SIF3 Training Exercises

Version 1.0

SIF3 Training Exercises - Java

**Author:** Joerg Huber, SIF Solution Architect

**Revision:** 1.1 (draft)

**Published:** Mar 2015

Copyright © 2015, Systemic Pty Ltd

Table of Contents

General Notes 4

1. Exercise 1: SIF3 Framework Installation & Verification 4

1.1. SIF3 Framework Installation 4

1.1.1. Setup Training Project for your Locale (Australia) 4

1.1.2. DB Config 4

1.2. Modify Configurations 5

1.2.1. JDBC Setup 5

1.2.2. Configuration Files 5

1.3. Classpath, Config/Property Files, Deployment 5

1.4. Verify Installation 6

2. Exercise 2: First Consumer – StudentPersonal 6

2.1. Write StudentPersonalConsumer 6

2.2. Write DemoConsumer 6

2.3. Run DemoConsumer 7

2.4. Advanced Exercise 7

3. Exercise 3: First Provider – SchoolInfo 7

3.1. Write SchoolInfoProvider 7

3.2. Deploy SchoolInfoProvider 8

3.3. Test your SchoolInfo Provider 8

3.3.1. Postman (Chrome Plugin) – Easy Test 8

3.3.2. Write SchoolInfo Consumer – More Work 8

3.4. Advanced Exercise 9

4. Exercise 4 (Optional): Consumer Multi-Object CRUD 9

5. Exercise 5: Environment Template Management 9

5.1. Manage Consumer Environment Templates 10

5.2. Manage DIRECT Provider Environment Templates 10

6. Exercise 7: ServicePath 11

6.1. Provider Implementation 11

6.2. Consumer Implementation 11

7. Exercise 6: Connect-A-Thon 12

7.1. General Preparation 12

7.2. Prepare your Provider to participate in the local network 12

7.3. Prepare you Consumer to connect to another Provider 12

7.4. Confirm that it is really working… 12

Appendix A: Classpath & Deployment 13

Ant Build Script and Tasks 13

Provider 13

Consumer 14

Appendix B: REST Client – Chrome Postman 15

General Notes

Throughout the exercises in this document the following notations are used when referring to file locations:

* **<installDir>** refers to the directory where you have installed the SIF3Training project.
* **<envConfigDir>** refers to the directory that is listed in the env.store.dir property in the environment.properties file.

‘Appendix A’ lists a number of options to consider for running either a consumer or provider. All exercises will use some deployment and will require various components to run, so it is strongly suggested to quickly browse through that Appendix to determine the steps or setups you require to run you project successfully.

‘Appendix B’ has some useful information about a very powerful REST Client that runs as an extension to the Chrome browser. Some exercises within that document point to this REST client for some tests or verifications.

# Exercise 1: SIF3 Framework Installation & Verification

**Task:**

* Install and configure components of SIF3 Framework or use pre-installed SIF3Training project.
* Prepare training material for Australian Data Model & Exercises.
* Configure components (i.e. DB, property files etc)
* Start Demo Provider
* Start Demo Connector

## SIF3 Framework Installation

If you are using the SIF3Training project all you need to do is creating appropriate tables and inserts to install the framework. Libs, Config Files, Web configuration etc. are already there, so there is no need to copy these files from the SIF3Framework to this project as highlighted in the presentation.

### Setup Training Project for your Locale (Australia)

The training project caters for various locales (Australia, US). The core difference between the locales is the data model. Also exercises are geared towards the locales data model. To configure the training material for Australia, please follow the steps below:

1. Open the file <instalDir>/ant.properties
2. Set the property called “locale” to AU (i.e. locale=AU).
3. Run the ant task called “99-prepare-training”.
4. Within your IDE Refresh the view of your project to ensure that all copies files are refreshed and picked up.

You should have your training material ready for the Australian Data Model and exercises.

### DB Config

If you do not have a local DB installed you can use the SQLite DB that comes with the SIF3Training project. Please note that you **SHOULD NOT use SQLite for your real implementation**. It is intended to be used only for the purpose of this training.

1. Start your DB Explorer of choice and connect to your database
   1. SQLite users: Your DB file is located at <installDir>/DB/Data/SIF3Infra.sqliteDB. No username and password required to open it with your DB explorer. A JDBC driver can be found in the <instalDir>/war/WEB-INF/lib called sqlite-jdbc-3.7.2.jar.
2. Create a DB/Schema called SIF3 (or any other name you like).
3. Run <installDir>/DB/DDL/SIF3InfrastructureERM\_DDL\_<db\_product>.sql script.
4. Run <installDir>/DB/DLL/Initial\_Inserts.sql

Now your DB should be configured for this training course.

