SIF3 Framework (Java) - v

Document Revision

**Author:** ,

**Revision:** (draft)

**Published:** Jun 2016

Copyright © 2016,

Table of Contents

[1. Introduction 5](#_Toc452452975)

[1.1. History 5](#_Toc452452976)

[1.2. Document & Framework History 5](#_Toc452452977)

[1.3. Licensing 6](#_Toc452452978)

[1.3.1. SIF3 Framework 6](#_Toc452452979)

[1.4. Why SIF3 Framework? 6](#_Toc452452980)

[2. Assumption & Constraints 6](#_Toc452452981)

[2.1. Java Version 7](#_Toc452452982)

[2.2. Functionality 7](#_Toc452452983)

[2.3. Web-/Application Container Requirements 7](#_Toc452452984)

[3. Structure, Installation & Getting Started 8](#_Toc452452985)

[3.1. Framework Structure 8](#_Toc452452986)

[3.2. Building Components of the Framework 9](#_Toc452452987)

[3.2.1. SIF Data Model 9](#_Toc452452988)

[3.2.2. Consumer Build 9](#_Toc452452989)

[3.2.3. Demo Provider Build 10](#_Toc452452990)

[3.3. Installation & Getting Started 10](#_Toc452452991)

[3.3.1. Installation 10](#_Toc452452992)

[3.3.2. Getting Started - Create your SIF Project 10](#_Toc452452993)

[4. Concepts & Terminology 11](#_Toc452452994)

[5. Framework Classes/Packages and Usage 11](#_Toc452452995)

[5.1. General Process of Developing SIF3 Services 11](#_Toc452452996)

[5.2. Proposed Package Structure 12](#_Toc452452997)

[5.3. Environments 13](#_Toc452452998)

[5.3.1. Environment Store Setup 13](#_Toc452452999)

[5.3.1.1. Environment Template Directory (File System) 13](#_Toc452453000)

[5.3.1.1.1. Environment Template Store Structure 13](#_Toc452453001)

[5.3.1.1.2. Consumer Environment Store Structure 14](#_Toc452453002)

[5.3.1.1.3. Provider Environment Template Store Structure 14](#_Toc452453003)

[5.3.1.2. Environment and Session Store (Database) 16](#_Toc452453004)

[5.3.1.3. Direct Environment Provider Template Store (Database) 16](#_Toc452453005)

[5.3.1.3.1. Default Environment Template Configuration 17](#_Toc452453006)

[5.3.1.4. SQLite DB 17](#_Toc452453007)

[5.4. Data Model 18](#_Toc452453008)

[5.4.1. Data Model POJOs 18](#_Toc452453009)

[5.4.2. Marshal & Unmarshal Factories 18](#_Toc452453010)

[5.4.3. Other Media Type than XML or JSON 19](#_Toc452453011)

[5.5. Building a Consumer 19](#_Toc452453012)

[5.5.1. Constraint 19](#_Toc452453013)

[5.5.2. Components of a Consumer 19](#_Toc452453014)

[5.5.2.1. FIRST CALL: ConsumerLoader Class 19](#_Toc452453015)

[5.5.2.2. The AbstractConsumer & AbstractEventConsumer Classes to wire up the Framework 20](#_Toc452453016)

[5.5.2.2.1. Required Methods of a Consumer 20](#_Toc452453017)

[5.5.2.2.2. Methods of the AbstractConsumer & AbstractEventConsumer 21](#_Toc452453018)

[5.5.2.2.3. Available Consumer CRUD Methods 21](#_Toc452453019)

[5.5.2.2.4. Methods required for DELAYED Responses 27](#_Toc452453020)

[5.5.2.2.5. Additional required Methods for an AbstractEventConsumer 28](#_Toc452453021)

[5.5.2.2.6. Available Method Overrides 29](#_Toc452453022)

[5.5.2.3. Actual Implementation 30](#_Toc452453023)

[5.5.2.4. Consumer Executable Example 30](#_Toc452453024)

[5.5.3. Consumer Events and Delayed Response Processing & Threads 30](#_Toc452453025)

[5.5.3.1. Queue Strategies 31](#_Toc452453026)

[5.5.3.2. Message Readers 31](#_Toc452453027)

[5.5.3.3. Local Workers 32](#_Toc452453028)

[5.5.3.4. Thread Examples 32](#_Toc452453029)

[5.6. Building a Provider 33](#_Toc452453030)

[5.6.1. Constraint 34](#_Toc452453031)

[5.6.2. Components of a Provider 34](#_Toc452453032)

[5.6.2.1. Configure your Provider Environment 34](#_Toc452453033)

[5.6.2.1.1. web.xml File 34](#_Toc452453034)

[5.6.2.2. The BaseProvider & BaseEventProvider Class 35](#_Toc452453035)

[5.6.2.2.1. No Events: BaseProvider Class 35](#_Toc452453036)

[5.6.2.2.2. Events: BaseEventProvider Class 36](#_Toc452453037)

[5.6.2.2.3. Available Method Overrides (BaseEventProvider Class Only) 36](#_Toc452453038)

[5.6.2.3. SIFEventItarator Class 37](#_Toc452453039)

[5.7. Service Paths 37](#_Toc452453040)

[5.7.1. Provider Environment Configuration (Service ACLs) 37](#_Toc452453041)

[5.7.2. The QueryCriteria Class 38](#_Toc452453042)

[5.7.3. Provider Implementation 39](#_Toc452453043)

[5.7.4. Consumer Implementation 39](#_Toc452453044)

[5.8. Query by Example (QBE) 39](#_Toc452453045)

[5.8.1. What is QBE? 40](#_Toc452453046)

[5.8.2. Provider Implementation 40](#_Toc452453047)

[5.8.3. Consumer Implementation 40](#_Toc452453048)

[5.8.4. Query Conditions & Wildcards 41](#_Toc452453049)

[5.9. “Changes Since” Functionality 41](#_Toc452453050)

[5.9.1. What is “Changes Since? 41](#_Toc452453051)

[5.9.2. Provider Implementation 42](#_Toc452453052)

[5.9.2.1. Methods of the ChangesSinceProvider Interface 42](#_Toc452453053)

[5.9.3. Consumer Implementation 43](#_Toc452453054)

[5.10. Custom/External Security Service Integration 43](#_Toc452453055)

[5.10.1. Overview & Motivation 44](#_Toc452453056)

[5.10.2. DIRECT Environment Provider 44](#_Toc452453057)

[5.10.2.1. Step 1: Implement a Security Class – Extend AbstractSecurityService 44](#_Toc452453058)

[5.10.2.2. Step 2: Tell the framework to use the Security Class 45](#_Toc452453059)

[5.10.2.3. When is the External Security Service Used? 46](#_Toc452453060)

[5.10.2.4. SIF3\_APP\_TEMPLATE Table – Bearer Authentication 46](#_Toc452453061)

[5.10.2.5. SIF Express Considerations 46](#_Toc452453062)

[5.10.3. Consumer 47](#_Toc452453063)

[5.10.3.1. Step 1: Implement a Security Class – Extend AbstractSecurityService 47](#_Toc452453064)

[5.10.3.2. Step 2: Tell the framework to use the Security Class 48](#_Toc452453065)

[5.10.4. Brokered Provider 48](#_Toc452453066)

[5.11. Security – HTTPS Configuration 49](#_Toc452453067)

[5.11.1. Provider 49](#_Toc452453068)

[5.11.2. Consumer or Provider connection to a Broker 49](#_Toc452453069)

[5.11.3. Client Certificate Exchange – Mutual Authentication 50](#_Toc452453070)

[5.12. Provider Request/Response Auditing 50](#_Toc452453071)

[5.12.1. Auditing Class & Enable/Disable Auditing 51](#_Toc452453072)

[5.12.2. Audit Record 51](#_Toc452453073)

[5.13. Hibernate Properties Injection 51](#_Toc452453074)

[5.14. Compression (GZIP) 52](#_Toc452453075)

[5.14.1. Enable Compression on Consumer 52](#_Toc452453076)

[5.14.2. Enable Compression on Provider 52](#_Toc452453077)

[5.14.2.1. Enable on Proxy 52](#_Toc452453078)

[5.14.2.2. Enable in Web- or Application Container 52](#_Toc452453079)

[5.14.2.3. Enable in Framework 53](#_Toc452453080)

[5.14.2.4. Events 53](#_Toc452453081)

[6. Deployment 54](#_Toc452453082)

[6.1. Servlet 3.0 Requirement for Providers 54](#_Toc452453083)

[6.2. Servlet Context 54](#_Toc452453084)

[6.3. Java classpath Configuration 55](#_Toc452453085)

[6.4. Customise Property Files to your environment 55](#_Toc452453086)

[6.4.1. Customise sif3infra.hibernate.xml 55](#_Toc452453087)

[6.4.2. Customise environment.properties 55](#_Toc452453088)

[6.4.3. Customise consumer properties file 55](#_Toc452453089)

[6.4.3.1. Properties for Demo only 56](#_Toc452453090)

[6.4.4. Customise provider property file 56](#_Toc452453091)

[6.4.4.1. Properties for Demo only 56](#_Toc452453092)

[6.4.5. Customise log4j.properties file (optional) 56](#_Toc452453093)

[6.5. Jersey, JAX-RS and other Library considerations 57](#_Toc452453094)

[6.6. Tomcat 6 or below, Jetty 7 or below 57](#_Toc452453095)

[6.7. Tomcat 7 and above or Jetty 8 and above Deployments 57](#_Toc452453096)

[6.8. JBoss AS 6 and above Deployments 58](#_Toc452453097)

[6.9. Verify Consumer & Provider Property Files 58](#_Toc452453098)

[7. Appendix A – Environment Properties File 59](#_Toc452453099)

[8. Appendix B – Consumer’s Property File 60](#_Toc452453100)

[9. Appendix C – Provider’s Property File 64](#_Toc452453101)

# Introduction

## History

The SIF3 Framework is a basic Java Framework intended to help developing SIF 3.x 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). In late 2014 SIF xPress capabilities were added. Throughout 2015 more capabilities mainly relating to various query mechanisms have been added. In December 2015 (v0.7.0) DELAYED response functionality has been introduced to the framework. DELAYED I/O is only supported where the environment provider supports that functionality (i.e. BROKERED environments). Other functionality is added incrementally and can be seen from the Document & Framework History below.

## Document & Framework History

|  |  |  |  |
| --- | --- | --- | --- |
| **Document**  **Version** | **Date** | **Author** | **Comments** |
| 0.1 | Feb 2014 | J. Huber | First main version of framework |
| 0.2 | May 2014 | J. Huber | Extended to Brokered Functionality including Events. |
| 0.3 | August 2014 | J. Huber | * Flexible Environment Template Manager for Direct Environment Provider. * SIF Express extensions. * Minor bug fixes. |
| 0.4 | December 2014 | J. Huber | * Removed SIF AU Data Model from Framework * Added JSON support * Better error logging in many areas. |
| 0.5 | March 2015 | J. Huber &  B. Carter | * Added support for SIF 3.x Service Path (see section 5.7). * Added External Security Service functionality (see section 5.10.2) for DIRECT providers. * Open up mime types for Object Services. |
| 0.6 | March 2015 | J. Huber & Ben Carter | * Auditing capability for Object Providers. * Providers require web-/application container with servlet 3.0 support. * Support for custom HTTP headers & URL parameters. |
| 0.7 | July 2015 | Joerg Huber | * Added Compression Capability. * Externalised Infrastructure Data Model. * Added External Security Service functionality (see section 5.10.3) for consumers. * Added DELAYED request/response functionality. |
| 0.8 | January 2016 | Joerg Huber | * Added HTTP HEAD functionality * Added Support for “ChangesSince” for Providers |
| 0.9 | March 2016 | Joerg Huber | * Added flexibility to some “Changes Since” methods for the Provider. * Allow Hibernate Properties injection. See section 5.13 for details. |
| 0.10 | June 2016 | Joerg Huber | * Added additional parameter to all methods of all provider style classes to enable custom HTTP headers to be set for response. |

## 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 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 SIF 3.x 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 bug reports 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 SIF 3.x 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).

## Java Version

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. Since v0.9.0 the JDK version must be 1.7 or above. From v0.9.0 onwards the framework will no longer be compiled or tested on Java versions older than 1.7. It has only undergone limited testing with Java 8.

## Functionality

As of June 2016 (beta version 0.10.0 of the framework) the SIF3 Framework includes the following functionality:

* Uses SIF 3.1 Infrastructure (current and latest version)
* **Immediate** Request/Response in a **direct & brokered** environment
* **Delayed** Request/Response in a **brokered** environment.
* 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.
* Providers and Consumers can operate in a DIRECT and BROKERED environment with above functionality and constraints.
* REST
* XML & JSON (Note: JSON is limited to Object and Infrastructure services. JSON is not yet supported for Events according to the SIF Specification 3.0.1 or 3.1).
* Support for Service Path as specified in the SIF 3.x Specification for consumers and providers.
* Support for External Security Service integration (See **warning** at the start of section 5.10)
* Support for ‘Request & Response Auditing’ for Object Providers (see section 5.12 for details).
* Support for “Query By Example” (SIF 3.2 draft functionality; see section 5.8 for details).
* Support for “Changes Since” for Providers (SIF 3.2 draft functionality; see section 5.9 for details)

Notable functionality that is not yet supported includes:

* No Support for Dynamic Queries
* No Support for xQueries & query templates (see SIF 3.x Specification for details on this functionality).
* No Support for Functional Services
* No Support for Utility Services.

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

## Web-/Application Container Requirements

Object providers are ‘servers’ in the traditional sense and therefore must be deployed into a web- or application container. The SIF3 Framework does not use any enterprise beans and therefore does not require an application server. It can be packaged as a ‘web application’ and deployed in a web container such as Tomcat if desired.

Since v0.6.0 the framework requires a web- or application container that supports the **servlet 3.0 specification**. The list below shows the most common open-source web- and application container and the minimal version that supports the servlet 3.0 specification:

* Tomcat 7 and above
* Jetty 8 and above
* Glassfish 3 and above
* JBoss 6 and above

For all commercial web- and application containers please refer to the product’s specification to ensure which version supports the servlet 3.0 specification.

# Structure, Installation & Getting Started

This section outlines the high level structure of the SIF3 Framework source code as well as the components you require from the SIF3 Framework to setup your SIF project.

## Framework Structure

This section shortly lists the main structure and how code, libraries, configuration files etc. are structured and organised within the framework. As with all 3rd party frameworks and libraries you do not need the actual source code but a set of framework libraries, config files and other dependent 3rd party libraries. The SIF3 Framework is no exception.

Before you can install the SIF3 Framework you need to download it from <https://github.com/Access4Learning/sif3-framework-java> or a locale (AU, UK, US) specific fork of the framework. 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.
* **sif3InfraModel/src**: POJOs (generated by JAXB) and Marshallers for Infrastructure Data Model classes. Classes in this package should not be used or exposed to the higher levels of the framework. They are only used internally.
* **sifIsif3InfraCommon/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.

