OPC UA
SDK for Java

Starting Guide

Version 4.0.0

Table of Contents

This guide is intended for new Java developers and covers the basics of setting up a new Java project in the Eclipse IDE in order to start using the Prosys OPC UA SDK for Java.

This is a basic guide. It is a trade-off between doing things quick and easy versus doing things according to what is considered to be modern Java development.

1. Tools for Java Development

1.1. Integrated Development Environment (IDE)

Java is a programming language with very good tools. Except for some specific cases, it is generally recommended that you use an Integrated Development Environment (IDE) program to write your own Java applications instead of a bare text editor. IDEs offer a lot of support in the form of autocompletion, following code paths, displaying (java)docs, compiling while you are typing to spot errors early, etc.

This guide uses Eclipse as the example IDE. It is free and open source. The reason for selecting Eclipse is that the writers of this guide have the most experience with it.

Other popular IDEs are the IntelliJ IDEA and the NetBeans IDE.

1.2. Java Runtime Environment (JRE) and Java Development Kit (JDK)

You need a JRE in order to run Java programs and a JDK to compile them. Eclipse is a Java program by itself and requires a JRE to run. It does have it's own compiler, so technically you do not need a JDK to start out, however, in general you will need it at some point. Therefore, it is best to just install it right away.

There are multiple JRE/JDK vendors to choose from. Obtaining and installing one is somewhat outside of this guide. Due to the Oracle's current licensing policies, you'll probably want to at least start with an OpenJDK. There are multiple vendors for OpenJDK, choosing one is outside of this guide, but e.g. https://adoptopenjdk.net/ should work. If you are running on Linux, your distro's package manager typically has an OpenJDK available.

This guide expects you to have Java version 8 runtime.

The Prosys OPC UA SDK for Java (4.0.0) is expected to work on Java versions from 6 to 8. At the moment of writing this guide, later versions of Java are not (yet) actively tested to work with the Prosys OPC UA SDK for Java. Java is in general a very backwards compatible language, but in version 9 it introduced some compatibility-breaking changes.

After obtaining and installing a JDK, you should be able to run `java -version` on the command line. You may skip checking this and just try to start Eclipse. This is the first thing to try, because Eclipse is a Java program by itself. Eclipse requires that the `bin` folders of the installed JRE and JDK are placed in your PATH or alternatively configured in the 'eclipse.ini' file (in the Eclipse installation folder).

This tutorial uses Java version 8 because it is required by the latest Eclipse version as of the writing of this guide. This does also mean that by default your applications also require Java 8 in order to run. Nevertheless, it is possible to use different Java version to compile a project, but running Eclipse itself does require Java 8.

1.3. Build and dependency management tools

Typically, a Java project uses some kind of tool for building the final package of the project (to be used by users or as a dependency to another project). The same tool is usually handling all the dependencies that the project might have.

This guide ignores this part and instead stops at the point where you can run the samples provided in the SDK package from Eclipse. However, learning to use these tools is eventually very recommended (they all have their advantages and disadvantages and having a basic understanding of all of them is beneficial).

Common tools in this category are:

- Apache Maven
- Gradle
- Apache Ant (sometimes combined with Apache Ivy)

2. Using the Prosys OPC UA SDK for Java in Eclipse IDE

This chapter demonstrates how to create a new Java project in the Eclipse IDE and how to import the Prosys OPC UA SDK for Java into the project so that you can easily get started on testing the features of the SDK. After completing the steps in this chapter, you should be ready to go through the tutorials for developing OPC UA client and server applications found next to this guide in the SDK distribution package.

The step-by-step instructions provided in this chapter are based on Eclipse 2018-12 Release (4.10.0) running on Windows 10. The steps might be slightly different on other versions of Eclipse or in other operating systems.