## Modify Configurations

### JDBC Setup

* Ensure that you have an appropriate JDBC driver (library) in the <installDIR>/war/WEB-INF/lib directory. You can find some samples in the <installDIR>/lib/jdbc directory.
* Make hibernate aware of the database connection from previous section. Configure your JDBC configuration in <instalDir>/config/hibernate/sif3infra.hibernate.cfg.xml.

### Configuration Files

There are three config files that need to be checked. One generic file, one for the consumer and one for the provider.

**Generic File**: environment.properties

Located: <installDir>/config

Ensure that the property “**env.store.dir**” points to the directory where you intend to store all your environment template XMLs. If you use the SIF3Training project of the box then that value would be <installDir>/config/environments.

**Consumer File**: StudentConsumer.properties

Located: <installDir>/config/consumers

Ensure that the property “**env.baseURI**” has the correct hostname/IP Address and port number to your web- or application container (generally just **localhost:8080** for tomcat). Currently it will have “localhost:9080”. You need to change that to your web server’s port.

**Provider File**: StudentProvider.properties

Located: <installDir>/config/providers

Ensure that the property “**env.connector.url**” has the correct hostname/IP Address and port number to your web- or application container (generally just localhost:8080 for tomcat). ). Currently it will have “localhost:9080”. You need to change that to your web server’s port.

Also for running the provider for StudentPersonal objects you need to set the property “**provider.student.file.location**” to <installDir>/TestData/xml/input/StudentPersonals.xml. This file holds 608 students. We use these for a number of exercises later.

## Classpath, Config/Property Files, Deployment

Please refer to Appendix A for details on config/property files and deployment of your provider.

## Verify Installation

**Provider Verification**

Assuming all of the steps in the previous sections are complete, you can now deploy your provider to your web- or application container. Most likely your IDE has a plugin for your web-/application container, so you can run the provider, which is a webapp directly within your IDE. Alternatively you build the provider according to the instructions in Appendix A and deploy to your container. Observe the output during start-up. It should not show any errors if all is configured properly.

Type the following URL into your browser’s location bar:

[http://localhost:<port>/SIF3Training/sif3Demo/requests/StudentPersonals](http://localhost:%3cport%3e/SIF3Training/sif3Demo/requests/StudentPersonals)

The above should return the following error message: “**No or invalid Authorization Token provided**”

=> Provider appears to be fine, i.e. it responded with correct message!

**Consumer Verification**

1. Run the **TestConsumerConnector** class in the sif3test source (sif3.test.env package).
2. Should show you a lot of output in the log/screen relating to environment creation
3. No errors => All good.
4. Check the SIF3\_SESSION table in the database
5. Should hold 2 rows with values. Check the ENVIRONMENT\_XML column that holds a full and valid environment XML.
6. You may want to remove the rows again as this is a test only

=> delete from SIF3\_SESSION

# Exercise 2: First Consumer – StudentPersonal

**Task**: Write a basic Consumer for Student Personal (Consumer Class, Executable Class)

## Write StudentPersonalConsumer

**Source Directory**: sifdemo/src

**Class**: sif3demo.consumer.StudentPersonalConsumer

* Lookout for all “**// TODO Auto-generated method stub”** tags in the class and implement appropriate methods.
* For marshal and unmarshal methods use DataModelMarshalFactory and DataModelUnmarshalFactory class.
* For getSingleObjectClassInfo() and getMultiObjectClassInfo() methods use appropriate constants in the ModelObjectConstants class.
* No code for shutdown() method required at this point. You can leave it empty.

## Write DemoConsumer

**Source Directory**: sifdemo/src

**Class**: sif3demo.service.DemoConsumer

* Lookout for all “**//TODO: Exercise 2:…”** tags in the class and implement appropriate methods.
* Implement getStudent() and getStudents(). Each should only be made up of a single line of code! **THAT IS IT!** Uncomment printResponse() method for a nice output.

