SIF 3.0 Framework (.NET)

Version 1.1.0

Demo Usage Guide

**Author:** Rafidzal Rafiq, SIF Solution Architect

**Revision:** 1.00 (draft)

**Published:** May 2016

Copyright © 2016, Systemic Pty Ltd

Table of Contents

1. Introduction 3

1.1. Target audience 3

1.2. Scope 3

2. Technical details 4

2.1. Configuring an Environment 4

2.2. Starting the services 4

2.2.1. Starting the Environment Provider 5

2.2.2. Starting the Service Provider 5

2.2.3. Starting the Service Consumer 5

2.3. Reviewing expected behaviour 6

2.4. Reviewing HTTP messages in Fiddler 6

2.4.1. Filtering for SIF messages 6

2.5. Explaining service choreography 7

2.6. Transitioning from the demo 7

2.6.1. Configuring an Environment 7

2.6.2. Implementing an Object Service Provider 7

2.6.3. Implementing a Service Consumer 8

2.7. Running the demo over a LAN 10

2.7.1. Configure local IIS Express instance 10

2.7.2. Grant remote access 10

2.7.3. Configure firewall access 11

2.7.4. Configure Visual Studio 14

2.7.5. Configure the StudentPersonal Consumer 14

2.7.6. Configure the Environment definition 14

# 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):

* Configuring an Environment
* Starting the services
* 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.

# Technical details

In running this demo, the following steps are performed:

* Create an Environment
* Start the Environment Provider
* Start an Object Service Provider
* Run a Service Consumer

The sections that follow provide greater detail on each step.

## 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\DemoUsSetup.bat
* Scripts\BAT\Demo execution\DemoUkSetup.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 by running *Scripts\BAT\Code Generation\CompileDemos.bat*

**NOTE:** When running the scripts mentioned above you will be asked to provide administrator privaleges. This is reuired to add/update entries to the computer’s host file to enable monitoring HTTP messages with Fiddler (see section 2.4).

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\DemoEnviroment.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
* Scripts\BAT\Demo execution\DemoAuProvider.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
* Scripts\BAT\Demo execution\DemoUkConsumer.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.

## Reviewing HTTP messages in Fiddler

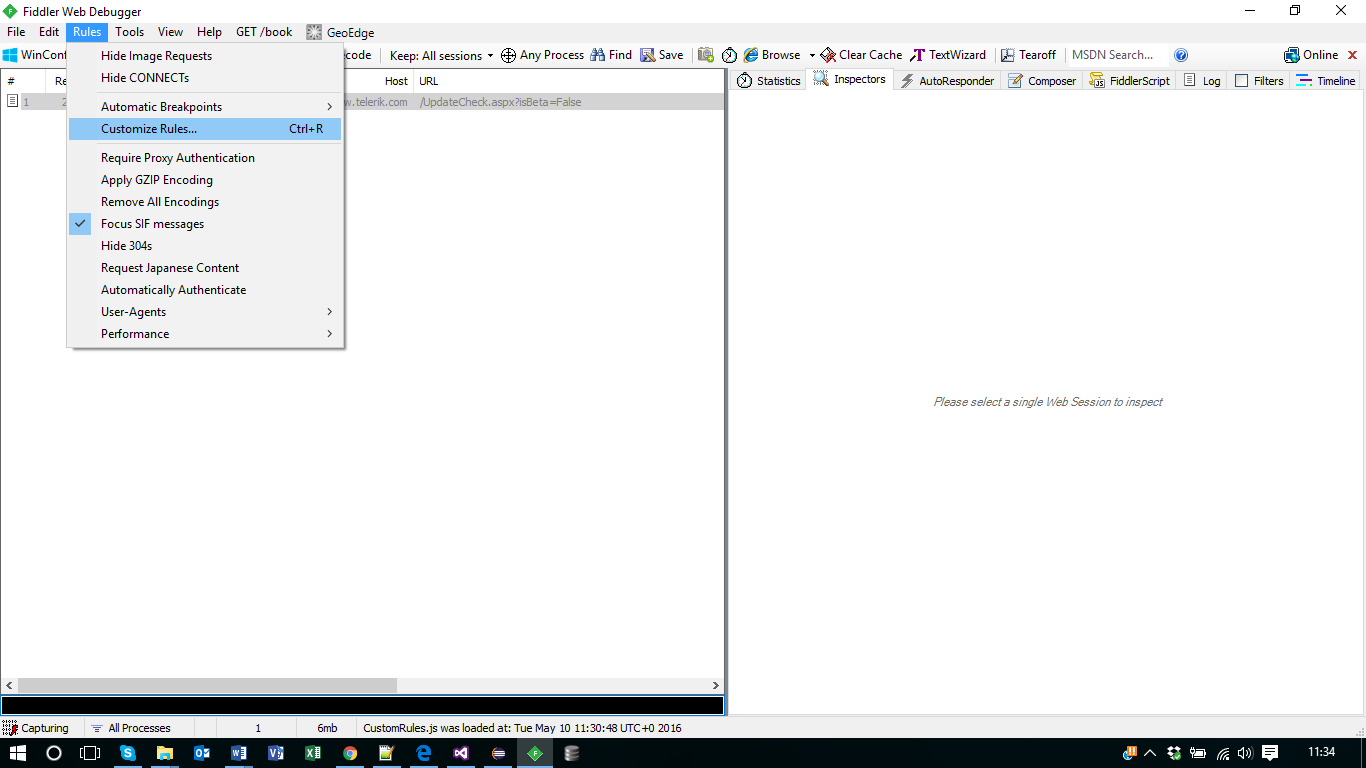
Created by Telerik, Fiddler is a free web debugging proxy available from <http://www.telerik.com/fiddler>. It works by setting up a system proxy, allowing the messages being sent/received between consumer, environment and provider to be monitored.