2.1. Creating New Java Project

To create a new Java project for using the SDK in Eclipse, follow the steps below:

1. Go to the File menu and click New → Java Project.

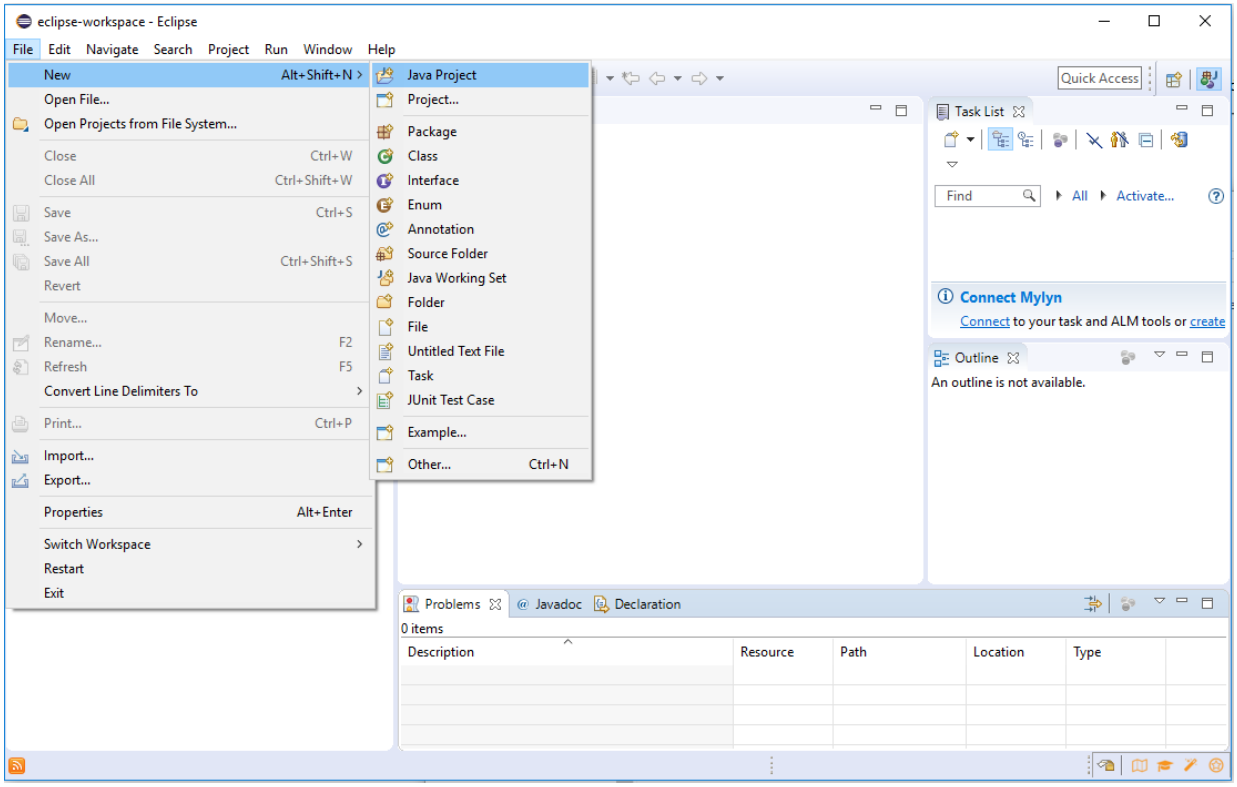

Figure 1. New Java Project.

2. Enter a Project name, e.g. "SDK Test".
3. The Use default location checkbox is enabled by default and points to the Eclipse workspace, which is suitable for this tutorial. You may untick the checkbox and define another location where the project will be saved.