## Run DemoConsumer

First ensure your provider is deployed in the web- or application container. Ensure it is started correctly. Please refer to ‘Appendix A’ for further details on how to deploy the provider. Once it is deployed you should see the following output somewhere in the Provider output:

**> DEBUG provider.StudentPersonalProvider:79 - Loaded 608 students into memory.**

If you don’t see above line check your config/providers/StudentProvider.properties file. Ensure that the property “provider.student.file.location” point to the correct location (<installDir>/TestData/xml/input/StudentPersonals.xml). If it doesn’t then fix the path and restart your provider. Also ensure that your log4j configuration is set to DEBUG for the sif3demo package as well as the sif3 (framework) package.

Finally you can run your **DemoConsumer** class. Because it has a main() it can run immediately as a standalone executable within your IDE. You should see a lot of output on the command line. The important output is various XML data which should show you a single student <StudentPersonal> and a list of students <StudentPersonal**s**>.

## Advanced Exercise

If you have finished the above exercise successfully and have more time, why don’t you try to “Update” a student?

# Exercise 3: First Provider – SchoolInfo

**Task**:

* Write an Object Provider (CRUD) for SchoolInfo (or any other object of your choice).
* Deploy new Object Provider.
* Test using Postman or write a SchoolInfoConsumer.

## Write SchoolInfoProvider

**Source Directory**: sifdemo/src

**Class**: sif3demo.provider.SchoolInfoProvider

* Lookout for all “**// TODO Auto-generated method stub”** tags in the class and attempt to implement methods listed below.
* For marshal and unmarshal methods use DataModelMarshalFactory and DataModelUnmarshalFactory class.
* For getSingleObjectClassInfo() and getMultiObjectClassInfo() methods use appropriate constants in the ModelObjectConstants class.
* Implement the following CRUD operations:
  + createSingle()
  + retrieve()
* No code for shutdown() method required at this point. You can leave it empty.
* Note: You may want to look into the StudentPersonalProvider class for some ideas…

## Deploy SchoolInfoProvider

Before you can deploy you must ensure that the Object Provider is aware of the newly implemented provider.

* Add the SchoolInfoProvider class name to the appropriate property in the StudentProvider.properties file (provider.classes).
* **NOTE**: That you may need to copy the StudentProvider.properties file to your web- or application container’s location where you copied the file as part of Exercise 1.

Restart/Redeploy the Object Provider in your web- or application server. Ensure it is started correctly. If it is you should see the following output somewhere in the Provider output:

> DEBUG provider.StudentPersonalProvider:79 - Loaded 608 students into memory.

> DEBUG provider.SchoolInfoProvider:88 - Loaded 2 schools into memory.

Note that you can see Students and School being loaded now. The latest Object Provider is now dealing with these two SIF Objects!

If you don’t see above line check your config/providers/StudentProvider.properties file. Ensure that the property “provider.school.file.location” point to the correct location (<installDir>/TestData/xml/input/SchoolInfos.xml). If it doesn’t then fix the path and restart your provider. . Also ensure that your log4j configuration is set to DEBUG for the sif3demo package as well as the sif3 (framework) package.

## Test your SchoolInfo Provider

### Postman (Chrome Plugin) – Easy Test

Postman is a plugin for Chrome (<https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=en>). This is the perfect plugin to test your provider.

1. Check the sessionToken and password in the SIF3\_SESSION table.
2. Use the sessionToken and password and create the Authorization header in Postman (Basic Auth tab at the top).
3. Use the following Base URL to test your new provider (you may need to change the port number!): <http://localhost:9080/SIF3Training/sif3Demo/requests/SchoolInfos>
4. Use POST, GET etc from the Postman GUI and play with the values and see what your SchoolInfo provider returns.

Note to create (POST) a SchoolInfo you must use the URL [http://localhost:9080/SIF3Training /sif3Demo/requests/**SchoolInfos/SchoolInfo**](http://localhost:9080/SIF3Training%20/sif3Demo/requests/SchoolInfos/SchoolInfo) (an oddity that is required in SIF to cater for single and batch operations). You must also provide a payload which is a SchoolInfo XML.

Note to get a SchoolInfo you must use the URL [http://localhost:9080/SIF3Training /sif3Demo/requests/SchoolInfos/<GUID>](http://localhost:9080/SIF3Training%20/sif3Demo/requests/SchoolInfos/%3cGUID%3e) (you may need to change the port number!).

### Write SchoolInfo Consumer – More Work

You can write a SchoolInfo consumer as you did for the StudentPersonal consumer in Exercise 2. This is a bit of work but mostly a copy-paste exercise.