**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. To use the SIF3 Framework in your own SIF Project it is expected that you copy the content of this directory into your SIF Project.

**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.

**Java 3rd Party Libraries Directory**

The “**lib**” directory has the required java 3rd party libraries required by the SIF3 Framework. If you develop a consumer you may need some or all of these libraries in your classpath. For your own project you may not need the lib/ant-contrib-1.0b3/ant-contrib-1.0b3.jar as it is only used for the ant build script of the framework but not for the runtime of any consumer or provider. Further the lib/jdbc holds a number of JDBC drivers. Only copy the one you require into your project or add your very own jdbc driver library.

**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 build the framework libraries yourself. 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.

**Release Directory**

This directory holds a pre-built version of the latest release of the framework’s libraries. You can take and include them in your project. There is no need for you to build them if you use these libraries.

## Building Components of the Framework

If you have modified the source of the framework 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.

### SIF Data Model

The test and demo source code directories depend on the SIF AU 1.3 Data Model. For that reason the sifDataModel\_au1.3.jar is part of the framework. This jar has been taken/built from the GitHub repository <https://github.com/nsip/SIF3DMGenerator-Java>. This project is responsible to build Data Model libraries based on XSD files and add appropriate classes to allow the SIF3 Framework to use these data model libraries out of the box.

### Consumer Build

As part of the SIF3 Framework download you have the required libraries already pre-built in the **release** directory but 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 built libraries will be placed into the **build/dist** directory. The name of the jar files that will be built are:

* **sif<infra-version>Common-<framework-version>.jar**
* **sif<infra-version>Infra-model-< framework-version>.jar**
* **sif<infra-version>Infra-common-< framework-version>.jar**
* **sif<infra-version>Infra-rest-< framework-version>.jar**
* sif3Demo-<version>.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. You will also need a Data Model library. For details refer to section 3.2.1. 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 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.

## Installation & Getting Started

If you start your own SIF Project and intend to use the SIF3 Framework there are a few steps you need to follow to set your project up. These steps mainly include a simple installation of some database scripts as well as copy appropriate areas/components of the SIF3 Framework into your own SIF Project. The final step is configuring you project to work as a SIF service (consumer or provider). This section states these steps shortly but for more details subsequent sections of this developer’s guide will be required.

### Installation

The SIF3 Framework requires a database for various tasks but mainly to manage your SIF3 Session that is core to all SIF3 Services. The first step for you is to select a database of your choice (Oracle, MySQL, Postgres etc.) and create a dedicated schema on that database. Once the schema is created you run the following steps:

1. Run the DB/DDL/SIF3InfrastructureERM\_DDL\_<db\_product>.sql script. This will create all required tables, sequences, indexes etc.
2. Run the DB/DLL/Initial\_Inserts.sql script. This will perform initial inserts to make at least one template known to the system.

Now the database is ready to be used by your SIF Project. There is one more step to tell the framework how to connect to the DB but this step is detailed in the next section under ‘Configuration’. More detailed information can also be found in section 5.3.1.2.

### Getting Started - Create your SIF Project

When you want to develop SIF services (consumers and/or providers) you need to create a SIF Project. This can either be a simple Java project if you only write consumers or it can be a web-project if you intend to develop data object providers. Either way you need certain components from the SIF3 Framework. It is suggested that you copy the following components from the SIF3 Framework into your SIF project:

* **SIF3 Framework Libraries**: Copy the following libraries from the framework’s “**release**” directory into your SIF Project:
* **sif<infra-version>Common-<framework-version>.jar**
* **sif<infra-version>Infra-model-< framework-version>.jar**
* **sif<infra-version>Infra-common-< framework-version>.jar**
* **sif<infra-version>Infra-rest-< framework-version>.jar**
* **Data Model Library**: Note that the sifDataModel\_au1.3.jar is currently included in the war/WEB-INF/lib directory. This jar file is taken from the SIF3DMGenerator GitHub project (see section 3.2.1 for details). If you need another data model you need to replace this one with your required data model.
* **Dependent 3rd party libraries**: The SIF3 Framework is dependent on a number of 3rd party libraries such as log4j, commons etc. If you develop SIF services as part of your project using the SIF3 Framework, your own project may already hold some of the required 3rd party library. Please refer to the “**lib**” and its sub-directories as well as the “**war/WEB-INF/lib**” directory. Copy all 3rd party libraries from there, that are not yet part of your project, into you project. You don’t need the lib/ant-contrib-1.0b3/ant-contrib-1.0b3.jar as it is only used for the ant build script of the framework but not for the runtime of any consumer or provider. Further the lib/jdbc holds a number of JDBC drivers. Only copy the one you require into your project or add your very own jdbc driver library. Please refer to section 6 for additional information about libraries and if they need to be included in your project or not.
* **Configuration**: Finally it is suggested that you copy the entire “**config**” and its sub-directories into a location of your choice within your project. Note you do not need to call the directory ‘config”’ within your project. You can name it what you like.
  + To make the framework aware of your database as outlined in previous section you need to add/change the appropriate JDBC URL and credentials in the ‘config/hibernate/sif3infra.hibernate.cfg.xml file.
  + You also need to configure the ‘config/environments.properties’ file to point to your very own config directory.

Now your project is mostly configured to use the SIF3 Framework as part of your SIF project. The reminder of this developer’s guide goes into much more details how you use the framework to achieve particular tasks such as writing consumers and data object providers. Also, you need to configure and customise the environment setup of your consumer or data object provider. Details for that configuration and customisation can be found in section 5.3. It is highly recommended that you read carefully through that section to ensure you have configured and customised your consumer/provider correctly. Finally to deploy your consumer/provider it is suggested to read section 6.

# Concepts & Terminology

If you are new the SIF3 Service development then you will need to get used to some terms common in the SIF 3 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)

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.
* 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 also section 3.2.1).

## 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 8, provider see section 9).

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

### Environment Store Setup

It is important to note that the current version 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 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/3.1">

<solutionId>testing</solutionId>

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

<consumerName></consumerName>

<applicationInfo>

**<applicationKey></applicationKey>**

<supportedInfrastructureVersion>3.1</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 in blue are mandatory as per specification for SIF 3.1. 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 8). 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 5.3.1.1.2 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 9 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 current version of the framework requires a database to store runtime as well as some configuration related data. There are two main scripts that must be run to setup the framework’s database:

* SIF3InfrastructureERM\_DDL\_<dbType>.sql: This script will create all required tables, sequences and indexes etc. for the database product of your choice.
* Initial\_Inserts.sql: This script will populate some of the tables with default data.

You may have already done this step as part of section 3.3.1. If so you do not need to do the above steps again. You can use whatever database product you like (MySQL, Oracle, SQLServer etc). There are various DDL scripts in this framework under ‘DB/DLL’ to create appropriate tables for the DB product of your choice. 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. The insert script is generic and should run for most DB products.

Once you have run the DDL and Insert 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 default configuration is ready to go. You may want to changes some of the configuration for the Direct Providers. Please refer to the next section for details.

#### Direct Environment Provider Template Store (Database)

As part of the scripts run in section 5.3.1.2 you will be configured for a basic default environment template for a direct environment provider. The Direct Environment Provider implementation can deal with any number of environment templates. The following two tables configure the available templates for a direct environment provider:

* **SIF3\_ENV\_TEMPLATE**: Lists the file names of available templates in the ‘**provider/template/direct**’ directory (see section 5.3.1.1.3 for details).
* **SIF3\_APP\_TEMPLATE**: This table links the templates listed in the SIF3\_ENV\_TEMPLATE table to a solution, application etc.

There is a ‘hierarchical’ algorithm how templates are found in the SIF3\_APP\_TEMPLATE table. The following columns can be used to identify a template for a client: SOLUTION\_ID, APPLICATION\_KEY, USER\_TOKEN and INSTANCE\_ID. The APPLICATION\_KEY is the only mandatory value. The following algorithm applies:

1. The lowest level of the hierarchy is when the client (consumer) uses all of the following values: SOLUTION\_ID, APPLICATION\_KEY, USER\_TOKEN and INSTANCE\_ID. In this case the INSTANCE\_ID is ignored in determining the actual entry in the database because according to SIF 3.x the INSTANCE\_ID is only used for cases where two different devices access the same environment. The environment composition for both instances is assumed to be the same and therefore the only columns to be used for the determination of the actual environment are SOLUTION\_ID, APPLICATION\_KEY and USER\_TOKEN. Note the client will still get its distinct environment instance with a dedicated Session Token and Environment ID but the ACLs and all other properties of the environment will be identical.
2. If there is no template listed for this combination of step 1 then the next level is attempted: SOLUTION\_ID, APPLICATION\_KEY.
3. It must be noted that the SOLUTION\_ID is optional. It is possible that the APPLICATION\_KEY is the only value provided by a client and that there is a template for that. In the database table it may have a SOLUTION\_ID set but the consumer doesn't provide it and only provides the APPLICATION\_KEY. In this case the SOLUTION\_ID in the table will be returned to the client as part of the environment ‘create’ response. Note if neither the client nor the database holds a solution id then the solution id in the actual template XML file is used if there is one set.

All of the above may sound complex but at the end it is an environment administrator that will configure this and it is something that is configured once and then will hardly ever be touched again.

##### Default Environment Template Configuration

As part of the initial installation a default environment template configuration is being provided. The default template is called devLocal.xml (see SIF3\_ENV\_TEMPLATE) and it is linked to the solution ‘test’ for the application with the key of ‘TestSIS’ with the password ‘Password1’ (see SIF3\_APP\_TEMPLATE). With this configuration you will be able to start your provider immediately and connect to it. For your real testing you may want to change that to a different template or add more to the two tables.

#### SQLite DB

As part of this framework a SQLite database is provided. This database can be used immediately and won’t require you to have a database or run any database install scripts. The database can be found in the directory ‘DB/Data’ and has the name **SIF3Infra.sqliteDB**. This database is only provided, so that you can run the demos or do some basic prototyping, pilot work etc. immediately and don’t need to install another database or run the DB scripts listed in section 5.3.1.2. For the framework to use the SQLite database you must update the ‘**config/hibernate/sif3infra.hibernate.cfg.xml**’ and uncomment the section with the “SQLite Embedded” connection detail block. Change the property “connection.url” to the correct path where your SIF3Infra.sqliteDB file resides. Also ensure that the **sqlite-jdbc-3.7.2.jar** is in your classpath (consumer) or war file (provider). It can be found in the ‘lib/jdbc’ directory of the framework.

**IMPORTANT NOTE**

While you can use the provided SQLite database to run demo code and/or do some prototyping it is **not intended or recommended** to be used in a production environment. SQLite is a basic file based database and does not support many features a proper database offers (i.e. proper row/table locking).

## 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 Specification1)
* A good starting point to find the SIF AU 1.3 (Australia) or SIF NA 3.2 (North America) Data Model that can be used is the GitHub repository <https://github.com/nsip/SIF3DMGenerator-Java>. The demo classes in the SIF3 Framework use the SIF AU 1.3 data model from that GitHub repository.

1) The SIF3 Framework can also deal with other media types for Object Services, assuming the concrete implementation of the marshal and unmarshal class deals with the media type. All infrastructure services will only deal with XML/JSON, though.

### 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 SIF 3.x 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. The GitHub repository <https://github.com/nsip/SIF3DMGenerator-Java> is such an example for Java. The demo classes in the SIF3 Framework use that library. 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 unmarshal factory to use so that it can do the required serialisation on the wire. See section 5.5.2.2 and 5.6.2.2 for details.

The data model classes from the SIFDataModelGenerator GitHub repository have the marshal and unmarshal interfaces already implemented. They are part of the data model’s jar file (i.e. the **sif.dd.au30.conversion.DataModelMarshalFactory** class in the sifDatamodel\_au1.3.jar). If you find the need to write your own marshaller and/or unmarshaller you must implement the following interfaces:

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

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

### Other Media Type than XML or JSON

The SIF3 Framework allows other media types than XML/JSON to be used for all Object Services. An Object Service is a consumer or provider that deals with a **Data Model** (i.e. StudentPersonal Service). The SIF 3.x Infrastructure Services that are fully encapsulated with the SIF3 Framework (i.e. Environment Service, Queue Service, Subscription Service etc.) are limited to XML/JSON only.

If the need arises to ‘transport’ another Object type, such as a CSV Content for example, over the wire then an appropriate CSV Marshaller and/or Unmarshaller must be written that can deal with that format. The CSV Consumer/Provider must be linked with that marshaller/unmarshaller. Section 5.5.2.2 (consumer) and 5.6.2.2 (provider) illustrates how a service is linked with a data model.

**Examples:**

The **systemic.sif3.demo.rest.conversion** package has an example of a CSV Marshaller/Unmarshaller. The class **systemic.sif3.demo.rest.provider.CSVStudentProvider** and **systemic.sif3.demo.rest.consumer.CSVStudentConsumer** illustrate how this CSV marshaller and unmarshaller are linked with the appropriate object service (consumer or provider).

## 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 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. An example can be found 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. Refer to section 8 for a list of all consumer properties that can be used to configure the composition and behaviour of a consumer.
* 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

The next sections illustrate the classes to use and considerations to take into account for building a consumer.

#### 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:

##### Required Methods of a Consumer

Because a consumer class must extend the AbstarctConsumer or AbstractEventConsumer class the developer is required to implement the methods listed below for the reasons given:

**Constructor**

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

**getSingleObjectClassInfo()**

* 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()**

* 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()**

Tells the framework which marshal factory implementation to use to convert your data model object to XML and/or JSON.

**getUnmarshaller()**

Tells the framework which unmarshal factory implementation to use to convert a XML and/or JSON string to your data model object.

**shutdown()**

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

##### Methods of the AbstractConsumer & AbstractEventConsumer

When a developer implements a consumer that extends the AbstractConsumer or AbstractEventConsumer class then the following methods are immediately available for that consumer:

* All CRUD Operations for a particular Data Object (i.e. StudentPersonal) as specified by the SIF Specification (see section 5.5.2.2.3).
* A number of methods that can be overridden for specific customisation (see section 5.5.2.2.6)

##### Available Consumer CRUD Methods

Each available CRUD method of the consumer is shortly described in this section. Additional details can be found in the Javadoc of the AbstractConsumer or Consumer interface class.

**Method Name:** **createSingle**

**Description:**

This method creates a given object in the specified zones and contexts.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| data | No | The object that shall be created. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the object is created in the default zone and context |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each created object in a given zone/context combination.

**Method Name:** **updateSingle**

**Description:**

This method updates the object given by its resourceID in the specified zones and contexts, assuming that the object has the same resourceID in all of the zones & contexts.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| data | No | The actual object (i.e. Student). It holds the values of the object that need to be updated. It can either hold the full object or only parts of the object that needs updating. |  |
| resourceID | No | The Id (i.e. RefId, UUID, etc.) of the object to be updated. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the object is updated in the default zone and context |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each updated object in a given zone/context combination.