To ensure that Fiddler can monitor the messages correctly the configuration scripts described in section 2.1 add aliases to the localhost address into the computer’s hosts file.

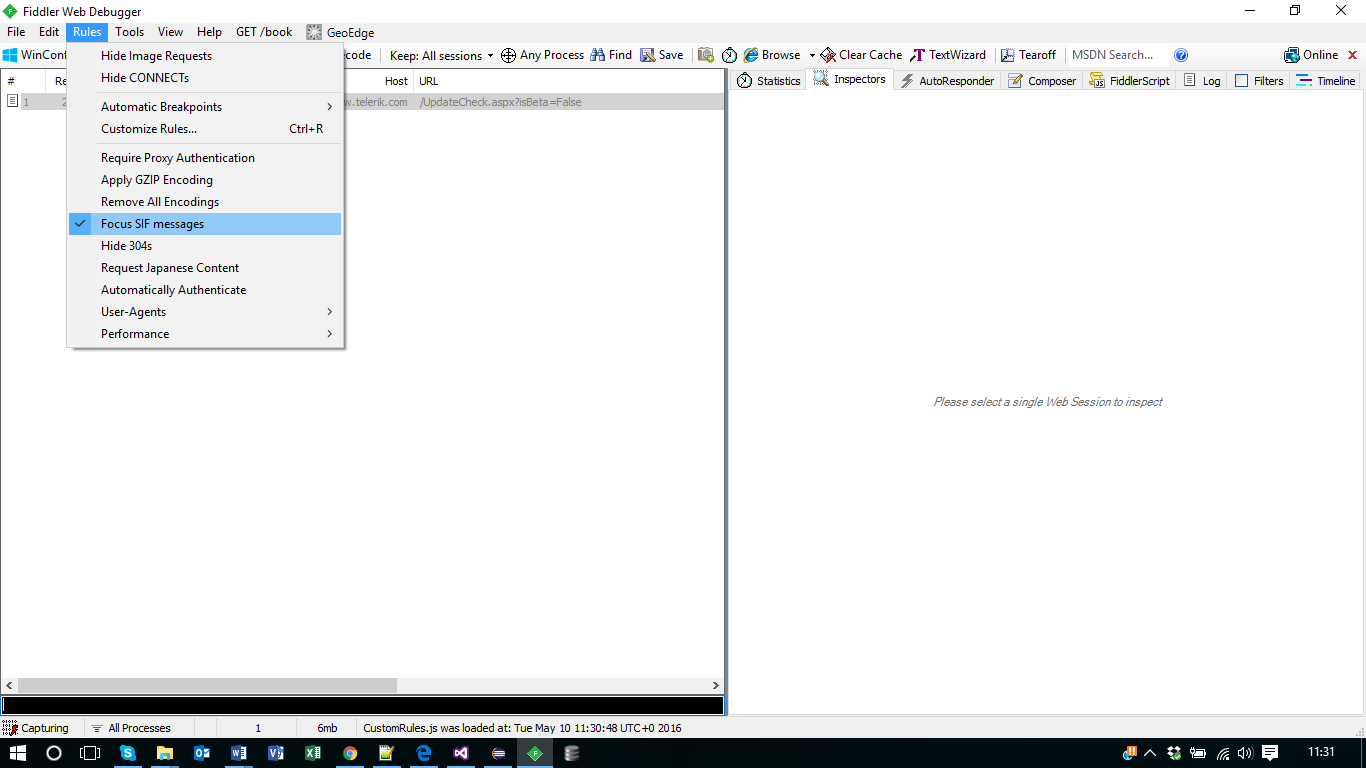
### Filtering for SIF messages

Fiddler can use scripts to extend the application, one is provided in *Tools/Fiddler/CustomRules.js* to filter HTTP messages by known ports. To use this feature first install the *Syntax-Highlighting Add-Ons* available from <http://www.telerik.com/fiddler/add-ons>.

Open Fiddler and open the script editor by selecting *Rules > Customize Rules…* (or *ctrl-R*).



The entire contents of the script can be replaced by the script found in *Tools/Fiddler/CustomRules.js*. If all has been installed correctly then there should be a new enable/disable option called *Focus SIF messages* under the *Rules* menu, as seen below:



## 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.

## 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
* 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

**IMPORTANT NOTE: As of May 2016 until further notice**

The functionality described within this section is an early adoption of the “Functional Services” functionality. SIF 3.2 will support Functional Services, but in the interim you are advised to consult with your provider/consumer if Functional Services are supported.

Details are correct as of code committed 10 May 2016. The code, and its documentation, is subject to change until delivery of final product to the DfE in July.

To implement a Functional Service Provider, a Web API Visual Studio Project needs to be created. When adding a Controller to the project, it must extend JobsController and pass in an appropriate BasicFunctionalService implementation. In terms of SIF 3.2, the Controller effectively becomes a Functional Service Provider and partial Services Connector implementation. Below is a Codechart that shows the the hierarchy of these classes:



In the *Provider* class all CRUD operations should be overridden, but forwards their invocation to the JobsController. This is so that correct route annotations can be attached to these methods, see PayloadProvider for an example of this. Note that *Provider* class name must end in the word “Provider”, e.g. PayloadProvider, for it to be recognised within the framework as a provider service.

