SIF3 Framework (Java)

Version

**Author:** ,

**Revision:** 0.2 (draft)

**Published:** Mar 2014

Copyright © 2014,

Table of Contents

[1. Introduction 4](#_Toc387755510)

[1.1. History 4](#_Toc387755511)

[1.2. Licensing 4](#_Toc387755512)

[1.2.1. SIF3 Framework 4](#_Toc387755513)

[1.3. Why SIF3 Framework? 4](#_Toc387755514)

[2. Assumption & Constraints 4](#_Toc387755515)

[2.1. Functionality 5](#_Toc387755516)

[3. Installation 5](#_Toc387755517)

[3.1. Building Components of the Framework 6](#_Toc387755518)

[3.1.1. Consumer Build 6](#_Toc387755519)

[3.1.2. Demo Provider Build 7](#_Toc387755520)

[4. Concepts & Terminology 7](#_Toc387755521)

[5. Framework Classes/Packages and Usage 7](#_Toc387755522)

[5.1. General Process of Developing SIF3 Services 7](#_Toc387755523)

[5.2. Proposed Package Structure 8](#_Toc387755524)

[5.3. Environments 8](#_Toc387755525)

[5.3.1. Environment Store Setup 9](#_Toc387755526)

[5.3.1.1. environment.properties File 9](#_Toc387755527)

[5.3.1.2. Environment Template Directory (File System) 9](#_Toc387755528)

[5.3.1.3. Environment and Session Store (Database) 12](#_Toc387755529)

[5.4. Data Model 12](#_Toc387755530)

[5.4.1. Data Model POJOs 12](#_Toc387755531)

[5.4.2. Marshal & Unmarshal Factories 13](#_Toc387755532)

[5.5. Building a Consumer 13](#_Toc387755533)

[5.5.1. Constraint 13](#_Toc387755534)

[5.5.2. Components of a Consumer 13](#_Toc387755535)

[5.5.2.1. Configure your Consumer Environment 14](#_Toc387755536)

[5.5.2.2. FIRST CALL: ConsumerLoader Class 17](#_Toc387755537)

[5.5.2.3. The AbstractConsumer & AbstractEventConsumer Classes to wire up the Framework 18](#_Toc387755538)

[5.5.2.4. Actual Implementation 19](#_Toc387755539)

[5.5.3. Consumer Events Processing & Threads 20](#_Toc387755540)

[5.5.3.1. Queue Strategies 20](#_Toc387755541)

[5.5.3.2. Message Readers 21](#_Toc387755542)

[5.5.3.3. Event Processing Worker 21](#_Toc387755543)

[5.5.3.4. Thread Examples 21](#_Toc387755544)

[5.6. Building a Provider 22](#_Toc387755545)

[5.6.1. Constraint 23](#_Toc387755546)

[5.6.2. Components of a Provider 23](#_Toc387755547)

[5.6.2.1. Configure your Provider Environment 23](#_Toc387755548)

[5.6.2.2. The BaseProvider & BaseEventProvider Class 26](#_Toc387755549)

[5.6.2.3. SIFEventItarator Class 27](#_Toc387755550)

[5.7. Security – HTTPS Configuration 27](#_Toc387755551)

[5.7.1. Provider 27](#_Toc387755552)

[5.7.2. Consumer or Provider connection to a Broker 28](#_Toc387755553)

[6. Deployment 29](#_Toc387755554)

[6.1. Servlet Context 29](#_Toc387755555)

[6.2. Jersey, JAX-RS and other Library considerations 29](#_Toc387755556)

[6.3. Tomcat 6.x Deployment 29](#_Toc387755557)

[6.4. Tomcat 7.x and above and JBoss Deployment 30](#_Toc387755558)

[6.4.1. Tomcat 30](#_Toc387755559)

[6.4.2. JBoss (RESTEasy) 31](#_Toc387755560)

[6.5. Verify Consumer & Provider Property Files 31](#_Toc387755561)

# 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 0.1) in September-October 2013. This work has been sponsored by [NSIP](http://www.nsip.edu.au/). In March-May 2014 some additional work has been undertaken to increase the functionality of the framework to add events and broker connection capability (alpha 0.2). As of May 2014 the framework is limited to **immediate** responses for DIRECT and BROKERED environments. Delayed I/O is not yet supported (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 a 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 NSIP’s GitHub site (<https://github.com/nsip/sif3-framework-java>) 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 the near future. 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 only undergone limited testing with Java 7.

## Functionality

As of May 2014 (alpha version 0.2 of the framework) the SIF3 Framework is limited to the following functionality:

* Uses SIF 3.0.1 Infrastructure (current and latest version)
* Immediate Responses only (no delayed responses)
* Provider and Consumer can operate in a DIRECT and BROKERED environment.
* 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 specifies the details of an XML<->JSON mapping).
* No xQueries & query templates (see SIF3 Spec for details on this functionality)
* No service paths (see SIF3 Spec for details on this functionality)
* Events:
  + **Provider**: Can only publish events in a brokered environment. The direct environment provider implementation of this framework does not support events.
  + **Consumer**: Can subscribe to events from any environment provider that supports events.

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

# Installation

Before you can install the SIF3 Framework you need to download it from <https://github.com/nsip/sif3-framework-java>. 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 are a couple of notable exceptions. They are the **AbstractConsumer** and the **AbstractEventConsumer** that a developer must extend for the development of Consumer style service.
* **demo/src**: Demo code to showcase how a consumer (StudentPersonalConsumer) and provider (StudentPersonalProvider) is implemented using the SIF3 Framework.
* **test/src**: Various test classes to test parts/components of the framework. Mainly used for development.
* **sif3Datamodel/src**: This package does **NOT** form part of the framework. I.e. is not required for the framework to function. 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. Details about the configuration files and their content can be found in appropriate sections of this document.

**Documentation Directory**

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

**Generator 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 may need some or all of these libraries in your classpath.

**Web-Application Directory**

Providers are “servers” and therefore must be deployed in a web- or application container. The “**war**” directory holds all the required structure, descriptors and libraries for a provider to be deployed in a web- or application container. Note this directory is only in that project to showcase how a provider could be deployed as a “war”. You are not required to use this exact setup for your development or deployments. More details on how to configure your own project for deployment can be found in section 6.

**Build Directory**

The “**build**” directory is the directory where classes and the final jars 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 have ant (<http://ant.apache.org/>) installed.

### Consumer Build

As part of the download you should have the required libraries already pre-built in the build/dist directory. If you want to build them yourself or if you have changed anything in the framework and need a rebuild then you can use ant to do so. 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**
* sif3Demo-<version>.jar
* sifDataModel\_au1.3.jar

Please note that you only need the libraries marked in **bold** for your project. Copy them over to your SIF project to have the framework available. The sifDataModel\_au1.3.jar is encapsulating parts of the SIF AU 1.3 data model. It is not complete at the moment. You can use it in your project if you wish but you don’t have to. It is an optional library. The sif3Demo-<version>.jar holds the demo classes that showcase the SIF3 Framework. It is not required in your project at all. In fact you should not copy it into your own project. There is a high likelihood that the data model and demo code will be removed from this project in the near future and will become an own little project on GitHub.