**Method Name:** **deleteSingle**

**Description:**

This method deletes the object given by its resourceID in the specified zones and contexts, assuming that the object has the same resourceID in all of the zones & contexts.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| resourceID | No | The Id (i.e. RefId, UUID, etc.) of the object to be deleted. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the object is deleted in the default zone and context |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each deleted object in a given zone/context combination.

**Method Name:** **createMany**

**Description:**

This method will create many objects in one call. The 'data' parameter is a collection-style object that is defined in the data model (i.e. StudentPersonals which is a collection of StudentPersonal).

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| data | No | The 'collection' object. Each object in that collection will be created. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are created in the default zone and context |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each set of objects created in a given zone/context combination. A particular response is a list of response statuses, a status per object, as defined by the SIF Specification.

**Method Name:** **updateMany**

**Description:**

This method will update many objects in one call. The 'data' parameter is a collection-style object that is defined in the data model (i.e. StudentPersonals which is a collection of StudentPersonal). Each object in the collection can either hold the full object or only parts of the object that needs updating.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| data | No | The 'collection' object. Each object in that collection will be updated. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are updated in the default zone and context |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each set of objects updated in a given zone/context combination. A particular response is a list of response statuses, a status per object, as defined by the SIF Specification.

**Method Name:** **deleteMany**

**Description:**

This method removes all objects in the resourceIDs list in one call.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| resourceIDs | No | A list of resourceIdx for the objects to be removed. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are deleted in the default zone and context |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each set of resourceIDs deleted in a given zone/context combination. A particular response is a list of response statuses, a status per resource, as defined by the SIF Specification.

**Method Name:** **retrievByPrimaryKey**

**Description:**

This method returns an object with the given resourceId from all zone/context combinations given by the call.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| resourceID | No | The Id of the object to be returned. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the resource is retrieved from the default zone and context. |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each resourceID retrieved in a given zone/context combination.

**Method Name:** **retrieve**

**Description:**

This method returns any number of objects. This is achieved in terms of 'paging' through the list of objects. The consumer is expected to provide paging information to tell the provider which objects in the list shall be returned. The first page has the number 0.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| pagingInfo | Yes | Page information to be set for the provider to determine which results to return. | If null then it is assumed that the consumer wants all objects. |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are retrieved from the default zone and context. |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| queryIntention | Yes | Indicating what the intention of the query and follow-up queries is. | Can be set to null which will default to 'ONE-OFF'. |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each zone/context combination. The data in the response is a ‘collection-type’ style object as defined in the data model (i.e. StudentPersonals).

**Method Name:** **retrieveByServicePath**

**Description:**

This method returns any number of objects based on the implied query of the service path or in case of this method the queryCriteria. Section 5.7.2 & 5.7.4 has a detailed description on how to use Service Path functionality in a consumer.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| queryCriteria | No | The query based on the intended service path. See section 5.7.2 for details. |  |
| pagingInfo | Yes | Page information to be set for the provider to determine which results to return. | If null then it is assumed that the consumer wants all objects. |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are retrieved from the default zone and context. |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| queryIntention | Yes | Indicating what the intention of the query and follow-up queries is. | Can be set to null which will default to 'ONE-OFF'. |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each zone/context combination. The data in the response is a ‘collection-type’ style object as defined in the data model (i.e. StudentPersonals).

**Method Name:** **retrieveByQBE**

**Description:**

This method returns a list of objects based on example object given to the method. Section 5.8 has a detailed description on how to use QBE functionality. This retrieval method supports all the standard parameters as any other ‘retrieve’ method such as paginInfo, zoneCtxLists etc.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| exampleObject | No | The sample object that shall be used as the query criteria. See section 5.8.4 for details about the sample object and how a query is build based on it. |  |
| pagingInfo | Yes | Page information to be set for the provider to determine which results to return. | If null then it is assumed that the consumer wants all objects. |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are retrieved from the default zone and context. |
| requestType | Yes | Indicating if the request is synchronous (IMMEDIATE) or if it shall be shall executed asynchronously (DELAYED). | **If set to DELAYED then see section 5.5.2.2.4.** |
| queryIntention | Yes | Indicating what the intention of the query and follow-up queries is. | Can be set to null which will default to 'ONE-OFF'. |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

A list of responses with one response for each zone/context combination. The data in the response is a ‘collection-type’ style object of the given ‘exampleObject’ parameter as defined in the data model. I.e. if the ‘exampleObject’ is of type StudentPersonal (singular form) then the returned object is of type StudentPersonals (plural form or collection type).

**Overloaded Methods**

All methods listed in this section are overloaded. The last parameter of each method is called ‘customParameters’ and is rarely used. Instead of a developer having the need to always set this parameter to null every method listed in this section has an overloaded version that does not have this parameter.

For example the overloaded method of

**retrievByPrimaryKey( String resourceID,**

**List<ZoneContextInfo> zoneCtxList,**

**CustomParameters customParameters)**

is

**retrievByPrimaryKey( String resourceID,**

**List<ZoneContextInfo> zoneCtxList)**

Note the missing parameter ‘customParameters’ in the second version. In this case the ‘customParamters’ will be assume to be null.

**Method Name:** **getServiceInfo**

**Description:**

This method will invoke the REST HEAD call to retrieve some information about a service. It will not return a payload as per HTTP Specification of the HEAD method. Besides a status code and an optional status message only a set of HTTP Header properties will be set in the returned responses. These HTTP Header properties can be retrieved as part of the returned response object (response.getHdrProperties()).

Because this method almost mirrors the retrieve() for the object service all parameters that would make up the retrieve() method are supported. The exception is the requestType and queryIntention parameter that are allowed in the retrieve() method. They do not make any sense for this method and are therefore omitted.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| pagingInfo | Yes | Page information to be set for the provider to determine which results to return. Note while this parameter might be set the provider may not use them since this method won’t return a payload. It is up to the implementation on the provider if the pagingInfo parameter is used to return some information about the potential result set. |  |
| zoneCtxList | Yes | A list of Zone/Context pairs. If either the zone and/or the context is null then the default zone and/or context is assumed. In this case the corresponding matrix parameter in the final URI is not set.  If this list is not empty then the action is performed for all Zone/Contexts pairs listed. | If null then the objects are retrieved from the default zone and context. |
| customParameters | Yes | Custom HTTP Header fields and Custom URL Query parameters that will be added to the request. | If null then no custom HTTP or Query parameters are added. |

**Return:**

Responses corresponding to the zoneCtxList parameter. Each response does only hold a status code, an optional status message and a list of HTTP header values. All other properties of the returned response object might be null. The HTTP header values can be retrieved with the response. getHdrProperties() method.

##### Methods required for DELAYED Responses

The methods listed in this section are required for the processing of DELAYED responses. If a consumer does not make use of DELAYED request/response each of the methods listed in this section can simply be “nulled out” in the consumer. The “processDelayed” methods are called by the framework each time a response message is retrieved from a message queue that is maintained on the environment provider (i.e. SIF Broker). For the configuration and behaviour of delayed message queues and processing please refer to 8 where the properties of DELAYED queues are described in details.

**Method Name:** **processDelayedCreateMany**

**Description:**

This method is called when a DELAYED response to a “CREATE” request is retrieved from the consumer’s message queue.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| statusList | No | A list of statuses for the original CREATE request. This list holds a status for each objects created in a given zone and context, as defined by the SIF Specification. |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

**Method Name:** **processDelayedUpdateMany**

**Description:**

This method is called when a DELAYED response to an “UPDATE” request is retrieved from the consumer’s message queue.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| statusList | No | A list of statuses for the original UPDATE request. This list holds a status for each objects updated in a given zone and context, as defined by the SIF Specification. |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

**Method Name:** **processDelayedDeleteMany**

**Description:**

This method is called when a DELAYED response to a “DELETE” request is retrieved from the consumer’s message queue.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| statusList | No | A list of statuses for the original DELETE request. This list holds a status for each objects updated in a given zone and context, as defined by the SIF Specification. |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

**Method Name:** **processDelayedQuery**

**Description:**

This method is called when a DELAYED response to a “QUERY” request (paged or not-paged) is retrieved from the consumer’s message queue. If a “Query By Example” style query was requested by the consumer then the DELAYED response to such a QBE is also retrieved by this method.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| dataObject | No | The data is a ‘collection-type’ style object as defined in the data model (i.e. StudentPersonals). |
| pagingInfo | No | Paging information. Can be used to interrogate which page the result relates to and how many objects there are in the ‘dataObject’ parameter. |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

**Method Name:** **processDelayedServicePath**

**Description:**

This method is called when a DELAYED response to a “SERVICEPATH QUERY” request (paged or not-paged) is retrieved from the consumer’s message queue.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| dataObject | No | The data is a ‘collection-type’ style object as defined in the data model (i.e. StudentPersonals). |
| queryCriteria | No | The original ‘servicePath’ criteria as use by the consumer when the request was issued. It might be useful to the consumer to know to which criteria/servicepath the original request related to. The actual servicepath notation as a String, can also be retrieved from the ‘receipt’ parameter. |
| pagingInfo | No | Paging information. Can be used to interrogate which page the result relates to and how many objects there are in the ‘dataObject’ parameter. |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

**Method Name:** **processDelayedError**

**Description:**

This method is called when an ERROR to a DELAYED request is retrieved from the consumer’s message queue.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Nullable** | **Description** |
| error | No | The error object with the error details. The ‘receipt’ parameter needs to be interrogated to determine the details of the original request to determine what the error relates to (service name, requested action etc.). |
| receipt | No | Metadata information about the response relating to the original request. This includes but is not limited to the zoneId, contextId, original request ID etc. as well as all HTTP Headers of the response. |

##### Additional required Methods for an AbstractEventConsumer

**processEvent ()**

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.
* **metadata:** Metadata about the event. Typically it may hold the generatorId if provided by the publisher of the event. In future more properties might be added to this parameter.
* **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 ()**

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.

}

##### Available Method Overrides

The SIF3 Framework uses the consumer’s property file to configure many HTTP Header values required by the SIF 3.x specification. The property file is also used to drive many default behaviours of the consumers. There are situations where the default behaviour/values should be ‘overruled’ by a particular object consumer while others should use the default behaviour/values. One such case could be where all consumers deal with the standard SIF Data Model, which supports XML and JSON, but one consumer must deal with a CSV Content. The default behaviour of the Framework is to use the media type listed in the consumer’s property file which can only be XML or JSON. To override that for the CSV Consumer the following method(s) in the CSV Consumer class must be overridden:

@Override

public MediaType **getRequestMediaType**()

{

return getMarshaller().getDefault(); //Marshalling request

}

@Override

public MediaType **getResponseMediaType**()

{

return getUnmarshaller().getDefault();//Unmarshalling response

}

The getMarshaller() or getUnmarshaller() methods would return the CSV Marshal/Unmarshal Factory class a developer would have implemented. The SIF3 Framework will use the media type returned by these methods rather than what is set in the consumer’s property file.

The list below shows all methods that can override values/behaviours of the consumer’s property file. The name of the methods is mostly self-explanatory. Also refer to the Javadoc of the AbstractConsumer for each method.

* getGeneratorID()
* getRequestMediaType()
* getResponseMediaType()
* getMustUseAdvisory()
* getCompressionEnabled()

**Experimental** (not part of the SIF 3.0.1 specification)

* getApplicationKey()
* getAuthenticatedUser()

#### 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 5.5.2.1 for details.

#### Consumer Executable Example

There is an example of a Consumer Executable Skeleton in the demo/src directory. The class **systemic.sif3.demo.rest.consumer.StudentConsumerService** illustrates how a standalone consumer service could be implemented. It shows how the startup, shutdown etc. are used in the correct way. Use CTRL-C to shut down the consumer. There is a script in script/service directory to start the consumer (start\_StudentConsumer.bat). The second parameter in that script is the name consumer property file that shall be used with the example consumer. Change it to whatever property file you wish to use.

**Note 1:**

The script to start the StudentConsumerService uses the libraries in the “release” directory. If you want to use the latest libraries you may have changed and built then change the “startConsumer.bat” to use the libraries of the build/dist directory instead (see EXE\_JAR\_PATH variable). Also ensure that property files are configured correctly (see section 6.4.2, 6.4.3 and section 8 for details).

**Note 2:**

This example consumer is not really doing much until you connect it to a broker that may have a provider that sends events. In that case it might start processing events but the default behaviour is just to start and sit there.

### Consumer Events and Delayed Response Processing & Threads

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

* SIF 3 allows concurrent readers on a SIF Queues.
* Increase performance of message 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 Local Workers

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

#### Queue Strategies

Currently the only supported queue strategy by the framework is called ADAPTER\_LEVEL. This queue strategy is set with the ‘events.queue.strategy’ (events) or ‘delayed.queue.strategy’ (delayed responses) property in the consumer’s property file. Future extensions to the framework may offer two more strategies. They could be 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 message queue for events and one message queue for delayed responses per environment. All events or delayed responses of all zones will be directed through the corresponding 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)

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)

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 SIF 3 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

SIF 3 allows multiple concurrent reader threads on each message queue. This is new in SIF 3 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‘ or ‘delayed.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!

#### Local Workers

The final level of thread creation is scoped to the actual consumer class that implements the event or delayed response 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 ‘thread’ related properties in the consumer’s property file

**Example 1: Queue Strategy = ADAPTER\_LEVEL**

The image depicts an example of the simplest queue strategy (ADAPTER\_LEVEL). See also section 5.5.3.1 for details. It shows 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, JBoss and to some extend with Jetty. 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

This section discusses the components that need to be considered when developing a provider. As of May 2014 the provider supports event creation if connected to a brokered environment.

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

* Configure your Provider Environment. This framework deals with DIRECT environments (provider is also a basic environment provider) and BROKERED environments (provider connects to a SIF3 Broker). The properties that can be set/used to configure the composition as well as the behaviour of your provider are listed in section 9.
* 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. See section 5.6.2.2 for details.
* Implements **SIFEventIterator** classes for objects that need to be sent as events. See section 5.6.2.3 for details.
* Deployment.

The following sections will illustrate how the various classes and methods of a provider are used and configured.

