Getting Started with the SAS® Life Science Analytics Framework Java API

Since you are reading this document, you've already expanded the client distribution file (lsaf-java-api-client-<release>.zip) into a directory. In that directory is now an "lsaf-java-api-client-<release>" subdirectory which will be referred to as *HOME*.

In your *HOME* directory you should see:

- README
- docs [a directory containing javadoc]
- lib [a directory of jars the API needs to run]
- log4j2.properties [a file to configure client logging]

This document assumes that the server-side portion of the API has already been installed on your application server. Speak to your system administrator if this has not been done.

The API requires that you have Java version 11 installed. At your command prompt you can type "java –version" to confirm this.

Verifying Your Client Installation

To run one of the sample programs or verify your client installation:

1. From a command prompt, change directory to *HOME/lib*.
2. Run the following command:

    java -jar sas.lsaf.api.client.jar -h

You should see a list of commands that this sample program provides. This indicates the client install has been done correctly.

To verify your deployment and your connection to the SAS LSAF server, run the command:

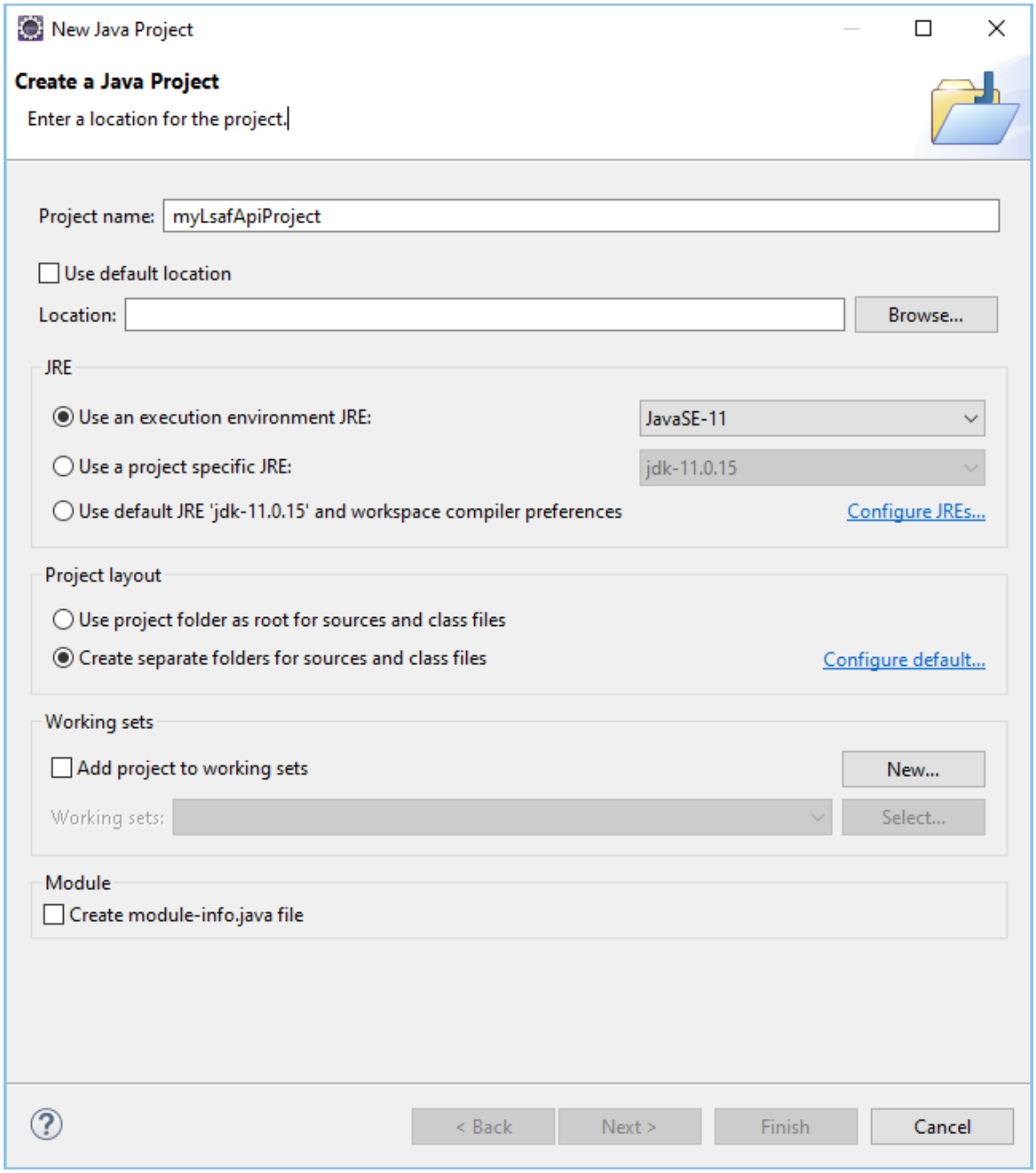
java -jar sas.lsaf.api.client.jar –u userid –p password –s https://<LSAF_HOST_NAME> –test