### Demo Provider Build

If you build the Demo 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. Please refer to section 6 for some instructions on how to build the war file for your runtime environment.

# 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/Infrastructure%203.0.1/SIF_3-0-1_Infrastructure_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 class for each SIF Object.
* For each provider you need to determine if its objects are published as events or are published/modified on request only (different abstract classes 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 abstract classes 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 resides)
* project.**consumer** (package where the code of your consumers resides)
* project.**datamodel** (package where the code for your data model, marshallers and unmarshallers resides)

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

* project.**business** (package where business logic code resides)
* project.**model** (package where model code resides, ie. models reflecting your internal object structure)
* project.**dao** (package where DAO code resides 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).

## Environments

The SIF Environment forms a core part of SIF3. Everything you do is in the context of an environment. Before any operations can be performed, consumer and/or provider, there must be a valid SIF Environment. The environment is either being provided by an Environment Provider (Brokered Environment) or the Provider itself (Direct Environment). A service’s first step is to authenticate against an environment and then get an authorisation. Only once this step is successfully performed a service can participate in a SIF Environment. A consumer must further investigate the environment to figure out what it is allowed to do (access rights to services), where services are located (URIs) etc. All of this adds significantly to an implementation of either a consumer or a provider. This SIF3 Framework has a number of environment manager built-in (Consumer Environment Manager, Brokered Provider Environment Manager etc), so that the developer doesn’t have to deal with all of this and much more. It is fully abstracted within this framework. The implementer only needs to ensure that the environment is configured properly. Each service, consumer or provider, has two components to configure the environment, and after this is done, the implementer doesn’t have to write any code to deal with environments. The two core components are:

* Service Property File
* Environment Store Setup

The service property files are discussed in details in appropriate sections (consumer see section 5.5.2.1.1, provider see section 5.6.2.1.1).

This section is mainly concerned on how to setup your environment store.

### Environment Store Setup

It is important to note that the current version (May 2014) of the framework uses a combination of file based environment information (environment templates) and database base information (runtime sessions and environments) to manage environments. To get this working some initial installation and database configuration is required. There are a set of configuration files that need to be setup and/or configured correctly to get the full environment store to work. The next three sections lead you through the setup of the environment store.

#### environment.properties File

This property file holds some high level information on where your Environment Templates are stored (see next section for the term ‘Environment Template’). If your services use HTTPS instead of HTTP to connect to end-points then the environment.properties file holds the information which key and trusted stores you want to use for certificate lookup and validation. When you deploy your SIF project you must ensure that this properties file is on your deployment’s **classpath**. Within the framework an example of this file can be found in the **config** directory. This file is **KEY** to the entire environment management of the framework. The next section lists information about each property that can/must be set in this file.

**env.store.dir**

This property must point to a valid physical location on your file system. This is the ‘root’ directory of all your environment templates, consumer and provider property files. All environment template data will be accessed in sub-directories of this ‘root’ directory.

The next four values are all required if you configure your consumer/provider to use HTTPS (secured & encrypted connections) instead of HTTP (unsecured & unencrypted connections). Please refer to section 5.7.1 and 5.7.2 for details on how to configure HTTPS for consumers and providers.

**key.store**

This is the physical location where your Java keystore is located.

**key.store.password**

This is the password to use to access the Java keystore.

**trust.store**

This is the physical location where your Java Trusted Certificate Authority store is located.

**trust.store.password**

This is the password to use to access the Java Trusted Certificate Authority store.

Please note, you can point to existing key and trusted stores or you can create your own stores scoped for your SIF services only. If you set your consumer or provider to use HTTPS the above properties will be used to configure and validate HTTPS connections.

#### Environment Template Directory (File System)

Environment Templates are XML files that hold information about a specific SIF3 environment configuration. The consumer’s and/or provider’s environment store might load environment information into memory during operation but the persistence of the information is in XML format. The XML is the exact structure as specified in the SIF3 Infrastructure XSD. The content of the environment XML templates vary significantly depending if the service is a consumer, a provider in a brokered environment or a provider in a direct environment. A consumer (direct or brokered) and a brokered provider have a very basic XML template, while a direct provider has a much more complex template (more on this later).

##### Environment Template Store Structure

The environment template store on your file system has a given structure. Most of it is automatically setup by the environment store managers and a developer doesn’t have much to do with it. Still it is important to know what that general structure is and how things function so that you can quickly and easily add new environment templates to your configuration. The image below illustrates a typical environment template store structure. It depicts the structure of a consumer’s (‘consumer’ sub-directory) as well as a provider’s environment template store (‘provider’ sub-directory).



##### Consumer Environment Store Structure

The core directory that must exist under the ‘consumer’ sub-directory is called ‘**template**’. This directory holds environment templates that can be used by any consumer to connect to an environment. The typical content of such a template XML is shown below:

<environment xmlns="http://www.sifassociation.org/infrastructure/3.0.1">

**<solutionId>testing</solutionId>**

**<authenticationMethod>Basic</authenticationMethod>**

<consumerName></consumerName>

<applicationInfo>

**<applicationKey></applicationKey>**

**<supportedInfrastructureVersion>3.0.1</supportedInfrastructureVersion>**

**<dataModelNamespace>http://www.sifassociation.org/au/datamodel/1.3 </dataModelNamespace>**

**<transport>REST</transport>**

**<applicationProduct>**

**<vendorName>Systemic Pty Ltd</vendorName>**

**<productName>Test Driver</productName>**

**<productVersion>0.1alpha</productVersion>**

**</applicationProduct>**

</applicationInfo>

</environment>

The elements highlighted are mandatory. The party that implements the Environment Provider will let you know what value to use in the <solutionId> and <applicationKey> node. You can create as many such templates as you like or to be exact the provider knows about (you can only connect to environments that are known to the environment provider!).

Whenever a consumer is started/initialised for the first time it will look up the **consumer/template** directory to find the appropriate environment template (the consumer’s specific property file will indicate which template to lookup, see also section 5.5.2.1.1). This template is then used to connect with the environment provider.

##### Provider Environment Template Store Structure

The environment template store structure of a provider can be a bit more complex. It depends if the provider is participating in a brokered or direct environment. For each setup there is a sub-directory where appropriate environment template XMLs are stored. The directories are called ‘**provider/template/direct**’ and ‘**provider/template/brokered**’.

**Brokered Providers**

If you are dealing with a provider that connects to a SIF 3 Broker then you are dealing with a ‘Brokered Provider’. In this case it connects to the broker the exact same way as a consumer and the environment template XML has the identical structure and mandatory elements as a consumer. See section 5.3.1.2.2 for details.

**Direct Providers**

If your provider is a direct provider, meaning it is a direct environment as specified in the SIF3 Base Architecture then your environment template XML must hold some additional information that is returned to a consumer when it creates an environment. One important component is the set of infrastructure URIs.