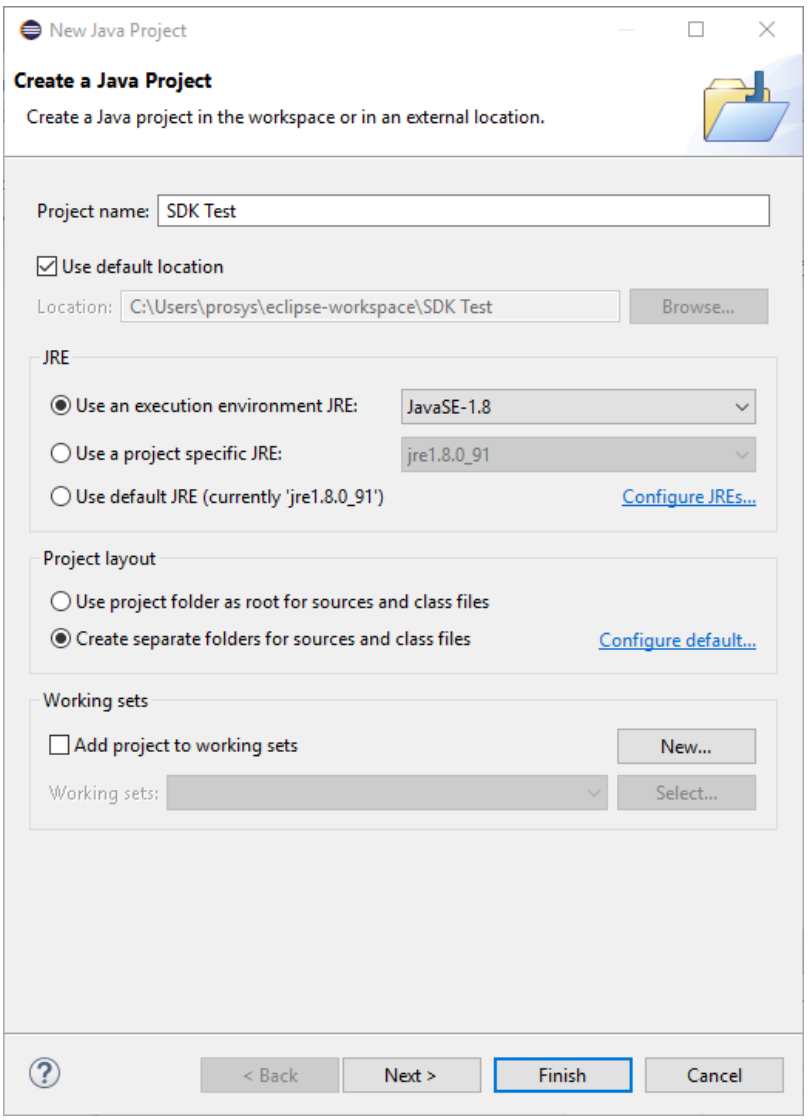

Figure 2. Settings for new Java project.

4. The other settings are fine with their default values.

2.2. Importing the Prosys OPC UA SDK for Java

After creating a new empty Java project in Eclipse, follow these steps to import the required SDK files to the project so that you can start testing the functionality:

1. Open the project folder (either in Eclipse or your operating system's file manager).
2. Create two new folders called lib and javadoc at the project root folder. A src folder should already be created, but if not, then create it as well.

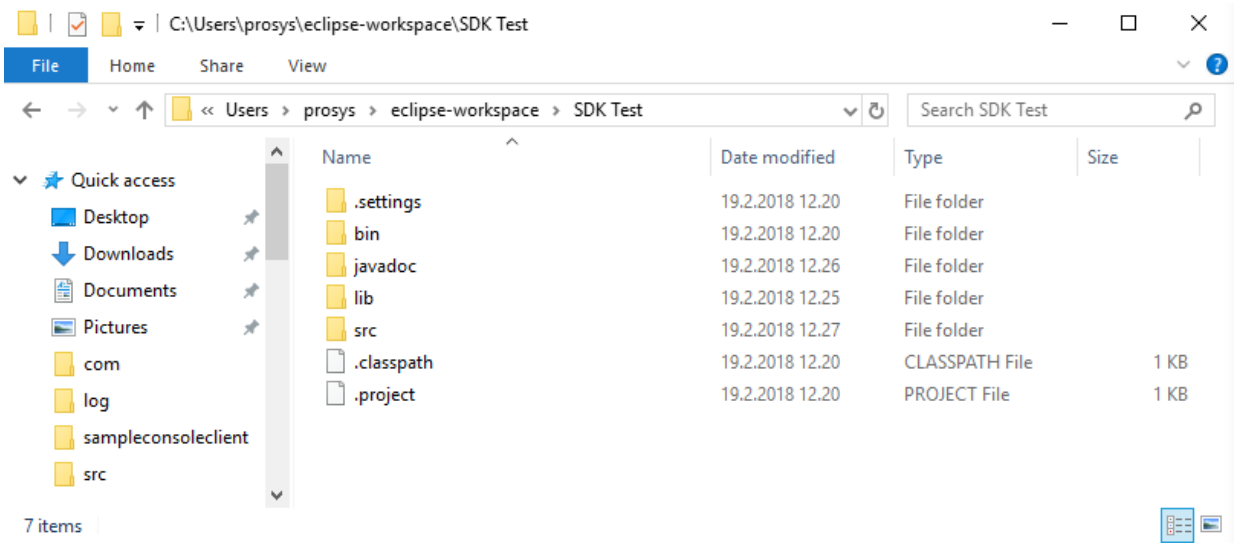

Figure 3. Project root folder.