#### Configure your Provider Environment

##### web.xml File

A provider has a property file that tells the application what a provider consists of, what environments it deals with etc. Section 9 documents all properties that can be used in the provider’s properties file. Once that file is configured the developer must make the provider aware which property file to use. This is done by adding a particular section to the provider’s **web.xml**. The SIF3 Framework has a dedicated Servlet Context that is part of the web.xml to initialise and finalise the provider (refer to section 6.2 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 these classes 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 used 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:

**Constructor:**

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

**Methods relating to the data model and SIF object the provider deals with:**

* getMarshaller (link data model to provider)
* getUnmarshaller (link data model to provider)
* getSingleObjectClassInfo (link a SIF Object to provider)
* getMultiObjectClassInfo (link a collection of SIF Objects to provider)

**Methods relating to the actual CRUD Operations for single and collection style SIF Objects:**

* retrievByPrimaryKey
* createSingle
* updateSingle
* deleteSingle
* retrieve
* createMany
* updateMany
* deleteMany

**Housekeeping methods:**

* getCustomServiceInfo (Response to HTTP HEAD call)
* shutdown

They are all mostly self-explanatory and the Javadoc for the Provider Interface gives a detailed description what each of these methods and their parameters are 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 BaseEventProvider forces you to implement the Provider & Event Provider Interface methods. These are the same methods as with the BaseProvider but additionally the following methods are required:

* 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 and their parameters are intended for. Each event provider class that is implemented must also implement a SIFEventIterator class. The intend and usage of this class is illustrated in section 5.6.2.3.

##### Available Method Overrides (BaseEventProvider Class Only)

The SIF3 Framework uses the provider’s property file to configure many HTTP Header values required by the SIF 3.x specification. The property file is also used to drive many default behaviours of the providers. There are situations where the default behaviour/values should be ‘overruled’ by a particular object provider while others should use the default behaviour/values. One such case could be where all providers send events with the standard SIF Data Model, which supports XML and JSON, but one provider sends CSV Content in the event. The default behaviour of the Framework is to use the media type listed in the provider’s property file which can only be XML or JSON. To override that for the CSV Event Provider the following method(s) in the CSV Provider class must be overridden:

@Override

public MediaType **getRequestMediaType**()

{

return getMarshaller().getDefault(); //Marshalling request

}

@Override

public MediaType **getResponseMediaType**()

{

return getUnmarshaller().getDefault();//Unmarshalling response

}

The getMarshaller() or getUnmarshaller() methods would return the CSV Marshal/Unmarshal Factory class a developer would have implemented. The SIF3 Framework will use the media type returned by these methods rather than what is set in the provider’s property file.

The list below shows all methods that can override values/behaviours of the provider’s property file. The name of the methods is mostly self-explanatory. Also refer to the Javadoc of the BaseEventProvider for each method.

* getGeneratorID()
* getRequestMediaType()
* getResponseMediaType()
* getCompressionEnabled()

**Experimental** (not part of the SIF 3.0.1 specification)

* getApplicationKey()
* getAuthentictedUser()

**Note (June 2015):**

While the SIF3 Framework allows events to be sent (provider) and processed (consumer) for media types other than JSON and/or XML it is currently not supported by any SIF3 Brokers and therefore that functionality is not yet available in brokered environments.

#### SIFEventItarator Class

In case of events the number of objects that may be published as an event can be large. Past implementations have shown that publishing of 500,000 objects is not unheard off, especially when dealing with Daily Attendance data. Resource restrictions would most likely not allow your service to read all 500,000 objects into memory for them to be published. To allow a more resource/memory friendly implementation of event data loading in your event publisher, the framework makes use of the SIFEventItarator class. Each provider class that extends the BaseEventProvider class must implement the getSIFEvents() method. This method has the following signature:

public **SIFEventIterator<L>** getSIFEvents()

The SIFEventIterator class allows you to implement an ‘iterator’ style implementation on how to retrieve your event objects and load them into memory. It is up to the actual implementation of that iterator how the memory friendly loading is done. Batch loading, paging, DB cursors etc. are all options that can be used. Please note that the SIFEventIterator is not an “iterator” style class as known from the java.util.Iterator interface. It is a much more basic iterator class specifically geared to the required functionality for event providers.

The javadoc for the SifEventIterator<L> class has extensive documentation on how to use and implement each method (3 methods) of that class.

## Service Paths

The Service Path functionality is a query mechanism that is commonly used in REST. It allows querying objects based on its “association”. For example you could use a service path query, to return a list of student attendances for a particular student.

A service path query is performed by prepending the filter objects and ref id’s to the request URL before the object being queried.

Example:

http://.../StudentPersonals/24ed508e1ed04bba82198233efa55859/StudentDailyAttendances

The above URL would return all student daily attendances for the student with the refId=24ed508e1ed04bba82198233efa55859.

There are a few extra steps involved in order to enable service path filtering. The next few sections illustrate how to implement that functionality on an object provider and/or an object consumer.

### Provider Environment Configuration (Service ACLs)

In the provider environment template XML you must define the service access rights for each desired service path as a special object of type SERVICEPATH. The name of the object is of the format “FilterObject/{}/ReturnObject” where return object is the object being returned, and filter object is the object to be used for filtering, the “{}” braces represent the position where the unique Id is inserted. You can also have multiple filter objects defined in the service path. The only access right that can be assigned to a service path object is “QUERY”.

Example Environment Template XML (snippet):

<services>

...

<service name="StudentPersonals/{}/StudentDailyAttendances"

contextId="DEFAULT"

type="***SERVICEPATH***">

<rights>

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

</rights>

</service>

<!-- Multi-Level service path -->

<service name="StudentPersonals/{}/SchoolInfos/{}/StudentDailyAttendances"

contextId="DEFAULT"

type="***SERVICEPATH***">

<rights>

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

</rights>

</service>

...

</services>

The first service defined above is how you define the service path query explained in section 5.7. The second service defined above would allow you to further filter the StudentDailyAttendances to a particular school.

**Note:**

More information about service paths and how they are intended to be used and configured can be found in the SIF Specification Base Architecture document.

### The QueryCriteria Class

The classes exposed to the developer make URLs completely transparent. This is also true for the service path functionality. Instead of knowing how exactly a service path URL is constructed the developer only deals with a query object based on which the SIF3 Framework will create the correct service path URL. The service path functionality on an object provider as well as a consumer will use a class called **QueryCriteria** to deal with the details of the query that is implied with a service path. To illustrate the usage of that class consider the following example:

**Example:**

Let’s consider a consumer wants a list of all students of a class. Using the SIF AU 1.3 data model this would equate to the following service path URL:

http://.../TeachingGroups/{teaching\_group\_refID}/StudentPersonals.

**Consumer:**

A consumer must create & populate a QueryCriteria object and pass it to the retrieveByServicePath() method of the AbstractConsumer. The code would look something like this:

QueryCriteria criteria = new QueryCriteria();

criteria.addPredicate(new ServicePathPredicate(“TeachingGroups”,

<teaching\_group\_refID>));

//Call method of AbstractConsumer.

List<Response> responses = consumer.**retrieveByServicePath**(criteria,...);

Note that the <teaching\_group\_refID> would be replaced with the teaching group’s refID.

**Object Provider:**

The object provider will receive the QueryCriteria object as a parameter of the retrieveByServicePath() method. For the given example the provider that services above query would be the StudentPersonal Provider (see also section 5.7.3). The method that will be invoked by the SIF3 Framework is the retrieveByServicePath() which will also have the QueryCriteria as a parameter. A provider may serve more than one service path (i.e. see example in section 5.7.1). In such a case the retrieveByServicePath() method must inspect the QueryCriteria Object to determine the “where-clause” that should be applied to the data to be retrieved and returned. Some pseudo code to do so is listed below (refer to javadoc for details about the various classes relating to the QueryCriteria object):

List<QueryPredicate> predicates = queryCriteria.getPredicates();

// ensure it is a valid condition (i.e. has at least one condition)

if ((predicates != null) && (predicates.size() >= 1)) {

if ("SchoolInfos".equals(predicates.get(0).getSubject())) {

// Get students for a school

}

else if ("TeachingGroups".equals(predicates.get(0).getSubject())) {

// Get students for a class

}

}

### Provider Implementation

The query implementation for service paths is performed in the provider for the object being returned. This is always the last name in the service path. In the examples in section 5.7.2 this would be the provider for StudentPersonals.

To enable service path query handling in your provider, your provider must implement the **sif3.common.interfaces.QueryProvider** interface. This interface defines query style methods which must be implemented. The method that will be required is called **retrieveByServicePath()**. Please refer to the javadoc for the **QueryProvider** class for details about usage of the method as well as the parameters.

An example of a service path implementation within an object provider can be found in the **retrieveByServicePath()** method in the **StudentPersonalProvider** class of the **systemic.sif3.demo.rest.provider** package.

### Consumer Implementation

You can find a basic example of how you could implement searching by service path in a consumer in the class **sif3.infra.test.rest.consumer.TestStudentPersonalConsumer**. Check out the getStudentsByServicePath() method, it builds a basic query given one parent type and it’s key. The important thing is that your consumer class simply calls the retrieveByServicePath() method of the AbstractConsumer with an appropriate QueryCriteria object (see section 5.7.2 for details).

## Query by Example (QBE)

**IMPORTANT NOTE: As of June 2015 until further notice**

The functionality described within this section is an early adoption of the “Query By Example” (QBE) functionality due to some pressing requirements from some real implementations. SIF 3.2 will support QBE but until the release of SIF 3.2 you are advised to consult with your provider/consumer if QBE is supported. Also the QBE is only supported in a DIRECT environment until the official release of SIF 3.2. This Java framework as well as the corresponding .Net framework already support this functionality.

### What is QBE?

The “Query by Example” (QBE) concept is not a new concept. The concept is known in other technologies and/or frameworks such as Entity Framework (.Net), Hibernate (database abstraction framework) which is available in .Net, Java (https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-examples), etc.

The concept is based on providing the “Query Executor” an example or template of the data to be returned. For example if we want all **female** students with the **legal family name** of “**Jones**” to be returned we would send the following payload to the object provider (we use the SIF AU 1.3 data model in the example):

<StudentPersonal>

<PersonInfo>

<Name **Type="LGL"**>

**<FamilyName>Jones</FamilyName>**

</Name>

<Demographics>

**<Sex>2</Sex>**

</Demographics>

<PersonInfo>

</StudentPersonal>

The result of that query should be a list of students where the three conditions in the above example match.

### Provider Implementation

The query implementation for QBE is performed in the provider class for the object being returned. For example if a QBE shall be performed for StudentPersonals (AU Data Model example) then the provider class that deals with the StudentPersonal object will implement the QBE.

To enable QBE query handling in your provider, your provider must implement the **sif3.common.interfaces.QueryProvider** interface. This interface defines query style methods which must be implemented. The method that will be required is called **retrieveByQBE()**. Please refer to the javadoc for the **QueryProvider** class for details about usage of the method as well as the parameters.

If your provider doesn’t support QBE your implementation of the **retrieveByQBE()**method should throw a UnsupportedQueryException exception. Also if the ‘exampleObject’ given to the retrieveByQBE() method holds elements your provider does not allow to query on a UnsupportedQueryException exception should be thrown. It is up to the provider class to inspect the ‘exampleObject’ to determine if the query template is supported or not.

### Consumer Implementation

You can find a basic example of how you could implement searching by QBE in a consumer in the class **sif3.infra.test.rest.consumer.TestStudentPersonalConsumer**. Check out the getStudentsByQBE() method which builds a basic StudentPersonal object where only the family name is provided, meaning that all students with a family name that matches the given value shall be returned. The important thing is that your consumer class simply calls the retrieveByQBE() method of the AbstractConsumer with the a StudentPersonal as the ‘exampleObject’.

### Query Conditions & Wildcards

As per SIF 3.2 specification (to be release later in 2015) the example object provided is the query template meaning that all the elements/attributes that are provided as part of the object are to be used in the query condition. Each element/attribute in the payload forms a specific query condition. The conjunction between the conditions is ‘**AND**’. There is no ‘or’ or any other conjunction between conditions supported. This is the nature of the QBE concept. The comparison operator for each condition is a ‘**LIKE**’ as known in SQL Syntax. The only supported wildcard in the value of an element/attribute is the ‘**%**’ which stands for any number of characters. The following assumptions and constraints apply to the LIKE comparator in QBE:

* A ‘LIKE’ shall be interpreted as an **EQUAL** if the SIF Object element/attribute type is anything other than a string style type (i.e. dates, numbers etc.).
* For string type elements the ‘LIKE’ with wildcard has the following meaning:
  + ‘ABC’: No wildcards in the value is equivalent to an ‘EQUAL’.
  + ‘%ABC’: Wildcard at the start of the value means ‘ENDS IN’. In this case anything that ENDS IN ‘ABC’.
  + ‘ABC%’: Wildcard at the end of the value means ‘STARTS WITH’. In this case anything that STARTS WITH ‘ABC’.
  + ‘%ABC%’: Wildcard at the start and end of the value means ‘CONTAINS’. In this case anything that CONTAINS ‘ABC’.
  + ‘AB%C’: Wildcard anywhere in the value means ‘STARTS WITH’ and ‘ENDS IN’. In this case anything that STARTS WITH ‘AB’ followed by any number of characters and ENDS IN ‘C’.
* Case Sensitivity: No case sensitivity is implied or mandated. It is up to the implementation of the provider if case sensitivity is required/applied or not.

## “Changes Since” Functionality

**WARNING: As of January 2016 until further notice**

The functionality described within this section is proposed for SIF 3.2 Infrastructure release. That release is expected in early 2016. As of January 2016 this framework has an early implementation of the “Changes Since” functionality but care must be taken that this implementation may have to change slightly once the SIF Specification is officially released. Early adoption of the functionality described in this section may mean that developers have do perform changes to the code base once the final release of this functionality is available.

### What is “Changes Since?

The “Changes Since” functionality is an alternate method to SIF Events. Instead of a provider pushing data changes to consumer using SIF Events, the “Changes Since” functionality allows a consumer to request data that has changed since a “given point”. “Changes Since” is intended to be used where SIF Events are not available which is most likely in DIRECT environments. If SIF Events are available then that is the preferred and recommended way to get changes to a consumer. “Changes Since” should only be used where SIF Events are not available.

The "Changes Since" allows a consumer to request data which has changed since a "given point". The "given point" is an opaque value that is provided by the provider to a consumer. The opaque value indicates a point since the last changes have been requested. That opaque value can be a timestamp, a version number, an offset etc. It is entirely up to the provider to determine what this opaque marker is. The important thing is that it is the provider who will return changed data (payload) and the next valid opaque marker (HTTP Header) to the consumer in the response to a "changes since" request. The SIF3 Framework abstracts all the low level details how this opaque marker is retrieved from a request and reported back to the consumer at the right time to meet the SIF Infrastructure Specification.

### Provider Implementation

The implementation for “Changes Since” functionality is performed in the provider class for the object being returned. For example if a “Changes Since” shall be supported for StudentPersonals (AU Data Model example) then the provider class that deals with the StudentPersonal object will implement the “Changes Since” functionality.

To enable “Changes Since” in your provider, your provider must implement the **sif3.common.interfaces.ChangesSinceProvider** interface.

Example:

public class StudentPersonalProvider extends BaseProvider implements **ChangesSineProvider**

{

...

}

This **ChangesSinceProvider** interface defines a few methods that must be implemented to support the “Changes Since” functionality. These methods are described in the next section. Additional details can be found in the javadoc of the **ChangesSinceProvider** class.

