SIF3 Framework (Java)

Version

**Author:** Joerg Huber,

**Revision:** 0.1 (draft)

**Published:** Nov 2013

Copyright © 2013,

Table of Contents

1. Introduction 3

1.1. History 3

1.2. Licensing 3

1.2.1. SIF3 Framework 3

1.3. Why SIF3 Framework? 3

2. Assumption & Constraints 3

2.1. Functionality 4

3. Installation 4

3.1. Building Components of the Framework 5

3.1.1. Consumer Build 5

3.1.2. Provider Build 5

4. Concepts & Terminology 6

5. Framework Classes/Packages and Usage 6

5.1. General Process of Developing SIF3 Services 6

5.2. Proposed Package Structure 6

5.3. Data Model 7

5.3.1. Data Model POJOs 7

5.3.2. Marshal & Unmarshal Factories 7

5.4. Building a Consumer 8

5.4.1. Constraints 8

5.4.2. Consumer Components 8

5.5. Building a Provider 8

5.5.1. Constraints 8

5.5.2. Provider Components 8

5.6. Environments 9

5.6.1. Constraints 9

5.6.2. Configuration 9

# Introduction

## History

The SIF3 Framework is a basic Java Framework intended to help developing SIF3 Services/Adapters in an efficient manner.

[Systemic Pty Ltd](http://www.systemic.com.au) has implemented the first version (alpha) in September-October 2013. This work has been sponsored by [NSIP](http://www.nsip.edu.au/). As of November 2013 the framework is limited to functionality for **DIRECT** environments with **immediate** responses (see SIF3 Specification for details of these terms).

## Licensing

### SIF3 Framework

The SIF3 Framework is an open-source framework and therefore is free of charges and no licence purchase is required. It is licensed under the [Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0).

## Why SIF3 Framework?

The **SIF3 Framework** abstracts low level infrastructure functionality of SIF3. It provides an easy to use API to efficiently implement service consumers and providers. It adds further functionality behind the scene such as interfaces to deal with large data sets, multi-threading etc. This functionality is fully transparent to a developer. This allows the service developer to wire up the various components of a service (consumer and/or provider) in an efficient manner by only writing the minimal amount of code to have the skeleton of an service ready for deployment. Many components and their behaviour are controlled by a configuration file rather than writing a large amount of code. The developer can then concentrate on the business logic or data access layer to retrieve/store data from/to their system rather than spending time writing agent infrastructure code.

# Assumption & Constraints

It is assumed that the developer has some familiarity with the concepts of REST as well as the SIF3 Specification.

The framework is a work-in-progress. There are no guarantees that things work without any errors and I’m sure there are some bugs in there. It is open-source and once downloaded from the SIF-AU Confluence Group site (**TBD**) you get the source code as well. You are welcome to modify it as you see fit for your project if required.

It needs to be remembered that this framework is not guaranteed to be maintained any further, maybe it is, and maybe it is not. At present there is some commitment from NSIP to ensure that the framework will be maintained for at least another two years. If you want to use it you do so at your own will and bugs can be added to the GitHub repository but it is not guaranteed that they will be fixed. The framework and code is there to get you started but it is not required to be used for your SIF3 service/adapter development. You can always use your own libraries or toolkits to implement SIF3 services.

The framework has been developed based on the concepts and ideas of the SIFCommon Framework that has successfully been used in SIF 2.x implementations. You may find things that don’t fit your requirements. The framework allows overriding of methods in many places to make them behave, as you require without abandon the framework altogether (that’s what OO and inheritance is good at).

Finally it is also assumed that you are familiar with Java, have a JDK installed and ideally that you have Ant in case you want to modify and re-build the framework libraries. The JDK version should be 1.6 or above. The framework has not been compiled nor can it be compiled with a JDK version older than 1.6. It has also NOT been tested with Java 7.

## Functionality

As of November 2013 (alpha version of framework) the SIF3 Framework is limited to the following functionality:

* Direct Environment only
* Immediate Responses only (no delayed responses)
* REST (SOAP missing in SIF 3.0 Specification)
* XML (that is what SIF 3.0 specifies, but you can add JSON support with the marshal/unmarshal classes, discussed later in this document. If you add JSON support you do so at **your own risk** until SIF 3.0 specified the details of an XML<->JSON mapping).
* No xQueries
* No Events & Queues

More of the missing functionality might be added in subsequent releases.