3. Copy the contents of the samples/sampleconsoleserver/src/main/java (available only with the server version of the SDK) and samples/sampleconsoleclient/src/main/java of the SDK package into the src folder (no need to copy the .bat, .sh and pom.xml files from the base folder). You can replace the files in the destination if prompted.
4. Copy the contents of the lib folder of the SDK package into the lib folder of the project.
5. Copy the contents of the javadoc folder of the SDK package into the javadoc folder of the project.
6. If you did the operations in your operating system's file manager, refresh the project in Eclipse (by selecting it and either pressing F5 or via right-click menu).

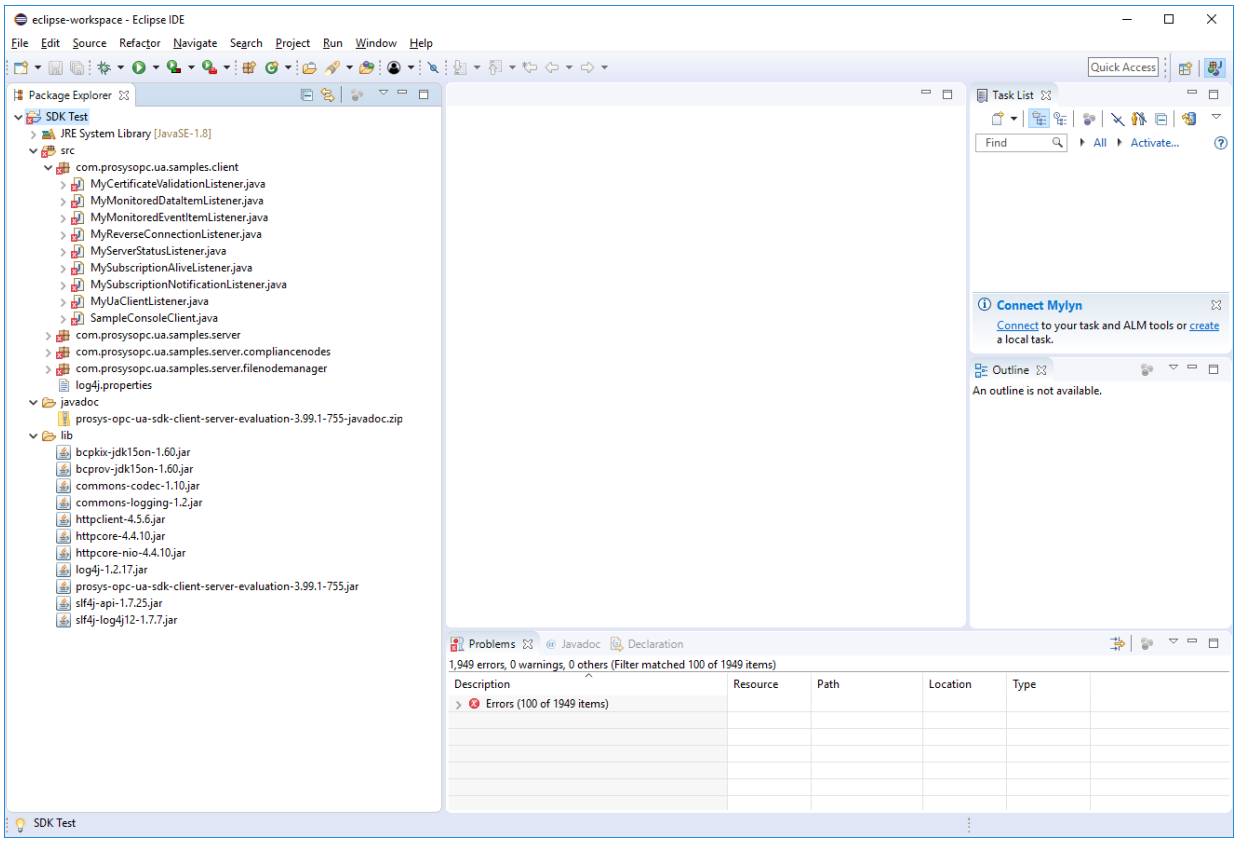

Figure 4. View of the Eclipse Package Explorer after adding the required files to the project folder and refreshing the project.

