(.NET)

Version

**Author:** ,

**Revision:** 1.00 (draft)

**Published:** May 2016

Copyright © 2016,

Table of Contents

1. Introduction 3

1.1. Target audience 3

1.2. Scope 3

1.3. History 3

1.4. Document & Framework History 4

2. Configuring an Environment 8

3. Starting the services 8

3.1. Starting the Environment Provider 8

3.1.1. Via Script 8

3.1.2. Via Visual Studio 9

3.2. Starting the Service Provider 9

3.2.1. Via Script 9

3.2.2. Via Visual Studio 9

3.3. Starting the Service Consumer 9

3.3.1. Via Script 9

3.3.2. Via Visual Studio 10

4. Reviewing expected behaviour 10

5. Reviewing HTTP messages in Fiddler 10

5.1. Filtering for SIF messages 10

6. Explaining service choreography 11

7. Transitioning from the demo 11

7.1. Configuring an Environment 11

7.2. Implementing an Object Service Provider 11

7.3. Implementing a Service Consumer 13

7.3.1. Service Consumer properties 13

7.4. Implementing a Functional Service Provider 14

7.5. Implementing a Functional Service Consumer 16

8. Running the demo over a LAN 17

8.1. Configure local IIS Express instance 17

8.2. Grant remote access 17

8.3. Configure firewall access 17

8.4. Configure Visual Studio 21

8.5. Configure the StudentPersonal Consumer 21

8.6. Configure the Environment definition 21

9. Code documentation 22

# Introduction

The demo projects included with the SIF 3.0 Framework were developed to illustrate framework usage. They were designed to run out of the box with minimal configuration. From an understanding of these projects, a developer should be able to create simple Service Consumers and Object Service Providers relatively quickly.

The demo projects are broken into projects for the Australian locale (AU) or the American locale (US). Demo projects for the UK locale have yet to be implemented. When following the instructions for running the demo projects, ensure the appropriate projects are used.

This document describes various technical aspects of the demo projects including (but are not limited to):

* Create and configuring an Environment
* Starting the services, including the Environment Provider and Object Service Providers
* Run a Service Consumer
* Reviewing expected behaviour
* Explaining service choreography
* Transitioning from the demo to a working application

## Target audience

The intended audience for this document are developers who want to get hands-on experience with SIF 3.0 development in .NET. A basic understanding of SIF 3.0 concepts and terminology would be beneficial for understanding how the demo projects interact.

## Scope

The scope of this document is to provide instructions on how to run the demo projects. It is outside the scope of this document to describe the SIF 3.0 specification or the underlying technologies used by the framework. It is expected that before attempting to run the demo projects, the developer has read the Sif3Framework .NET Developer’s Guide.

## History

The SIF3 Framework is a basic .NET Framework intended to help developing SIF 3.x Services/Adapters in an efficient manner.

[Systemic Pty Ltd](http://www.systemic.com.au) implemented the first version (0.1.0) in January 2014. Functionality has been added incrementally and can be seen from the Document & Framework History below.

ZiNET Data Solutions Limited implemented the Functional Services functionality as part of a Department For Education (DfE, UK) project in April–July 2016.

