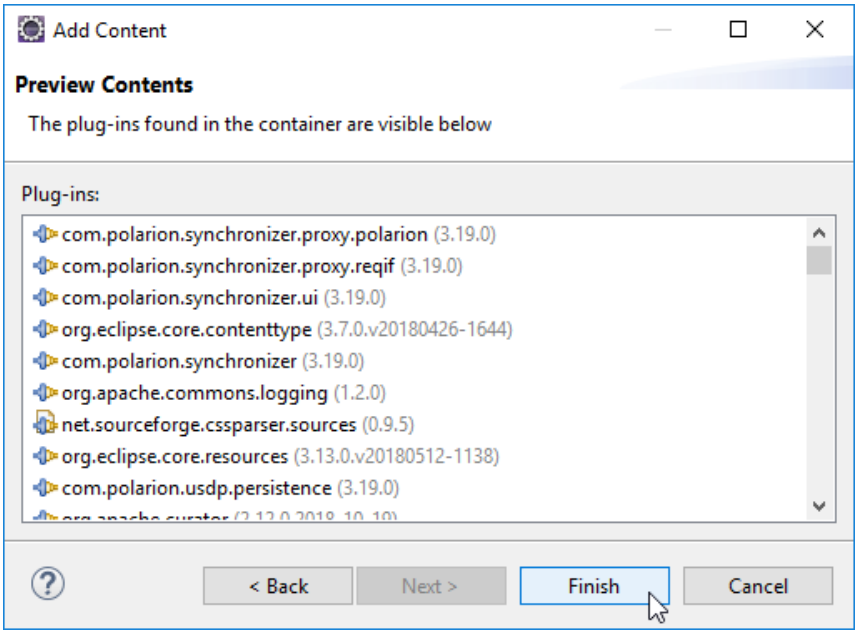

Figure WP-2: Currently Installed Polarion Plug-ins

9. A list of currently installed Polarion plug-ins appears. Click **Finish**.

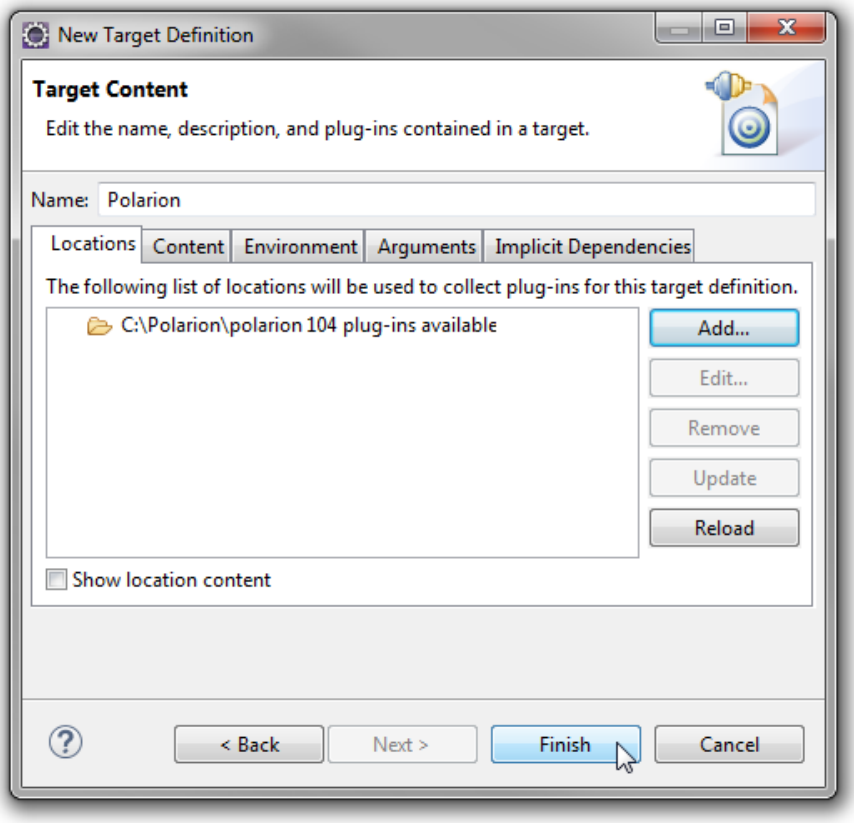

Figure WP-2: Confirm the Selected Path

10. The selected path and the number of discovered plug-ins available appear. Confirm that the path is correct and click **Finish**.