1. Create a new consumer class SchoolInfoConsumer (see also StudentPersonalConsumer in exercise 2).
2. Create new or modify existing DemoConsumer executable (see also DemoConsumer in exercise 2).
3. Add new consumer to StudentConsumer.properties.
4. Run your new DemoConsumer… and check results.

## Advanced Exercise

If you have finished the above exercise successfully and have more time, why don’t you try to implement remaining methods in the SchoolInfoProvider class?

# Exercise 4 (Optional): Consumer Multi-Object CRUD

If there is time or you are ahead of schedule with the other exercises you may try the following to further extend a consumer…

**Task**:

* Go back to DemoConsumer and complete methods for deleteStudents().
* Go back to DemoConsumer and complete methods for updateStudents().

**Note:**

The provider we use does only ‘fake’ the responses to multi-object CRUDs. This is done so that it also produces some errors in the response for illustration purpose.

**Exercise**: Delete a list of Students

* Find deleteStudents() method in DemoCosumer Class.
* Determine what parameters the consumer.deleteMany() method requires (i.e. consult javadoc of SIF3 Framework and look at AbstractConsumer).
* Start implementing/populating data structures for deleteMany() parameters.
* Run DemoConsumer and observe what is returned (i.e. check output on command line).

**Exercise**: Update a list of Students

* Find updateStudents() method in DemoCosumer Class.
* Populate StudentPersonalCollectionType object.
* Option 1: Programmatically using

ObjectFactory objFactory = **new** ObjectFactory();

objFactory.createStudentPersonalCollectionType();

* Option 2: Read students from a file. See loadStudents() method and set the constant MULTI\_STUDENT\_FILE\_NAME at the top of the class to the correct location.
* Determine what parameters the consumer.updateMany() method requires (i.e. consult javadoc of SIF3 Framework and look at AbstractConsumer).
* Start implementing/populating data structures for updateMany() parameters.
* Run DemoConsumer and observe what is returned (i.e. check output on command line).

# Exercise 5: Environment Template Management

The exercises in this section are all about how environment templates are managed in the SIF3 Framework. There is a difference between environments for Consumers and environments for the DIRECT environment provider. More details can be found in the Developer’s Guide in section “5.3 Environments”.

## Manage Consumer Environment Templates

**Task:**

* Investigate a Consumer Environment Template and the consumer.properties file.
* See how environment template XML and properties are used.
* Try to create new template or change existing template
* Cannot run it until Provider is configured as well (see exercise 5.2)

Step 1:

Open the environment template called demo.xml in the <**envConfigDir**>/consumer/template directory. Note that some elements are empty (not populated i.e. consumerName, applicationKey etc.).

Step 2:

Open the StudentConsumer.properties file in the <**installDir**>/config/consumers.

1. Can you find the properties that relate to the missing bits in the demo.xml template?
2. Can you see how the property file links to the environment template?

Step 3:

1. Attempt to create a new environment template XML.
2. Where and what would you need to change in the StudentConsumer.properties to make that new environment template active?
3. In exercise 5.2 we will make the DIRECT environment provider aware of the new consumer configuration (applicationKey, password).

## Manage DIRECT Provider Environment Templates

**Task:**

* Investigate a Provider Environment Template the Database Configuration.
* See how environment template XMLs are linked with applications (DB Configuration).
* Try to create new template, based on the demo.xml template.
* Try to change ACLs in your new template and see how they filter to the consumer

**Exercise**: Create a new environment template XML and use it.

1. Create new environment template XML file based on the demo.xml and store it in <**envConfigDir**>/provider/direct/template directory.
2. Add your new template to SIF3\_ENV\_TEMPLATE table.
3. Link your new environment template with a **new** APPLICATION\_KEY, PASSWORD . Configure this in the SIF3\_APP\_TEMPLATE table.
4. Now go to your consumer in exercise 5.1 and change appropriate values in the consumer’s properties file.
5. Restart your provider (stop/start tomcat/JBoss)
6. Run your DemoConsumer from exercise 2. It should now use the new environment. Verify by checking SIF3\_SESSION table. Look at the content of the ENVIRONMENT\_XML column.

**Exercise**: Change ACLs

1. Change the ACL in your new environment template from previous exercise.
2. Restart your provider (stop/start tomcat/JBoss)
3. Run your DemoConsumer from exercise 2. Verify by checking SIF3\_SESSION table. Look at the content of the ENVIRONMENT\_XML column. You should see the new ACLs.