## Document & Framework History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 0.1.0 | Jan 28, 2014 | R. Rafiq | Beta release to be used for collaborative review only and not intended for commercial use. |
| 0.2.0 | Jan 29, 2014 | R. Rafiq | Submitted a partial implementation of the POST action for the Environments Controller. |
| 0.3.0 | Feb 03, 2014 | R. Rafiq | Update the POST action of the EnvironmentsController to authenticate a user and return an appropriately populated Environment response. |
| 0.4.0 | Feb 06, 2014 | R. Rafiq | Completed implementation of the GET action for an environment.  Fixed an issue with authentication based upon session token |
| 0.5.0 | Feb 15, 2014 | R. Rafiq | Tweaked the overall design of the core code to properly enable the implementation of the DELETE action for an Environment. |
| 0.6.0 | Feb 20, 2014 | R. Rafiq | Redesigned the persistence layer (repositories) to add the flexibility to inject a different SessionFactory.  Prepared the code for a demo provider. |
| 0.7.0 | Mar 12, 2014 | R. Rafiq | Added a demo provider project to help illustrate how the framework can be used to provide StudentPersonal data.  Re-factored some code as a result of this work. |
| 0.7.1 | Jul 16, 2014 | R. Rafiq | Minor updates to documentation within the code, including code documentation. |
| 0.8.0 | Aug 05, 2014 | R. Rafiq | Extract demo projects into separate Solution.  Extract data model and infrastructure projects into separate Solution.  Fixed issue with BaseController.  Changed constructors in the persistence, Service and Controller layers to accommodate Dependency Injection.  Updated in-code documentation.  Updated SessionFactorys to cater for use as an executable as well as for deployment to IIS.  Updated version of Web API and SQLite.  Added NHibernate configuration and DDLs for use with SQLite, SQL Server LocalDB, SQL Server, Oracle and MySQL.  Added a project make setting up the demo database easier.  Added demo databases for SQLite and SQL Server LocalDB.  Renamed the Sif.Framework.Core project to Sif.Framework. |
| 0.9.0 | Aug 06,2014 | R. Rafiq | Added assembly information (including version number) to each project assembly.  Updated README.md to provide a more comprehensive version control history.  Recompiled and re-referenced libraries in SharedLibs.  Better organised the Scripts directory. |
| 0.10.0 | Aug 07, 2014 | R. Rafiq | - Fixed issues with referencing of Sif.Specification.Infrastructure assembly.  Renamed StudentPersonal.cfg.xml to Demo.cfg.xml to make the file name less specific.  Fixed issue of incorrectly referenced Sif.Framework assembly in the Demo Provider.  Added scripts to ease demo execution. |
| 0.10.1 | Aug 19, 2014 | R. Rafiq | Added a draft version of the Developer's Guide.  Added a draft version of the Demo Usage Guide. |
| 0.11.0 | Aug 27, 2014 | R. Rafiq | Added a generic Consumer to the framework.  Added a utility class for HTTP operations.  Created a new StudentPersonal demo Consumer.  Updated .gitignore so that "x64" directories are no longer ignored (caused problems with SQLite DLLs).  Re-ordered the projects listed in the VS Solutions to manage the default projects run when debugging. |
| 0.12.0 | Aug 30, 2014 | R. Rafiq | Upgraded the framework to use SIF Infrastructure 3.0.1.  Updated the data models to the latest version of the SIF AU 1.3 Data Model.  Re-designed the XML serialisation code to provide for better extensibility.  Fixed an issue whereby the root element of collections returned by Controllers started with "ArrayOf".  Made enhancements to the demo Setup. |
| 0.13.0 | Aug 31, 2014 | R. Rafiq | Fixed an issue with clean-up if a Consumer fails to register with the Environment Provider.  Fixed an error in the returned Environment object on Consumer register.  Added exception handling to the demo Consumer to ensure proper clean-up after an error. |
| 0.13.1 | Sep 01, 2014 | R. Rafiq | Updated the Developer's Guide from user feedback.  Updated the Demo Usage Guide to include instructions for implementing a Consumer. |
| 0.13.2 | Sep 01, 2014 | R. Rafiq | Updated the Demo Usage Guide to include instructions for running the demo over a LAN. |
| 0.14.0 | Sep 11, 2014 | R. Rafiq | Created a new Solution to contain a reference implementation of the SBP (only partially implemented).  Upgraded Web API to version 5.2.2 on all appropriate projects.  Added a SchoolInfo Consumer and Provider to the demo Solution. |
| 0.15.0 | Oct 17, 2014 | R. Rafiq | Upgraded NHibernate to version 4.0.1.4000 on all appropriate projects.  Upgraded SQLite to version 1.0.94.0 on all appropriate projects.  Added log4net to some projects.  Added debug statements in some projects using log4net.  Added error messages to the payload of GenericController response messages.  Changed the existing demo Consumer and Provider projects to be AU locale specific.  Made the AU Consumer and Provider projects simpler by removing the use of a database for retrieving sample student data from.  Added new demo Consumer and Provider projects for the US locale.  Updated the demo Setup project to cater for the AU and US locales.  Added data models generated from the SIF US 3.2 XSDs to the Sif3Specification Solution.  Updated the demo execution batch scripts to cater for the new demo projects.  Updated the SharedLibs libraries. |
| 0.15.1 | Oct 20, 2014 | R. Rafiq | Added the Training Exercises (US) document. |
| 0.16.0 | Nov 15, 2014 | R. Rafiq | Implement the ability to handle payload-free POST requests for the EnvironmentsController (Simple SIF).  Upgrade the GenericConsumer to take a solutionId.  Improved logging of error messages.  Added the Training Exercises (AU) document. |
| 0.16.1 | Dec 22, 2014 | R. Rafiq | Added documentation to explain how to specify MIME Types using URL postfix extensions (Simple SIF).  Enabled URL postfix extensions for MIME Types in the demo AU Provider. |
| 0.17.0 | Jan 09, 2015 | R. Rafiq | Added exception classes to better manage exception handling and information.  Implemented global error handling guidelines for Web API 2 handlers and loggers.  Added utility classes to help collate and extract error information from response messages.  Updated the ConsumerApp to display more meaningful error details.  Updated the EnvironmentsController to support better error messages. |
| 0.17.1 | Jan 10, 2015 | R. Rafiq | Based on feedback, deleted the SbpFramework Solution and instead incorporated its code into the Sif3FrameworkDemo Solution to reduce complexity and confusion.  Based on feedback, removed shared code (projects) from the Sif3FrameworkDemo Solution to better reflect implementations where Consumers and Providers are developed by different vendors. |
| 0.17.2 | Jan 26, 2015 | R. Rafiq | Fixed an issue introduced in version 0.17.0 whereby the demo AU Consumer referenced a non-existant file - SifFramework.brokered.config.  Updated the demo AU Consumer and Provider to better reflect exception handling and logging enhancements in the SifFramework library.  Updated the demo US Consumer and Provider to match the changes in the AU versions. |
| 0.18.0 | Mar 29, 2015 | R. Rafiq | Updated the GenericConsumer to allow registration to a SIF Broker, as well as (direct) to an Environment Provider.  Updated the GenericController to allow creation of Service Providers that can connect to a SIF Broker, as well as run (directly) as an Environment/Service Provider.  Added functionality for Consumers and Providers to store the session token (received after service registration) locally so that state can be maintained between Consumer and Provider sessions.  Improved exception handling and logging in the SifFramework library.  Updated the demo Consumers and Providers to reflect these changes. |
| 0.18.1 | Mar 30, 2015 | R. Rafiq | Updated the US Provider project with configuration changes that should have been made in the last submission.  Updated all documentation to reflect recent changes. Documentation on SIF Broker integration is still incomplete. |
| 0.19.0 | May 03, 2015 | R. Rafiq | Updated the service interface to facilitate paging of retrieved data.  Updated the GenericConsumer to make paged retrievals by default.  Updated the GenericController to handle (GET) requests for paged data.  Updated the demo Consumers and Providers to reflect these changes. |
| 0.19.1 | May 11, 2015 | R. Rafiq | Updated the SIF AU 1.3 data model of the Sif3Specification Solution. |
| 0.20.0 | May 17, 2015 | R. Rafiq | Updated the GenericConsumer to add a new Retrieve method that accepts an "example" object.  Updated the Get method of the GenericController to handle a payload when a method override is requested.  Updated the demo AU Provider configuration to manage redirection when a method override is requested.  Updated the demo AU Consumer with an example call that uses the new Retrieve method. |
| 0.20.1 | Jul 17, 2015 | R. Rafiq | Added a beta version of the SIF AU 1.4 data model to the Sif3Specification Solution. |
| 0.21.0 | Sep 23, 2015 | R. Rafiq | Updated the SIF AU 1.3 and 1.4 data models of the Sif3Specification Solution.  Updated all unit tests and (AU) demo projects to reference the updated 1.4 data models.  Fixed an issue with a missing namespace on serialisation of data model collections.  Enhanced the demo AU Consumer to demonstrate connection with HITS. |
| 0.22.0 | Sep 28, 2015 | R. Rafiq | Updated the Framework service layer, Consumers and Providers to handle Service Paths.  Updated (AU) demo projects to demonstrate Service Path usage. |
| 1.0.0 | Jan 17, 2016 | R. Rafiq | Redesigned Consumer implementation to handle multiple object operations.  Redesigned Provider implementation to handle multiple object operations.  Updated AU and US demo projects to reflect mutliple object operations.  Updated documentation to reflect changes.  Fixed issue with mustUseAdvisory implementation. |
| 1.1.0 | Jan 29, 2016 | R. Rafiq | Upgraded Web API to version 5.2.3 on all appropriate projects.  Enhanced and configured the WebApi implementation to recognise Matrix Parameters.  Updated Consumers to pass Zone and Context with all requests using Matrix Parameters.  Updated Providers to handle receiving Zone and Context as Matrix Parameters.  Updated AU and US demo projects to reflect the use of Matrix Parameters. |
| 1.1.1 | Feb 03, 2016 | R. Rafiq | Added the SIF US 3.3 data model to the Sif3Specification Solution.  Updated US demo projects to use SIF US 3.3 model objects.  Added a Service Path exercise to the AU and US training exercises. |
| 1.1.2 | Feb 04, 2016 | R. Rafiq | Applied code change due to compiler error that occurs in VS`2013 but not VS 2015. |
| 1.2.0 | Jul ??, 2016 | ZiNet Data Solutions Limited | Extend to support UK data model 2.0  Implement UK demo provider/consumer projects  Implemented Functional Services  Provide demo of functional services in the UK demo projects  Revised the scripts to facilitate easier development and demo execution  Updated documentation to include functional services |

# Configuring an Environment

Before a Service Consumer and Object Service Provider can interact, an Environment must be created to manage the session information between them. Once an Environment has been defined, then the Service Consumer and Object Service Provider are able to register to that Environment. This registration is managed by an Environment Provider.

