SIF 3.0 Framework (.NET)

Version

**Author:** , SIF Solution Architect

**Revision:** 1.5

**Published:** Dec 2017

Copyright © 2017,

Table of Contents

1. Introduction 3

2. Exercise 1: SIF 3 Framework Installation & Verification 3

2.1. Prerequisites 3

3. Exercise 2: Student Consumer 3

3.1. Prerequisites 3

3.2. Define the student data model 4

3.3. Implement the student Consumer 4

3.4. Implement a console application 4

3.5. Test the student Consumer 4

3.6. Advanced exercise 4

4. Exercise 3: School Provider 4

4.1. Prerequisites 5

4.2. Define the school data model 5

4.3. Implement the school service interface 5

4.4. Implement the school Provider 5

4.5. Create a new Environment for the school Provider 5

4.6. Start the school Provider 5

4.7. Test the school Provider 6

4.7.1. Postman (Chrome Plugin) 6

4.7.2. Implement a school Consumer (optional) 7

4.8. Advanced Exercise 7

5. Exercise 4: Service Paths 7

5.1. Pre-requisites and configuration 7

5.2. Provider implementation 8

5.3. Consumer implementation 8

6. Important notes 8

6.1. Issues with the Environment Provider 8

# Introduction

Before attempting the exercises in this document, it is assumed that the following documents have already been read:

* Developer Guides\SIF Framework Installation and Verification
* Developer Guides\Implementing a SIF Consumer
* Developer Guides\Implementing a SIF Provider

The exercises presented here run the demo projects included with the SIF 3 Framework, and copy heavily from the information presented in the above documents.

# Exercise 1: SIF 3 Framework Installation & Verification

**Task:**

* Download the SIF 3 Framework
* Load the SIF 3 Framework into Visual Studio
* Create an Environment
* Start the Environment Provider
* Start the demo student Provider
* Run the demo student Consumer

## Prerequisites

Work through the instructions from the “SIF Framework Installation and Verification” document.

# Exercise 2: Student Consumer

**Task:**

* Implement a student Consumer from scratch

## Prerequisites

Work through the instructions from the “Implementing a SIF Consumer” document.

## Define the student data model

For implementing a Consumer, a SIF data object needs to be specified. In this case, create a student data model based upon the SIF AU StudentPersonalType type. A requirement for the use of this Framework is that the model object used has to implement the IDataModel interface. For an example implementation, refer to the Consumer demo project.

Create a new class (e.g. StudentPersonal.cs) and implement the IDataModel interface (where the RefId property equates to the RefId of StudentPersonalType).

It is important that the model object used will serialise (XML) to meet the AU Data Model Specification. To assist with this requirement, the Sif.Specification.DataModel.Au.vx\_x\_x Project can be used as a reference for any data model objects used.

## Implement the student Consumer

To implement the Consumer, create a new class (e.g. StudentPersonalConsumer.cs) that extends the BasicConsumer class with the previously defined data model class (e.g. StudentPersonal.cs) as the generic type. Implement the constructors to simply call upon the “base” constructors of the BasicConsumer class.

## Implement a console application

Create a class with a Main() method that will be run the student Consumer. In the Main() method, instantiate the new Consumer class (e.g. StudentPersonalConsumer.cs) by calling the constructor that takes multiple string parameters, and pass it a parameter of “Sif3DemoConsumer” (to match the Environment settings used by the existing demo Provider project).

Call the Register(), Query() and then Unregister() methods, writing the student details retrieved to the console.

**IMPORTANT:** Ensure that the Unregister() method is always called from within a “finally” block that encompasses all Consumer (service) calls. This will prevent the Consumer from having an invalid state should the Consumer fall over at any point. For more information, refer to section “6.1 Invalid session state”.

## Test the student Consumer

To test the newly created student Consumer, repeat Exercise 1 replacing the demo Consumer with this Consumer.

## Advanced exercise

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

# Exercise 3: School Provider

**Task:**

* Implement a school Provider (or any other object of your choice)
* Test using Postman, or implement a school Consumer

## Prerequisites

Work through the instructions from the “Implementing a SIF Provider” document.