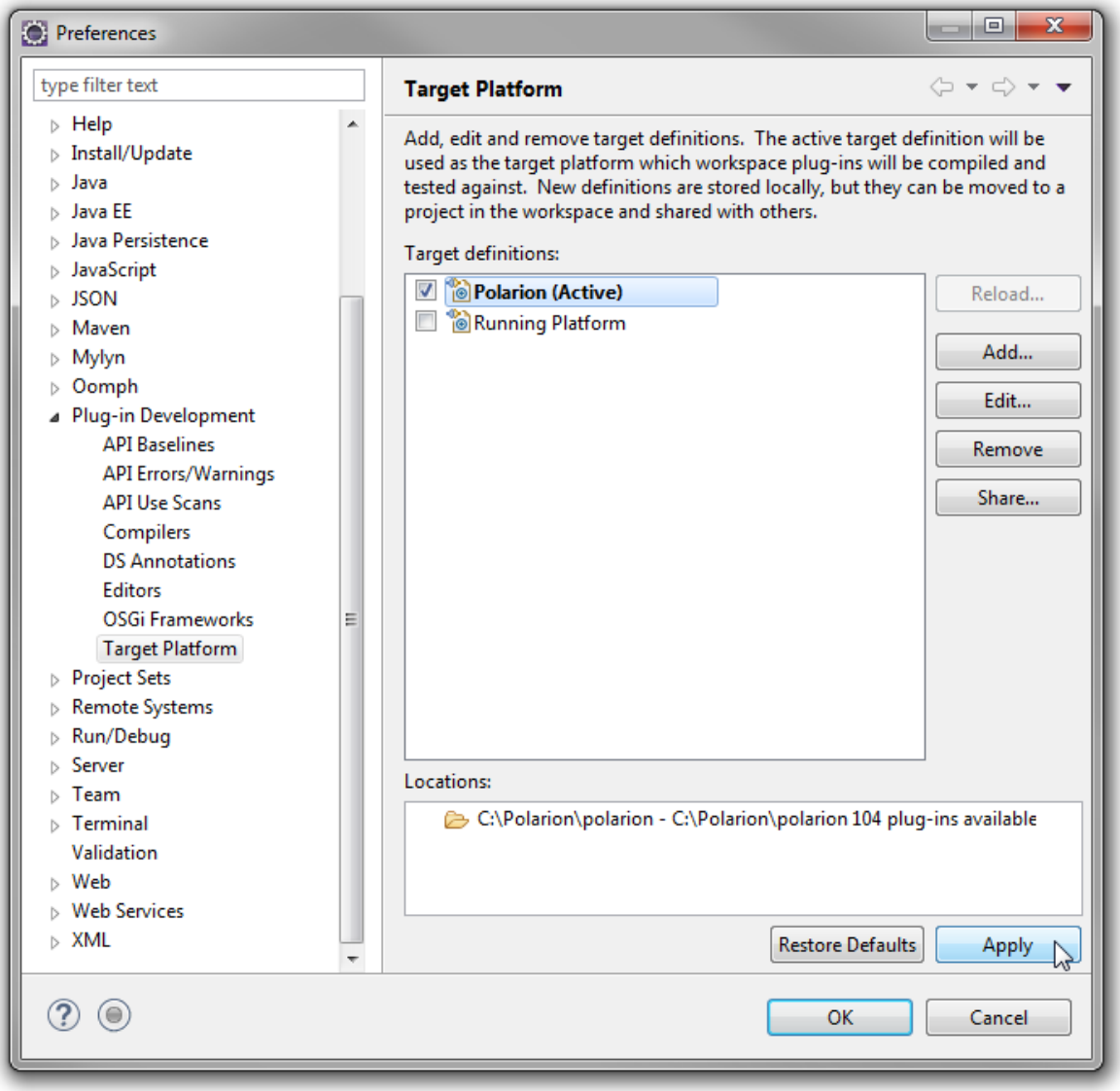

Figure WP-2: Select the Target Platform

11. Check the box beside the newly added path and click **Apply**.

Deployment to Installed Polarion

You can deploy a plugin to Polarion in two ways. First you can export a project as **Deployable Plugins and Fragments**. The second way is described in the following section *Execution from Workspace*. To export the plug-in, perform these steps:

1. Select **File** > **Export...**
2. In the dialog that appears, select **Deployable Plugins and Fragments** in **Plug-in Development** section and click the **Next** button.
3. Mark your project (e.g. for **Servlet** example it will be `com.polarion.example.servlet`), and as the destination directory specify the `polarion` folder of your Polarion installation directory (usually in `C:\Polarion\polarion`)
4. At the **Options** card be sure, that **Package plug-ins as individual JAR archives** is unchecked. Click *Finish*.
5. Because this is a new polarion plug-in extension, you have to restart your Polarion server.

NOTE: Servlets loaded by Polarion are cached in: `[Polarion_Home]\data\workspace\.config`. If this folder is not deleted before deploying a servlet extension (plugin) and restarting Polarion, then either the servlets will not be properly loaded, or the old ones will be loaded.

Execution from Workspace

The second way to deploy the plug-in to Polarion is to launch Polarion directly from your Eclipse workspace. This

method has the added advantage of debugging the code directly in Eclipse.

1. Select **Debug** > Open **Debug Configurations**..
2. Create a new Eclipse application (double click on *Eclipse Application*)
3. You should set:

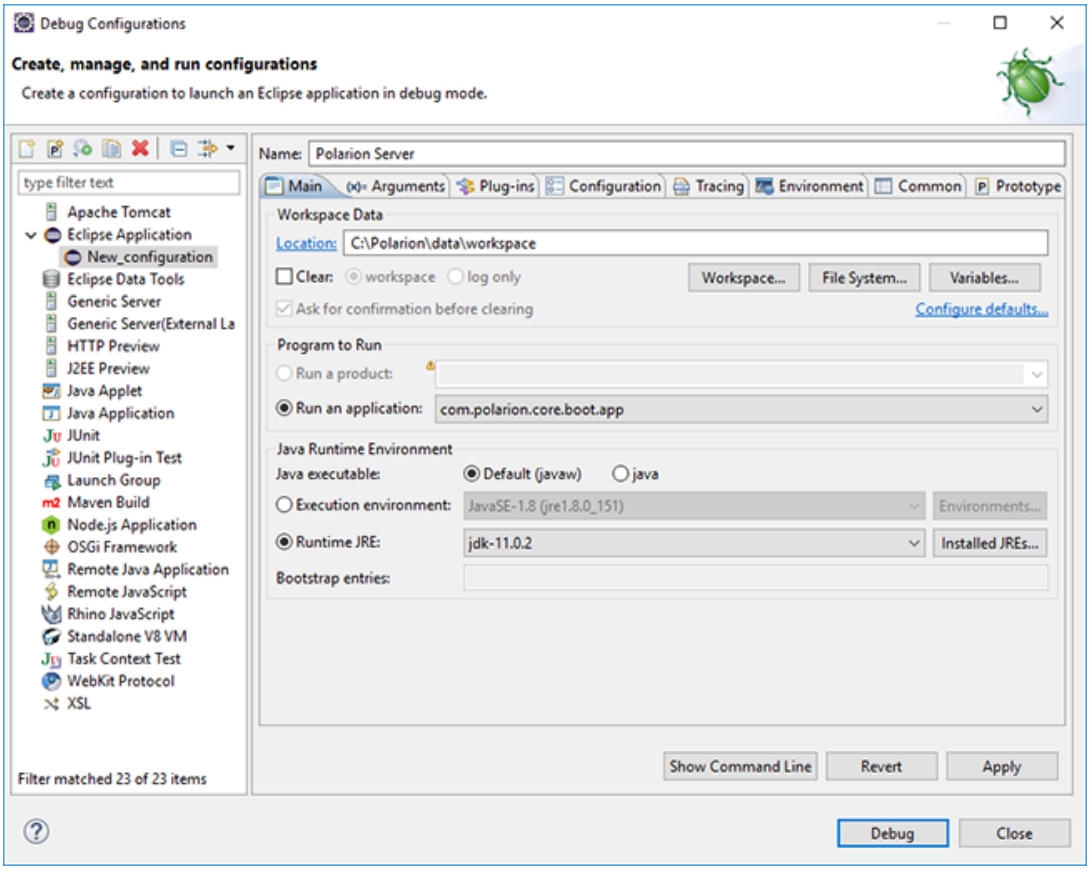

Figure WP-2: Debug - Main page

   - **Name** to `Polarion Server`
   - **Workspace Data Location** to `C:\Polarion\data\workspace` (assuming that your Polarion is installed in `C:\Polarion\`).
   - **Run an application** to `com.polarion.core.boot.app` in the **Program to Run** section.
4. Finally set your Runtime JRE. On the second, "**Program Arguments**" tab, set the following arguments:

In the **Program Arguments** section:

Windows:

```
-os win32 -ws win32 -arch x86 -appId polarion.server
```

Linux:

```
-os linux -ws gtk -arch x86_64 -appId polarion.server
```

In the **VM Arguments** section:

Windows:

```
-Xms256m -Xmx640m
```

```
-Dcom.polarion.home=C:\Polarion\polarion
```

Linux

```
-Xms256m -Xmx640m
```

```
-Dcom.polarion.home=/opt/polarion/polarion -Dcom.polarion.propertyFile=/opt/polarion/etc/polarion.properties
```

5. You must now change the parameters to the Polarion server based on your installation. You can check the settings with the following screenshot:

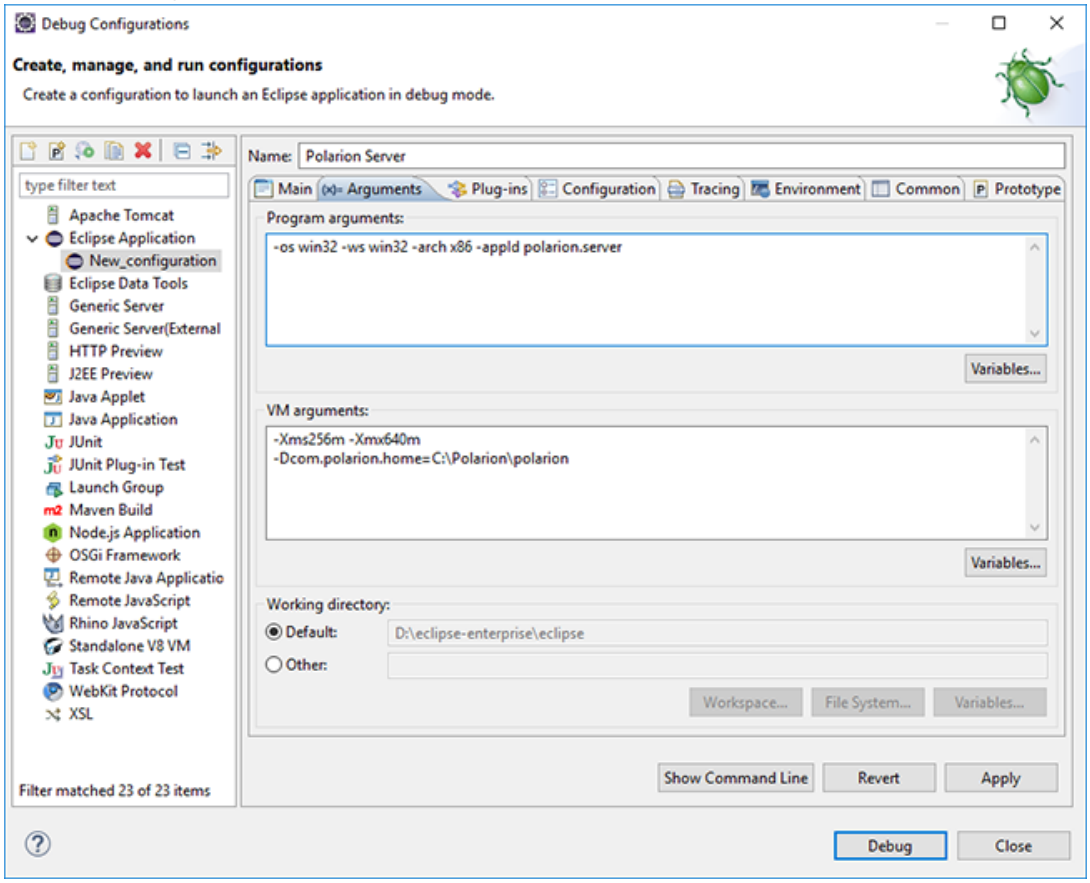

Figure WP-3: Debug - Arguments page

6. On the third "**Plugins**" tab, make sure, you have also selected "**Target Platform**" plugins.

7. Select all, and then click the **Validate Plug-ins** button. If there are