Environment definition generally falls under the domain of a SIF Administrator. However, creation of an initial Environment for this demo is performed by running one of the following scripts

* Scripts\BAT\Demo execution\DemoAuSetup.bat
* Scripts\BAT\Demo execution\DemoUkSetup.bat
* Scripts\BAT\Demo execution\DemoUsSetup.bat

These scripts use the Sif.Framework.Demo.Setup project to create and populate a demo database with an initial Environment definition. By default, the database used can be found under the *Data\Databases\SQLite* directory. An SQLite database is used so that no configuration changes are required to be able to run this demo out of the box.

**NOTE:** Before running the scripts mentioned above, the Sif.Framework.Demo.Setup project needs to be built for you system. This can be done within Visual Studio or headless by running *Scripts\BAT\Code Generation\CompileDemos.bat* [[1]](#footnote-1)

All mandatory information associated with the applications (e.g. applicationKey, sharedSecret) have been predefined for these demos.

If this demo needs to be run against a different database, then the SifFramework.cfg.xml file (in all demo projects) needs to be updated accordingly.

# Starting the services

These demos are made up of 3 components; a consumer of data, a provider of data and an Environment Provider. The consumer is a standalone application, whereas both providers are Web API services that require IIS.

For these demos, the providers are run using IIS Express from within Visual Studio 2015.

## Starting the Environment Provider

The Environment Provider can be launched from Visual Studio or via a script.

### Via Script

The environment provider can be started by running *Scripts\BAT\Demo execution\EnviromentProvider.bat*

### Via Visual Studio

To do so, load the Sif3Framework.sln Solution into Visual Studio and simply run it (by clicking on the Internet Explorer button shown below).

Internet Explorer button

If the Internet Explorer button is not visible, it will be necessary to make the Sif.Framework.EnvironmentProvider project of the Solution the single start-up project.

The Sif.Framework.EnvironmentProvider project has been configured to run in IIS Express on a particular port (as specified in the project properties). This port is referenced in the Environment configured from the previous section.

For the moment, ignore the “HTTP Error 403.14 – Forbidden” message that appears in the popped up Internet Explorer window on start-up. As this is a Web API service, the fact that it does not have a home page is not relevant for this demo.

## Starting the Service Provider

The Service Provider can be launched from Visual Studio or via a script.

### Via Script

The service provider can be started by running one of the following scripts:

* Scripts\BAT\Demo execution\DemoUkProvider.bat
* Scripts\BAT\Demo execution\DemoUsProvider.bat

### Via Visual Studio

To do so, load the Sif3FrameworkDemo.sln Solution into Visual Studio and simply run it (by clicking on the Internet Explorer button shown below). By default, the Sif.Framework.Demo.Au.Provider has been configured as the start-up project. This can be changed if the US (Sif.Framework.Demo.Us.Provider) or UK (Sif.Framework.Demo.Uk.Provider) locale is to be used.

Internet Explorer button

As with the Sif.Framework.EnvironmentProvider, the Sif.Framework.Demo.Au.Provider, Sif.Framework.Demo.Uk.Provider and Sif.Framework.Demo.Us.Provider projects have been configured to run in IIS on particular ports.

## Starting the Service Consumer

The Service Provider can be launched from Visual Studio or via a script.

### Via Script

Once the providers have been successfully started, run one of the following scripts to start a sconsumer:

* Scripts\BAT\Demo execution\DemoAuConsumer.bat
* Scripts\BAT\Demo execution\DemoUsConsumer.bat

These scripts run either the Sif.Framework.Demo.Au.Consumer, Sif.Framework.Demo.Uk.Consumer or the Sif.Framework.Demo.Us.Consumer project (respectively) of the Sif3FrameworkDemo.sln Solution.

### Via Visual Studio

To do so, load the Sif3FrameworkDemo.sln Solution into Visual Studio and simply run the desired consumer by right clicking and selecting Debug > Start new instance.

# Reviewing expected behaviour

The scenario illustrated in this demo is one where a consumer is making a call for student data in a direct environment. As such, once this data has been consumed, then the demo has been completed.

In this case, the Service Consumer simply prints the student details to the console in XML format. Once the details have been printed, the Object Service Provider instances can be stopped.

If you wish to inspect the messages passed between consumer and provider please use a tool such as Fiddler, further help on setting this up is found at <https://github.com/ZiNETHQ/SIF3DiagnosticTools>.

# Explaining service choreography

The Environment Provider manages the interaction between the Service Consumer and Object Service Provider. The workflow for this interaction is as follows:

1. The Service Consumer makes a request to create an Environment;
2. Using the response details returned, the Service Consumer makes a request to get all student records from the Object Service Provider; and
3. The Service Consumer makes a request to remove the created Environment.

The functional service demonstraton (UK locale only) follows a similar pattern, but demonstrates actions on phases as well as states within a job:

1. The Consumer makes a request to create an Environment;
2. Using the response details returned, the Consumer makes a request to create a new functional service instance (a new job);
3. Using the created job object the Consumer will send data to each of the three defined phases:
   1. “default” accepts and responds with text/plain data. Some operations on this phase have been disabled by the job’s ACL and are expected to fail.
   2. “xml” accepts applicaton/xml data and responds with text/plain data. The consumer will send an XML formatted LearnerPersonal object (UK Data Model), which is processed by the phase and the name of the learner returned.
   3. “json” accepts applicaton/json data and responds with text/plain data. In the same way as the “xml” phase, the consumer will send a JSON formatted LearnerPersonal object (UK Data Model), which is processed by the phase and the name of the learner returned.
4. The Consumer will then update the status of the “json” phase, indicating that it has failed;
5. The Consumer will then delete the job object and create 5 more in a batch operation;
6. On receiving the refIDs of the created job objects the Consumer will attempt a batch delete operation fo 4 of them plus a fake job object’s ID. This is expected to succeed for the 4 existing job objects and fail (with an appropriate error response) on the 5th;
7. The provider will time out the remaining job once its defined timeout period has expired; and
8. The Consumer makes a request to remove the created Environment.