*Infrastructure Service URIs*

The environment template of a direct provider **must have** a section where all the valid infrastructure service URIs are listed. This section will look something like this:

<infrastructureServices>

<infrastructureService name="environment">**environments**</infrastructureService>

<infrastructureService name="requestsConnector">**requests**</infrastructureService>

<infrastructureService name="provisionRequests">**provision**</infrastructureService>

<infrastructureService name="queues">**queues**</infrastructureService>

<infrastructureService name="subscriptions">**subscriptions**</infrastructureService>

</infrastructureServices>

The parts in **bold and red** are URI segments. They must remain like this and cannot be changed. These template URIs will be expanded to a full URIs. They will be prefixed with the value that is set in the env.connector.url property of the provider’s property file (see section 5.6.2.1.1 for details about this property and its use). The final environment XML with the fully expanded infrastructure URIs is returned to the consumers.

*Service Access Rights (Access Control Lists – ACLs)*

The second important section is the <provisionedZones> node. It states all the rights a consumer has for each SIF Object in each zone (ACL in SIF 2.x). The structure is as followed:

<provisionedZones>

<provisionedZone id="MyZone">

<services>

<service name="StudentPersonals" contextId="DEFAULT" type="OBJECT">

<rights>

<right type="QUERY">APPROVED</right>

<right type="CREATE">APPROVED</right>

<right type="UPDATE">APPROVED</right>

<right type="DELETE">APPROVED</right>

<right type="PROVIDE">REJECTED</right>

<right type="SUBSCRIBE">SUPPORTED</right>

<right type="ADMIN">REJECTED</right>

</rights>

</service>

<service name="SchoolInfo" contextId="DEFAULT" type="OBJECT">

<rights>

<right type="QUERY">APPROVED</right>

<right type="CREATE">APPROVED</right>

<right type="UPDATE">APPROVED</right>

<right type="DELETE">APPROVED</right>

<right type="PROVIDE">SUPPORTED</right>

<right type="SUBSCRIBE">SUPPORTED</right>

<right type="ADMIN">APPROVED</right>

</rights>

</service>

</services>

</provisionedZone>

<provisionedZone id="MyOtherZone">

...

</provisionedZone>

</provisionedZones>

You need to configure this to set the ACLs you would like to give to the consumers. Valid values for rights and right types can be found in the SIF3 specification. Ensure that such an ACL is configured in each environment template. Examples can be found in the directory ‘**config/environment/provider/template/direct**’.

While all of the above may sound complicated, once configured the developer will rarely need to care about any environment template management anymore. It is a one-of setup at the start of your deployment/implementation.

#### Environment and Session Store (Database)

The latest version of the framework requires a database to store runtime environments and SIF3 sessions as well as queue and subscription information in cases where events are enabled. You need to run a DDL script to create tables, sequences, indexes etc. in a database schema of your choice. You can use whatever database product you like (MySQL, Oracle, SQLServer etc). There are a set of DDL scripts in this framework under ‘DB/DLL’ to create appropriate tables. If you use another database product for which there is no DDL provided, then you can use an existing DDL and modify it to match your database product. There are no scripts provided for Postgres. You can use the Oracle scripts as they work for Postgres as well.

Once you have run the DDL scripts you need to set the correct connection URL, username, password etc. in the ‘**config/hibernate/sif3infra.hibernate.cfg.xml**’ file to match your database connection URL. Finally you need to add an appropriate JDBC driver library to your project.

Now your runtime environment and SIF3 session store is ready to go.

## Data Model

Before you can develop any consumers or providers you need to choose a data model your services work with. For the purpose of this developer’s guide we use the AU 1.3 data model. Whatever data model you use, you will be required 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 used the OpenADK as 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 is 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 “over the wire” 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 will require you to implement a few methods that tell it which marshal and un-marshal factory to use so that it can do the required serialisation on the wire. See section 5.5.2.3 and 5.6.2.2 for details.

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 is just a “client” in the traditional sense.

### Constraint

In the current version (May 2014) the framework doesn’t provide classes to deploy the consumer as a standalone executable. This may be added in later versions of the framework. If you require the consumer to run as a standalone executable you are required to write the appropriate class with a main() method. There is an example in the **test/src** directory. See the class **sif3.infra.test.rest.consumer.TestStudentPersonalConsumer**

### Components of a Consumer

There are a few components that are required to get your consumer working and configured. These components are:

* Configure your Consumer Environment
* Decide on the SIF Objects you need to deal with
* For each SIF Object your service deals with you need to implement a class that extends either the **AbstractConsumer** (no event processing) or the **AbstractEventConsumer** (events supported) class to wire up the Framework

Each of the above steps is illustrated in the next few sections.

#### Configure your Consumer Environment

##### Consumer Properties File

Each consumer has a property file that must be part of the consumer’s classpath. Your consumer is initialised with this property file as part of the ConsumerLoader class (see section 5.5.2.2 for details). There is an example of such a file in the **config/consumers** directory. It has some extensive comments for each property. This section further discusses each property on how to use it. Note that in the documentation of the properties it often refers to the “provider”. In this context a “provider” can be a direct provider or a broker in case of a brokered environment.

**adapter.type**

Valid values for this property are either **consumer** or **provider**. In this case it must be “consumer” because we deal with a consumer.

**adapter.id**

The unique ID of the consumer. This value will be used when the consumer creates an environment on the environment provider. The value of that property is inserted into the **<consumerName>** node of the consumer’s environment template XML before it is sent to the provider.

**adapter.use.https**

This property has the value ‘true’ or ‘false’. If it is set to ‘true’ then the consumer will attempt to establish a secured (HTTPS) connection to the provider. If it is set to ‘false’ then an unsecured connection will be established. Please ensure if you use ‘true’ (HTTPS) that your env.baseURI property has an URI of the form http**s**://... instead of http://...

**adapter.checkACL**

Turn on (true) or off (false) ACL check on the consumer. Default is true. If this property is set to ‘true’ then the consumer will check many permissions before the provider is called. Only if the permission of the given operation is set to ‘APPROVED’ in the environment’s ACL then a call to the provider is performed. Ideally this should be set to true to avoid obsolete calls to the provider but for some testing purpose it might be useful to turn this check off (set to false).

**adapter.mustUseAdvisoryIDs**

This property indicates if the consumer manages all the RefIds/UUIDs of the Data Model Objects and therefore the provider must accept them as they are when a ‘create’ operation is requested (HTTP POST) by the consumer. If it is set to ‘**true**’ then the provider will be requested to use the consumer’s RefIds/UUIDs. If it is set to ‘**false**’ then the provider is expected to allocate the RefIds/UUIDs for the objects when they are created. If this property is not provided then it will be defaulted to ‘**false**’.

**adapter.deleteEnvironment.onShutdown**

Indicates if the environment shall be deleted (true) on the provider and in the environment store (DB) when the consumer shuts down. **USE WITH CARE!** If an environment is removed, all associated data is lost (queues, messages in queues, delayed responses etc). This property should only be TRUE in a direct environment without events and delayed I/O. In most implementations where consumers are used to integrate systems (i.e. not tablet applications) that property would be set to FALSE.