# Exercise 7: ServicePath

The exercises in this section are all about how implement service path functionality using the SIF3 Framework. There is a difference between the implementation for Consumers and for the DIRECT provider. More details can be found in the Developer’s Guide in section “5.7 Service Paths”.

## Provider Implementation

We start with the implementation of the provider (DIRECT) for the service path. Once this is working we can implement the consumer and use it against the provider.

**Task:**

* Use the StudentPersonalProvider class to implement a service path for …/TeachingGroups/{}/StudentPersonals which means get all students of a class.
* Checkout the QueryProvider Interface, what methods it enforces and what parameters it supports.
* Work with the QueryCriteria class and process a Query Object.
* Write code to return a list of StudentPersonal Objects
* Use your REST web-service tool of choice (i.e. Postman) to invoke and test the service path functionality of the provider.

**Tips:**

* For simplicity reasons the StudentPersonalProvider has already a list of students for a class loaded. This list is called “teachingGroupStudents”. You can use that to create a list of students to be returned to the caller.
* You can peek into the retrieve() method of the StudentPersonalProvider to see how a list of students can be fetched and a correct student list is created.
* The demo.xml has already an entry for the required service path. Look into the demo.xml to find it and see how it is structured.

**Note:**

In a real implementation you would need to query your database for example with the correct where-clause derived from the QueriCritera object to get a list of students for a given class.

## Consumer Implementation

Exercise 6.1 should be completed before this exercise otherwise there is no provider that can be used to send a service path query to.

**Task:**

* Use the DemoConsumer class to implement a service path for …/TeachingGroups/{}/StudentPersonals which means get all students of a class.
* Checkout the consumer.retrieveByServicePath() method and its parameters.
* Work with the QueryCriteria class and create a Query Object for the given service path.

**Tips:**

* There is already a ‘skeleton’ for the service path functionality in the DemoConsumer class. Look in the getStudentsByServicePath() method and fill in the missing bits.
* To test it uncomment the appropriate line in the main() method of the DemoConsumer Class and run it.
* Observe the output. Also observe the output of the provider to confirm it is being invoked.

# Exercise 6: Connect-A-Thon

**Task**:

* Either connect your consumer to another participants provider or
* Prepare your provider, so that other participants can connect their consumers

## General Preparation

A local network is required for this exercise. Connect to it and find out what your machine’s IP address in that local network is. Make a note of that IP address.

Note: The name and password for the local network will be provided to you during the training course.

## Prepare your Provider to participate in the local network

* In the StudentProvider.properties file modify the following two properties, so that they use your IP address instead of ‘localhost’: env.connector.url & env.connector.url.secure
* Determine the consumers that want to connect. Set an applicationKey and password and associate an environment template with each in the SIF3\_APP\_TEMPLATE.
* **Remove all sessions from the SIF3\_SESSION table. (delete form SIF3\_SESSION)**
* Let the participants that connect to your provider know all of the above (your IP address, the port number, the applicationKey and password). They will need this in their configuration to be able to connect to your provider.
* Restart/Redeploy your provider. It should be ready to accept calls from the other consumers.
* **NOTE**: That you may need to copy the StudentProvider.properties file to your web- or application container’s location where you copied the file as part of Exercise 1.

## Prepare you Consumer to connect to another Provider

* Get the following information form the provider you intend to connect to: ApplicationKey, Password, IP Address and port number of provider.
* Set the appropriate property in the StudentConsumer.properties file with these values. The IP address is used in the env.baseURI. You need to replace ‘localhost’ with that IP address. Note the port may change as well!
* **Remove all sessions from the SIF3\_SESSION table. (delete form SIF3\_SESSION)**
* Start your DemoConsumer. It should now connect to the other participant’s provider.

## Confirm that it is really working…

* Check your SIF3\_SESSION table. You should only have one session in there (before you had two at least).
* Monitor the output in your command window. It should show you IP addresses you connect to etc. They should all have something else than ‘localhost’ in there.
* Communicate with your buddy if he/she can see your calls and if they are processed.

Appendix A: Classpath & Deployment

# Ant Build Script and Tasks

The SIF3Training Project has an ant build script with a number of handy ant tasks. You may or may not need some of these ant tasks during this training course. Please refer to the Provider and Consumer sections in this appendix for details where you may need an ant task to build and/or deploy your provider or consumer successfully.