# Transitioning from the demo

## Configuring an Environment

As mentioned previously, the management of Environments generally falls under the domain of SIF Administrators. However, projects that utilise a direct environment may not be large enough to justify the need for a SIF Administrator. In-lieu of one, this framework provide tools from within the Sif.Framework.Demo.Setup project to manage Environment definitions.

The DatabaseCreator.cs class puts an Environment definition into the framework’s database using Environment template files in XML format. The XML format matches that of the Environment definition outlined in the SIF 3.0.1 specification. The template files used for this demo can be found in the Sif.Framework.Demo.Setup project under the *Data files* directory.

## Implementing an Object Service Provider

To implement an Object Service Provider, a Web API Visual Studio Project needs to be created. When adding a Controller to the project, it must extend Provider (or BasicProvider) and pass in an appropriate IProviderService (or IBasicProviderService) implementation. In terms of SIF 3.0, the Controller effectively becomes an Object Service Provider.

Both BasicProvider and IProviderService classes use generics and require an appropriate object model as the type. The object model defined has to be a SIF Object as the framework serialises this object when communicating with a Service Consumer. If the object provided is not a SIF Object, it will not be recognised.

As a guide, the model objects contained in the Sif3Specifiation Solution can be used to create a model object appropriate for the BasicProvider and IProviderService classes. The model objects from this project were generated from the SIF 3.0 specification (defined in XML Schema) and would therefore be compatible with the XML produced. The model objects contained in the demo projects were created in this manner.

The NHibernate library is an inherent part of the framework, and the inclusion of the SifFramework.cfg.xml file is mandatory otherwise the Environment Provider will not be able to manage the Environment session for an Object Service Provider. Both the Object Service Provider and the Environment Provider need to reference the same database.

Please note that due to the different methods the .NET Framework uses to XML serialise objects, the Global.asax.cs file that comes with the Project needs to be updated to specify a particular serialisation method. This is achieved by the following lines of code:

XmlMediaTypeFormatter formatter =

GlobalConfiguration.Configuration.Formatters.XmlFormatter;

formatter.UseXmlSerializer = true;

Please note that due to the default behaviour of the inherent XML serialiser used by Web API, the additional lines of code below are also required in the Global.asax.cs. Without these lines, a StudentPersonalsProvider would return a list of student records with a root element of <ArrayOfStudentPersonal> instead of the required <StudentPersonals>. A similar entry would need to be added for the SIF Object type handled by other Controllers (such as the K12StudentsController).

XmlRootAttribute studentPersonalsXmlRootAttribute = new XmlRootAttribute("StudentPersonals") { Namespace = SettingsManager.ProviderSettings.DataModelNamespace, IsNullable = false };

ISerialiser<List<StudentPersonal>> studentPersonalsSerialiser = SerialiserFactory.GetXmlSerialiser<List<StudentPersonal>>(studentPersonalsXmlRootAttribute);

formatter.SetSerializer<List<StudentPersonal>>((XmlSerializer) studentPersonalsSerialiser);

For an example of other additional settings required, refer to the Global.asax.cs file of the Sif.Framework.Demo.Au.Provider project.

Lastly, the following third-party libraries are required by the Project for the framework to operate:

* AutoMapper (3.1.1)
* NHibernate
* Log4net
* Sif.Framework
* Sif.Specification.Infrastructure

## Implementing a Service Consumer

To implement a Service Consumer, the Consumer (or BasicConsumer) class needs to be extended. As with the Provider, it uses generics and requires a SIF Object as the type. It is then a matter of implementing an executable that will call the necessary methods of the Consumer (e.g. to return a list of students).

To instantiate an instance of a Consumer, certain information is required to help associate the Consumer with an Environment (that will be created). The constructor which takes an Environment object has been provided so that all necessary information can be provided conveniently within a single object. The constructor which takes specific properties has been provided to help identify the minimum properties needed to associate the Consumer with an Environment.

With both constructors, a properties file provided for the Consumer can be used to default missing properties. The “template” properties defined below will be used if defined and not provided through the constructors.

### Service Consumer properties

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

#### consumer.environment.deleteOnUnregister

This property determines whether the Environment associated with the Service Consumer is deleted after the Service Consumer has been shut-down. If the Environment is deleted, then the next time the Service Consumer registers, a new session token will be created and returned to the Service Consumer. If the Environment is not deleted and an attempt to register the same Service Consumer is made, and error is returned indicating that the Service Consumer has already been registered.

In a Direct Environment, there is no adverse impact from the deletion of the Environment on shut-down. However, in a Brokered Environment, keeping the Environment for subsequent use may be essential.

#### consumer.environment.sharedSecret

This property defines the password used to register the Service Consumer with the Environment. This value is mandatory and should be provided by the Administrator of the Environment.

#### consumer.environment.url

They property defines the URL of the Environment Provider service. This value is mandatory and should be provided by the Administrator of the Environment.

#### consumer.environment.template.applicationKey

This property defines a unique identifier for the application/Consumer. This property, in combination with the instanceId, userToken and solutionID, is used to uniquely identify a Service Consumer instance (and therefore associated Environment).

#### consumer.environment.template.authenticationMethod

This property defines the method used by the Environment Provider to authenticate a Service Consumer.

NOTE: For the current release of this framework, this value must always be Basic.

#### consumer.environment.template.consumerName

This property is used to provide a user label for the Service Consumer. At this point in time, it does not have any functional impact.

#### consumer.environment.template.dataModelNamespace

This property defines the namespace of the SIF Object managed by the Service Consumer.

#### consumer.environment.template.supportedInfrastructureVersion

This property defines the version of the SIF Infrastructure that this framework supports. As this framework currently on supports the current version of the SIF Infrastructure, this value should always be “3.0.1”.

## Implementing a Functional Service Provider

To implement a Functional Service Provider, a Web API Visual Studio Project needs to be created. There is no need to implement a controller/provider classes as this is done for you through the FunctionalServiceProvider. This class will route all traffic addressed to your functional service. You need only extend the class FunctionalService to get started. Below is a Codechart[[2]](#footnote-2) that shows the hierarchy of this class:



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 assoiated operations. To implement a functional service you must implement a class that extends the class FunctionalService 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, and optionally a third.

That is, the class FunctionalService implements the IFunctionalService and extends the SifService class. In doing so it inherits CRUD functionality for the SIF3 Job object and implenents the necessary functionality to support phase and state CRUD operations. The methods GetServiceName(), Configure(Job) and optionally JobShutdown(Job) are methods you will be required to implement when extending the FunctionalService class as follows:

|  |  |
| --- | --- |
| Method | Purpose |
| GetServiceName() | Should be overridden to provide the single name of the functional service.  **Example** If this method returns “Payloads” the service will expect jobs with the name “Payload” and be accessible from the URL /services/Payloads/. |
| Configure(Job) | 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 “phaseName” to the job, that is required. Consumers will have the right to send create and query messages to the phase (but not update etc.). Consumers can also send create messages to the states of this phase. The initial state of the phase is NOTSTARTED. |
| JobShutdown(Job) | This optional 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 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.  It is possible to initiate a job shut down process asynchronously. That is, when a shutdown request happens your code shoul initiate the shutdown process, extend the job’s timeout by calling:    This essentially puts the job back into the pool of jobs to be considered again for a timeout at a later date. If the job has been shutdown at that point the method should complete without exception to result in the job object’s deletion. |

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



IPhaseActions implementations encapsulate the CRUD operations available for a phase. In this case a set of actions for the phase named “phaseName” 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 a RejectedException 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 PhaseActions class.