**env.events.supported**

This property indicates if the provider supports events. Only if this property is set to true then the consumer will start its event processor and events can be received and processed. Please note that the property ‘**events.enabled**’ must also be set to true for events to fully work!

**env.xml.file.name**

The name of the environment template file to be used with this environment. This is the full file name with the extension “.xml”. This file must exist in the environment template store for consumers and it must have a certain structure. Details can be found in section 5.3.1.2.2.

**env.application.key**

The application key this consumer **MUST** use. The party/vendor that implements/provides the environment provider will let you know what that value is. It is used to create the authentication token when an environment is created. The value of that property is inserted into the **<applicationKey>** node of the consumer’s environment template XML before it is sent to the environment provider.

**env.userToken & env.instanceId** (Optional)

These are additional values used to further define what exact instance of an environment shall be created by the environment provider. These values are inserted into the <userToken> and <instanceId> node of the consumer’s environment template XML before it is sent to the environment provider. These are optional values and may rarely be used.

**env.pwd**

The password to authenticate with a specific environment. The party/vendor that implements/provides the environment provider will let you know what that value is. It is used to create the authentication token when an environment is created or when any other interaction with the provider is performed.

**env.mediaType**

The media type use with this environment. Can be XML or JSON.

Note:

Currently SIF3 only specifies XML. If you implement JSON as a media type you do this at your own risk as it may not conform to the future definition of the JSON mappings in the SIF Specification.

**env.authentication.method**

The authentication method to be used for this environment. Valid values are ‘Basic’ and ‘SIF\_HMACSHA256’ (case sensitive!). ‘Basic’ is the default. The value of that property is inserted into the **<authenticationMethod>** node of the consumer’s environment template XML before it is sent to the provider.

**env.baseURI**

The base URL to connect to the environment provider. This will be given to you by the vendor that provides the environment provider. There is a high likelihood that this URI may look in something like http(s)://…/environments/environment but it doesn’t have to be.

The next set of properties relate to event processing only. Please note while you can set these values it is not guaranteed that events are being supported by the event provider. You must consult with the vendor that provides you with an implementation of an environment provider if events are supported. Often in DIRECT environments the event functionality might not be there. In a BROKERED environment it is expected that event functionality is available. See also property ‘env.events.supported’.

**events.enabled**

Indicate if event subscription is enabled. Even if an environment supports events, this consumer might not be interested in any events. Default is false (don't subscribe to events). This is useful to turn off events on a consumer (i.e. temperately for some testing). Even if this property is to true it is not guaranteed that events are being received. The ‘env.events.supported’ must also be set to true for events to be received.

**events.queue.strategy**

SIF3 allow many ways on how to create queues and direct events to them. This framework only supports a set of the most common strategies. Currently only 'ADAPTER\_LEVEL' is implemented which mean a single queue for this consumer is created. Future expansions can be 'ZONE\_LEVEL' (One queue per zone), SERVICE\_LEVEL (one queue per service in each zone) etc. Default is ADAPTER\_LEVEL. Please refer to section 5.5.3.1for more information about queue strategies.

**events.queue.name**

This property is used give the queues a certain name. This might be the actual queue name if the strategy is ADAPTER\_LEVEL. It might be the queue's prefix if any other queue strategy is used where multiple queues are configured. If not provided the name is assumed to be the same as the adapter.id property with all white spaces removed. This name is for display purposes only on the broker. It will make it easy to find your specific queue. This name will also be used in some debug/info entries in logs on the broker and/or consumer.

**events.queue.type**

There are two ways how a consumer can receive messages from a queue. IMMEDIATE or LONG\_POLLING. Refer to SIF3 specification for details on the two options. Default is IMMEDIATE. This is the same behaviour as known in SIF 2.x for Pull agents.

**events.queue.subscribers**

SIF3 allows concurrent subscribers/readers on each queue. This property indicates the number of subscribers/readers per queue. The default is 1. Care must be taken with other numbers as it could mean that events are processed out of order! The environment provider might override that value with a lesser value.

**events.polling.frequency**

The frequency with which the queues are being polled to check if there are messages available. This value is in seconds and might be overwritten by the environment provider to a larger value. Default is 60 seconds (once a minute).

**events.longPolling.timeout**

Number of seconds the connections shall remain open for LONG\_POLLING queues before it is closed. See SIF3 Spec for more details about LONG\_POLLING queues. Default is 120 seconds (2 minutes).

**events.subscriptions.removeOnShutdown**

When a consumer is shut down then events will be queued on the provider. This is generally the desired behaviour. There might be situations where a shutdown of the consumer also means no events shall be queued during the time the consumer is offline. To enable this, the subscriptions to queues and therefore events must be removed from the appropriate SIF queue on the provider. This property indicates if subscriptions shall be removed when the consumer shuts down. **USE WITH CARE!** If set to TRUE then no events will be queued on the provider. Default is false which indicates that subscriptions shall remain when the consumer shuts down and therefore events will queue up on the provider.

The next section of the consumer property file relates to the wiring of the consumer. It tells the event processor what consumer classes make up the actual consumer. The event processor will then inspect each consumer class and configure it accordingly. Each consumer is most likely made up of several classes, each implementing the functionality of for a particular SIF Object Type (see also section 5.5.2.3 for details).

**consumer.basePackageName**

The name of the base/root package where all consumer classes can be found. It is suggested that all consumer classes (one per SIF Object Type) are being placed under one package. This property is the fully qualified path (dot notation) of that consumer class package.

**consumer.classes**

This property is a comma separated list of class names of all consumer classes that make up the final consumer. The consumer.basePackageName will be prefixed to get the final fully qualified class name of all consumer classes.

**consumer.startup.delay**

Not used, yet.

**consumer.local.workerThread**

The number of threads processing messages (events) locally. This is the number of threads for each object type (consumer class in the consumer.classes list). If this value is not set then 1 is assumed (default). If a particular consumer requires a different number of worker threads then it can be set individually by means of appending ".<consumerName>" at the end of this property. For example if the daily attendance consumer shall have 5 worker threads then it would be set with the following property and value: consumer.local.workerThread.DailyAttedanceConsumer=5 (Note that there must be a class called DailyAttendanceConsumer in the consumer.classes list).

**Note:**

You can add any other properties to this file. They are available to you in each consumer class through the **getServiceProperties()** method.

#### FIRST CALL: ConsumerLoader Class

This is the most important class for any consumer implementation. It must be called once per consumer deployment unit. It will ensure that the consumer configures itself as desired. It will connect and sync up with the environment provider (DIRECT or BROKERED) and if the consumer implements event functionality it will initialise the event processor. This class is a static class and ensures that it is only being called/initialise once per deployment unit. There are three methods in this class that are of importance:

Method **initialise(String consumerPropertyFileName)**

This is the first method your consumer must call. The consumerPropertyFileName is the name of the consumer’s property file name WITHOUT the extension of “.properties”. It will return true if the initialisation succeeds. If false is returned it must be assumed that initialisation is not successful and your consumer should not continue. Please note that if any of your consumer classes extend the **AbstratctEventConsumer** class (events enabled) then the successful execution of this method will **immediately** kick off the event processing, assuming events are queued on the provider.