7. Add each file from the lib folder to the projects Build Path (select one or more and right-click and click Build Path → Add to Build Path).

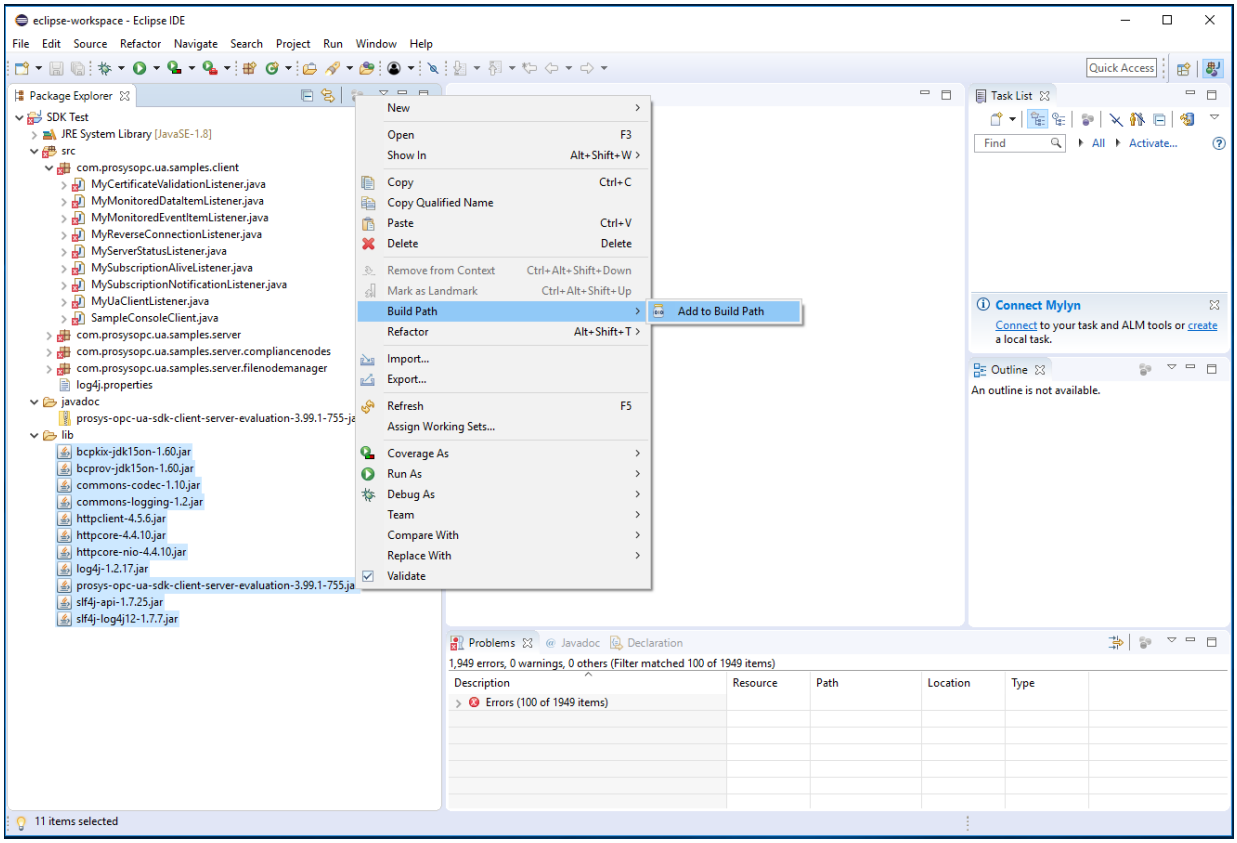

Figure 5. Add library to Build Path.

8. Finally, you can expand the src folder and find the SampleConsoleClient and SampleConsoleServer Java classes. You can run these by right-clicking them and selecting Run As → Java Application.