In the *FunctionalService* class the TypeName property should be overridden to provide the single name of the functional service. For example, if the URL is to be /services/Payloads/… then TypeName should return “Payload”. The addPhases(Job) method is called when a new job is created to ensure that each new job is instantiated with all appropriate phase definitions and appropriate access rights. Each line of code in this method should look like below:



The constructor of the *FunctionalService* class should populate the PhaseActions map with the name of the phase and the associated actions it performs. For example:



Here the job is being given a phase called “default”, whose CRUD operations should be handled by an instance of the class DefaultActions. Each action should be an implementation of the IPhaseActions interface. A BasePhaseActions class is provided for convenience which implements all methods with not supported exceptions allowing only the desired methods be implemented. The structure of the phase action classes is shown in the Codechart below:



Each action exects a (possibly null) payload that must be deserialized within the phase action, and the response serialized to string for sending back to the consumer. 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 the Global.asax.cs file. For example, if your project includes a reference to WebApiContrib (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 received by the *Provider*
2. *Provider* calls the correct method on *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.

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

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

### Implementing a Functional Service Consumer

**IMPORTANT NOTE: As of May 2016 until further notice**

The functionality described within this section is an early adoption of the “Functional Services” functionality. SIF 3.2 will support Functional Services, but in the interim you are advised to consult with your provider/consumer if Functional Services are supported.

Details are correct as of code committed 10 May 2016. The code, and its documentation, is subject to change until delivery of final product to the DfE in July.

To implement a Functional Service Consumer, the BasicJobConsumer class needs to be extended. As is the case in the Provider must override the TypeName property with the (singular) name of the service. For example if the service is expected to sit at /services/Payloads then TypeName must return “Payload”. A Codechart explaining the hierarchy of the Functional Service Consumer classes is given below:



No other extension is required as all functionality is implemented in BasicJobConsumer.