Method **isInitialised()**

This method can be used to check if initialisation has succeeded. Note the initialise() method also returns this value.

Method **shutdown()**

The last thing your consumer implementation should do before it shuts down is calling this method to ensure that all resources are freed up and that the consumer can shut down gracefully.

For an example on how these methods are used please refer to some of the test consumer in the test/src directory.

#### The AbstractConsumer & AbstractEventConsumer Classes to wire up the Framework

The next step is to decide what SIF Objects you want to consume. For each SIF Object you will be required to implement a consumer class that extends either the **AbstractConsumer** or the **AbstractEventConsumer** (events enabled) class. With these classes you wire up your implementation with the framework or in other words you make the framework aware of your SIF Objects, data model etc. Each class you have implemented should be added to the **consumer.classes** property in your consumer’s property file to activate it as part of your consumer.

The **AbstractConsumer** and the **AbstractEventConsumer** class provide you with an implementation of some standard and common functionality of the Consumer and/or Event Interface. The main purpose of these classes is to:

* Add some abstract methods that make the consumer Data Model aware.
* Implement low level functionality to deal with Bulk Operations to abstract infrastructure details.
* Thread management for event processing (AbstractEventConsumer)

In the **demo/src** directory you can find some examples on how to implement the abstract methods for these classes. Check out the StudentPersonalConsumer (events enabled), SchoolInfoConsumer (no events) classes. The minimum methods that must be implemented are:

**Constructor:**

Takes no arguments. Please ensure that your consumer implementation uses the empty constructor. It is required that way by the framework.

**getSingleObjectClassInfo() Method:**

* Links up the name of a SIF Object Type with the internal data model class.
* Used for all Single Object operations within the framework

**getMultiObjectClassInfo() Method**

* Links up the name of Collection Type Objects with the internal data model class.
* Used for all bulk operations and events within the framework

**getMarshaller() Method:**

Tells the framework which marshal factory implementation to use to convert your data model object to XML and/or JSON (JSON not supported by SIF3, yet).

**getUnmarshaller() Method**

Tells the framework which un-marshal factory implementation to use to convert a XML and/or JSON string to your data model object (JSON not supported by SIF3, yet).

**shutdown() Method**

This method is called when the consumer is shut down. It can be used to free allocated resources by this consumer.

**processEvent () Method – AbstractEventConsumer Only**

Note this method must only be implemented if your consumer extends the AbstractEventConsumer. This framework’s event processor will call this method if it receives an event for a SIF Object type that is managed by this consumer class. There are a number of parameters that are given to this method:

* **sifEvent**: The actual event data. Please refer to the javadoc of the SIFEvent<L> class for details.
* **zone**: The zone from which this event has been received. The framework ensures that this value is never null even if the event was for the default zone. In that case this parameter will be set to the default zone data.
* **context**: The context for which this event has been received. The framework ensures that this value is never null even if the event was for the default context. In this case this parameter will be set to the DEFAULT context.
* **msgReaderID**: An ID that indicates which message queue reader thread has received the event message. It is informative only.
* **consumerID**: An ID that indicates which consumer thread is processing this event. It is informative only.

**createEventObject () Method – AbstractEventConsumer Only**

Note this method must only be implemented if your consumer extends the AbstractEventConsumer. By now you may have noticed that the event related classes of this framework use generics. Until Java 7 it was not possible to create instances of a generic class at runtime using reflection. To enable the framework to run in a pre-Java 7 JVM and still make use of generics this class must be implemented. It enables the framework to call this method which has detailed knowledge about the generic implementation and can create the event object based on the passed in parameters. The implementation of this class should simply create a SIFEvent<L> object and assign the parameters of this method to the SIFEvent object. A typical such implementation could look like this (assume StudentPersonal SIF Object which equates to the StudentCollectionType class):

public SIFEvent<StudentCollectionType> createEventObject(Object sifObjectList,

EventAction eventAction,

UpdateType updateType)

{

if (sifObjectList != null)

{

if (sifObjectList instanceof StudentCollectionType)

{

int size = ((StudentCollectionType)sifObjectList).getStudentPersonal().size();

return new SIFEvent<StudentCollectionType>((StudentCollectionType)sifObjectList,

eventAction,

updateType,

size);

}

}

else

{

logger.error("The given event data is null. Cannot create event object. Return null");

}

return null; // if something is wrong then we get here.

}

#### Actual Implementation

The final bit is your actual implementation where you write the code to connect the consumer to your data store and read or persist the data from the various calls of your consumer. You can find a basic example how this could look like in the **test/src** directory. Check out the class **sif3.infra.test.rest.consumer.TestStudentPersonalConsumer**.

**Note:**

The above class reads the data from a file. At the top of the class are the paths where the file is located. You may need to modify that path to your installation to make this class work properly.

Further note the constant **CONSUMER\_ID** that is defined at the top of this Test class. This ID is/must be the same as the name of the consumer property file that will be used for this consumer (without the extension “.properties”).

Finally notice the **ConsumerLoader.initialise(CONSUMER\_ID)** call as the first line in the main(). It must be there to ensure that the framework initialises all the components at start up. Refer to section 5.5.2.2 for details.

### Consumer Events Processing & Threads

The SIF3 Framework makes extensive use of multi-threading for event processing. This has a number of reasons but the two most important ones are:

* SIF3 allows concurrent readers on the SIF Queues.
* Increase performance of event processing where “order is not important”.

This section shortly discusses how the multi-threading works and how it can be configured using the consumer’s property files. It is important that the developer understands how the multi-threading strategies in that framework work, so that the best benefit can be achieved but also that not too many obsolete threads are created that may negatively impact the service’s resources.

The number of threads that a consumer will create depends on three factors, each of which can be controlled through the consumer’s property file. These three factors are:

* Queue Strategy
* Number of Concurrent Message Readers on each queue
* Number of Concurrent Event Processing Workers

Each of the above is shortly discussed in the next sections.

#### Queue Strategies

Currently (May 2014) the only supported queue strategy for the framework is called ADAPTER\_LEVEL. This queue strategy is set with the ‘events.queue.strategy’ property in the consumer’s property file. Future extensions to the framework will offer two more strategies. They are called ZONE\_LEVEL and SERVICE\_LEVEL. The three strategies dictate how many queues will be created on the provider and how they are associated with data.

**Strategy: ADAPTER\_LEVEL**

This strategy will create one single message queue per environment. All events of all zones will be relayed through this queue. This is the most basic strategy and will be sufficient in most implementations. Since most environment deal with one zone only this strategy will be fine.

**Strategy: ZONE\_LEVEL** (not yet supported as of May 2014)

This strategy will create a single message queue per zone that is known in an environment. This is almost the same as known from SIF2.x ZIS’. If a consumer’s environment deals with one zone only then this strategy will equate to the same as the ADAPTER\_LEVEL strategy.