You should see a success message indicating that the connection was successful.

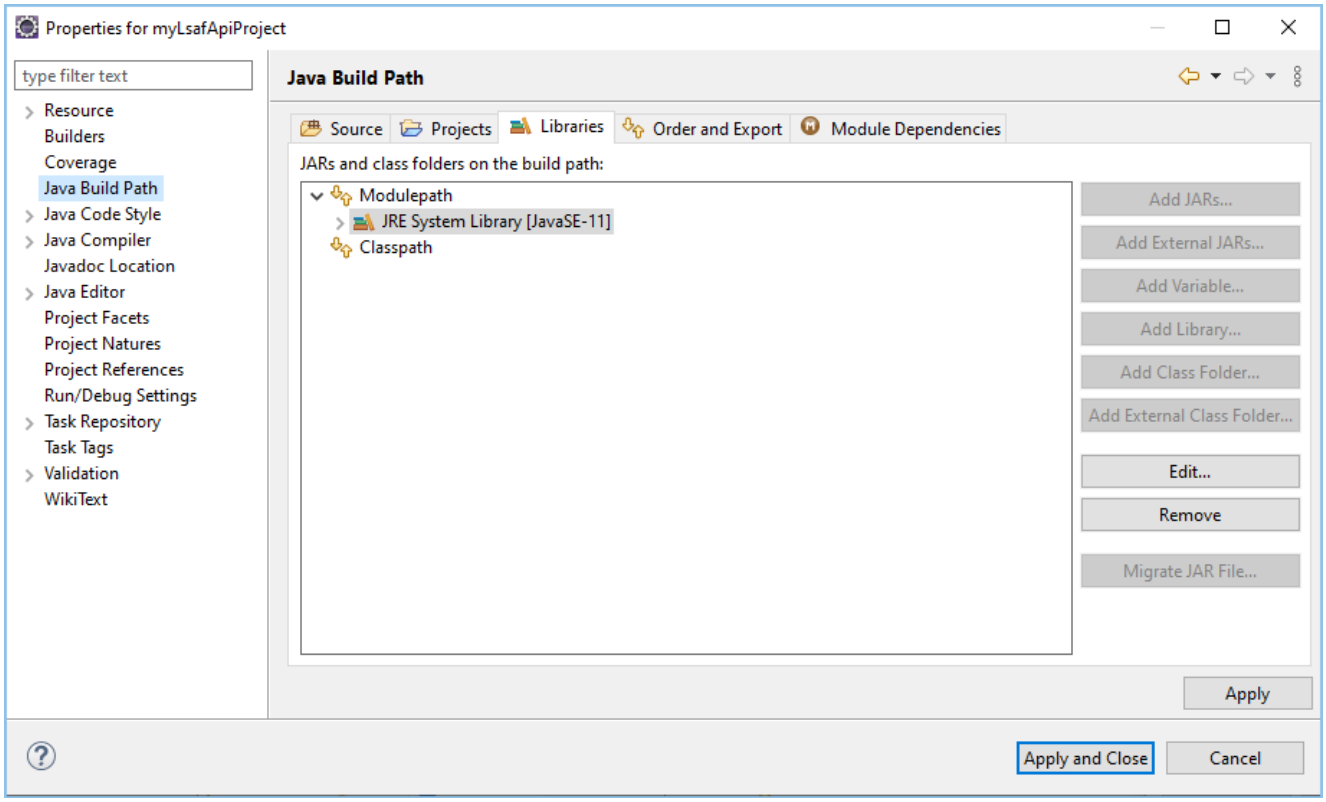
Setting up a Development Environment

The SAS Life Science Analytics API requires the use of a