## 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.

### Grant remote access

As an Administrator, run the following commands:

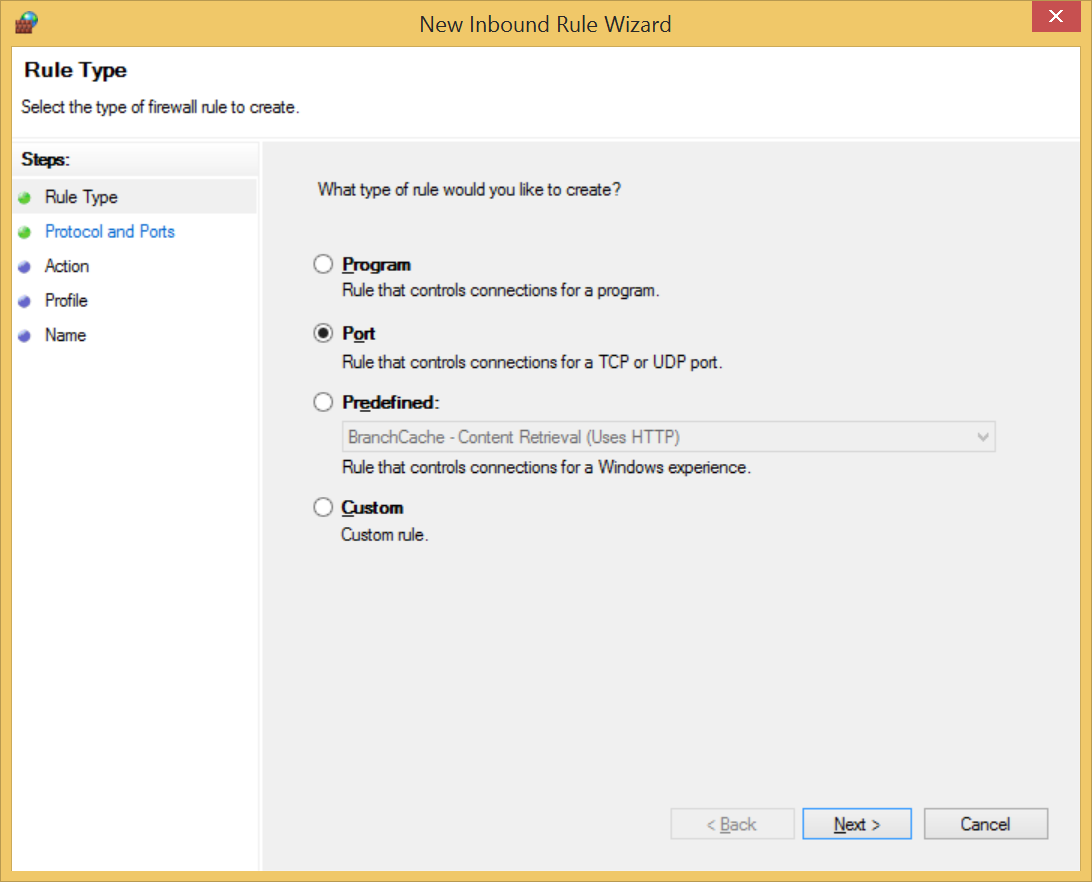
netsh http add urlacl url=http://<computer\_name>:62921/ user=everyone