## Define the school data model

For implementing a Provider, a SIF data object needs to be specified. In this case, create a school data model based upon the SIF AU SchoolInfoType type. A requirement for the use of this Framework is that the model object used has to implement the IDataModel interface. For an example implementation (of a student), refer to the Provider demo project.

Create a new class (e.g. SchoolInfo.cs) and implement the IDataModel interface (where the RefId property equates to the RefId of StudentPersonalType).

It is important that the model object used will serialise (XML) to meet the AU Data Model Specification. To assist with this requirement, the Sif.Specification.DataModel.Au.vx\_x\_x Project can be used as a reference for any data model objects used.

## Implement the school service interface

Create a class (e.g. SchoolInfoService) that implements the IBasicProviderService interface with the previously defined data model class (e.g. SchoolInfo.cs) as the generic type. For these exercises, it would be sufficient to only implement the Query() method.

## Implement the school Provider

To implement the Provider, create a new class (e.g. SchoolInfosProvider.cs) that extends the BasicProvider class with the previously defined data model class (e.g. SchoolInfo.cs) as the generic type. In the default constructor, call upon the “base” constructor of the BasicProvider class that accepts a service interface (in this case, the newly created SchoolInfoService).

As the Web API specification relies heavily on coding convention, a very important point to take into account when implementing a Provider:

1. The prefix to “Provider” defines both the SIF data model used and the Web Service URL, and MUST therefore be named SchoolInfosProvider (not SchoolInfoProvider).

## Create a new Environment for the school Provider

Open the Project properties of this school Provider and copy the Project Url from the Web section.

Edit the *Data files\AU\Sif3DemoConsumer\EnvironmentResponse.xml* file of the Sif.Framework.Demo.Setup Project and replace the *requestsConnector* URL with the URL of the new school Provider (keeping /api). Re-run the *Scripts\BAT\Demo execution\DemoAuSetup.bat* script.

## Start the school Provider

To test the newly created school Provider, repeat Exercise 1 replacing the demo Provider with this Provider (ignoring the running of a Consumer).

## Test the school Provider

### Postman (Chrome Plugin)

To test the school Provider without an actual Consumer, a REST Client is required. For these exercises, the Postman application will be used (<https://www.getpostman.com>). The following steps need to be performed:

* Create an Environment
* Retrieve all Schools

#### Create an Environment

Use Basic Auth with a Username of “Sif3DemoConsumer” and a Password of “SecretDem0”. These authentication settings have been pre-defined in the demo database using the Sif.Framework.Demo.Setup project (refer to the *Data files\AU\Sif3DemoConsumer\EnvironmentRequest.xml* file).

Use a URL of <http://localhost:62921/api/environments/environment>.

Use the POST HTTP operation.

Provide the following (raw) Body with the HTTP request.

<?xml version="1.0"?>

<environment xmlns="http://www.sifassociation.org/infrastructure/3.2.1">

<solutionId>Sif3Framework</solutionId>

<authenticationMethod>Basic</authenticationMethod>

<instanceId></instanceId>

<userToken></userToken>

<consumerName>TemplateDemoConsumer</consumerName>

<applicationInfo>

<applicationKey>Sif3DemoConsumer</applicationKey>

<supportedInfrastructureVersion>3.2.1</supportedInfrastructureVersion>

<dataModelNamespace><http://www.sifassociation.org/datamodel/au/3.4>

</dataModelNamespace>

<transport>REST</transport>

</applicationInfo>

</environment>

Specify the following headers:

* Authorization - Basic U2lmM0RlbW9Db25zdW1lcjpTZWNyZXREZW0w
* Content-Type – application/xml
* Accept – application/xml

The Authorization header specified above is generated from Basic Auth with the provided Username and Password from above.

Once ready, send the POST request and save a copy of the response payload (which should match the *Data files\AU\Sif3DemoConsumer\EnvironmentResponse.xml* file).

#### Retrieve all schools

From the response payload of the Environment POST operation (above), store the <sessionToken> value and the requestsConnector URL.

Using Basic Auth, set the Username to this <sessionToken> value and a Password of “SecretDem0”.