# Installation

Before you can install the SIF3 Framework you need to download it from **TBD**. The downloaded zip file needs to be expanded/extracted to a directory. The expanded directory contains the following sub-directories (list not complete):

**Source Directories (each will produce a separate jar file)**

* **sif3Common/src**: Common classes that are transport layer (REST, SOAP) independent. Most Interfaces and common types used throughout the framework reside in this directory.
* **sif3InfraCommon/src**: Common classes for infrastructure implementation. Classes in this package should not be used or exposed to the higher levels of the framework. They are only used internally.
* **sifInfraREST/src**: Classes in this package deal specifically with the REST implementation of SIF3. Most of it should not be exposed to higher levels of the framework. There is one exception which is the **AbstractConsumer** that a developer must extend for the development of Consumer style service. This will be fixed for future releases of the framework.
* **demo/src**: Demo code to showcase how a consumer (StudentPersonal) and provider (StudentPersonal) is implemented using the SIF3 Framework.
* **test/src**: Various test classes to test things of the framework. Mainly used for development.
* **sif3Datamodel/src**: This package does NOT from part of the framework. The SIF3 framework is independent from the data model. This package is only there so that the demo package can showcase how different data models are supported with the same infrastructure.

**Configuration Directory**

The “**config**” directory is core to the framework. It holds a number of configuration files that are required by the framework to function. For details about the configuration files and their content refer to section **TBD**.

**Documentation Directory**

The “**documentation**” directory has the javadoc for the framework as well as this developer’s guide.

**Documentation Directory**

The “**Generator**” directory has various XSD files (**data** directory), scripts and libraries to build the POJOs for the infrastructure in the **sifInfraCommon/src** package. It also has an example of the SIF AU 1.3 XSD to generate the SIF AU 1.3 POJOs that are used in the demo code.

**Java 3rd Party Libraries Directory**

The “**lib**” directory has all the required java 3rd party libraries for the framework. If you develop a consumer you need to have these libraries in your classpath.

**Web-Application Directory**

Providers are “servers” and therefore must be deployed in a web- or application container. The “**war**” holds all the required structure, descriptors and libraries for a provider to be deployed in a web- or application container. If you write a provider you must copy this directory content into your own application.

**Build Directory**

The “**build**” directory is the directory where classes and the final jar are located. If you run the ant script to build the jar files then they will be located in this directory under the “**dist**” sub-directory. See the next sections for build instructions.

## Building Components of the Framework

Once you have modified the source code you need to build the various jar files and include the latest version in your own project. There is an ant script provided to build the jar files. The build.xml is located at the root level of the zip. For it to work you must ant (<http://ant.apache.org/>) installed.

### Consumer Build

If you want to build a consumer you need to build the jar files of the framework first. Use the ant task **03-jar-components** (the default task) to build the framework libraries. The name of the jar files that will be built are:

* sif3.0Common-<version>.jar
* sif3.0Infra-common-<version>.jar
* sif3.0Infra-rest-<version>.jar
* sif3.0Common-<version>.jar
* sif3Demo-<version>.jar
* sifDataModel\_au1.3.jar

You need to drop the above jar files into your project if you write a consumer for the AU data model.

### Provider Build

If you build a provider you need to use the ant task **80-build-war** to build the “war” file. This will build a war file called **sif3InfraREST.war**. This will contain ALL required jar files. You simply drop the war file into your web- or application container.

# Concepts & Terminology

If you are new the SIF3 Service development then you will need to get used to some terms common in the SIF3 realm. Those terms are Adapter, Environment, Service Provider, Service Consumer, Direct, Broker etc. In short those terms relate to each other and a detailed overview can be found on the SIF Association Website in the [Read This First](https://www.sifassociation.org/Specification/Documents/ReadThisFirst.pdf) guide.

# Framework Classes/Packages and Usage

This section describes how the SIF3 Framework classes are intended to be used to write SIF3 Consumers and Providers. It is also recommended to use the Javadoc that is provided as part of the SIF3 Framework for additional information. The Javadoc is basic but should be sufficient to get you going once you have read through this developer’s guide.

## General Process of Developing SIF3 Services

When developing a SIF3 Service the following questions need to be answered first:

* What SIF Environment do I want/need to connect to?
* What Data Model do I need (AU, US, UK) and which version?
* Which SIF Objects need to be exchanged (students, enrolments etc.)?
* Which of those SIF Objects need to be provided (Providers) or consumed (Consumers)? There will be one publisher and/or subscriber for each SIF Object.
* For each provider you need to determine if its objects are published as an event or are published/modified on request only (different interface methods need to be implemented)
* For each consumer you need to determine if the SIF Objects are received by listening to events or by requesting them (different interface methods need to be implemented)

Once the above questions have been answered you should be ready to develop the skeleton for the services for consumers and/or providers with the classes of the SIF3 Framework. With the skeleton I mean the wiring of those components. Once the wiring is done the actual work of querying the database and/or updating the database remains and is not covered as part of the SIF3 Framework.

The following sections describe how to develop services using the SIF3 Framework. The descriptions are all based on the Demo classes that are part of this framework.