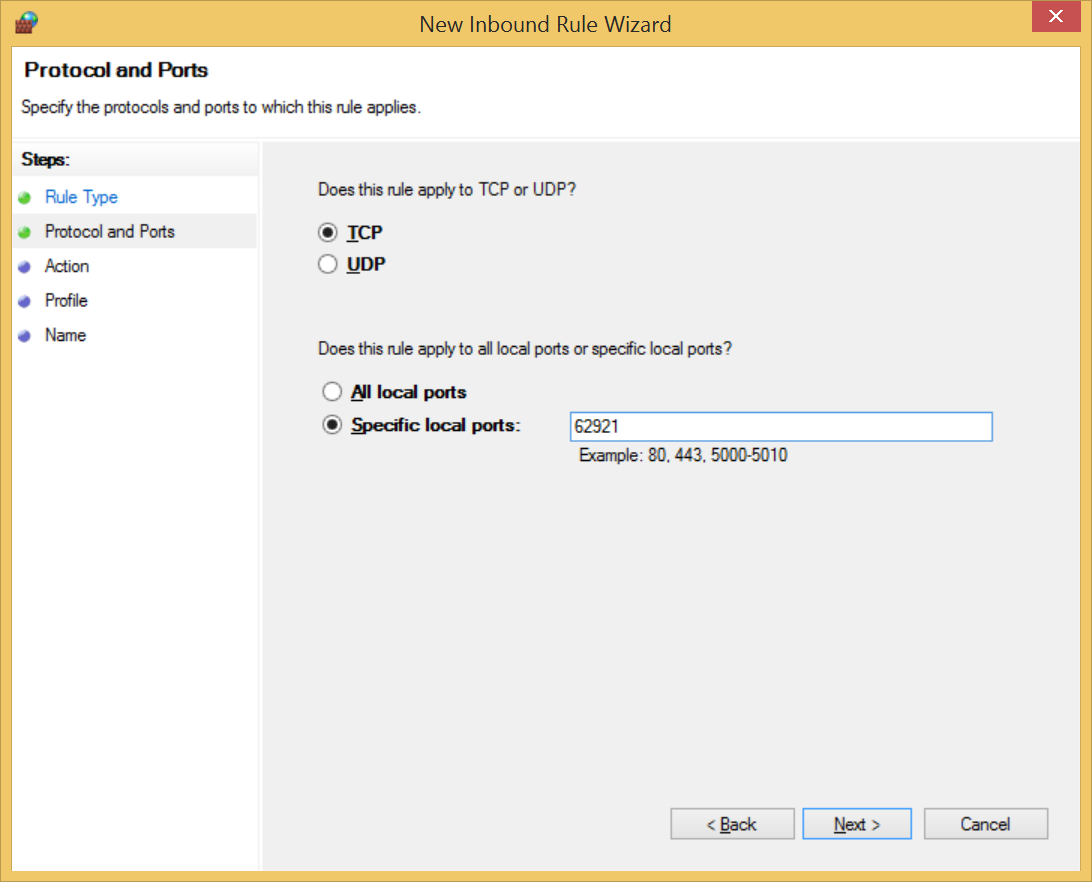
netsh http add urlacl url=http://<computer\_name>:50617/ user=everyone

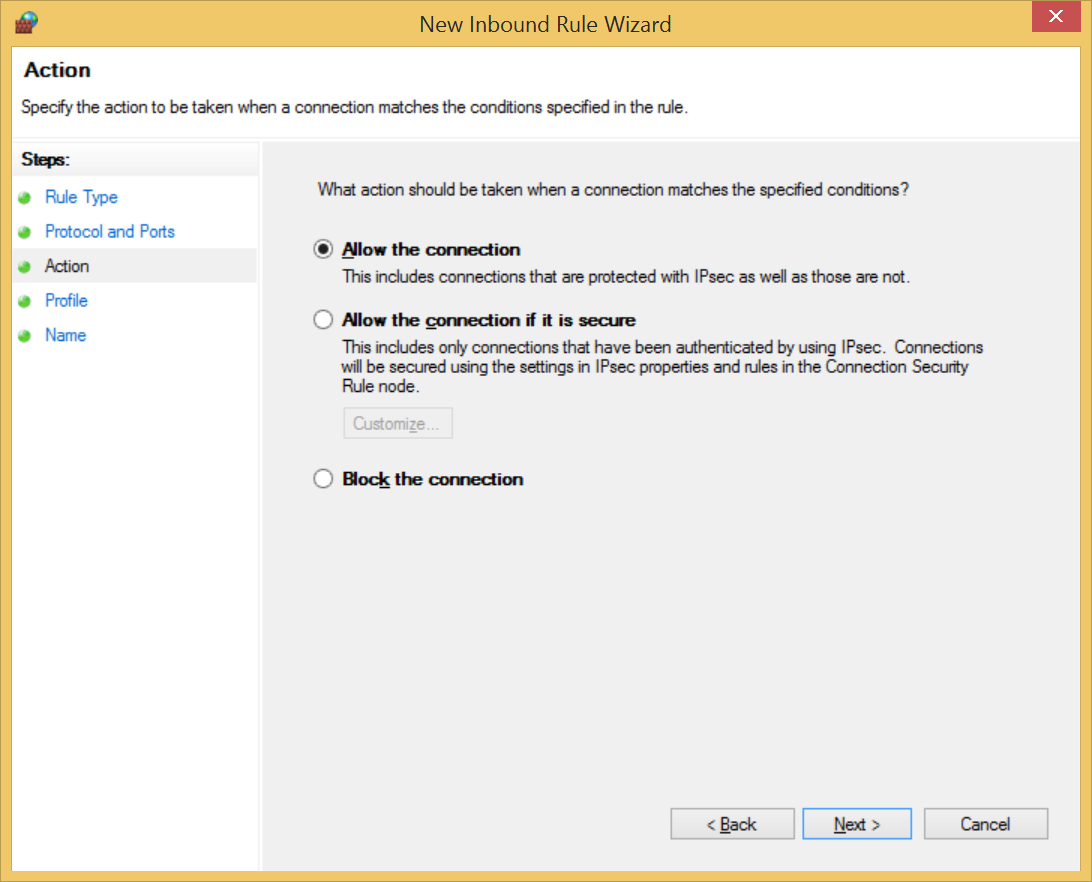
netsh http add urlacl url=http://<computer\_name>:53180/ user=everyone