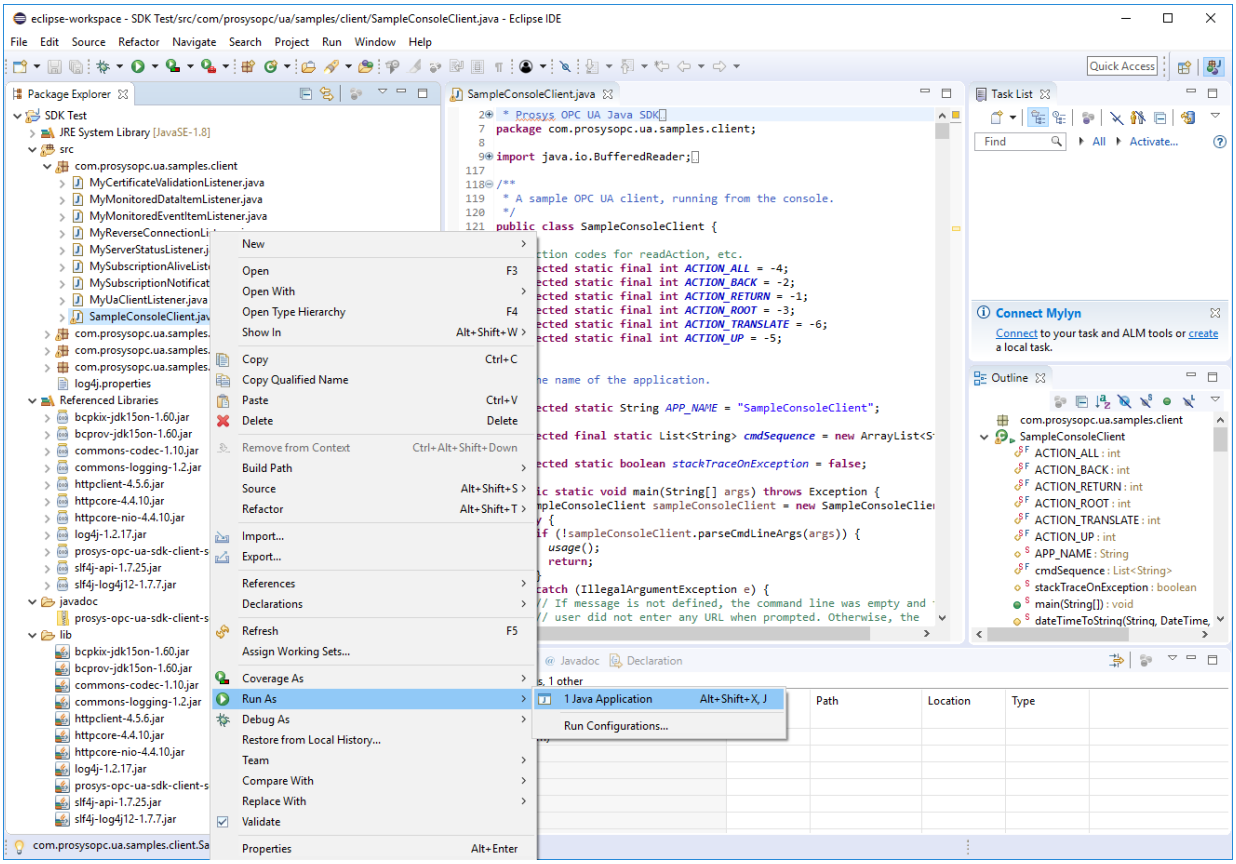

Figure 6. Running the sample application.

2.3. Adding Javadoc documentation

For development, it is highly recommended that the Javadoc code documentations are attached to the provided libraries. The SDK package provides Javadocs for the Prosys OPC UA SDK for Java. The Javadocs can be attached to their respective libraries in Eclipse with these steps:

1. Right-click on the library for the SDK or the Stack in the Referenced Libraries section of the Eclipse Package Explorer. The prosys-opc-ua-sdk-client-server-evaluation-4.x.x-xxx.jar is the libraries for the SDK.
2. From the pop-up menu select Properties.