## Proposed Package Structure

To get the best benefit from the SIF3 Framework it is recommended to use the following package structure for your consumer and provider service development:

* project.**provider** (package where the code of your providers is)
* project.**consumer** (package where the code of your consumers is)
* project.**datamodel** (package where the code for your data model, marshallers and unmarshalles is)

The above is the absolute minimum. I would further suggest having the following packages:

* project.**business** (package where business logic code is)
* project.**model** (package where model code is, ie. models reflecting your internal object structure)
* project.**dao** (package where dao code is to access your DB)
* project.**mapping** (package where code map between your model objects and SIF Objects)

**Note:**

* The above is just a suggestion but it will help you to keep track of your code and project.
* Within this framework the “**datamodel**” with its marshallers and unmarshallers can be found in **sif3Datamodel/src**. For the purpose of this user’s guide we use the SIF AU 1.3 data model. The POJOs and the marshallers/unmarshallers have been generated/coded using JAXB and the SIF AU 1.3 XSD (see **Generator** directory for example of the XSD).

## Data Model

Before you can develop any consumers or providers you need to choose a data model you work with. For the purpose of this developer’s guide we use the AU 1.3 data model. Whatever data model you use you need to do the following steps to make your consumers and providers aware of the data model they need to deal with:

* Have a library that encapsulates your data model (i.e. POJOs)
* Implement a Marshal and Unmarshal Factory that converts your POJOs into valid SIF XML/JSON according to your locale’s SIF Specification

### Data Model POJOs

First you must have something that deals with your data model. Generally you would have a set of POJOs that encapsulate your locale’s data model. In the past, SIF Classic, the OpenADK was such a library. With SIF3 the OpenADK cannot be used as it is. It is not necessary either because it is hoped that the data model libraries can be generated based on the SIF Data Model specification which is available as an XSD. There are many libraries out there in any number of programming languages that generate POJO style classes based on an XSD. In fact this framework has such an example in the sif3Datamodel/src package for the SIF AU 1.3 data model. The classes in that package have been generated using JAXB and the SIF AU 1.3 XSD. You can find the ant task to generate the POJOs in the Generator/script directory of this framework. This is just one way to get a data model library. Other options are:

* Write the POJOs manually (might be a lot of work but they can be customised to what you need and you may only implement the POJOs you need for your services)
* Get hold of a 3rd party library that has implemented your locale’s POJOs
* Generate them with your tool of choice.

Whatever your approach to get the Data Model Library the next step you need is a marshal and unmarshal implementation for these POJOs, so that you can transport them over the wire. The next section illustrates what needs to be done to achieve that.

### Marshal & Unmarshal Factories

The SIF3 Framework makes full use of what is called Marshal and Unmarshal Factories. They form the link between the infrastructure (this framework) and the data model used to be transported over the wire. These factories ensure that the infrastructure is independent from the data model. The infrastructure of the framework only deals with interfaces but not with implementations. Before a SIF Java Object (POJOs from previous section) can be sent to the “other side” it must be marshalled to XML/JSON and on the receiver’s side unmarshalled from XML/JSON to a SIF Java Object. The Marshal and Unmarshal Factories are intended to do exactly that. Each consumer and provider **must** be initialised with a marshal and unmarshal factory so that it can do the required serialisation on the wire.

You can find an example for each factory in the **sif3Datamodel/src** package for the SIF AU 1.3 data model (**sif.dd.au30.conversion.DataModelMarshalFactory**). When you write a marshaller and/or unmarshaller you must implement the following interfaces:

Marshaller: **sif3.common.conversion.MarshalFactory**

Unmarshaller: **sif3.common.conversion.UnmarshalFactory**

## Building a Consumer

First of all it must be considered that a consumer can be deployed as a standalone executable or be part of an application. Depending on your setup it does not require to be deployed in a web- or application container like a provider does. The consumer for is just a “client” in the traditional sense.

### Constraints

In the current version (Nov 2013) the framework doesn’t provide classes to deploy the consumer as a standalone executable. This may be added in later versions of the framework.

### Consumer Components

#### Classes

**TBC**

#### Configuration Files & Setup

**TBC**

#### Deployment

**TBC**

## Building a Provider

A provider is a “server” in the traditional sense and therefore must be deployed in a web- or application container. The SIF3 Framework has appropriate descriptors ready (web.xml) so that such a deployment should be straight forward. You may need to customise it to map to your environment.

### Constraints

As of November 2013 (alpha version) the framework has only been tested with Tomcat. This doesn’t mean it won’t work with other web- or application containers, but no guarantee can be given. Care has been taken not to introduce tomcat specific scripts, code etc. so that it should work with other containers.

### Provider Components

#### Classes

**TBC**

#### Configuration Files & Setup

**TBC**

#### Deployment

**TBC**

## Environments

### Constraints

**TBC**

### Configuration

**TBC**