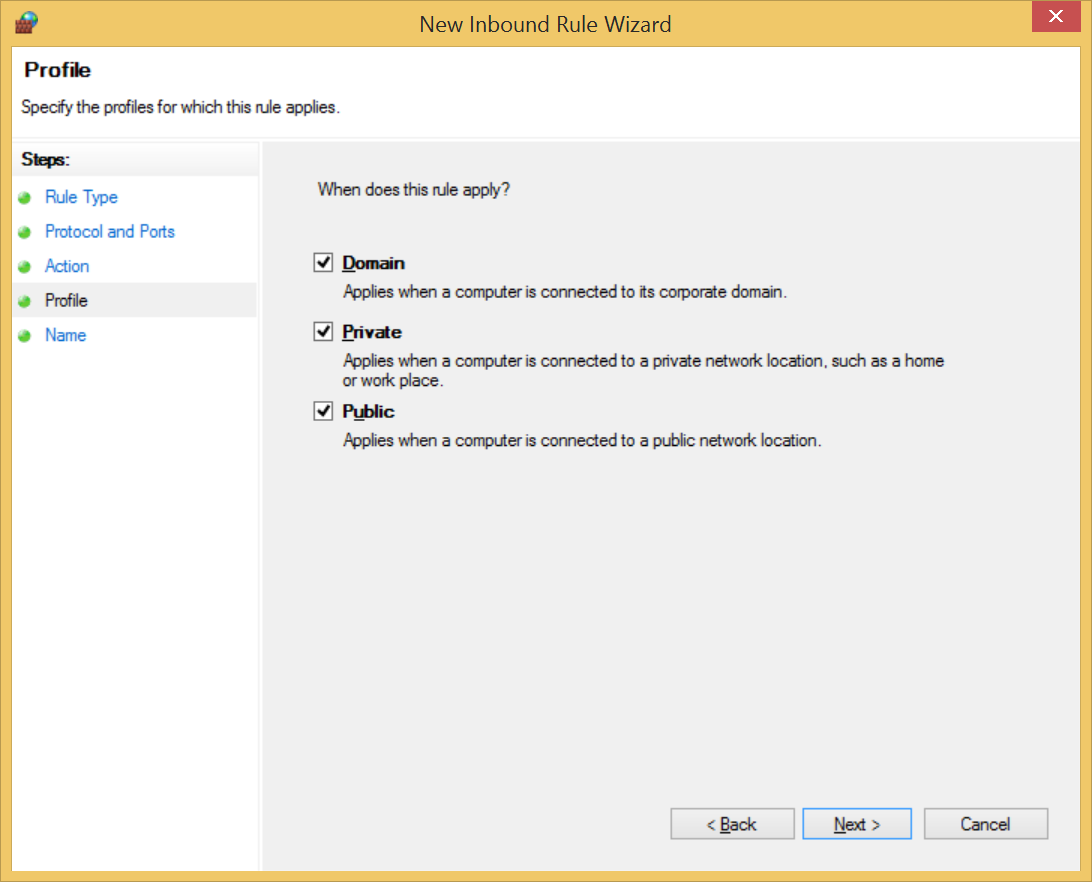
### Configure firewall access

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 and Sif.Framework.Demo.Us.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/EnvironmentResponse.xml
* Data files/US/EnvironmentResponse.xml
* Data files/UK/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.