Each action in a phase exects 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.

If support is required for other media types than XML and JSON then additional formatters will need to be added to Application\_Start() in your Global.asax.cs file. For example, if your project includes a reference to the WebApiContrib library (as the Sif.Framework.Demo.Uk.Provider project does) then it is easy to support plain text by adding:



The choriography 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 *Provider*
2. *Provider* processes the HTTP message and calls the correct method on the *FunctionalService*
3. *FunctionalService* 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.

Note that the NHibernate library is an inherent part of the framework, and the inclusion of the SifFramework.cfg.xml file is mandatory otherwise jobs will not be persisted. To achieve this Functional Services share the same database reference as the rest of the framework.

Lastly, the following third-party libraries are required by the Project for the framework to operate:

* AutoMapper (3.1.1)
* NHibernate
* Log4Net
* Sif.Framework
* Sif.Specification.Infrastructure

### Functional Service Provider properties

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".

#### provider.job.classes

Determines which functional service classes are to be loaded into the application. Value is a string containing either:

1. A “|” separated list of Assemply Quallified Class Names.
2. The keyword “any”, which will attempt to find and load all functional services in the system. Which services are logged, usefull in testing.

If unspecified the default is “any”.

#### provider.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.

#### provider.job.timeout.enabled

Determines if jobs should be checked to see if they have timed out. Value is a Boolean (true/false). If unspecified the default is true.

#### provider.job.timeout.frequency

Determines how often to check for timed out jobs. Value is numeric in seconds. If unspecified the default is 60.

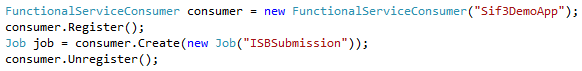
#### provider.startup.delay

Determines how long to wait in seconds between starting each functional service. Value is numeric in seconds. If unspecified the default is 10.

## Implementing a Functional Service Consumer

A functional service consumer does not have to be implemented as a generic one is provided for you called FunctionalServiceConsumer. To use the functional service consumer you need only instantiate it, register it, and work with its API. The functional Service to which any given request is made is configured by the name of the job object assed as a parameter. When an individual job is required, such as when operating on a job’s phases, the job object’s refid is used. This allows the consumer to be reused against any functional service it has rights to access within its environment.

For example, to register a functional service consumer with the application key “Sif3DemoApp”, and create a new “ISBSubmission” job in the default zone/context is as simple as:



**Note:** The job name is always **singular**, the service name is **plural**. That is, in the case of creating an “ISBSubmission” job the expected service name is “ISBSubmission***s***”.

### Registration/Deregistration

|  |  |
| --- | --- |
| Name | Description |
| Register() | Register this Consumer. |
| Unregister() | Unregister this Consumer. |

### Job CRUD operations

In all operations the zone and context are strings that indicate the zone and context in which to issue the request. They can be null values, in which case the default zone is assumed.

|  |  |
| --- | --- |
| Name | Description |
| Create(Job, Zone, Context) | Create a single Job with the defaults provided. |
| Create(List(Job), Zone, Context) | Create multiple Jobs with the defaults provided. |
| Query(Job, Zone, Context) | Gets a single Job object by its refid. |
| Query(jobName, navigationPage,  navigationPageSize,  Zone, Context) | Get all jobs from the service that handles the specified jobname (a string) with (nullable) page number and page size parameters. |
| QueryByExample(Job,  navigationPage,  navigationPageSize,  Zone, Context) | Get a all Jobs that match the example provided. |
| Update(Job, Zone, Context) | Update single job object is **not supported** for Functional Services. Throws a HttpResponseException with Forbidden status code. |
| Update(List(Job), Zone, Context) | Update multiple job objects is **not supported** for Functional Services. Throws a HttpResponseException with Forbidden status code. |
| Delete(Job, Zone, Context) | Delete a Job. The job needs only its name and refid defined. |
| Delete(List(Job), Zone, Context) | Delete a series of Jobs. Each job needs to have the same name and each should have a refid defined. |

### Phase CRUD operations

In all operations the zone and context are strings that indicate the zone and context in which to issue the request. They can be null values, in which case the default zone is assumed. The body parameter is a string (possibly the result of serialization by your consumer’s business logic) to be sent to the Provider. It can be null. Th arguments contentTypeOverride and acceptOverride are both strings 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.

|  |  |
| --- | --- |
| Name | Description |
| CreateToPhase(Job, phaseName, body,  zone, context,  contentTypeOverride,  acceptOverride) | Send a create operation to a specified phase on the specified job with a (possibly null) payload. |
| DeleteToPhase(Job, phaseName, body,  zone, context,  contentTypeOverride,  acceptOverride) | Send a delete operation to a specified phase on the specified job with a (possibly null) payload. |
| RetrieveToPhase(Job, phaseName, body,  zone, context,  contentTypeOverride,  acceptOverride) | 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(Job, phaseName, body,  zone, context,  contentTypeOverride,  acceptOverride) | Send a update operation to a specified phase on the specified job with a (possibly null) payload. |

### Phase State CRUD operations

In all operations the zone and context are strings that indicate the zone and context in which to issue the request. They can be null values, in which case the default zone is assumed.

|  |  |
| --- | --- |
| Name | Description |
| CreateToState(Job, phaseName, state,  zone, context) | Send a create operation to the state of the specified phase on the specified job. The state object cannot be null. Returns the current state of the phase, which may not be the state sent for creation. |

# Running the demo over a LAN

The Object Service Providers in this Sif3FrameworkDemo Solution are run using IIS Express within Visual Studio. As such, only Service Consumers on the same machine as the Object Service Providers will be able to connect to the Object Service Providers. To test the Object Service Providers against Service Consumers on another machine, the following steps are required.

## Configure local IIS Express instance

Configure IIS Express to bind to your machine (computer name) and appropriate port (that running the Object Service Providers). Your computer name can be found in the system settings (Control Panel > System and Security > System). The appropriate port numbers can be found in the properties of the Sif.Framework.EnvironmentProvider, Sif.Framework.Demo.Au.Provider and Sif.Framework.Demo.Us.Provider projects.