Java 11 execution environment. For this document we'll assume you are using Eclipse.

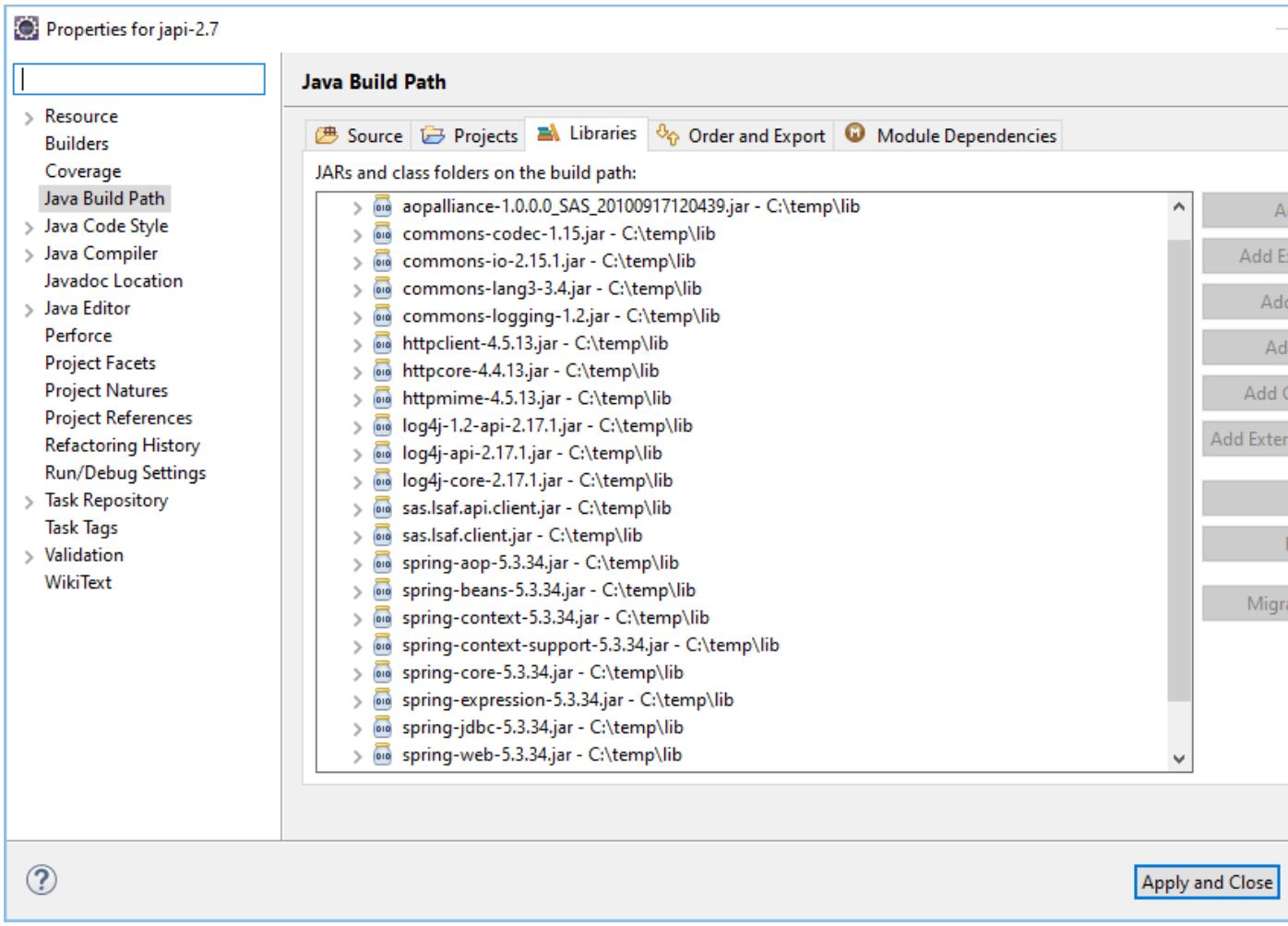
1) In Eclipse, create a new java project, specifying a Java 11 execution environment:



2) Once the project is made, open the project properties and make sure the Java 11 library is in the java build path setting:



3) Click the "Add External JARs…" button. Navigate to the lib directory of *HOME*, select all the jars there, and add them to your project (selection of jars may differ slightly depending on the release of the API in use).


Properties for japi-2.7
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Perforce
Project Facets
Project Natures
Project References
Refactoring History
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
Module Dependencies
JARs and class folders on the build path:
aopalliance-1.0.0.0_SAS_20100917120439.jar - C:\temp\lib
commons-codec-1.15.jar - C:\temp\lib
commons-io-2.15.1.jar - C:\temp\lib
commons-lang3-3.4.jar - C:\temp\lib
commons-logging-1.2.jar - C:\temp\lib
httpclient-4.5.13.jar - C:\temp\lib
httpcore-4.4.13.jar - C:\temp\lib
httpmime-4.5.13.jar - C:\temp\lib
log4j-1.2-api-2.17.1.jar - C:\temp\lib
log4j-api-2.17.1.jar - C:\temp\lib
log4j-core-2.17.1.jar - C:\temp\lib
sas.lsaf.api.client.jar - C:\temp\lib
sas.lsaf.client.jar - C:\temp\lib
spring-aop-5.3.34.jar - C:\temp\lib
spring-beans-5.3.34.jar - C:\temp\lib
spring-context-5.3.34.jar - C:\temp\lib
spring-context-support-5.3.34.jar - C:\temp\lib
spring-core-5.3.34.jar - C:\temp\lib
spring-expression-5.3.34.jar - C:\temp\lib
spring-jdbc-5.3.34.jar - C:\temp\lib
spring-web-5.3.34.jar - C:\temp\lib
Apply and Close

4) Finally, you can create your main program, starting with logging on to your instance of the application:

To simplest way to start is to logon via the LsafClient. LsafClient allows a static logon and static access to services like the RepositoryService. For example:

```
import com.sas.lsaf.LsafClient;
import com.sas.lsaf.content.repository.RepositoryService;
...
LsafClient.logon("https://yourLSAFMachine.domain.com", "myLSAFuserId",
"myLSAFpwd".getBytes());
RepositoryService repositoryService = LsafClient.getRepositoryService();
repositoryService.checkout("/YOURORG/Files/Folder/someProgram.sas");
...
```

Once logged in, a single user session is established and used when accessing the services. Note subsequent calls to `logon` will logoff the current session and establish a new session.

Alternatively, you can log on using `SessionFactory.logon(URL,String, byte[])` which will create and return a `Session`. From the `Session`, you can access services and manage the session.

```
import com.sas.lsaf.client.*;
import com.sas.lsaf.content.repository.RepositoryService;
...
Session session = SessionFactory.logon(new
URL("http://yourLSAFMachine.domain.com"),
                                        "myLSAFuserId",
"myLSAFpwd".getBytes());
ServiceManager serviceManager = session.getServiceManager());
RepositoryService repositoryService = serviceManager.getRepositoryService();
repositoryService.checkout("/YOURORG/Files/Folder/someProgram.sas");
...
```

Each `SessionFactory` logon creates a new `Session`.