#### Methods of the ChangesSinceProvider Interface

This section shortly describes the methods of the ChangesSinceProvider interface. Additional details can be found in the javadoc of the class.

**Method Name:** **changesSinceSupported**

**Description:**

This method returns true if the particular object provider class supports changes since functionality. If it returns false then the SIF3 Framework assumes that “Changes Since” aren’t supported for this particular object provider and all other methods defined by this interface class can simply return null.

**Parameters: None**

**Return:**

See description section.

**Method Name:** **getLatestOpaqueMarker**

**Description:**

This method must return an opaque value (String). It is the value/id/date etc. of the latest change known for this object type. Generally the implementation of this method will look up a change log or the like to retrieve latest “Change Log ID” or the latest “date-time-stamp” in the change log or any other value determined by the provider. This value will be returned to a consumer. The SIF3 Framework will call this method at the right time and return it to the consumer in the correct way to meet the SIF Specification. Developers don’t need to worry about the internals of this.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| zone | No | The Zone from which the marker shall be returned. |  |
| context | No | The Context for which the marker shall be returned. |  |
| pagingInfo | Yes | Page information. Might be useful in determining the marker value. | If null then it is assumed that the consumer wants all objects. |
| metadata | No | Metadata relating to the request. Note that most of the properties might be null. |  |

**Return:**

See description section.

**Method Name:** **getChangesSince**

**Description:**

This method is used to retrieve a list of changed SIF Objects that have changed since the given “changedSinceInfo.changedSinceMarker”. The returned object must be a "Collection" style SIF Object (i.e. StudentPersonalCollectionType) even if there is only one change. The changes returned must be all objects of this OBJECT provider that have undergone any changes since the given "changedSinceInfo.changesSinceMarker". Paging might be used for this method indicating that the consumer will call this method multiple times to get all the changes since the "changedSinceInfo.changesSinceMarker". Paging starts with page number 1 (pagingInfo.currentPageNumber). If there are no more changes based on the changedSinceInfo.changesSinceMarker then this method must return null. If the data set to be returned is considered too large by the provider (implementation dependent) then a DataTooLargeException must be raised. This exception is then translated into an appropriate HTTP Status within the framework to meet the SIF Specification.

**Parameters:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nullable** | **Description** | **Default** |
| zone | No | The Zone from which the request is being issued. |  |
| context | No | The Context for which the objects shall be returned. |  |
| pagingInfo | Yes | Page information. Indicates to the provider which results to return. | If null then it is assumed that the consumer wants all objects. |
| changedSinceInfo | No | Data required to determine the changes to be returned. This holds the opaque changes since value that was given to the consumer at some stage via the getLatestOpaqueMarker() method of this class. |  |
| metadata | No | Metadata relating to the request. Note that most of the properties might be null. |  |
| customResponseParams | No | Values to be returned as part of the response. These are generally just HTTP Header fields. If a developer sets the HTTP Header of a well defined SIF3 HTTP Header (i.e. providerId, timestamp) then the framework may override these with its own value to ensure the correct use and workings of the framework. It is the developer who will populate the object. When it is passed to this method it won’t be null but might be empty. |  |

**Return:**

The returned object must be a "Collection" style SIF Object (i.e. StudentPersonalCollectionType) even if there is only one change. The changes returned must contain all objects of this OBJECT provider that have undergone any changes since the given "changedSinceInfo.changesSinceMarker", potentially applying paging.

### Consumer Implementation

**Not yet supported.**

## Custom/External Security Service Integration

**WARNING: As of March 2015 until further notice**

The functionality described within this section is a custom implementation and does only work for DIRECT environments. As the custom/external security service is not yet part of the SIF Specification as of June 2015 the functionality provided by the framework in relation to this type of security may change in future to fully comply with the SIF Specification. Using the functionality described in this section must be considered as experimental and should be used with care. No guarantee can or is being given that it will work for all circumstances.

### Overview & Motivation

SIF 3.x supports various ways of security. It allows for HTTPS (SSL) to secure the actual data on the ‘wire’ (see section 5.11 on how to use HTTPS with this framework). For authentication of adapters/services with the environment provider there are two standard authentication methods available, called ‘Basic’ and ‘SIF\_HMACSHA256’. There are strong drivers ‘in the field’ that would like to use other authentication services such as OAuth, SAML, Active Directory etc. For the purpose of this document and in context of the SIF3 Framework all these security and authentication services are considered ‘Custom’ or ‘External’ Security Services. This section of the Developer’s Guide illustrates how such external security services are enabled within the SIF3 Framework.

The general idea to enable these external security services is through the use of an implementation of a Java interface class that enforces a few methods to integrate external security services. The framework doesn’t really need to know how to implement and integrate with a particular external security service rather it uses appropriate interface methods to interact with external security services in an abstract manner. The developer on the other hand knows what external security service shall be used and can implement a concrete class that implements the security interface(s) and calls the specific external security service from within that implementation. This keeps the SIF3 framework clean from specific implementations such as OAuth, SAML etc. but also provides the flexibility to use a number of external security services.

### DIRECT Environment Provider

This section describes the steps required to configure a Provider in a DIRECT environment to utilise an external security service.

In a direct environment consumers will send requests to the provider to perform any CRUD operations. It is assumed that the consumer has used an external security service to authenticate itself and in return has received some sort of ‘access\_token’ from that security server. The consumer will only use that access\_token as the authorisation token to communicate and identify itself with the provider. Up to this point the provider has not been involved with the authentication and/or authorisation at all. The question is how do we ‘tell’ the provider what it needs to do with this type of authentication. The next few sections illustrate the steps required to enable external security services in a direct provider so that consumers that have used them can request data from a direct provider.

#### Step 1: Implement a Security Class – Extend AbstractSecurityService

To use an external security service a class **must be implemented that extends the** **AbstractSecurityService** class. This ensured that all required methods and the appropriate constructor is implemented for the SIF3 Framework to use an external security service. It is expected that the implementation of this security class is connecting to a security service such as OAuth and performs the operations enforced by the AbstractSecurityService class.

The extension of the AbstractSecurityService enforces the following methods (see Javadoc for details about parameters etc.):

* ***AbstractSecurityService(AdvancedProperties properties)***: This *constructor* **must exist**. It will be called by the framework. The ‘properties’ parameter is automatically set by the SIF3 Framework when it instantiates the security class. It will pass the properties of the provider’s property file to this constructor, so that they can be used within the other methods of this class. This constructor may or may not build up a permanent connection or connection pool to the external security service such as OAuth for efficiency reasons. It may initialise other resources etc.
* ***boolean validateToken(String securityToken, RequestMetadata requestMetadata)***: This method will be called by the SIF3 Framework to validate a security token. The security token is the access\_token from a consumer as it has been given to the consumer by the external security server (remember the consumer has called that security server before to authenticate itself). The provider must ensure that the consumer that requests any CRUD operations is a valid and authenticated user/consumer and therefore will call this ‘validateToken’ method at some stage. It is assumed that the implementation of this method will contact the security server such as OAuth to validate the security token. Please refer to the framework’s Javadoc for information about the ‘requestMetadata’ parameter. **Note this parameter is intended to be a ‘read-only’ parameter** and that information in that parameter could mostly be null if a specific property wasn’t available from a consumer’s request through either a URL Query Parameter or a HTTP Header field.
* ***TokenInfo getTokenInfo(String securityToken, RequestMetadata requestMetadata)***: In certain circumstances (i.e. a REST call to the create environment) may need additional information about the consumer’s security token. In these cases the ‘getTokenInfo’ method of the AbstractSecurityService class is called. As with the ‘validateToken’ method it is assumed that the implementation of the ‘getTokenInfo’ method connects to the external security server such as OAuth to retrieve information about the consumer’s security token. Please refer to the Javadoc of the TokeInfo class for details about the information that may be contained in the TokenInfo. Generally it would be information about a user, an application or both, or the SIF environment ID or SIF Session Token etc. What information is returned strongly depends on the functionality of the external security server. Please refer to the framework’s Javadoc for information about the ‘requestMetadata’ parameter. **Note this parameter is intended to be a ‘read-only’ parameter** and that information in that parameter could mostly be null if a specific property wasn’t available from a consumer’s request through either a URL Query Parameter or a HTTP Header field.
* ***TokenInfo generateToken(TokenCoreInfo coreInfo, String password)***: This method is only required by consumers. For a Direct Provider this method can simply return null as it is not used.

Other methods of interest are:

* ***AdvancedProperties getServiceProperties()***: This method can be used to get hold of the properties as set by the constructor of the AbstractSecurityService class. It might be useful to get hold of some of properties of the provider’s property file. This method is the way to get access to the content this property file.

#### Step 2: Tell the framework to use the Security Class

Once a concrete class that extends the AbstractSecurityService is implemented the SIF3 Framework must be configured to use that security class. This is done by means of setting the **adapter.security.service** property in the provider’s property file. The fully qualified class name of the security class from step 1 must be set in this property.

Example:

**adapter.security.service=com.demo.security.MyOAuthSecurityService**

#### When is the External Security Service Used?

It must be noted that an external security service is only used in certain circumstances. Only if the HTTP Authorization Header in the consumer’s request uses the ‘**Bearer**’ authentication method then the external security service will be used. If ‘Basic’ or ‘SIF\_HMACSHA256’ are used as the authentication method then the external security service won’t be invoked and the standard SIF 3.x authentication method will be utilised. The authentication method is determined by the first part (string before the ‘ ‘ character) in the HTTP Authorization header of each HTTP request. The following authentication methods are supported by the SIF3 Framework:

|  |  |  |
| --- | --- | --- |
| **Authentication Method** | **Part of SIF 3.0.1 Specification** | **HTTP Authorization Header Example** |
| Basic | Yes | **Basic** Z3Vlc3Q6cGFzc3dvcmQx |
| SIF\_HMACSHA256 | Yes | **SIF\_HMACSHA256** Z3Vlc3Q6WjNWbGMzUTZjR0Z6YzNkdmNtUXg= |
| Bearer | No | **Bearer** n4bQgYhMfWWaL+qgxVrQFaO/TxsrC4Is0V1sFbDwCgg= |

#### SIF3\_APP\_TEMPLATE Table – Bearer Authentication

The SIF3\_APP\_TEMPLATE table holds all valid applications that can connect to a direct provider. Besides the environment template to be used with each application it also states the authentication method to be used with that application (AUTH\_METHOD column). If an application shall use an external security service it is assumed that the authentication method is ‘Bearer’ (see previous section for details). It must be ensured that the AUTH\_METHOD column has the value of ‘**Bearer**’. In this case the PASSWORD column would not be set (i.e. remain null) because the external security service would deal with the user’s password and no longer the SIF3 Framework. The PASSWORD column is only required for Basic and SIF\_HMACSHA256 authentication methods.

#### SIF Express Considerations

If a consumer uses SIF Express then it may not set the HTTP Authorization Header instead it may set the access token as a URL Query Parameter. In this case the SIF3 Framework will automatically assume that the authentication method is ‘**Bearer**’ and therefore will use the external security service. This is the default behaviour. If another authentication method shall be used with the access token URL Query Parameter then an additional URL Query Parameter called ‘authenticationMethod’ must be provided, indicating the authentication method to be used with the given access token.

**Examples 1**:

http://.../requests/StudentPesonals?access\_token=Z3Vlc3Q6cGFzc3dvcmQx

The above would assume that the authentication method is ‘Bearer’ and therefore SIF3 Framework would behave as if the StudentPersonals service would have been called with an HTTP Authorization Header of ‘Bearer Z3Vlc3Q6cGFzc3dvcmQx’ and therefore call the external security service.

**Examples 2**:

http://.../requests/StudentPesonals?access\_token=Z3Vlc3Q6cGFzc3dvcmQx&**authenticationMethod=Basic**

This example uses the same access\_token but it also states that the authentication method to be used with that access token is ‘Basic’. In this case the SIF3 Framework would behave as if the StudentPersonals service would have been called with an HTTP Authorization Header of ‘Basic Z3Vlc3Q6cGFzc3dvcmQx’ and would use the standard SIF 3.x basic authentication.

In some cases it might be desirable that the default authentication method of the ‘access\_token’ is different to ‘Bearer’. To change the default authentication method of the ‘access\_token’ the adapter.default.accessToken.authentication.method property in the provider’s property file can be set to the desired authentication method (Basic, SIF\_HMACSHA256). See section 9 for more information on this and other properties of the provider’s property file.

### Consumer

This section describes the steps required to configure a consumer to utilise an external security service. Note that only consumers in a DIRECT environment can use that method.

The general sequence a consumer would perform to utilise external security services is:

* Contact an external security service (i.e. OAuth server) and provide some credentials.
* The security service will return an ‘access\_token’.
* The consumer uses the ‘access\_token’ as the authorisation token to communicate with the provider.

The question is how can the consumer get an ‘access\_token’ and how is it communicated to the provider. The next few sections illustrate the steps required to enable external security services for a consumer.

#### Step 1: Implement a Security Class – Extend AbstractSecurityService

To use an external security service a class **must be implemented that extends the** **AbstractSecurityService** class. This ensured that all required methods and the appropriate constructor is implemented for the SIF3 Framework to use an external security service. It is expected that the implementation of this security class is connecting to a security service such as OAuth and performs the operations enforced by the AbstractSecurityService class.

The extension of the AbstractSecurityService enforces the following methods (see Javadoc for details about parameters etc.):

* ***AbstractSecurityService(AdvancedProperties properties)***: This *constructor* **must exist**. It will be called by the framework. The ‘properties’ parameter is automatically set by the SIF3 Framework when it instantiates the security class. It will pass the properties of the consumer’s property file to this constructor, so that they can be used within the other methods of this class. This constructor may or may not build up a permanent connection or connection pool to the external security service such as OAuth for efficiency reasons. It may initialise other resources etc.
* ***boolean validateToken(String securityToken, RequestMetadata requestMetadata)***: This method is only required by providers. A consumer must not implement it and can just return true/false. It is not called anywhere in the SIF3 Framework for consumers.
* ***TokenInfo getTokenInfo(String securityToken, RequestMetadata requestMetadata)***: This method is only required by providers. A consumer must not implement it and can just return null. It is not called anywhere in the SIF3 Framework for consumers.
* ***TokenInfo generateToken(TokenCoreInfo coreInfo, String password)***: A consumer must implement this method. This method may contact the security server which can be an OAuth server, and LDAP server an Active Directory etc. It will then retrieve a security token and optionally an expire date for that token based on the 'coreInfo' and optionally the 'password' parameter given to this method. In return the TokenInfo object which will have the 'token' property set (the security token). Optional the 'tokenExpiryDate' may be set if the token has an expire date. If the 'tokenExpiryDate' is null it is assumed that the returned security token won't expire.

**NOTE**: The coreInfo parameter has an element called ‘otherInfo’. This will automatically hold all values from the consumer’s property file that have the prefix “**security.service.property**”. The name of the property in the ‘otherInfo’ hashmap is the name after that prefix.