To configure IIS Express, open the “%userprofile%\My Documents\IISExpress\config\applicationhost.config” file. Add the following entry to the *<site name="Sif.Framework.EnvironmentProvider">* section:

<binding protocol="http" bindingInformation="\*:62921:<computer\_name" />

The port value of 62921 should be that of the Environment Provider. The <computer\_name> should be replaced with the name of the host machine.

Add the following entry to the <site name="Sif.Framework.Demo.Au.Provider"> section:

<binding protocol="http" bindingInformation="\*:50617:<computer\_name>" />

The port value of 50617 should be that of the StudentPersonal Provider. The <computer\_name> should be replaced with the name of the host machine.

Add the following entry to the <site name="Sif.Framework.Demo.Us.Provider"> section:

<binding protocol="http" bindingInformation="\*:53180:<computer\_name>" />

The port value of 53180 should be that of the K12Student Provider. The <computer\_name> should be replaced with the name of the host machine.

Add the following entry to the <site name="Sif.Framework.Demo.Uk.Provider"> section:

<binding protocol="http" bindingInformation="\*:51424:<computer\_name>" />

The port value of 51424 should be that of the LeanerPersonalProvider and Payloads Functional Service Provider. The <computer\_name> should be replaced with the name of the host machine.

## Grant remote access

As an Administrator, run the following commands:

netsh http add urlacl url=http://<computer\_name>:62921/ user=everyone

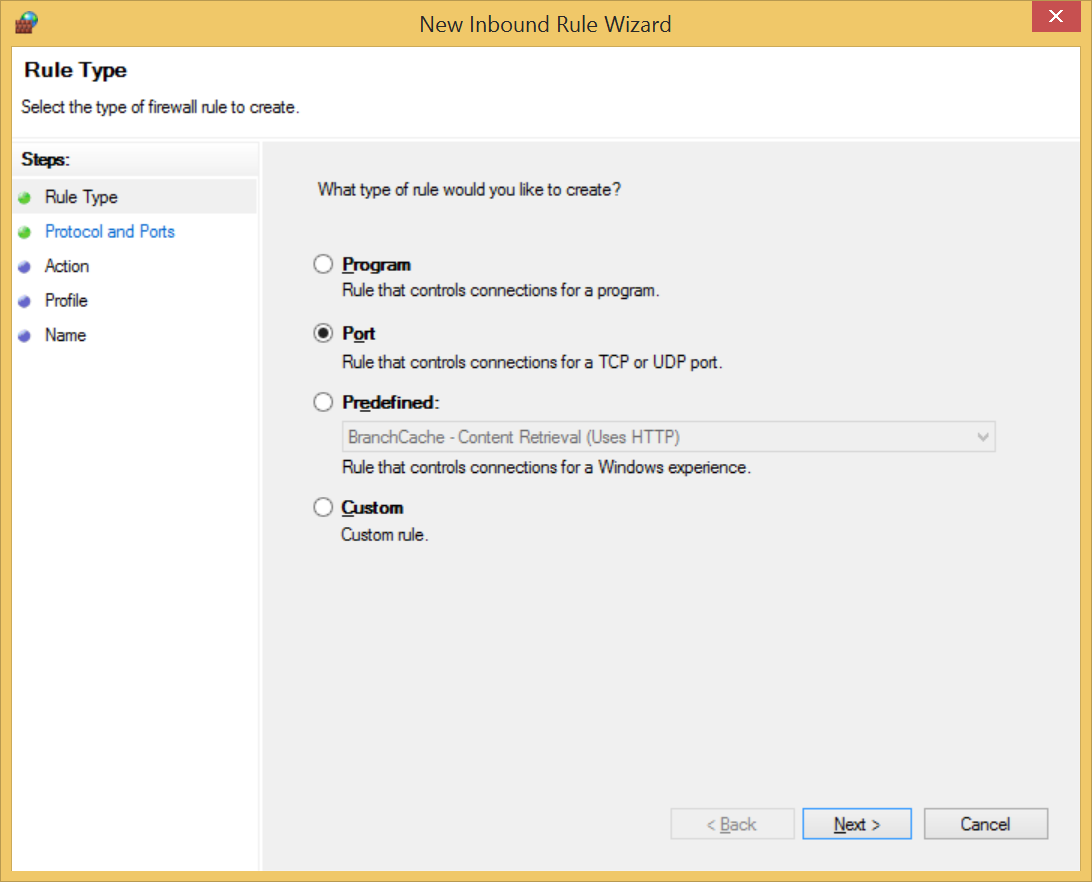
netsh http add urlacl url=http://<computer\_name>:50617/ user=everyone

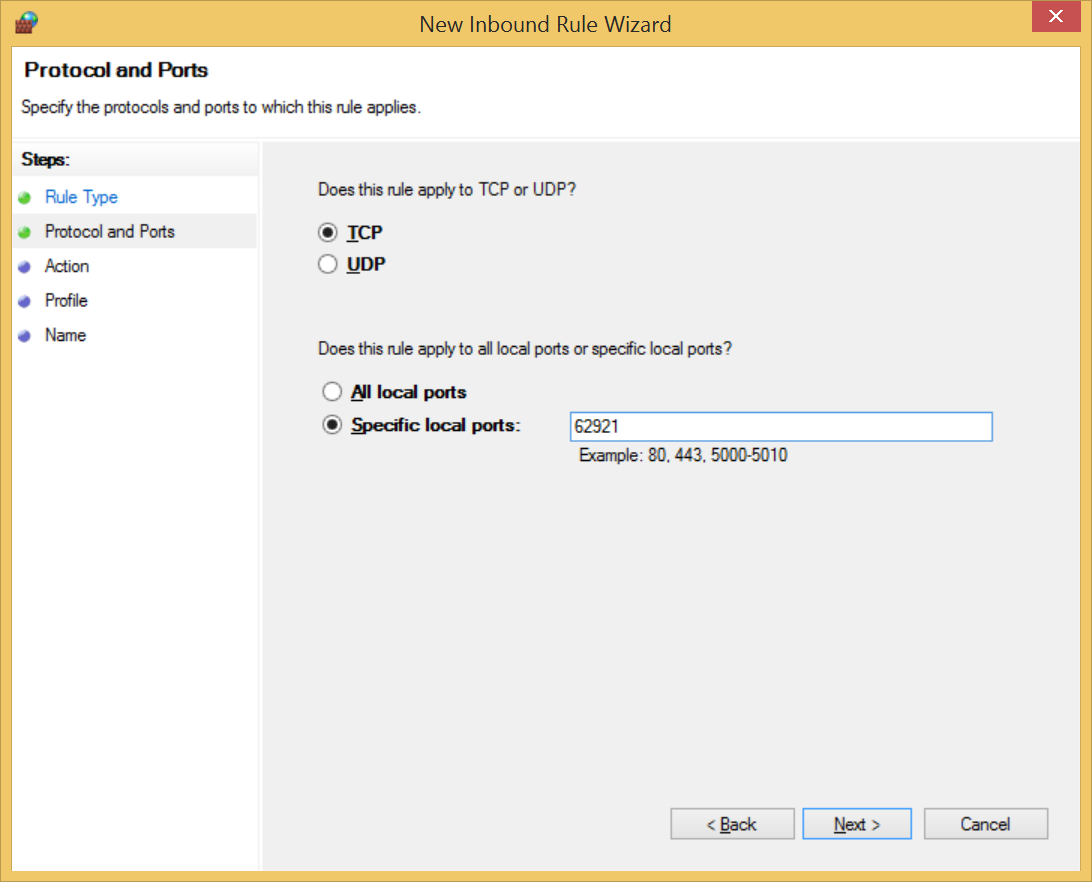
netsh http add urlacl url=http://<computer\_name>:53180/ user=everyone

