SIF3 Training Exercises

Version

**Author:** , SIF Solution Architect

**Revision:** 1.0 (draft)

**Published:** Oct 2014

Copyright © 2015,

Table of Contents

General Notes 3

1. Exercise 1: SIF3 Framework Installation & Verification 3

1.1. SIF3 Framework Installation 3

1.1.1. Setup Training Project for your Locale (North America) 3

1.1.2. DB Config 3

1.2. Modify Configurations 4

1.2.1. JDBC Setup 4

1.2.2. Configuration Files 4

1.3. Classpath, Config/Property Files, Deployment 4

1.4. Verify Installation 5

2. Exercise 2: First Consumer 5

2.1. Write StudentConsumer 5

2.2. Write DemoConsumer 5

2.3. Run DemoConsumer 6

2.4. Advanced Exercise 6

3. Exercise 3: CRUD Provider – School 6

3.1. Write SchoolProvider 6

3.2. Deploy SchoolProvider 7

3.3. Test your School Provider 7

3.3.1. Postman (Chrome Plugin) – Easy Test 7

3.3.2. Write School Consumer – More Work 7

3.4. Advanced Exercise 8

4. Exercise 4: Consumer Multi-Object CRUD 8

Appendix A: Classpath & Deployment 9

Appendix B: REST Client – Chrome Postman 11

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 North America 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 (North America)

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 North America, please follow the steps below:

1. Open the file <instalDir>/ant.properties
2. Set the property called “locale” to NA (i.e. locale=NA).
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 North America 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 not 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>/lib/jdbc 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 put 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).

**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).

Also for running the provider for Student objects you need to set the property “**provider.student.file.location**” to <installDir>/TestData/xml/input/k12Students.xml. This file holds … 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 section Appendix A and deploy to your container. Observe the output during startup. 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-US/sif3Demo/requests/k12Students](http://localhost:%3cport%3e/SIF3Training-US/sif3Demo/requests/k12Students)

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

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

## Write StudentConsumer

**Source Directory**: sifdemo/src

**Class**: sif3demo.consumer.StudentConsumer

* 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.StudentProvider:79 - Loaded 11 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/k12Students.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 <k12Student> and a list of students <k12Student**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: CRUD Provider – School

**Task**:

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

## Write SchoolProvider

**Source Directory**: sifdemo/src

**Class**: sif3demo.provider.SchoolProvider

* 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.

## Deploy SchoolProvider

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

* Add the SchoolProvider 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.StudentProvider:79 - Loaded … students into memory.

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

Note that you can see Students and Schools 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/k12Schools.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 School Provider

### Postman (Chrome Plugin) – Easy Test

Please refer to Appendix B for details on how to install Postman.

Postman 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) and click ‘Refresh headers’.
3. Use the following Base URL to test your new provider: <http://localhost:9080/SIF3Training-US/sif3Demo/requests/k12Schools>
4. Use POST, GET etc from the Postman GUI and play with the values and see what your School provider returns.

Note to create (POST) a School you must use the URL [http://localhost:9080/ SIF3Training-US /sif3Demo/requests/k12Schools/k12School](http://localhost:9080/%20SIF3Training-US%20/sif3Demo/requests/k12Schools/k12School) (remember the REST URL Presentation?). You must also provide a payload which is a School XML.

Note to get a School you must use the URL [http://localhost:9080/ SIF3Training-US /sif3Demo/requests/k12Schools/<UUID>](http://localhost:9080/%20SIF3Training-US%20/sif3Demo/requests/k12Schools/%3cUUID%3e) (remember the REST URL Presentation?).

### Write School Consumer – More Work

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

1. Create a new consumer class SchoolConsumer (see also StudentConsumer 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 SchoolProvider class?

# Exercise 4: Consumer Multi-Object CRUD

If you have time to spare why don’t you try the following optional exercise…

**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 K12StudentCollectionType object.
* Option 1: Programmatically using

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

objFactory.createStudentCollectionType();

* 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).

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 must 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-US.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-US.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-US.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>.