Example:

Consumer Property File has the following two properties set:

security.service.property.prop1=value1

security.service.property.prop2=value2

The ‘otherInfo’ element in the TokenCoreInfo class will hold the following two values (note the name of the property which has the prefix of “security.service.property” removed!):

prop1=value1

prop2=value2

They can be accessed in the following manner:

**coreInfo.getOtherInfo.get(“prop1”);**

Other methods of interest are:

* ***AdvancedProperties getServiceProperties()***: This method can be used to get hold of the properties as set by the constructor of the AbstractSecurityService class. It might be useful to get hold of some of properties of the consumer’s property file. This method is the way to get access to the content this property file.

#### Step 2: Tell the framework to use the Security Class

Once a concrete class that extends the AbstractSecurityService is implemented the SIF3 Framework must be configured to use that security class. This is done by means of setting the **adapter.security.service** property in the consumer’s property file. The fully qualified class name of the security class from step 1 must be set in this property.

Example:

**adapter.security.service=com.demo.security.MyOAuthSecurityService**

Secondly the **env.authentication.method** property in the consumer’s property must be set to ‘**Bearer**’.

Example:

**env.authentication.method=Bearer**

Once these two properties are set the consumer will use external security services and populate all HTTP header fields correctly. The developer does not need to implement anything else.

### Brokered Provider

*Not yet supported.*

## 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 9 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 may need the public component of the certificate. 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 and you may skip step 1. 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 use a certificate from a known certificate authority you could use the JRE’s trusted store and point to it. The JRE’s trusted store is in the location <jre>/lib/security/cacerts. In this case you may not need the keystore either (set key.store= in the environment.properties file).

1. 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 9 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.

### Client Certificate Exchange – Mutual Authentication

**Client-based mutual authentication** using certificates provides a high level of security. Not only are the messages that are transmitted encrypted (HTTPS) but the server (provider) is ensured to only allow access to data to a known and secure consumer.

To enable client certificate exchange, also known as **client-based mutual authentication**, HTTPS must be enabled on the consumer as well as on the provider. Please refer to previous sections on how this is achieved. The SIF3 Framework fully supports client-based mutual authentication. With this type of authentication the following actions occur between consumer (the client) and server (provider or broker):

1. A consumer requests access to data (i.e. Student Personal).
2. The web server presents its certificate to the consumer.
3. The consumer verifies the server’s certificate.
4. If successful, the ***consumer sends its certificate*** to the server.
5. The server verifies the consumer’s credentials/certificate.
6. If successful, the server grants access to the requested data by the consumer.

Enabling client-based mutual authentication is a matter of configuring the provider’s web- or application server to allow and enforce this. It is outside the scope of this document to describe how this is done. Please refer to your web- or application server’s administration guide for details.

There are a few points to remember though. Besides the server certificate to enable HTTPS an additional client certificate is required (see algorithm above). This client certificate must be known to the server. If a ‘self-signed’ certificate is used for development then this certificate must be imported to the **cacerts** keystore (trusted key store) which can be found in your JDK’s installation under **<jdk\_root>/jre/lib/security**. Only this way the web- or application server will accept a self-signed certificate. If the client certificate is issued by a trusted authority then this step can be ignored. On the consumer side the private key of the client certificate must be imported to the consumer’s keystore. Once the client certificate is imported to appropriate key stores the consumer and provider can use **client-based mutual authentication for security**.

If you encounter any issues with that type of security you can enable SSL style debugging on the consumer and/or server by setting the JVM property **-Djavax.net.debug=ssl**. If you start your web- or application server with that JVM property a lot of SSL and key store information is logged. The same is true if you start your consumer with the same JVM property.

## Provider Request/Response Auditing

The SIF3 Framework has a feature that allows some auditing of all request and responses that are received/sent by the **object provider**. This functionality was added due to the demand of some projects and their monitoring requirements as well as a strong integration testing feature. This is an optional feature which can be turned on and off as required. Auditing capabilities are currently not available for object consumers.

**Note:**

Attention must be taken when enabling this feature because it intercepts each HTTP request and HTTP response on the server (provider) and therefore a slight performance impact must be expected.

### Auditing Class & Enable/Disable Auditing

To enable auditing a class must be implemented that logs each auditing record. For this the implementation class must implement the ‘Auditor’ interface (sif3.common.interfaces.Auditor). This interface forces one method to be implemented. It is entirely up to the implantation how each auditing record is processed. Options include but are not limited to:

* Log to a file using log4j or something similar.
* Log to a database
* Write to any other IO.
* Etc.

An example of such logging of an audit record can be found in the systemic.sif3.demo.audit.LogAuditor class under the demo package. This implementation simply uses log4j to write the audit record to a file.

Once a specific audit logging class has been written it must be enabled by the framework. Enabling is as simple as adding the fully qualified class name of the implementation class to the provider’s property file in the ‘adapter.audit.service’ property. If that property is set then auditing is enabled, if it is not set or doesn’t exist auditing is disabled. To switch between enable/disable the provider must be redeployed.

### Audit Record

The AuditRecord class has a number of properties that are populated by the framework. The data that is made available fall into three categories:

* HTTP Request related data such as the HTTP method (GET, PUT, POST etc.), HTTP request headers etc.
* HTTP Response related data such a HTTP response status, HTTP response headers etc.
* SIF related data such as environment Id, session token, zone, context etc.

Any of these properties can be logged. There is no requirement that all properties are logged and how they are logged. It is up to the implementation of the ‘Auditor’ interface class and the requirements each project may have. An example can be found in the systemic.sif3.demo.audit.LogAuditor class under the demo package.

## Hibernate Properties Injection

There are situations where the deployment of an adapter (consumer or provider) to a given platform doesn’t want to use the sif3infra.hibernate.cfg.xml file to set certain properties such as username and password as they might not be considered secure of must be retrieved from a repository like place. In such a case these properties should be “injected’ programmatically to the hibernate configuration at start-up of the adapter. This framework allows that through an injection approach. To enable injection of hibernate properties the developer must implement the sif3.common.interfaces.HibernateProperties interface. An example can be found in the systemic.sif3.demo.hibernate.DemoProperties class. To “tell” the framework to use this implementation to retrieve certain properties the property called “adapter.hbr.propertyClass” in the consumer’s or provider’s properties file must be set to the implemented class. For above example the property would be set to:

adapter.hbr.propertyClass=systemic.sif3.demo.hibernate.DemoProperties

Hibernate for this SIF3 Framework will now be initialised using the properties of the sif3infra.hibernate.cfg.xml file as well as the properties of the implemented property class. If a certain property is set in the sif3infra.hibernate.cfg.xml file and in the property class then the value of the sif3infra.hibernate.cfg.xml will take precedence, meaning the corresponding value in the property class will be ignored.

## Compression (GZIP)

Compression can be used to transfer all payloads (request and response) between consumer and provider. Payload compression can be useful for large payloads to minimise bandwidth. This may or may not improve the request/response turnaround time. Once compression is enabled there is an overhead on the request and response processing on the consumer as well as the provider which will impact full turnaround times. For large payloads and low bandwidth environments the overhead created by compressing payloads will be offset by faster transfer speeds due to the considerable smaller payloads. Framework test have shown that the compression can add a few per-cent (3%-5%) overhead in processing a payload on the consumer as well as on the provider. For XML/JSON payloads the compression factor is about 20 which means a 1MByte payload can be compressed to about 50kByte. On tablets that may have a slower line this bandwidth safing can significantly improve the transfer speeds despite the compression processing overhead. Test have shown that transfers speeds can be increased by a factor 8-10 in low bandwidth cases (800%-1000%) which clearly demonstrates that the 4% compression overhead is negligible compared to the gain in throughputs.

### Enable Compression on Consumer

Enabling compression on a consumer is achieved by setting the **adapter.compression.enabled** property to **true** in the consumer’s property file. Please note that the only supported compression is GZIP. This is the most commonly used compression method and therefore should not be a limitation.

### Enable Compression on Provider

Since a provider is a ‘server’ in the traditional sense it also allows enabling compression in many ways. There are three common ways to enable compression on a provider. Each of them is shortly discussed in the next sections.

#### Enable on Proxy

Most proxies (i.e. Apache) allow compression to be managed natively at this point. This means the web- or application container will only see already de-compressed payloads and do not have to perform any compression at all. Often this is the most efficient way to deal with compression. It is outside the scope of this document to outline how a particular proxy can be configured to manage compression. Please refer to your proxy’s administration documentation for details.

#### Enable in Web- or Application Container

Similar as with proxies most web- or application containers support the functionality of compressing/de-compressing messages out-of-the-box. If you do not have a proxy that manages compression, you may want to turn compression management on within your web- or application container. It is outside the scope of this document to outline how a particular web- or application container can be configured to manage compression. Please refer to your product’s administration documentation for details.

#### Enable in Framework

If compression management is **not enabled at the proxy or web-/application container** then it can be turned on/off within the SIF3 Framework. Enabling compression is done by means of adding a compression filter to your web.xml file. The SIF3 Framework has a small library with such a compression filter. You can use this one or you can get an alternative compression filter or even develop one yourself. Below are the instructions on how to add the SIF3 Framework’s provided compression filter. You would follow similar steps with an alternative compression filter.

The compression filter that is part of the SIF3 Framework is the PJL Compression Filter ([http://sourceforge.net/projects/pjl-comp-filter](http://sourceforge.net/projects/pjl-comp-filter/) and <http://pjl-comp-filter.sourceforge.net>). For various properties on that filter please refer to the appropriate web-site. Below are only the core properties listed in the configuration of this filter in the web.xml but there are additional parameters available. To enable the PJL Compression Filter in the SIF3 Framework you must add the following section to your web.xml:

<filter>

<filter-name>**CompressingFilter**</filter-name>

<filter-class>com.planetj.servlet.filter.compression.CompressingFilter</filter-class>

<init-param>

<param-name>debug</param-name>

<param-value>false</param-value>

</init-param>

<init-param>

<param-name>compressionThreshold</param-name>

<param-value>0</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>**CompressingFilter**</filter-name>

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

</filter-mapping>

Add this section right after the servlet definitions. An example can be found in the SIF3 Framework’s war/WEB-INF/web.xml file. Ensure that you have the **pjl-comp-filter-1.8.1.jar** in your WEB-INF/lib directory. A copy of the library can be found in the lib/pjl directory.

**Parameters of Filter**

* **debug** (optional): if set to "true", additional debug information will be written to the servlet log. **Default: false**.
* **compressionThreshold** (optional): sets the size of the smallest response that will be compressed, in bytes. That is, if less than compressionThreshold bytes are written to the response, it will not be compressed and the response will go to the client unmodified. If 0, compression always begins immediately. **Default: 1024**.

Additional parameters can be found on the [PJL web-site](http://pjl-comp-filter.sourceforge.net/) under the CompressionFilter class.

#### Events

If your provider publishes events then the compression for the event payload can be turned on. When publishing events your provider behaves like a ‘client’ to the Broker and therefore compression must be turned on by means of setting the **adapter.compression.enabled** property to **true** in the provider’s property file. Please note that the only supported compression is GZIP. This is the most commonly use compression method and therefore should not be a limitation.

## Functional Services

From the SIF 3.2 specification[[1]](#footnote-1) Functional Services document “a Functional Service coordinates one or more data exchanges such that they have a beginning, middle, and an end.

A Functional Service consists of:

* a ‘functional service’ definition describing the service
* a ‘Job Object’ format for communicating intent, status, and results
* one or more ‘Phases’ - which accomplish the Job step-by-step”

Implementing a functional service provider and consumer are detailed in the following subsections.

### Building a Provider

All HTTP requests directed to functional services will be routed through an instance of the class sif3.infra.rest.resource.ServiceResource. This class will manage marshalling/unmarshalling data and redirecting the request to the appropriate method on an identified service provider class. Therefore, a Functional Service Provider implementation must extend the class sif3.infra.rest.provider.BaseFunctionalServiceProvider. The inheritance structure of this class and the methods that must be implemented are indicated in the Codechart[[2]](#footnote-2) below:



In the above diagram grey rectangles represent classes that exist in the framework, some of which are abstract/interfaces. Methods are represented by the ellipses, some of which are also abstractly defined. Ellipses with shadow represents a set of operations, in this case all CRUD and associated operations. To implement a functional service you must implement a class that extends BaseFunctionalServiceProvider such that it fits in the place of the white rectangle labelled “YourFunctionalService”. In implementing this class you must implement the two methods indicated by the white ellipses (JobShutdown and configure), and a set of provider operations.

That is, the class BaseFunctionalServiceProvider implements the FunctionalServiceProvider interface and extends the BaseProvider class. In doing so it inherits CRUD functionality for the SIF3 Job object and implements the necessary functionality to support phase and state CRUD operations. The methods configure(SIF3Job) and jobShutdown(SIF3Job) are methods you will be required to implement, as well as the usual methods for a provider class in the framework (shutdown(), getSingleObjectClassInfo(), and getMultiObjectClassInfo()). Each of these is explained below:

|  |  |
| --- | --- |
| Method | Purpose |
| configure(SIF3Job) | This method configures (‘decorates’) a job instance with all the necessary phases, timeout, etc. To add a phase to the job code like the following is required:    This code demonstrates adding a phase named “default” to the job, which is defined as required (true). Consumers will have the right to send create, query and update messages to the phase (but not delete etc.). Consumers can also send create messages to the states of this phase. An initial state, e.g. NOTSTARTED, and description can optionally be included. |
| JobShutdown(Job) | This method is called when deleting a job. This method should do any checks that are required in the business logic of the functional service to make the given Job safe to delete. Throwing any exception will cause the Job shutdown process to fail for that job and the job will not be deleted. |
| shutdown() | Called when the provider is shutting down. Should release all resources used by the provider, e.g. database connections. |
| getSingleObjectClassInfo() | Returns a ModelObjectInfo instance with a **singular** name (i.e. the name of job objects) and the objectType argument must be JobType.class. For example: |
| getMultiObjectClassInfo() | Returns a ModelObjectInfo instance with a **plural** name (i.e the name of the functional service) and the objectType argument must be JobCollectionType.class. For example: |

A class that extends BaseFunctionalService should populate the protected phaseActions dictionary. This associates a phase name with a specific implementation of the PhaseActions interface.



PhaseActions implementations encapsulate the CRUD operations available for a phase. In this case a set of actions for the phase named “default” have been defined in a class called DefaultActions. A BasePhaseActions class is provided that implements all CRUD methods so that you need focus only on those you wish to support. All methods other methods will throw an UnsupportedQueryException causing an appropriate error to be sent back to the Consumer. The structure of the phase action classes is shown in the Codechart below:



That is, each phase action implementation you create (a set of implementations represented by the white rectangle with a shadow) should extend the abstract BasePhaseActions class.