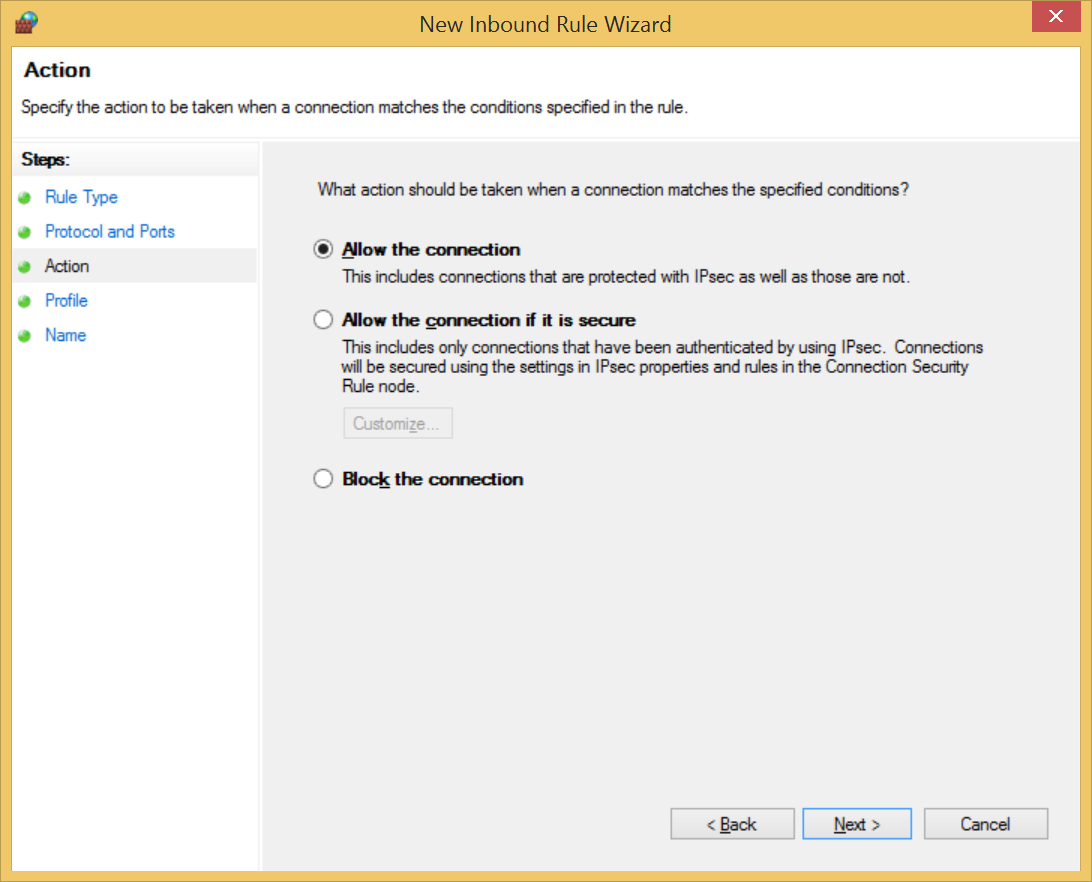
netsh http add urlacl url=http://<computer\_name>:51424/ user=everyone

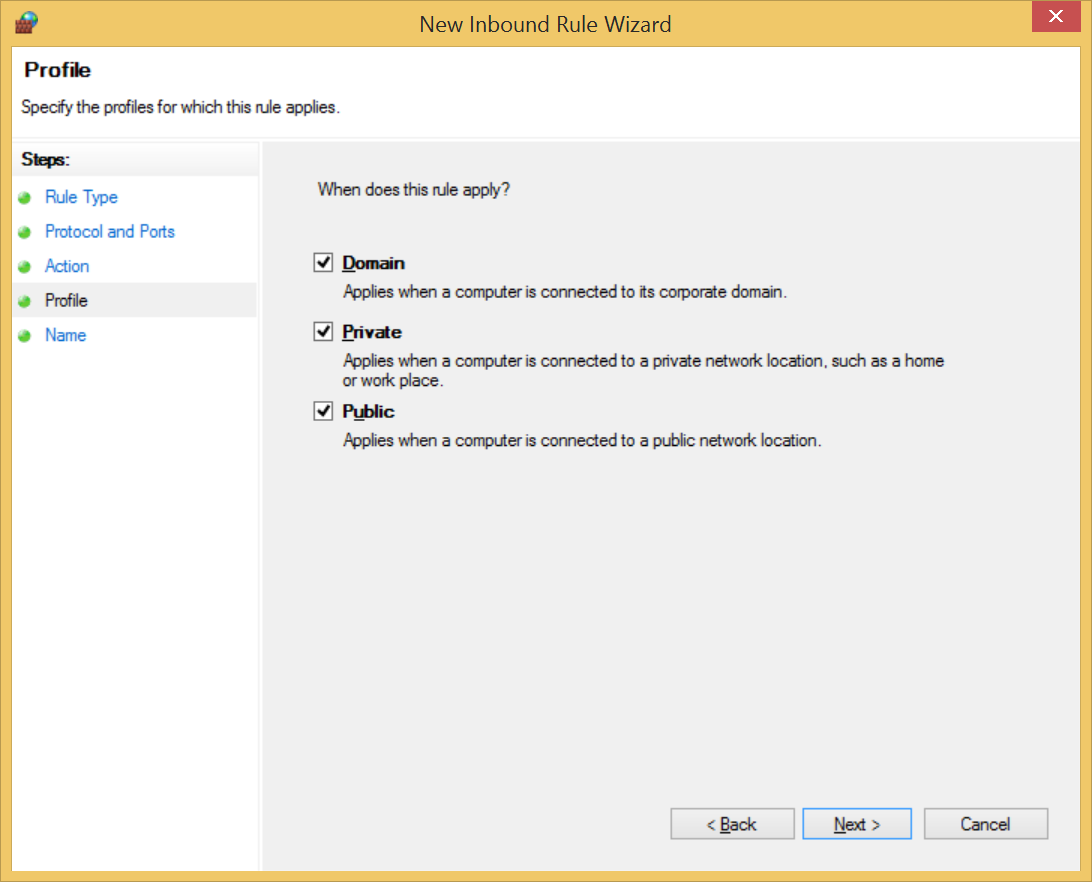
## Configure firewall access

The firewall needs to be configured for the ports used. This can be performed by adding new Inbound Rules for both ports in the Windows Firewall with Advanced Security window (Control Panel > System and Security > Windows Firewall > Advanced settings).