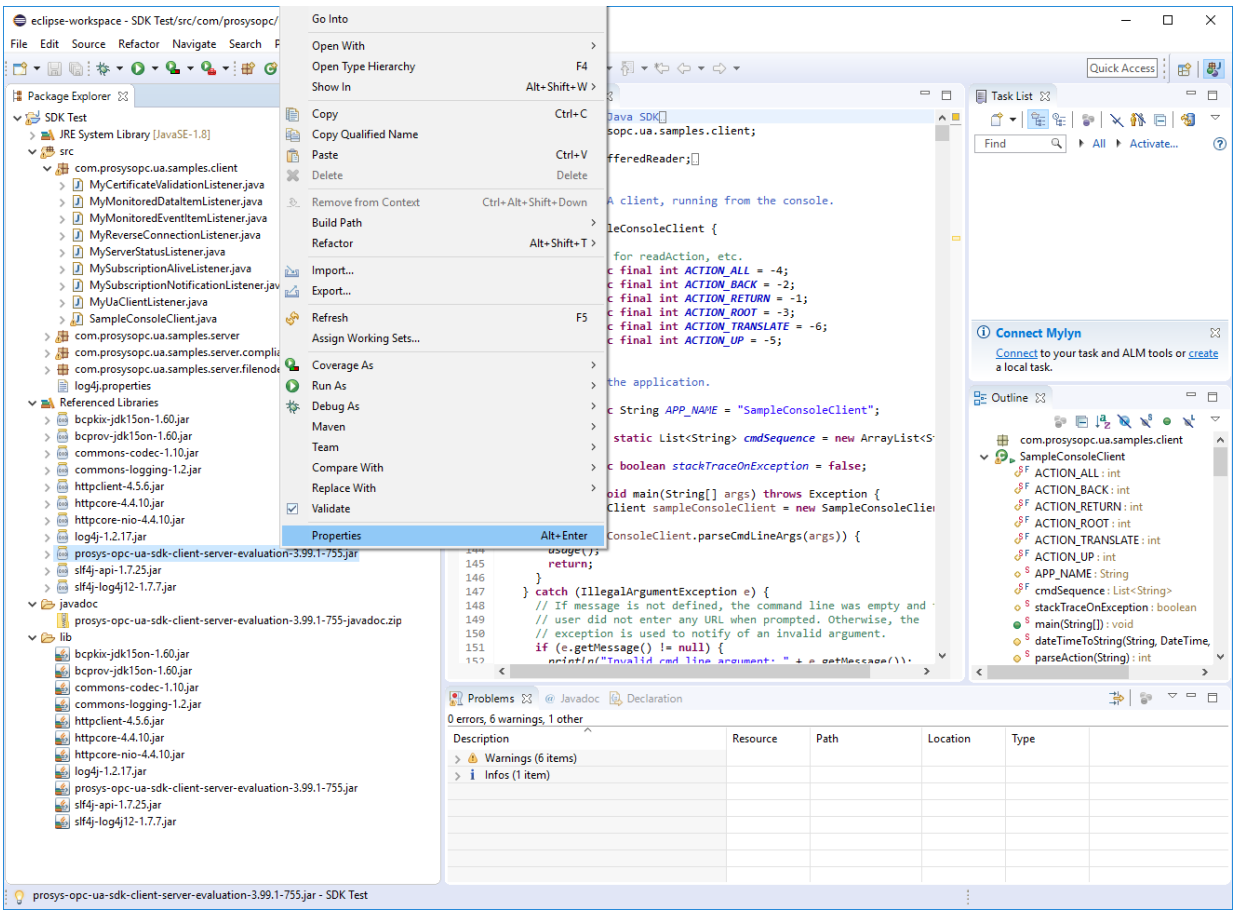

Figure 7. Library Properties.

3. Click Javadoc Location from the left-side menu.
4. Select Javadoc in archive.
5. Select Workspace file.
6. Click Browse next to the Archive path: field.
7. Locate the javadoc folder under the root folder (named according to what you defined in the beginning).
8. Inside the javadoc folder, you should click the Javadoc file that should be associated to the current library, i.e. prosys-opc-ua-sdk-client-server-evaluation-4.x.x-xxx-javadoc.zip for the SDK.
9. Click OK.

