# Simple WCF Halloween Name Service Application & Client "Tester" Console Application

## About

This project presents a simple, but fun WCF Halloween Name Service Application. The service provides either a single Halloween name (string), or an array of Halloween names. It is hosted using IIS Express to quickly demo and test the service with a client.

A client “tester” console application tests the service and provides output to the user.

## Architecture

The demo project consists of these component topics:

* Halloween Name WCF Service Application
  + Name (Data Model Contract)
  + Name Helper (Helper Class)
  + IHalloweenService (Interface for Service)
  + HalloweenService (Code that Implements the Service Interface)
* Client “Tester to Service” Console Application
  + Connected Service “Proxy Reference” HalloweenNameServiceRef
  + Main Program

## Halloween Name WCF Service Application

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

### Name (Data Model Contract)

The *Name* class is a simple DataContract with two DataMember representing a first name and a last name of a Halloween name implemented as automatic properties. The code is available on GitHub here [].

[insert code here]

### Name Helper (Helper Class)

The NameHelper is a static utility or helper class for the service application. The code is available on GitHub here [].

It initializes and loads two static lists of first names and last names from two separate text files in the App\_Data directory using reflection (See text files here []). The class uses a random number generator to randomize picking of names from both lists. The random names are then built into a Name object with the first and last name, then returned to the caller.

There are two static methods in the NameHelper class:

#### LoadNames

This method will create two lists (List<string>) representing the first names and last names of a Halloween name. It retrieves the names from two separate text files in the App\_Data directory using reflection. Note: The text files are Embedded Resources in the project.

[insert code here]

#### GetRandomName

This method will build and return a random generated *Name* object from predetermined List<string> of first names and last names that have been loaded from files in the project. It first determines if the static lists have been loaded, and if not, it calls the *LoadNames()* method to load the lists.

[insert code here]

### IHalloweenService (Interface for Service)

The ServiceContract for the Halloween Name Service allows for two possible options: a single Name entity and a list of Name entities. The first OperationContract will return a Name object, representing a randomly generated Halloween Name. The second OperationContract will return a List<Name> (list of randomly generated Halloween names) but since WCF cannot return generics by default, it will actually return an array of Name objects that the client can use to enumerate or convert to a generic of List<Name>. The code is available on GitHub here [].

[insert code here]

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

The service implementation code relies on the NameHelper class to further simplify and abstract the implementation. The code is available on GitHub here [].

[insert code here]

## Client “Tester to Service” Console Application

The client “tester to service” is a simple console application project in the same solution that connects to the Halloween Name service by use of a proxy generated by SVCUTIL. The client program will use this proxy 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” HalloweenNameServiceRef