## Configure Visual Studio

In the Web Server properties of both the Sif.Framework.EnvironmentProvider, Sif.Framework.Demo.Au.Provider, Sif.Framework.Demo.Us.Provider and Sif.Framework.Demo.Uk.Provider projects, modify the Project Url so that the computer name replaces “localhost”.

## Configure the StudentPersonal Consumer

Update the SifFramework.config file and replace the “localhost” part of the consumer.environment.url value to use the computer name.

## Configure the Environment definition

In the Sif.Framework.Demo.Setup project, update one of the following files:

* Data files/AU/Sif3DemoApp/EnvironmentResponse.xml
* Data files/US/ Sif3USDemoApp/EnvironmentResponse.xml
* Data files/UK/ Sif3DemoApp/EnvironmentResponse.xml

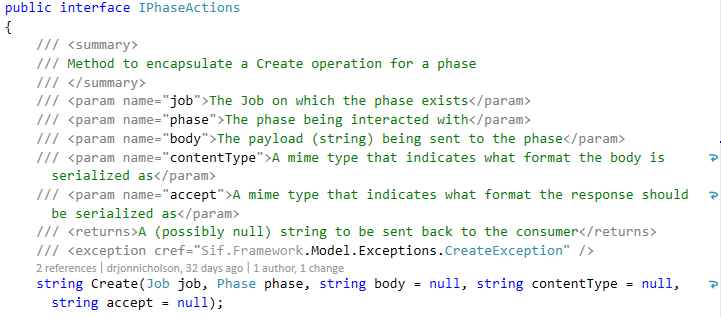
by replacing URL references to “localhost” with the computer name. Once done, run one fo the following:

* Scripts\BAT\Demo execution\DemoAuSetup.bat
* Scripts\BAT\Demo execution\DemoUsSetup.bat
* Scripts\BAT\Demo execution\DemoUkSetup.bat

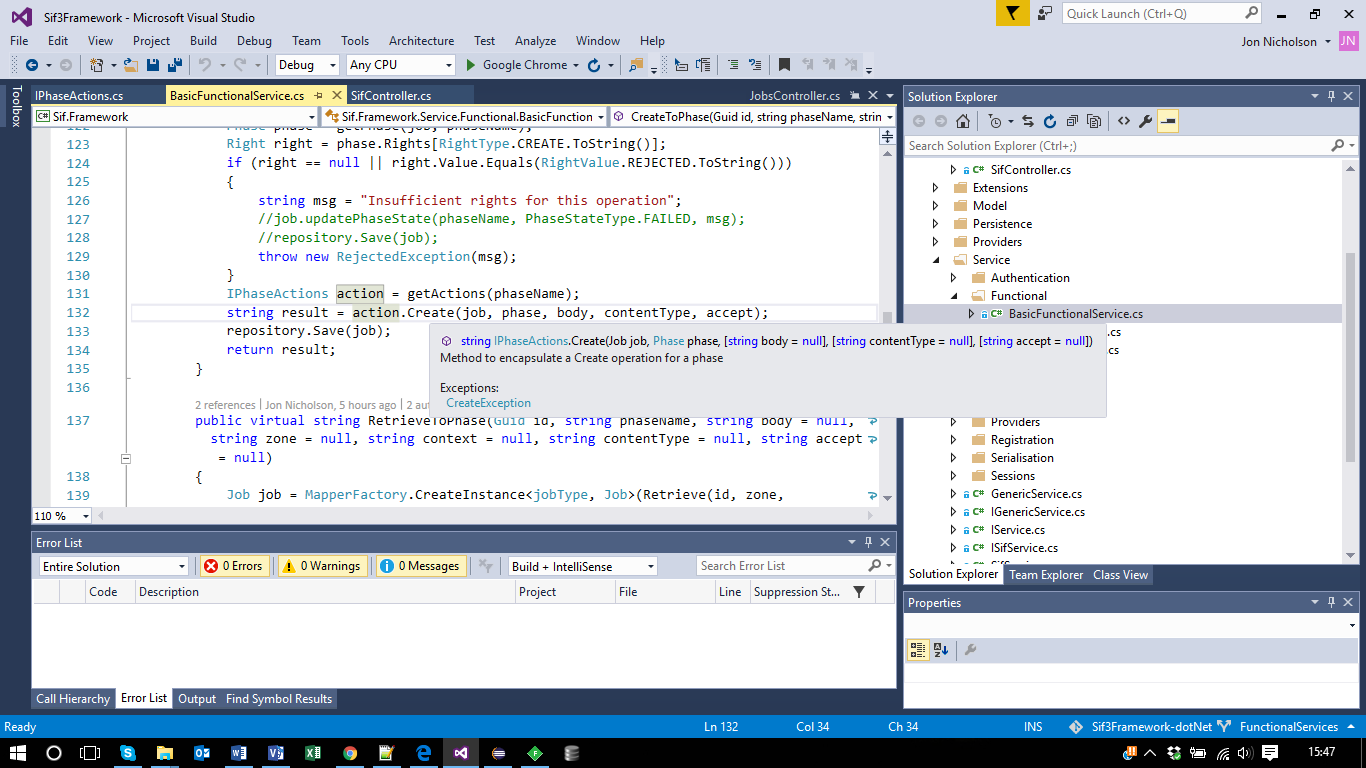
At this point, the demo can be re-run as per previous instructions.

# Code documentation

Where possible code has been commented using XML Documentation Comments (see <https://msdn.microsoft.com/en-us/library/b2s063f7.aspx>). This provides some more details on how to use certain classes and methods.



Visual Studio uses these comments to generate IntelliSense comments in Visual Studio. For example:



1. *CompileDemos.bat* requires editing so that it can find your copies of nugget.exe and msbuild.exe. You will also have to open the Demos solution in Visual Studio at least once to generate the correct ApplicationHosts.config file used by the other scripts. [↑](#footnote-ref-1)
2. See <http://lepus.org.uk> for a more thorough account of Codecharts. [↑](#footnote-ref-2)