# Provider

**Config Files and Classpath**

Deploying a provider means deploying a Web-Application (\*.war) to a web- or application container such as Tomcat, JBoss etc. For a successful deployment the provider web-application must have the following config/property files on the classpath:

* <installDir>/config/providers/StudentProvider.properties
* <installDir>/config/environment.properties
* All files in <installDir>/config/hibernate
* Optionally <installDir>/config/log4j.properties

There are various ways on how to get them onto the classpath of you deployment. These are listed below with additional deployment considerations.

**Option A: Using your IDE’s web container plugin**

**Eclipse & Tomcat Plugin**

If you use Eclipse and the Tomcat plugin then you can deploy your provider immediately in that built-in tomcat instance. No additional actions needed. The default configuration of the Training course is based on this.

**Another IDE & Tomcat Plugin**

Ensure that you have the files listed above added to your classpath of your project. Once this is done you should be able to deploy your provider immediately in that built-in tomcat instance. No additional actions needed.

**Eclipse & JBoss Plugin**

If you have a JBoss plugin in your Eclipse IDE then you are using JBoss’ JAXRS implementation (RestEasy). To make it all work properly you should run the **ant task called ‘prepare-for-jboss’ first**. This will ensure that there will be no conflicts with libraries. Refresh your project and deploy your provider.

**Another IDE & JBoss Plugin**

Ensure that you have the files listed above added to your classpath of your project. Now you follow the same steps as listed in the ‘**Eclipse & JBoss Plugin’** paragraph above. I.e. run appropriate ant task.

**Option B: Using an external web container**

If you wish to deploy your provider to an external web- or application container you need to build the provider web-application (SIF3Training.war) first and deploy it to your container. The instructions below give you the information to do this for a couple of web-containers (Tomcat, JBoss). If you use another web-container, you may be able to use one of the two options or you may need to change one or the other thing first.

**Config File & Classpath**

Whatever web-container you use you need to tell the container where your config/properties file are. There are two options to do so.

**Option 1:** Add files to classpath

With this option you have to configure your web-container to be able to load the config/property files from a given location. Please consult your web-container’s admin manual to determine how this is done. Once your web-container is configured accordingly you need to build the provider war file. Use the **ant task ‘70-build-for-Tomcat-no-config’ or ‘75-build-for-JBoss-no-config’** respectively and deploy the final war file. The built war file can be found in the directory <installDir>/build/dist and it should have the name ‘SIF3Training.war’.

**Option 2**: Package files into a jar

With this option you don’t need any further configuration in your web-container. All appropriate config/property files will be in a jar file that is added to the final war file. To achieve this simply use the **ant task ‘71-build-for-Tomcat-with-config’ or ‘76-build-for-JBoss-with-config’** respectively and deploy the final war file. The built war file can be found in the directory <installDir>/build/dist and it should have the name ‘SIF3Training.war’. Please note, if you use this option you must build the final war file and re-deploy it every time you change a config/properties file.

# Consumer

You should be able to run your consumer from within your IDE as a basic java application. They are simple executables that do not require a web- or application container. All you have to ensure is that the following config/properties file and the following jar files are on the classpath of your executable:

* <installDir>/config/consumerss/StudentConsumer.properties
* <installDir>/config/environment.properties
* All files in <installDir>/config/hibernate
* Optionally <installDir>/config/log4j.properties
* <installDir>/lib/jaxb/jaxb-impl.jar
* All jar files in <installDir>/war/WEB-INF/lib

If you use another IDE than Eclipse then you must ensure that you add the above list of files to your IDE’s project classpath. If you use Eclipse then all of it is pre-configured and no further action is required.

If you wish to run the DemoConsumer outside of your IDE you must first build the appropriate jar file. Use ant task ’03-jar-components’. This will build the appropriate jar file and will place it into the <installDir>/build/dist directory. Go to a command prompt and change to the script directory and run appropriate script (startDemoConsumer.bat or startDemoConsumer.sh).

**Note:**

The startDemoConsumer.sh script for linux has not been tested due to the lack of access to a linux environment. It may need one or the other tweak first.

Appendix B: REST Client – Chrome Postman

For some exercises it is suggested to use a REST Client to test your code. A good and extensive REST client is the Chrome extension called POSTMAN. You need Chrome as your browser to use it.

It can be downloaded from here: <https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=en>

Unfortunately you can only install it with a Google account/sign-in. If you do not have one there is the option to install it without a Google sign-in. For this you need to download the extension from Github directly and follow the instructions in the README.md. The Github location is <https://github.com/a85/POSTMan-Chrome-Extension>.