**Strategy: SERVICE\_LEVEL** (not yet supported as of May 2014)

This is the most complex strategy and is something that could only be achieved with SIF 2.x if each service was deployed as a separate agent. In SIF3 it is now possible to create a queue for each service if required. The number of queues this strategy will create is equal to all Object Services across all zones. So, if an environment can access 4 zones each of which has 7 services (SBP services) then this strategy will immediately create 28 queues (4 zones \* 7 services)!

#### Message Readers

SIF3 allows multiple concurrent reader threads on each message queue. This is new in SIF3 and was not possible in SIF2.x. The SIF3 Framework supports that option. The number of concurrent readers on each message queue is set in the ‘events.queue.subscribers‘ property in the consumer’s property file. Care must be taken as this will create that number of reader threads per message queue. The number of reader threads is the queue number (dictated by the Queue Strategy) multiplied with the number of that property. Please refer to section 5.5.3.4 for an example.

**Note:**

It is important to note that multiple readers increase the message throughput **BUT** it has also the danger that message order can no longer be guaranteed. Multiple concurrent queue readers should only be used where the order of messages is of no importance!

#### Event Processing Worker

The final level of thread creation is scoped to the actual consumer class that implements the event processing. As with queue readers the SIF3 Framework allows that multiple concurrent worker threads can be started to further increase the message throughput and processing. For example you can configure the framework to have three concurrent StudentPersonal Consumer threads running. This is configured with the ‘consumer.local.workerThread’ property in the consumer’s property file.

**Note:**

As with the concurrent message queue readers, multiple worker threads can increase the throughput of messages and their processing **BUT** it has also the danger that message order can no longer be guaranteed. Multiple worker threads should only be used where the order of messages is of no importance!

#### Thread Examples

This section illustrates the number of threads that may run in a consumer and how they relate to the queue strategy and the various properties in the consumer’s property file

**Example 1: Queue Strategy = ADAPTER\_LE**VEL

The image depicts an example of the simplest queue strategy (ADAPTER\_LEVEL). See also section 5.5.3.1 for details. It depicts the number of threads and how they interact with each other within the framework.



The above example shows 9 threads running (3 message reader threads and 6 event processor/worker threads). This means at any time there could be 6 messages being processed concurrently (6 event processor/worker threads). It also means that at any time the consumer can concurrently pull down 3 messages from the queue and then distribute them to available worker threads.

**Example 2: Queue Strategy = ZONE\_LEVEL (not yet supported)**

To be added once supported.

**Example 3: Queue Strategy = SERVICE\_LEVEL (not yet supported)**

To be added once supported.

## 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 the web.xml to your environment. Please refer to section 6 for details on how to configure and deploy providers with various web- and application containers.

### Constraint

As of May 2014 (alpha version 0.2) the framework has been tested with Tomcat and JBoss. 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. Please refer to section 6 for more details about deployments in various web- and application containers.

### Components of a Provider

**Note:**

This section deals with Provider Components specific to this framework as of May 2014. Some of the components/statements listed or stated here are only true for this framework and might sound slightly incorrect according to the SIF3 Specification. The “incorrectness” is only due to the way the current version of the framework implements certain concepts, but it doesn’t break the SIF3 Specification at all.

There are a few components that are required to get your provider working and configured. These components are:

* Configure your Provider Environment. As of April 2014 this framework deals with DIRECT environments (provider is also a basic environment provider) and BROKERED environments (provider connects to a SIF3 Broker).
* Decide on the SIF Objects you need to deal with
* Implement a **BaseProvider** (no events) or **BaseEventProvider** (event support) Class per SIF Object and wire it up to the Framework
* Deployment

Each of the above steps is illustrated in the next few sections.

#### Configure your Provider Environment

##### Provider Property File

As with a consumer the provider has a property file that tells the application what a provider consists of, what environments it deals with etc. The list below explains each property of the provider’s property file. It is also suggested that you look at the **StudentProvider.properties** file in the **config/providers** directory for some examples and more details and explanation of some properties.

**adapter.type**

Valid values for this property are either **consumer** or **provider**. In this case it must be “provider” because we deal with a provider.

**adapter.id**

The unique ID of the provider. This value will be used when the provider “talks” to the broker in a brokered environment. The value of that property is inserted into the **<consumerName>** node of the provider’s environment template XML before it is sent to the broker.

**adapter.checkACL**

Turn on (true) or off (false) ACL checks on the provider. Default is true. If this property is set to ‘true’ then the provider will check the permissions for the calling consumer to determine if ‘access’ is approved. This property should always be set to ‘true’ except in some testing scenarios where the ACL shall be ignored for all consumers.

**env.type**

This value can either be DIRECT or BROKERED. It indicates the type of the provider. In a DIRECT environment the provider will also act as a basic environment provider. In a BROKERED environment the provider will attempt to connect to a broker (see also env.baseURI property)

**env.events.supported**

Indicate if the environment provider supports events. As of May 2014 this framework does not support events if the env.type is set to DIRECT. If the provider connects to a broker and therefore acts as a BROKERED provider then events should be supported by the broker. In this case set this property to true. Valid values are true and false. The default is false.

**env.mediaType**

The media type used with this environment. Can be XML or JSON.

Note:

Currently SIF3 only specifies XML. If you implement JSON as a media type you do this at your own risk as it may not conform to the future definition of the XML->JSON mappings of the SIF Specification.

**env.xml.file.name**

The name of the environment template file to be used with this provider. This is the full file name with the extension “.xml”. This file must exist in the environment template store for providers and it must have a certain structure. Details can be found in section 5.3.1.2.3.

**env.authentication.method**

The authentication method to be used for this environment. Valid values are ‘Basic’ and ‘SIF\_HMACSHA256’ (case sensitive!). ‘Basic’ is the default. The value of that property is inserted into the **<authenticationMethod>** node of the provider’s environment template XML before it is sent to the consumer or broker.

**env.application.key**

The application key this provider **MUST** use. This property is used slightly differently depending if the provider is a DIRECT or BROKERED provider:

*DIRECT*:

In this configuration the application key is used to authenticate the consumer that attempts to connect to the provider.

*BROKERED*:

In this configuration the provider uses the application key to authenticate itself with the broker. In this case the party/vendor that implements/provides the environment provider will let you know what that value is. It is used to create the authentication token when an environment is created. The value of that property is inserted into the **<applicationKey>** node of the provider’s environment template XML before it is sent to the provider.

**env.pwd**

The password this provider **MUST** use. As with the application key the password is used slightly differently depending if the provider is a DIRECT or BROKERED provider:

*DIRECT*:

In this configuration the password is used to authenticate the consumer that attempts to connect to the provider.

*BROKERED*:

In this configuration the provider uses the password to authenticate itself with the broker. In this case the party/vendor that implements/provides the environment provider will let you know what that value is.

**env.connector.url**

This is the ***unsecured*** base URL of all connections (***HTTP***) to this provider and should be of the form ***http***://..... This is made up of several bits.

