# WCF Simple Greeting Service Library & Service Host Console Application Supporting Http and TCP Protocols & Client “Tester" Console Application

## About

This project presents a simple WCF Greeting Service that demos architectural styles of setting up, hosting, configuring services, and testing for dual protocols: http and tcp. The service accepts a simple string, which represents a name, and returns a simple string with a greeting that acknowledges the name. Instead of using IIS Express, the simple greeting service is hosted using a console application. The Visual Studio solution also has two client “tester” applications that test both the http and tcp protocol connections to the hosted service.

## Architecture

The demo project consists of these component topics:

* SimpleGreeting Service Library
  + ISimpleGreetingService (Interface for Service)
  + SimpleGreetingService (Code that Implements the Service Interface)
* ServiceHostApplication “Service Host” Console Application
  + Hosts the SimpleGreeting Service
  + Started/Stopped Using Console Window Interface
* HttpClient “Tester to Service” Console Application
  + Connected Service “Proxy Reference” SimpleGreetingServiceReference
  + Simple Program to test the service @ http endpoint
* TcpClient “Tester to Service” Console Application
  + Connected Service “Proxy Reference” SimpleGreetingServiceReference
  + Simple Program to test the service @ tcp endpoint

## SimpleGreeting Service Library

A WCF Service Library project was added to my Visual Studio solution. The code is available on GitHub here [].

[insert code here]

### ISimpleGreetingService (Interface for Service)

The ServiceContract for the Simple Greeting service has only one OperationContract: a method called “GreetMe” that accepts a string representing a name. The “GreetMe” returns string to the caller that includes a greeting plus the name. The code is available on GitHub here [].

[insert code here]

### SimpleGreetingService (Code that Implements the Service Interface)

The service implementation code details the “GreetMe” method. It accepts a string representing the name, and uses the default “Stranger” if no name is provided. It returns a simple string containing a greeting plus the name that the caller sent. The code is available on GitHub here [].

[insert code here]

## ServiceHostApplication “Service Host” Console Application

Instead of hosting with IIS Express, the Simple Greeting Service is self-hosted with specific ports, configuration, endpoints, etc. to allow access either via http or tcp protocols. The service is managed via a simple console application that starts and stops the service. The code is available on GitHub here [].

### Main Program

The main program spins up an instance of the SimpleGreeting Service Library alerting the user that it has successfully started the service and waits for the user input to stop the service. It catches any errors, alerts to the console window, and finalizes by properly closing the service host. The code is available on GitHub here [].

[insert code here]

### App Configuration “App.Config”

The app configuration “app.config” file for the service host is vital for the correct operations of the service host. The ServiceModel from the SimpleGreeting Service Library is copied and pasted to the actual executable host project, in this case, the ServiceHostApplication. The ServiceModel from the service library “dll” cannot run unless its is on an executable, so that is why the serviceModel configuration is copied to where the actual service will be hosted. It gives instructions on how to host it, kind of like a recipe.

The service model was customized and configured to have endpoints for both http and tcp protocols. The ABC’s (address, binding, contract) are specified for each endpoint. There are four endpoints, half of which contain the metadata exchange endpoint, while the other are services for the tcp and http. This app configuration has to be customized for the service project, as only basic configuration is auto-generated per the service library project template. The code is available on GitHub here [].

[insert code here]

## HttpClient “Tester to Service” Windows Console

The HttpClient “tester to service” is a simple client console application project in the same solution that connects to the “Simple Greeting Service” by use of a proxy generated by SVCUTIL. The client program will use this proxy and the http protocol to test each OperationContract or method available in the ServiceContract and return the results to the user on the console window.

Connected Service “Proxy Reference” SimpleGreetingServiceReference

I used the simple “Add Service Reference” wizard to create a Service Reference to an existing service in my Visual Studio solution. The existing service host application had to be started (*not in debug mode*) for the wizard to be able to connect to the metadata exchange via the base address provided and add a service reference. Note: the base address provided was specific to the protocol. For the http client, the metadata would be available after entering this address into the wizard:

http://localhost:8733/SimpleGreetingService/

Note: the protocol used was http. The client was able to first load, instance, and then recognize the SimpleGreeting service and build the service reference using the exposed meta-exchange data “WSDL”. The auto-generated code is available on GitHub here [].