Each action in a phase expects a (possibly null) payload that may require deserialization according to the business logic of the application. Serialisation and deserialization of these payloads is not handled automatically by the framework since the payload may be text, XML, JSON, a binary format file (image or ZIP), etc.. Similarly the response must be manually serialized to string for sending back to the Consumer (the String returned by each action). Implementations of an action can check that the contentType and Accept headers are as expected, and may adapt themselves accordingly.

The choreography between these classes for CRUD operations on a job (i.e. /services/<ServiceName>) is as defined in the framework for data objects. For phases the following holds:

1. Consumer sends message which is (possibly processed by a broker and) received by the *ServiceResource*
2. *ServiceResource* processes the HTTP message and calls the correct method on the *FunctionalServiceProvider*
3. *FunctionalServiceProvider* performs required checks, such as checking the phase ACL to confirm that the request is authorised and, if permitted, identifies a phase action for the named phase and passes on the request.
4. The phase action should check that the request/response media types are acceptable, perform any deserialization that might be required, operates over the data, and sends back a serialized response in the format the consumer expects.

#### Functional Service Provider Settings

The SifFramework.config file provided is mandatory for the operation of the Functional Service Provider.

The types of the functional services that are configured. Default is "any".

#### job.timeout.frequency

Determines how often to check for timed out jobs. Value is numeric in seconds. If unspecified the default is 60. To disable job timeouts use a value of 0.

#### job.binding

Determines if jobs should be bound to the consumer that created them. Value is a Boolean. If unspecified the default is “true”.

If “true” then once a job is created it gets bound to the application key/user token (in DIRECT environments) or source name (in BROKERED environments). If a request to operate on a job is received the consumer is checked against the job’s binding. If the Consumer does not “own” the job the request will fail and a suitable error is reported to the consumer.

If “false” then all jobs are visible/accessible to any Consumer that is given rights to access the functional service to which they belong.

### Building a Consumer

A unique functional service consumer must be implemented for each functional service to be consumed. To accomplish this the class AbstractFunctionalServiceConsumer should be extended. All methods that should be implemented are defined in the standard instructions for implementing a Data Object Consumer. The important distinction is that the getSingleObjectClassInfo() and getMultiObjectClassInfo() methods should return ModelObjectInfo instances that point to the JobType.class and JobCollectionType.class classes respectively.

Once implemented, to use the functional service consumer you need only instantiate it, register it, and work with its API. How to instantiate and register a functional service consumer is identical to an object service consumer, so the rest of this section will focus on the key API you will work with on the consumer end.

### Job CRUD operations

In each of the below operations the list of ZoneContextInfo instances is a list of zones and contexts in which to attempt to perform the action. This can be a null value, in which case the default zone is assumed. Each returns a list of SIF3 Response instances.

The Object argument for a “single” operation must be an instance of SIF3Job, whereas the Object argument for a “many” operation must be an instance of Collection<SIF3Job>.

All operations can also take an additional CustomParameters object, which allows custom header values to be exchanged. This is defined elsewhere in the documentation.

|  |  |
| --- | --- |
| Name | Description |
| createSingle(Object,  List<ZoneContextInfo>) | Create a single Job with the defaults provided. |
| createMany(Object,  List<ZoneContextInfo>) | Create multiple Jobs with the defaults provided. |
| retrieveByPrimaryKey(  resourceID,  List<ZoneContextInfo>) | Gets a single Job object by its resourceID. |
| retrieve(PagingInfo,  List<ZoneContextInfo>) | Get all jobs from the service that handles the specified jobname (a string) with (nullable) page number and page size parameters. |
| UpdateSingle(…) | Update single job object is **not supported** for Functional Services. Throws a UnsupportedOperationException. |
| UpdateMany(…) | Update multiple job objects is **not supported** for Functional Services. Throws a UnsupportedOperationException. |
| DeleteSingle(resourceID,  List<ZoneContextInfo>) | Delete a Job with the given RefId (resourceID). |
| DeleteMany(List<String>,  List<ZoneContextInfo>) | Delete a series of Jobs from the list of RefIds. |

### Phase CRUD operations

In each of the below operations the list of ZoneContextInfo instances is a list of zones and contexts in which to attempt to perform the action. This can be a null value, in which case the default zone is assumed. Each returns a string that is the direct output of the phase.

The arguments requestMediaType and requestMediaType are both instances of class MediaType that indicate what format the data is in and what format is expected back. These should be standard mime type values and are also nullable.

All operations can also take an additional CustomParameters object, which allows custom header values to be exchanged. This is defined elsewhere in the documentation.

|  |  |
| --- | --- |
| Name | Description |
| CreateToPhase(ResourceId, phaseName,  body, requestMediaType,  responseMediaType,  List<ZoneContextInfo> | Send a create operation to a specified phase on the specified job with a (possibly null) payload. |
| RetrieveToPhase(ResourceId, phaseName,  body, requestMediaType,  responseMediaType,  List<ZoneContextInfo> | Send a retrieve operation to a specified phase on the specified job with a (possibly null) payload.  The payload may be used to contain criteria to apply on the phase’s actions. |
| UpdateToPhase(ResourceId, phaseName,  body, requestMediaType,  responseMediaType,  List<ZoneContextInfo>) | Send a update operation to a specified phase on the specified job with a (possibly null) payload. |
| DeleteToPhase(ResourceId, phaseName,  body, requestMediaType,  responseMediaType,  List<ZoneContextInfo> | Send a delete operation to a specified phase on the specified job with a (possibly null) payload. |

### Phase State CRUD operations

In each of the below operations the list of ZoneContextInfo instances is a list of zones and contexts in which to attempt to perform the action. This can be a null value, in which case the default zone is assumed. Each returns a list of SIF3 Response instances.

All operations can also take an additional CustomParameters object, which allows custom header values to be exchanged. This is defined elsewhere in the documentation.

|  |  |
| --- | --- |
| Name | Description |
| CreateToState(resourceID, phaseName,  state, List<ZoneContextInfo>) | Send a create operation to the state of the specified phase on the specified job. The state object cannot be null. |

# Deployment

This section lists all the components you need to consider if you wish to deploy a SIF 3 Service (consumer and/or provider). It touches on the configuration of all applicable parts. Most of it is straight forward and only needs to be done once, yet care must be taken to consider all these areas.

## Servlet 3.0 Requirement for Providers

As stated in section 2.3 providers are deployed in a web- or application container. These containers must support the servlet 3.0 specification. Please ensure that the web.xml of your SIF3 provider states version 3.0 of the servlet specification. The first line of the web.xml should look something like this (note the **bold & red** components which indicate the servlet version):

<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**">

## Servlet Context

For a server component (i.e. provider) to be deployed successfully the web.xml must have a couple of specific entries to ensure that the provider is initialises and shut down properly. This section lists the sections you MUST add to your web.xml to enable the SIF3 framework as part your provider service.

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 5.6.2.1.1).

<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’.

## Java classpath Configuration

For the SIF3 Framework to work in your deployment unit you must configure the classpath of your deployment unit to include the following list of configuration files. You should be able to find example of these files under the framework’s ‘config’ directory. Note that most of these files are required to be on the classpath of a consumer and/or provider. Exceptions are listed with each configuration file.

* log4j.properties (not required with JBoss as it uses its own way of configuring log4j).
* environment.properties
* c3p0.properties
* hibernate.properties
* sif3infra.hibernate.xml
* consumer property file (consumer deployment only, example in config/consumers)
* provider property file (provider deployment only, example in config/providers)

**Note:**

Your project may have some of these property files (i.e. log4j.properties, hibernate.properties, c3p0.properties) already included. In that case you don’t need to use the one’s in the framework but you can use your very own property file. To turn of logging for some framework classes you just add appropriate entries in your own log4j.properties file as you would do with any other 3rd party library. The SIF3 Framework does not really use any properties from the hibernate.properties and c3p0.properties file but they must exist for hibernate to function.

## Customise Property Files to your environment

For a successful deployment and runtime behaviour of your SIF3 Services you need to customise some properties in a couple of properties file to your deployment environment. The files and properties that need to be configured are listed below. Please not that some of these properties are only applicable for the demo version of the framework and won’t be used in your final and real SIF3 Service/Project.

### Customise sif3infra.hibernate.xml

Ensure that the connection/jdbc details match with the notation of your DB of choice and the schema where you have installed the tables of this framework (see section 5.3.1.2 for details).

### Customise environment.properties

Ensure that the paths for the properties listed below point to a valid location of your deployment environment. The directories must exist and if they don’t you should create them. For the key.store and trust.store properties and their usage, please refer to section 5.11.2 for details.

* env.store.dir
* key.store
* trust.store

### Customise consumer properties file

This property file has very few properties that must be configured for your deployment environment. Ensure that the **env.baseURI** property points to a URI where a provider (DIRECT or BROKERED) is deployed. Also ensure that the **adapter.use.https** is set to ‘true’ or ‘false’ depending if you connect via http or https to your provider. You can find a detailed description of all properties in the consumer property file in section 8.

#### Properties for Demo only

To run the demo and test classes of the framework you need to ensure that the **test.tempDir.output** property points to a valid location. If your consumer subscribes to events the demo classes write the event data to a log file. This property indicates where these log files will be located.

### Customise provider property file

There are a few properties that must be configured so that consumers can connect to the provider as well as the provider can connect to the broker if it operates in a brokered environment. The list below illustrates what these properties must be:

* **env.baseURI**: Only used in a brokered environment. Must point to the URI of the broker.
* **env.connector.url**: This is the HTTP URI where your provider is deployed. This is also the URI that will be used in all infrastructure service URIs of the environment if your provider is a DIRECT provider (see section 5.3.1.1.3 for details).
* **env.connector.url.secure**: This is the HTTPS (secured & encrypted transport) URI where your provider is deployed. This is also the URI that will be used in all infrastructure service URIs of the environment if your provider is a DIRECT provider (see section 5.3.1.1.3 for details).

**Note:**

If your provider is a DIRECT provider then all consumers that wish to connect to your provider must have the value of the **env.connector.url** (HTTP) or the **env.connector.url.secure** (HTTPS) appended with ‘environments/environment’ in their **env.baseURI** property of their consumer’s property file.

For example if you deploy your provider using the domain ‘mybroker.com.au’ on port 443 (standard https port) then the env.baseURI property in the consumer’s property file should be something like this:

env.baseURI=https://mybroker.com.au/sif3/environments/environment

The ‘sif3’ part may or may not be there, or might be called something different, depending what you have your servlet mapping configured in the web.xml of your provider (see section 6.6 and/or 6.8 and 6.9), or how your proxy may map things to your environment. Also the port number 443 might be required. Since it is the standard https port most proxies would not need that port as part of the URI.

#### Properties for Demo only

For the demo provider to work it reads student and school data from a file rather than a database. The property provider.student.file.location and provider.school.file.location in the provider’s property file point to a location where there is a file with appropriate data. Ensure that you point to a correct location. The test data files can be found in the ‘TestData/xml/input’ directory of the SIF3 Framework.

### Customise log4j.properties file (optional)

The log4j.properties that is part of this SIF3 Framework has a path to a log file. Ensure that you change this to a location where you want the log files to be written to.

## 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 few open-source web- and application containers. A similar approach is suggested for any other web- or application container that are not listed in these sections such as Glassfish or any commercially available web-/application containers.

## Tomcat 6 or below, Jetty 7 or below

Please note that Tomcat 6.x or lower as well as Jetty 7.x or lower is **no longer supported** by the SIF3 Framework because the framework requires servlet 3.0 support. Servlet 3.0 support is only available in Tomcat 7 and above or Jetty 8 and above.

## Tomcat 7 and above or Jetty 8 and above Deployments

If you deploy into Tomcat **7 and above** or **Jetty 8 and above** then you must use the Jersey JAX-RS implementation (Tomcat and Jetty do 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.

Secondly you need to indicate which servlet to use for all request processing. You have two choices. The first choice is to specifically list the Jersey Servlet while the second choice is the standard default JAX-RS servlet notation (preferred). Add one of the two choices below to your web.xml:

**Option 1: Use Jersey specific Servlet**

<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>

**Option 2 (preferred): Use standard JAX-RS Servlet**

<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 implementation is provided (in this case the Jersey implementation).

## JBoss AS 6 and above Deployments

JBoss AS 6 and above supports the servlet 3.0 specification. Because JBoss has its own JAX-RS implementation (RESTEasy) one must ensure that the web.xml has the correct servlet notation as well as avoiding the inclusion of some Jersey libraries in the final deployment unit. This section highlights what to look out for if JBoss AS 6 and above is used. Please also refer to section 6.1 for details about the servlet 3.0 specification.

**Servlet to Use:**

Ensure that your web.xml indicates the correct JAX-RS servlet to be used. The following lines must be in you web.xml:

<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 implementation is provided (in this case the JBoss’s RESTEasy implementation).

**Avoid Jersey JAX-RS libraries in your deployment unit:**

After you configured your web.xml as outlined above you must also stop JBoss to use the Jersey Libraries that implement the JAX-RS. 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.jar and jersey-core.jar 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. Other typical libraries you may need to **remove** as they form part of JBoss’s implementation are **jaxb-impl.jar** and **log4j.jar**. Remove them from the war/WEB-INF/lib folder.

Build your war or ear file and deploy them to appropriate location of JBoss. Ensure that all the required config files as listed in section 6.3 are part of your classpath and start JBoss. Now you should be ready to go.

## Verify Consumer & Provider Property Files

In section 6.7 and 6.8 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 value in your servlet mapping.

# Appendix A – Environment Properties File

The ‘**environment.properties**’ file holds some high level information on where your Environment Templates are stored (see section 5.3.1.1 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. Please refer to section 5.11.1 and 5.11.2 for details on how to configure HTTPS for consumers and providers.

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.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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. |
| **key.store** | This is the physical location where your Java keystore is located.1) |
| **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. 1) |
| **trust.store.password** | This is the password to use to access the Java Trusted Certificate Authority store. |

1)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.