**env.connector.url.secure**

This is the ***secured*** base URL of all secured connections (***HTTPS***) to this provider and should be of the form ***https***://.... This is made up of several bits.

The next set of properties are specific to BROKERED providers. For DIRECT providers they are not used.

**adapter.use.https**

This property has the value ‘true’ or ‘false’. If it is set to ‘true’ then the provider will attempt to establish a secured (HTTPS) connection to the broker. If it is set to ‘false’ then an unsecured connection will be established. Please ensure if you use ‘true’ (HTTPS) that your env.baseURI property has an URI of the form http**s**://... instead of http://...

**adapter.deleteEnvironment.onShutdown**

Indicates if the environment shall be deleted (true) on the broker and in the provider’s environment store (DB) when the provider shuts down. **USE WITH CARE!** If an environment is removed, all associated data is lost (queues, messages in queues, delayed responses etc). This property should only be TRUE in direct environments without events and delayed I/O. In most implementations where providers are used to integrate systems (i.e. not tablet applications) that property would be set to FALSE.

**env.baseURI**

The base URL to connect to the broker’s environment provider. Ensure that the protocol matches what is indicated with the ‘adapter.use.https’ property. This is the URI that will be used to establish an environment with the environment provider at start-up of the provider. It most likely has the form http(s)://…/environments/environment but it doesn’t have to be like that.

The next section of the provider property file relates to the wiring of the provider. It tells the web-application what provider classes make up the actual provider. Each provider is most likely made up of several classes, each implementing the functionality of for a particular SIF Object Type (see also section 5.6.2.2 for details)

**provider.basePackageName**

The name of the base/root package where all provider classes can be found. It is suggested that all provider classes (one per SIF Object Type) are being placed under one package. This property is the fully qualified path (dot notation) of that provider class package.

**provider.classes**

This property is a comma separated list of class names of all provider classes that make up the final provider. The provider.basePackageName will be prefixed to get the final fully qualified class name of all provider classes.

**provider.startup.delay**

Each provider in the list above is started in their own thread. These threads are started with a little delay between them. The default is 10 seconds. This property allows overriding that value. The delay is in seconds.

The next section of the provider property file relates to the event configuration and is only applicable for environments that support events (i.e. Brokered Environments). A lot of these properties can be set at a global level (applicable for each provider class in the ‘provider.classes’ list) or for each individual provider in the ‘provider.classes’ list.

These properties have the notation: **event.<property\_name>[.<provider\_name>]=value** where the **<provider\_name>** is a name of one of the provider names listed in the **provider.classes** property.

**event.frequency**

Events are being sent/produced at a given interval. The event frequency (in seconds) can be set at a top level with this property. If not further specified for a particular provider then this value is used. To turn off events then this property can be set to 0.

**event.startup.delay**

Once a provider is started a delay might be required before events are published. This delay is set here in seconds. If this property is not set or provided then the default will be 5 seconds. If a different delay is required for each provider then this property can be set for each provider individually by adding ".<providerName>" at the end of this property. I.e. event.startup.delay.StudentPersonalProvider=30. This value is in seconds. The value cannot be set to less than 5 seconds. If it is set to less then it will be defaulted to 5 seconds.

**event.maxObjects**

This property indicates what the maximum number of objects per SIF Event message should be. This value is defaulted to 10 if not set. It can be overridden programmatically by an implementer if required. It can also be set at the provider level by adding ".<providerName>" at the end of this property. I.e. event.maxObjects.StudentPersonalProvider=25.

**Note:**

You can add any other properties to this file. They are available to you in each provider class through the getServiceProperties() method.

##### web.xml File

Because the provider is deployed in a web- or application container it requires a certain section in the **web.xml** to tell the application which provider property file to use. The SIF3 Framework has a dedicated Servlet Context that is part of the web.xml to initialise and finalise the provider (refer to section 6 for more details on the configuration of this servlet context). The name of the provider property file can be given to the servlet context with adding the following section to the web.xml:

<context-param>

<param-name>SERVICE\_PROPERTY\_FILE</param-name>

<param-value>**StudentProvider**</param-value>

</context-param>

The **<param-value>** node holds the name of the provider property file without the extension ‘.properties’.

#### The BaseProvider & BaseEventProvider Class

The next step is to decide what SIF Objects you want to provide. You will be required to implement a provider class that extends the **BaseProvider** (no events) or a **BaseEventProvider** (events enabled) class per SIF Object. With this class you wire up your implementation with the framework or in other word you make the framework aware of your SIF Objects, data model etc. It ensures the marshal/unmarshal factories to use to convert the SIF Model Objects to/from XML/JSON are known to the framework.

Internally the framework implements a generic resource that can deal with any object but it will use a factory to call the appropriate Provider Class that will deals with each SIF Object Type. All you need to do is implement the abstract/interface methods enforced by the BaseProvider/BaseEventProvider class and connect these methods to your data store to either read or modify it. No other coding is required.

##### No Events: BaseProvider Class

If you write a provider that doesn’t require events to be sent then you need to implement a class that extends the BaseProvider class (see **systemic.sif3.demo.rest.provider.SchoolInfoProvider** class for an example). Extending the BaseProvider forces you to implement the following Provider Interface methods:

* getMarshaller
* getUnmarshaller
* retrievByPrimaryKey
* createSingle
* updateSingle
* deleteSingle
* retrieve
* createMany
* updateMany
* deleteMany
* shutdown

They are all mostly self-explanatory and the Javadoc for the Provider Interface gives a detailed description what each of these methods is intended for.

##### Events: BaseEventProvider Class

If you write a provider that requires events to be sent then you need to implement a class that extends the BaseEventProvider class which adds a few methods to be implemented to deal with events (see **systemic.sif3.demo.rest.provider.StudentPersonalProvider** class for an example). Extending the BaseEvntProvider forces you to implement the Provider & EventProvider Interface methods. These are the same methods as with the BaseProvider but additionally the following methods are required as well:

* getSIFEvents
* modifyBeforePublishing
* onEventError

They are all mostly self-explanatory and the Javadoc for the EventProvider Interface gives a detailed description what each of these methods is intended for.

#### SIFEventItarator Class

**TBC**

## Security – HTTPS Configuration

There are a few configuration steps required to set-up your consumer and provider to use HTTPS (secured connections).

### Provider

This framework assumes that your provider is running in a web- or application container. To configure HTTPS you must configure your web- or application container to use https. This is generally done in terms of generating a private certificate and import it into an appropriate keystore and/or trusted certificate store. Also if you have a proxy in place, certificates might be managed in that component. In either case the configuration of HTTPS is not in your actual provider but in other components. It is outside the scope of this document to describe how your web-, app-container or your proxy is configured to use/accepts HTTPS.

The provider must only set the two properties called env.connector.url (http) and env.connector.url.secure (https) to appropriate values in the provider’s properties file (see section 5.6.2.1.1 for details).