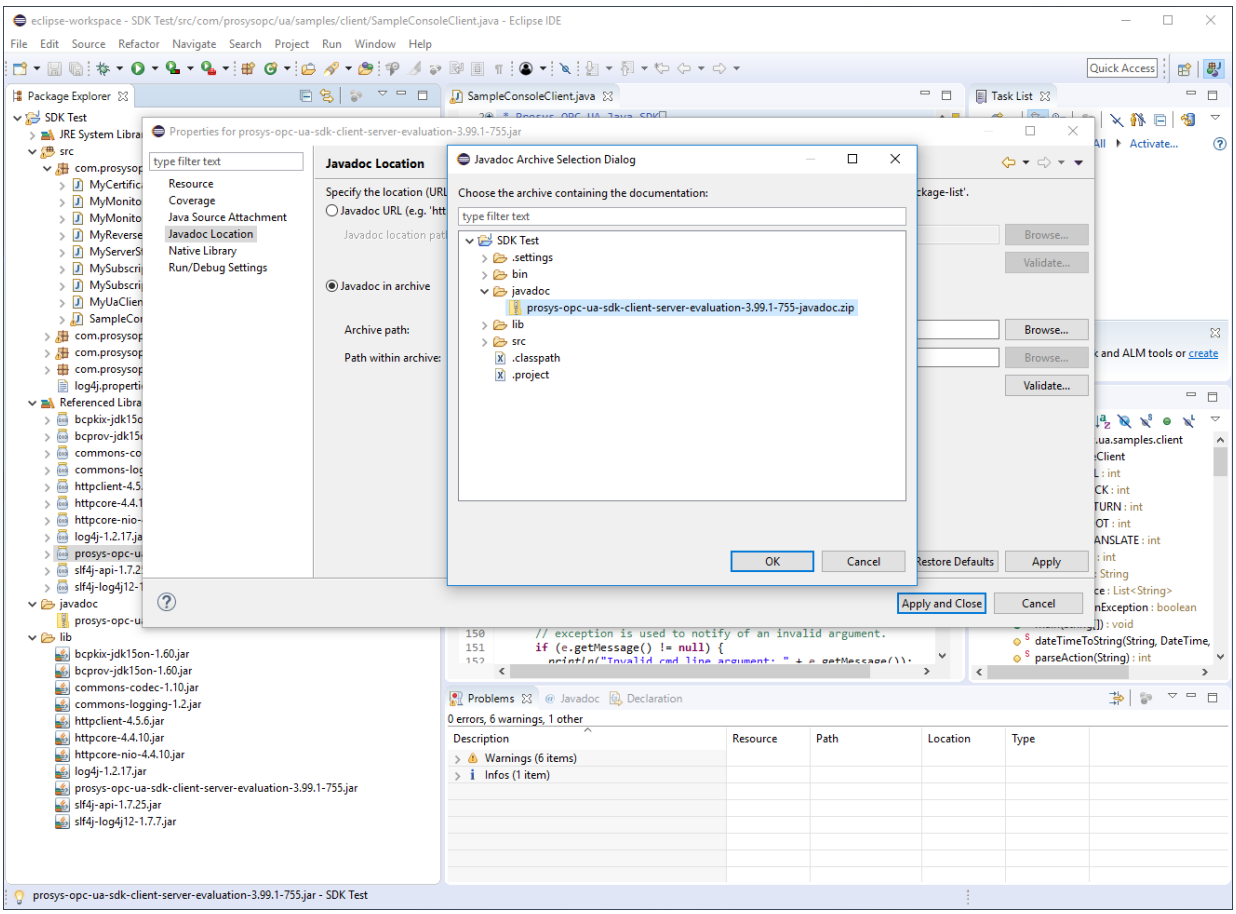

Figure 8. Setting the Javadoc for a library.

The project structure created here is suited for testing out the SDK. In general, more advanced development typically uses some build tool, which usually follows some conventions on how the project folders should be organized. In addition, the libraries are typically not within a project folder, but instead they are linked by the dependency management tool (which is typically also the build tool). Furthermore, they also typically link the Javadocs from the same place where they link the libraries. The reason for this is that typically the project folder is in a version control system (e.g. Git) and putting binaries to it is not generally seen as a good idea (size grows large when new versions of libraries are released). Instead the library and Javadoc dependencies are declared in a file for the dependency management tool.