For the URL, use that returned as the requestsConnector URL, e.g. [http://localhost:<port>/api/SchoolInfos](http://localhost:%3cport%3e/api/SchoolInfos).

Use the GET HTTP operation.

Specify the following headers:

* Authorization – as generated from Basic Auth with the provided Username and Password from above
* Content-Type – application/xml
* Accept – application/xml

Once ready, send the GET request. The response payload should contain a list of schools.

### Implement a school Consumer (optional)

Implement a school Consumer as per the student Consumer created in Exercise 2 and run against the newly created school Provider.

## Advanced Exercise

If you have finished the above exercise successfully and have more time, implement remaining methods in the school Provider class.

# Exercise 4: Service Paths

**Task:**

* Implement a Service Path endpoint in the student Provider
* Use the student Consumer to query the Service Path

## Pre-requisites and configuration

Use the student Consumer created in exercise 2.

Repeat the steps in exercise 3 to implement a student Provider. In addition, follow the instructions outlined in the “Enabling Service Paths.docx” document to configure the Provider project appropriately. For a short cut, simply use the student Provider included with the demo project.

## Provider implementation

To create a Service Path endpoint, implement the Retrieve(IEnumerable<EqualCondition>, uint?, uint?, string, string) method of the IBasicProviderService interface for the student service that is passed to the student Provider.

When evaluating the EqualCondition, note that the Left property refers to the SIF data model and the Right property refers to a RefId. To implement a Service Path of the form …/SchoolInfos/{}/StudentPersonals, the Left property of the EqualCondition would be “SchoolInfos” (not SchoolInfo), and the Right property would be the RefId of the queried school. Based on the Conditions passed, return a list of matching students (in this case, all students for the specified school).

To simplify your implementation, simply output the Conditions to the console to ensure that your Service Path endpoint is being hit with the correct values.

## Consumer implementation

The previous exercise should be completed before this to ensure that there is an appropriate Service Path endpoint to query against.

To query the Service Path endpoint of the student Provider, call the QueryByServicePath(IList<EqualCondition>) method of the student Consumer. To query a Service Path of the form …/SchoolInfos/{}/StudentPersonals, create an instance of EqualCondition whereby the value of the Left property is “SchoolInfos” (not SchoolInfo) and the value of the Right property is the RefId of the queried school. Based on the Conditions passed, the call should return all students for the specified school.

# Important notes

## Invalid session state

It is possible for the session state between a Consumer and the EnvironmentProvider to become invalid. The following scenarios are possible:

1. If the sessionToken does not exist in the Consumer’s SifFramework.config file (in the bin directory) but exists in the EnvironmentProvider database, a HTTP status of 409 Conflict may be returned.
2. If the sessionToken exists in the Consumer’s SifFramework.config file (in the bin directory) but does not exist in the EnvironmentProvider database, a HTTP status of 401 Unauthorized may be returned.
3. If the sessionToken in the Consumer’s SifFramework.config file (in the bin directory) does not match the sessiontToken in the EnvironmentProvider database, a HTTP status of 401 Unauthorized may be returned.

The first scenario may occur under the following circumstances:

1. The consumer.environment.deleteOnUnregister setting is “false” and the SifFramework.config file is reset (e.g. by rebuilding the project).

The second scenario may occur under the following circumstances:

1. The consumer.environment.deleteOnUnregister setting is “true” and the Consumer completes (or crashes) without calling Unregister(), but the sessionToken is removed from the EnvironmentProvider database (e.g. by running the DemoAuSetup.bat script).
2. The consumer.environment.deleteOnUnregister setting is “false”, but the sessionToken is removed from the EnvironmentProvider database (e.g. by running the DemoAuSetup.bat script).

The third scenario may occur under the following circumstances:

1. Manually changing the applicationKey setting in the Consumer’s SifFramework.config file.

As a workaround, should any of these scenarios arise during these exercises, it is highly recommended that the demo database be re-created (by executing the *Scripts\BAT\Demo execution\DemoAuSetup.bat* script) and the projects be re-compiled (to reset the SifFramework.config files).