Main Program

The main program in the client “tester” console application creates a proxy using the service reference that was previously created using the wizard, to connect to the simple greeting service. The proxy must specify the specific endpoint name on the service as it relates to the protocol.

SimpleGreetingServiceClient proxy = new SimpleGreetingServiceClient("basicHttpBinding\_ISimpleGreetingService");

The endpoint name for the specific endpoint differentiates the http protocol and make sure it does not connect to the default binding. The endpoint name is available in the app configuration file on the service host, so the client has to be aware of the endpoints. They can also learn this from the metadata exchange or WSDL on the service. The correct endpoint name for the protocol must be specified in the creation of the proxy to use that protocol.

After the proxy is setup, the tester console application prompts the user for their name and then makes a request to the service passing this string parameter. It receives the response from the service and outputs the result as a string to the console window. The code is available on GitHub here [].

[insert code here]

## TcpClient “Tester to Service” Windows Console

The TcpClient “tester to service” is a simple client console application project in the same solution that connects to the “Simple Greeting Service” by use of a proxy generated by SVCUTIL. The client program will use this proxy and the net.tcp protocol to test each OperationContract or method available in the ServiceContract and return the results to the user on the console window.

Connected Service “Proxy Reference” SimpleGreetingServiceReference

I used the simple “Add Service Reference” wizard to create a Service Reference to an existing service in my Visual Studio solution. The existing service host application had to be started (*not in debug mode*) for the wizard to be able to connect to the metadata exchange via the base address provided and add a service reference. Note: the base address provided was specific to the protocol. For the tcp client, the metadata would be available after entering this address into the wizard:

net.tcp://localhost:8755/SimpleGreetingService/

Note: the protocol used was tcp. The client was able to first load, instance, and then recognize the SimpleGreeting service and build the service reference using the exposed meta-exchange data “WSDL”. The auto-generated code is available on GitHub here [].

Main Program

The main program in the client “tester” console application creates a proxy using the service reference that was previously created using the wizard, to connect to the simple greeting service. The proxy must specify the specific endpoint name on the service as it relates to the protocol.

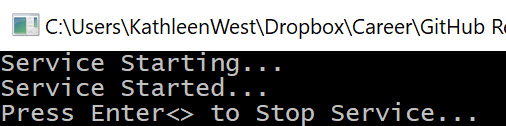
SimpleGreetingServiceClient proxy = new SimpleGreetingServiceClient("netTcpBinding\_ISimpleGreetingService");

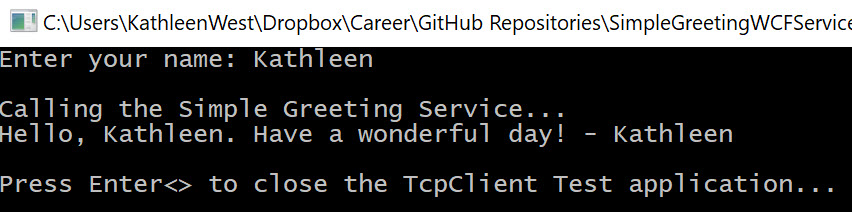
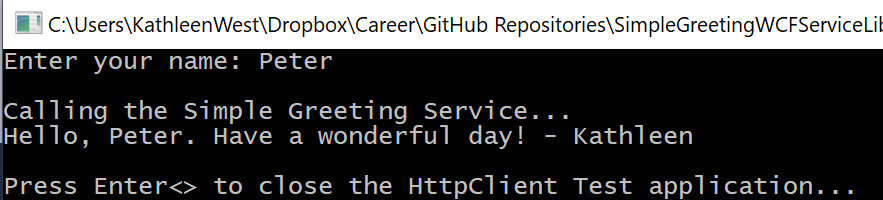
The endpoint name for the specific endpoint differentiates the net.tcp protocol and make sure it does not connect to the default binding. The endpoint name is available in the app configuration file on the service host, so the client has to be aware of the endpoints. They can also learn this from the metadata exchange or WSDL on the service. The correct endpoint name for the protocol must be specified in the creation of the proxy to use that protocol.

After the proxy is setup, the tester console application prompts the user for their name and then makes a request to the service passing this string parameter. It receives the response from the service and outputs the result as a string to the console window. The code is available on GitHub here [].

[insert code here]

## Demo





## Code

The entire project code repository is available on GitHub here.