some problems, uncheck the plugins which are in conflict.

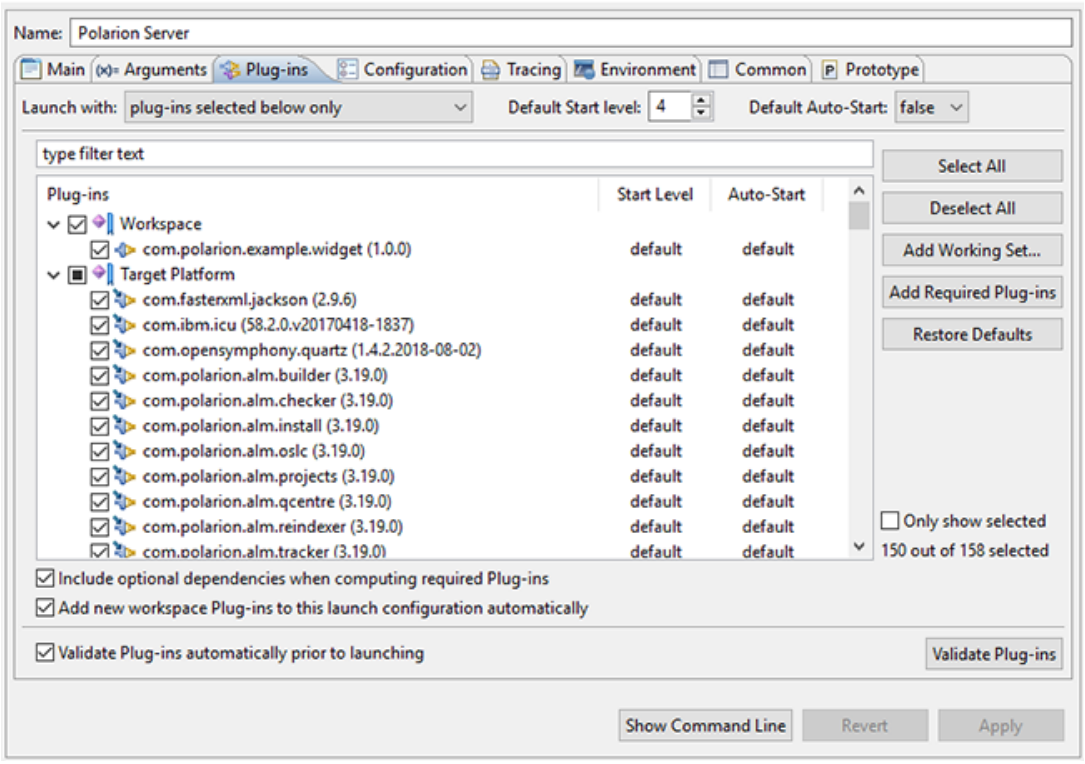

Figure WP-4: Debug - Plug-ins page

8. Other pages shouldn't be changed. Just click the **Debug** button, and go on with your new Polarion Server application.