### Consumer or Provider connection to a Broker

There are a number of steps required to configure your consumer (DIRECT & BROKERED) or provider (BROKERED) to use HTTPS. “Use HTTPS” in this context refers to the connection to your environment provider and/or broker.

Before you do anything you must have the public component of the certificate available to you. All the steps in this section assume you have such a certificate. Java has a tool called ‘keytool’ with which you can perform all necessary operations to import, view, remove, generate, list etc. certificates in your keystore. A good web-site on all of these commands can be found at <http://www.sslshopper.com/article-most-common-java-keytool-keystore-commands.html>. There is also a number of GUI driven tools that help you to perform all of the command with a GUI rather than a command line tool. One of these tools is Portecle (<http://portecle.sourceforge.net/>). Use either the Java command line tool ‘keytool’ or one of the GUI driven tools to perform the following steps to configure your certificate keystore and trusted certificate authority store:

1. Import the public component of the certificate into your keystore. Note you may have to create the keystore first (keystore.jks). Use one of the tools mentioned in the previous section to import the certificate and/or create the keystore.
2. If you are importing a certificate that has been issued with a known certificate authority such as VeriSign, Semantec, Thawte etc. then you are just about done. If you use a certificate from a lesser known issuer or a self-signed certificate then you must also import your certificate into the ‘trusted true store’ (cacerts or cacerts.jks). In that case you may also need/want to create the trusted certificate authority store with one of the above tools.
3. Once you have imported your certificate in applicable certificate stores you must tell your consumer/provider where these stores are located and what the password of these stores is. Open the environment.properties file and set the full path to the keystore (key.store property) and trusted keystore (trust.store property), as well as the password (key.store.password and trust.store.password property) for each store and save the environment.properties file. Note if you have used a certificate from a known certificate authority and have not imported it to your trust store then you can just point the trust.store property to the existing Java trust store located in the JRE’s install directory under <jre>/lib/security/cacerts.
4. Finally ensure that your consumer’s/provider’s property file sets the **env.use.https** property to **true** and that the **env.baseURI** points to the secure endpoint locations such as ***https***://<host>[:port]/.... Note that not only the http changes to https but potentially the port number of the URI as well (see section 5.5.2.1.1 for details about this property file).

Now restart your consumer/provider and all should start using HTTPS instead of HTTP. If your consumer has connected to an environment using HTTP before and now connects to it using HTTPS the environment store and its environment XML will automatically be updated with the new URL endpoints. There is no need to manually change anything in your environment store.

# Deployment

For a server component (i.e. provider) to be deployed successfully the web.xml must have a couple of specific entries. One set of entries are independent of the deployment container while one section depends on the web- or application container you use to deploy the service.

## Servlet Context

The SIF3 Framework has a specific Servlet Context that needs to be added to the web.xml. This servlet context ensures that provider classes are initialised and configured properly at start-up. This includes but is not limited to:

* Reading the provider property file
* Connect to the broker and verify the SIF Environment in case of a Brokered Provider
* Initialise the internal Session Manager
* If events are enabled, starting the event manager

The servlet context also ensures that all resources used by the provider are released once the web- or application container shuts down.

The following two sections **MUST** be added to your web.xml:

<listener>

<listener-class>sif3.infra.rest.web.ProviderServletContext</listener-class>

</listener>

This is the servlet context class for the SIF3 Framework. Now you need to add the section that indicates the provider property file you intend to use with this deployment (see also section 5.6.2.1.2).

<context-param>

<param-name>SERVICE\_PROPERTY\_FILE</param-name>

<param-value>**StudentProvider**</param-value>

</context-param>

The **<param-value>** node holds the name of the provider property file without the extension ‘.properties’.

## Jersey, JAX-RS and other Library considerations

JAX-RS is the Java API for RESTful services which provides support in creating web services according to the REST architectural pattern. Jersey is the reference implementation of JAX-RS from Oracle. The SIF3 framework uses the Jersey implementation to provide RESTful client functionality within the sifInfraREST module. The demo application also uses the Jersey implementation to provide JAX-RS Application (service) functionality. Only one JAX-RS Application class is allowed to be deployed inside a Java application container, this means that if you intend to deploy the demo application to a container that provides its own JAX-RS implementation (such as JBoss), you will need to make some minor changes to the **web-xml** before deploying. The next few sections illustrate how the SIF3 Framework can be used with a couple open-source web- and application containers. A similar approach is suggested for any other web- or application container that is not listed in these sections.

## Tomcat 6.x Deployment

If you deploy into Tomcat 6.x then you can choose to use the Jersey JAX-RS implementation (Tomcat does not have a JAX-RS implementation as part of it). In this case you need to include all 4 Jersey jar files that can be found in the WEB-INF/lib directory of this framework into your project’s WEB-INF/lib directory. You need to add the following section to your web.xml to indicate that you are using the Jersey Servlet to process all REST web-service calls:

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>sif3.infra.rest.resource</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- SIF Direct REST mapping according to SIF REST Direct Spec -->

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/sif3/\*</url-pattern>

</servlet-mapping>

## Tomcat 7.x and above and JBoss Deployment

For application containers that provide support for the servlet 3.0 specification the changes required to remove the Jersey Application class dependency are minimal. The servlet 3.0 specification has been supported since Tomcat 7.0 and JBoss Application Server 6.

The first change is to convert the web.xml to a Servlet 3.0 application descriptor. Remove the following line from the top of the web.xml file:

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

And replace it with the following line:

<web-app xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*

version=*"3.0"*>

Now you add or replace the jersey specific servlet definition with the servlet definition below:

<servlet>

<servlet-name>javax.ws.rs.core.Application</servlet-name>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>javax.ws.rs.core.Application</servlet-name>

<url-pattern>/sif3/\*</url-pattern>

</servlet-mapping>

This will automatically use whatever JAX-RS Application implementation is provided.

The third change is to stop the Jersey libraries from being packaged within the demo application. This is as simple as deleting the jersey-server.jar and jersey-servlet jar files from the war/WEB-INF/lib folder inside the SIF3 Framework project. The jersey-client and jersey-core libraries are still required by the sifInfraRest module but will not cause any conflicts with another JAX-RS implementation as they have utility and interface methods rather than JAX-RS implementations.

### Tomcat

To deploy a provider or consumer to a tomcat environment, you can follow the demo application. Place your configured environment.properties on the classpath and copy your built war file into the webapps folder.

### JBoss (RESTEasy)

To deploy a provider or consumer to a JBoss environment, you can follow the demo application, but you will need to make the changes as outlined in section 6.4. Place your configured environment.properties on the classpath, and deploy your built war or ear file.

## Verify Consumer & Provider Property Files

In section 6.3 and 6.4 the servlet definition has a mapping defined as <url-pattern>/sif3/\*</url-pattern>. You can change that mapping to whatever you require but if you do so, please ensure that the consumer and provider property files are updated to point to the right mapping context. The properties that must be updated are:

* env.baseURI (consumer property file)
* env.connector.url & env.connector.url.secure (provider property file)

Ensure that the last segment of these URLs is updated to reflect the servlet mapping.