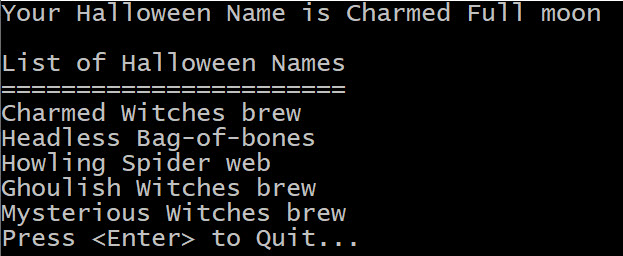
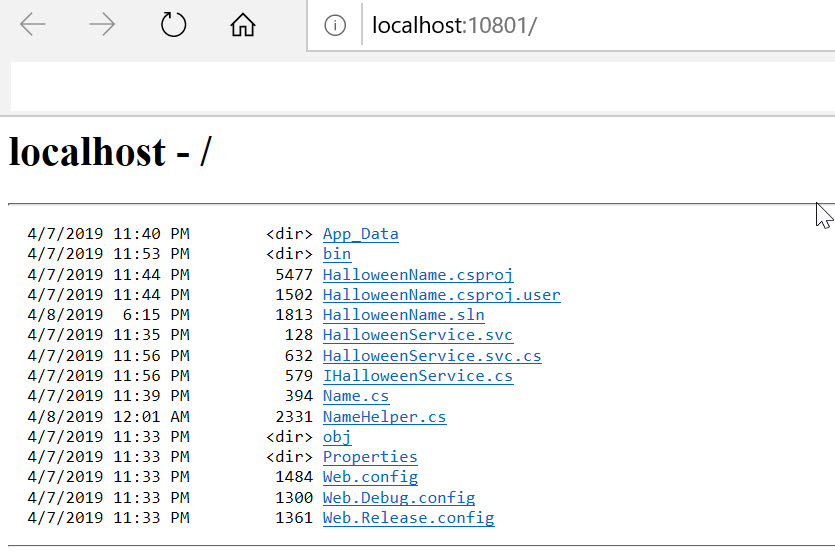
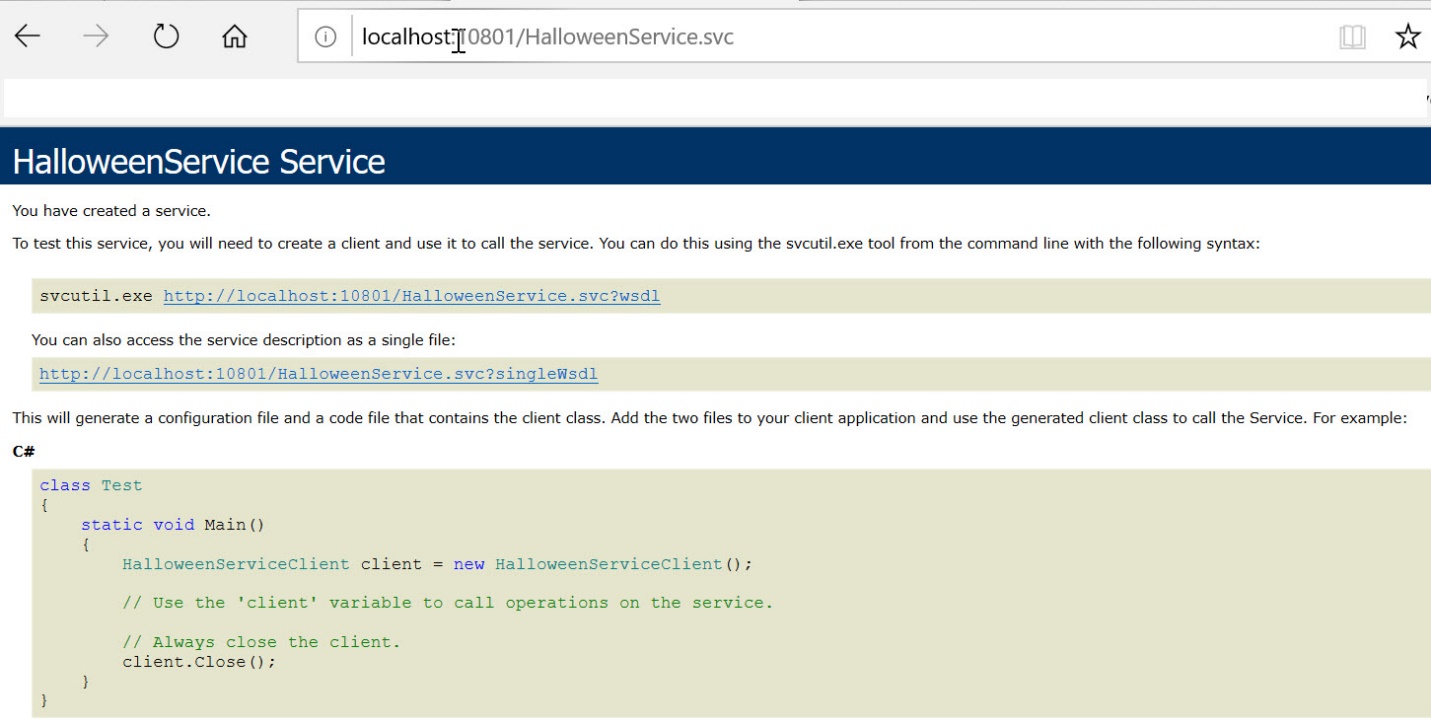
I used the simple “Add Service Reference” wizard to create a Service Reference to an existing service in my Visual Studio solution. After selecting the “Discover” button on the wizard, it was able to first load, instance, and then recognize the Halloween Name service and build the service reference using the exposed meta-exchange data “WSDL”. See the demo section for more information on how to use SVCUTIL without the wizard to generate client “tester” proxies to an existing service. The auto-generated code is available on GitHub here [].

Main Program

The main program in the client “tester” console application first connects to the Halloween Service using a client proxy with classes built from the “Add Service Reference” wizard. Based on this proxy, it then can call upon and retrieves either a single Halloween name or an enumerable list of Halloween Names. Note: The Halloween Service actually returns the List<Name> as an array[Name] and my client program converts it to a List (WCF does not work with generics by default). The code is available on GitHub here [].

[insert code here]

## Demo



## Code

The entire project code repository is available on GitHub here.