# Appendix B – Consumer’s Property File

Each consumer has a property file that must be part of the consumer’s classpath. A consumer is initialised with this property file as part of the ConsumerLoader class (see section 5.5.2.1 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 usage description 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.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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.noCertificateCheck** | This property allows the consumer to NOT check certificates against the trusted store. This can be useful during development where self-signed certificates are used. In a proper TEST or PROD environment this property should not be enabled. Valid values are true (don’t check certificates) and false (standard behaviour where certificates are checked/validated). If the property doesn’t exist then it is defaulted to false.  **Default: false** (check and validate certificate against the trusted store) |
| **adapter.checkACL** | Turn on (true) or off (false) ACL check on the consumer. 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).  **Default: true** |
| **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.  **Default: 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. |
| **adapter.generator.id** | This value is put into the HTTP header field called “generatorId” for each request to an object provider. This is an optional property that may help the object provider to determine where the request originated from. If not present or left empty then the “generatorId” HTTP header field won’t be populated. |
| **adapter.compression.enabled** | Indicate if this adapter should use compression for the payloads (true). If this property is set to true then the all calls will compress (gzip) each payload first before it is sent to the provider. It will also indicate to the provider that it can accept compressed payloads in the response.  **Default: false** |
| **env.xml.file.name** | The name of the environment template file to be used with this consumer. 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 5.3.1.1.2. |
| **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.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.authentication.method** | The authentication method to be used for this environment. Valid values are ‘Basic’, ‘SIF\_HMACSHA256’ or ‘Bearer’ (case sensitive!). 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. For ‘Bearer’ see section 5.10.3 for details.  **Default: Basic** |
| **adapter.security.service** | This property is used in conjunction with External Security Services. It must hold a fully qualified class name of a concrete implementation of the AbstractSecurityService class. Details about external security services and the use of this property can be found in section 5.10.3 and more specifically in section 5.10.3.15.10.2.1 and 5.10.3.2. |
| **adapter.hbr.propertyClass** | If hibernate properties shall be injected programmatically to the initialisation of the framework then this property must hold a fully qualified class name of a class that implements the HibernateProperties interface. See section 5.13 for details on the usage of this property and hibernate property injection in general. |
| **env.userToken & env.instanceId** | 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. |
| **security.service.property.<…>** | All properties with that prefix are passed to the OtherInfo element in the TokenCoreInfo class for the generateToken() method of the AbstractSecurityService class. See section 5.10.3.1 for details about this set of properties. |
| **env.mediaType** | The media type used by this consumer in this environment. Can be **XML** or **JSON**. It indicates the default mime type of all OBJECT Service and Infrastructure Service and requests that are sent to a provider and/or broker.  **Default: XML** |
| **env.mediaType.charset** | This property is used to set the real encoding used with the media type. The charset encoding value, if set, will be added to the media type of the “accept” and “content-type” http headers (i.e. application/xml; charset=UTF-8). If it is not set then no charset encoding will be added to the media type of the corresponding http headers.  Example: env.mediaType.charset=UTF-8. |
| **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 something like http(s)://…/environments/environment but it doesn’t have to be. |
| **env.create.conflictIsError** | This property is specific to the framework to indicate what to do if a HTTP Status of 409 is returned when an environment is created. The 409 is returned if an environment already exists and an attempt to create it again is made. The SIF Specification is not clear how to deal with this HTTP Status from a consumer’s point of view. Some environment providers my return a proper error message in the response payload, others may return the already existing environment in the return payload. This property indicates how to deal with the HTTP Status 409. By default it treats it as a proper Error (true) but if that behaviour shall be changed and the HTTP Status 409 shall be treated as a valid status with the response payload being a proper environment then this property can be changed to ‘false’.  **Default: true** |

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.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **events.enabled** | Indicate if event processing shall be enabled. Even if an environment supports events, this consumer might not be interested in any events. This is useful to turn off events on a consumer (i.e. temperately for some testing).  **Default: false** (don't subscribe to events) |
| **events.queue.strategy** | SIF 3.x 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 a consumer is created. Future expansions can be 'ZONE\_LEVEL' (One queue per zone), SERVICE\_LEVEL (one queue per service in each zone) etc. Please refer to section 5.5.3.1for more information about queue strategies.  **Default: ADAPTER\_LEVEL** |
| **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. 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.  **Default: Same as adapter.id with all spaces removed and a pre-fix of ‘EVENT\_’.** |
| **events.queue.type** | There are two ways how a consumer can receive messages from a queue. IMMEDIATE or LONG. Refer to SIF 3.x specification for details on the two options. This is the same behaviour as known in SIF 2.x for Pull agents.  **Default: IMMEDIATE** |
| **events.queue.subscribers** | SIF 3.x allows concurrent subscribers/readers on each queue. This property indicates the number of subscribers/readers per queue. Care must be taken if this number is more than 1 as it could mean that events are processed out of order! The environment provider might override that value with a lesser value. Please refer to section 5.5.3.2 and 5.5.3.4 for some more details on the frameworks multi-threading and configuration.  **Default: 1** |
| **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: 60** (once per minute) |
| **events.longPolling.timeout** | Number of seconds the connections shall remain open for LONG\_POLLING queues before it is closed. See SIF 3.x Spec for more details about LONG\_POLLING queues.  **Default: 120** (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: false** (subscriptions shall remain when the consumer shuts down) |

The next set of properties relate to delayed response processing only. Please note while you can set these values it is not guaranteed that delayed responses are being supported by the provider or the environment. You must consult with the vendor that provides you with an implementation of an environment provider if delayed responses are supported. Often in DIRECT environments delayed response functionality is unlikely to be available. In a BROKERED environment it is expected that delayed response functionality is available.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **delayed.enabled** | Indicate if delayed response processing shall be enabled. Even if an environment supports delayed responses, this consumer might not make use of this.  **Default: false** (don't enable delayed responses) |
| **delayed.queue.strategy** | SIF 3.x allow many ways on how to create queues and direct delayed responses to them. This framework only supports a set of the most common strategies. Currently only **'ADAPTER\_LEVEL'** is implemented which mean a single delayed response queue for a consumer is created. Future expansions can be 'ZONE\_LEVEL' (One queue per zone), SERVICE\_LEVEL (one queue per service in each zone) etc. Please refer to section 5.5.3.1for more information about queue strategies.  **Default: ADAPTER\_LEVEL** |
| **delayed.queue.name** | This property is used give the delayed response 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. 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.  **Default: Same as adapter.id with all spaces removed and a pre-fix of ‘DELAYED\_’.** |
| **delayed.queue.type** | There are two ways how a consumer can receive messages from a queue. IMMEDIATE or LONG. Refer to SIF 3.x specification for details on the two options. This is the same behaviour as known in SIF 2.x for Pull agents.  **Default: IMMEDIATE** |
| **delayed.queue.subscribers** | SIF 3.x allows concurrent subscribers/readers on each queue. This property indicates the number of subscribers/readers per queue. Care must be taken if this number is more than 1 as it could mean that responses are processed out of order! The environment provider might override that value with a lesser value. Please refer to section 5.5.3.2 and 5.5.3.4 for some more details on the frameworks multi-threading and configuration.  **Default: 1** |
| **delayed.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: 60** (once per minute) |
| **delayed.longPolling.timeout** | Number of seconds the connections shall remain open for LONG\_POLLING queues before it is closed. See SIF 3.x Spec for more details about LONG\_POLLING queues.  **Default: 120** (2 minutes). |
| **delayed.subscriptions.removeOnShutdown** | **NOT USED YET.**  **Default: false** |

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 5.5.2.2 for details).

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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 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). Please refer to section 5.5.3.3 and 5.5.3.4 for some more details on the frameworks multi-threading and configuration.  **Default: 1** |

The next three properties are a feature of this framework. It allows a consumer to connect to an existing environment without creating it from scratch. There are situations where an environment has been created and initialised by a 3rd party and this framework shall only connect to it with a given set of credentials.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **env.use.existing** | This property indicates if an existing environment on the environment provider shall be used. If it is set to TRUE then the framework will attempt to connect to the environment defined with the next two properties without creating it.  **Default: false** |
| **env.existing.sessionToken** | The session token to use with a pre-existing environment |
| **env.existing.environmentURI** | The full URI of the pre-existing environment. This URI holds the environment ID as part of the URI. This will be the same URI as given by the environment provider in the “environment” infrastructure service. |

**Custom Properties**

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

# Appendix C – Provider’s 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. It is important to note that some properties are only applicable for providers if they are DIRECT providers, other properties are only applicable for providers in a BROKERED environment and some properties are applicable for either environment type. All available properties are listed below, categorised based on the environment type the provider operates in.

The properties below are applicable for DIRECT and/or BROKERED environments.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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.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.  **Default: true** |
| **adapter.generator.id** | This value is put into the HTTP header field called “generatorId” for each **event** message that is published. This is an optional property that may help the consumer to determine from which object provider instance the event originated from. If not present or left empty then the “generatorId” HTTP header field won’t be populated. |
| **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 June 2015 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.  **Default: false** |
| **env.mediaType** | The media type used with this environment. Can be XML or JSON. It indicates:   * What mime type to use for Infrastructure Services * Mime type to use when publishing events   Note: The provider can respond to consumer’s request in the mime type indicated by the consumer. This property is not used to override that behaviour.  **Default: XML** |
| **env.mediaType.charset** | This property is used to set the real encoding used with the media type. The charset encoding value, if set, will be added to the media type of the “accept” and “content-type” http headers (i.e. application/xml; charset=UTF-8). If it is not set then no charset encoding will be added to the media type of the corresponding http headers. Note that it will be used for sending events and not for responding to requests since the consumer will tell the provider what encoding to use.  Example: env.mediaType.charset=UTF-8. |
| **adapter.audit.service** | This property holds the fully qualified name of the audit service implementation class to be used by this adapter. The class listed here must implement the Auditor interface. If no class is listed then it is assumed that audit logging is not enabled. Please refer to section 5.12 for details on how to implement, enable and disable auditing on an object provider. |
| **adapter.hbr.propertyClass** | If hibernate properties shall be injected programmatically to the initialisation of the framework then this property must hold a fully qualified class name of a class that implements the HibernateProperties interface. See section 5.13 for details on the usage of this property and hibernate property injection in general. |
| **env.create.conflictIsError** | In a DIRECT environment this property indicates if a 'conflict' or HTTP Status of 409 for a create environment shall create an error payload (true) or if it shall be considered a valid state (false). If it is considered a valid state then the payload of the response will hold the environment XML/JSON that is in conflict with the new create environment request. In a BROKERED environment the usage of this property is identical to the same property in the consumer’s properties file (see section 8).  **Default: true** (treat as error and return an error message in the response payload). |
| **adapter.custom.response.headers** | A set of custom HTTP Headers that are added to every response. If the name is one of the SIF 3.x pre-defined HTTP header names (i.e. timestamp, responseAction etc) then they will be overridden by the framework and the value in this list will be ignored. The various headers are separated by '|' and the value of a header is listed after the ':'. If a HTTP Header shall have more than one value then they are listed as a comma separated list.  EBNF notation: **<hdrName>:<value>{,<value>}{|<hdrName>:<value>{,<value>}}**  Example: Header1:value1,value2|Header2:value3|Header3:value4,value5  If this property is not set then no custom headers will automatically be added to the response. |
| **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 if events are enabled. These threads are started with a little delay between them. This property allows overriding that value. The delay is in seconds.  **Default: 10** |

The next set of provider properties relate 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.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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 in seconds. If a different delay than the default 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 than 5 seconds it will be defaulted to 5 seconds.  **Default: 5** |
| **event.maxObjects** | This property indicates what the maximum number of objects per SIF Event message should be. The default (10) 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.  **Default: 10** |

**BROKERED Environment Properties**

The following properties are only used if the provider is a ‘**BROKERED**’ provider. If the provider is a ‘DIRECT’ provider then these properties are either configured in the SIF3\_APP\_TEMPLATE and SIF3\_ENV\_TEMPLATE tables (see also section 5.3.1.3 for details) or are not applicable for a DIRECT environment at all.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **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 under the ‘brokered’ sub-directory and it must have a certain structure. Details can be found in section 5.3.1.1.3. For DIRECT environment this property is not used. It uses SIF3\_APP\_TEMPLATE table instead. |
| **env.authentication.method** | The authentication method to be used for this environment. Valid values are ‘Basic’, ‘SIF\_HMACSHA256’ and ‘Bearer’ (case sensitive!). 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. If ‘Bearer’ is used it is assumed that an external security service is used (see section 0 for details about external security services).  **Default: Basic** |
| **env.application.key** | The application key this provider **MUST** use. 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. The provider uses this 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. |
| **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.noCertificateCheck** | This property allows the provider to NOT check certificates against the trusted store. This can be useful during development where self-signed certificates are used. In a proper TEST or PROD environment this property should not be enabled. Valid values are true (don’t check certificates) and false (standard behaviour where certificates are checked/validated). If the property doesn’t exist then it is defaulted to false.  **Default: false** (check and validate certificate against the trusted store) |
| **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. |
| **adapter.compression.enabled** | Indicate if this adapter shall use compression (true) for when communicating with the broker (i.e. events). If this property is set to true then all payloads will be compressed (gzip) first before they are sent to the broker. It will also indicate to the broker that it can accept compressed payloads in the response.  **Default: false** |

The next three properties are a feature of this framework. It allows a provider when connecting to a broker to use an existing environment without creating it from scratch. There are situations where an environment has been created and initialised by a 3rd party and this framework shall only connect to it with a given set of credentials.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **env.use.existing** | This property indicates if an existing environment on the environment provider (broker) shall be used. If it is set to TRUE then the framework will attempt to connect to the environment defined with the next two properties without creating it.  **Default: false** |
| **env.existing.sessionToken** | The session token to use with a pre-existing environment |
| **env.existing.environmentURI** | The full URI of the pre-existing environment. This URI holds the environment ID as part of the URI. This will be the same URI as given by the environment provider in the “environment” infrastructure service. |

**DIRECT Environment Properties**

The following set of properties are only used for providers that are setup as **DIRECT** providers. They are ignored if the provider is a Brokered provider.

|  |  |
| --- | --- |
| **Property Name** | **Usage** |
| **adapter.security.service** | This property is used in conjunction with External Security Services (consumers use authentication method ‘Bearer’ instead of Basic or SIF\_HMACSHA256). It must hold a fully qualified class name of a concrete implementation of the AbstractSecurityService class. Details about external security services and the use of this property can be found in section 5.10.2and more specifically in section 5.10.2.1 and 5.10.2.2. |
| **adapter.default.accessToken.authentication.method** | This property is only used in a **DIRECT** environment and if a consumer uses SIF Express to invoke a service on the direct provider. SIF Express allows a service to be invoked with without the HTTP Authorization header but an ‘accessToken’ URL Query Parameter. In such a case the authentication method is assumed to be ‘Bearer’. To configure your direct provider to assume a different authentication method as its default this property can be set. Valid values are ‘Basic’, ‘SIF\_HMACSHA256’ and ‘Bearer’ (case sensitive!).  **Default: Bearer** |
| **adapter.authTokenOnURL.allowed** | SIF xPress allows the authentication token to be provides on the URL with the access\_token URL query parameter. This is not considered very secure and some implementations want to turn that option off. Setting this property to 'false' will disable this option. If it is set to 'true' then the access\_token URL query parameter is accepted by this adapter.  **Default: false** |
| **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. |
| **env.allow.autoCreate** | Valid values are true and false. This property is only used with external security services and if the consumer’s authentication method is ‘**Bearer**’. It Indicate if environment shall be created automatically (true) if requests are received for non-existent environments.  **Default: false** |

**Custom Properties**

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

1. Available at: <http://specification.sifassociation.org/Implementation/Infrastructure/3.2/Documents/FunctionalServices_3-2.pdf> [↑](#footnote-ref-1)
2. See <http://lepus.org.uk> for a more thorough account of Codecharts. [↑](#footnote-ref-2)