Assignment 22

WCF 2

1.WCF Service Task, CRUD operations on database table

Create a WCF service will

implement following methods

Define OperationContract AddEmployee takes employee object and add in employee table

Define OperationContract RetreiveEmployees

return list of employee objects

Define OperationContract RetreiveEmployeeByID takes id as parameter return list of employee

object

Define OperationContract UpdateEmployee takes employee object and id to edit row in database

Define OperationContract Delete

Employee takes employee id as argument and return status of the

row delete or not

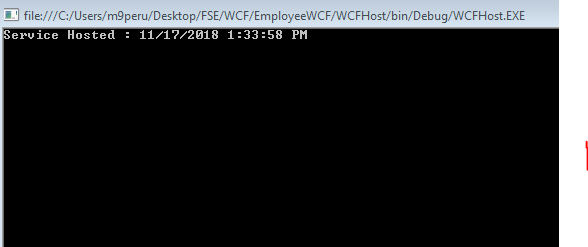
Create console application host(selfhosting) the service

Consume the service in ASP.NET MVC

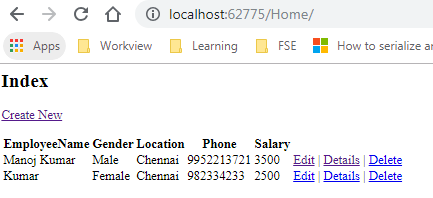
Design web form two perform crud operations

Output:

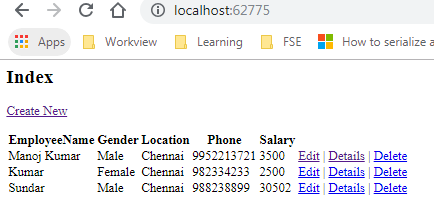
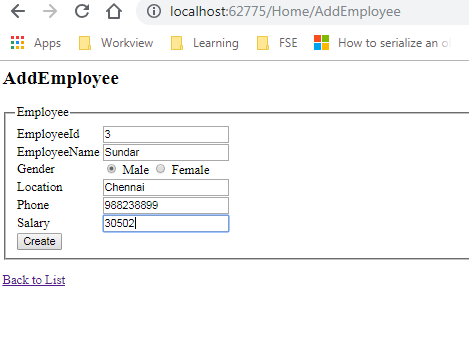
Host Output



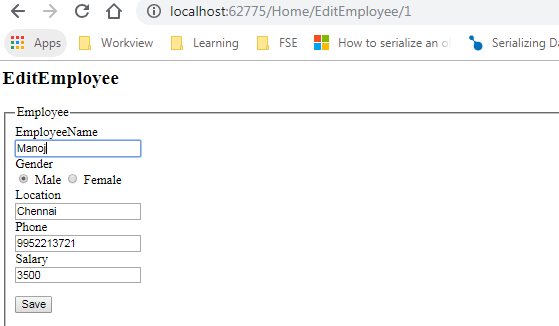
Client

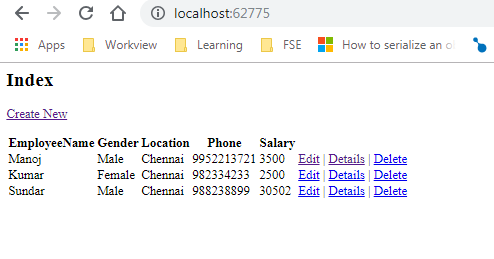


Create New

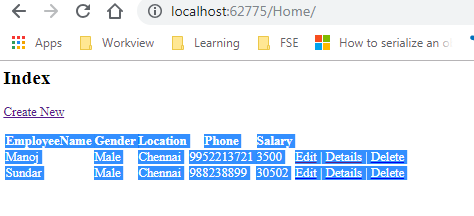


Edit Employee





Delete Employee



Code Snippet

Employee.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.Web;

namespace EmployeeWCF

{

[DataContract]

public class Employee

{

[DataMember]

public string EmployeeId { get; set; }

[DataMember]

public string EmployeeName { get; set; }

[DataMember]

public string Location { get; set; }

[DataMember]

public float Salary { get; set; }

[DataMember]

public bool Gender { get; set; }

[DataMember]

public string Phone { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace EmployeeWCF

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService1" in both code and config file together.

[ServiceContract]

public interface IEmployeeService

{

[OperationContract]

string GetData(int value);

[OperationContract]

string AddEmployeeRecord(Employee emp);

[OperationContract]

List<Employee> GetEmployees();

[OperationContract]

string DeleteEmployeeRecord(int Id);

[OperationContract]

Employee SearchEmployeeRecord(int Id);

[OperationContract]

string UpdateEmployeeRecord(Employee emp);

// TODO: Add your service operations here

}

// Use a data contract as illustrated in the sample below to add composite types to service operations.

[DataContract]

public class CompositeType

{

bool boolValue = true;

string stringValue = "Hello ";

[DataMember]

public bool BoolValue

{

get { return boolValue; }

set { boolValue = value; }

}

[DataMember]

public string StringValue

{

get { return stringValue; }

set { stringValue = value; }

}

}

}

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.ServiceModel.Web;

using System.Text;

namespace EmployeeWCF

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "Service1" in code, svc and config file together.

// NOTE: In order to launch WCF Test Client for testing this service, please select Service1.svc or Service1.svc.cs at the Solution Explorer and start debugging.

public class Service1 : IEmployeeService

{

public string GetData(int value)

{

return string.Format("You entered: {0}", value);

}

public string AddEmployeeRecord(Employee emp)

{

string result = "";

try

{

SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["SQLDBConnectionString"].ToString());

SqlCommand cmd = new SqlCommand();

string Query = @"INSERT INTO [FSE].[dbo].[Employees]([EmployeeId],[EmployeeName],[Location],[Salary],[Gender],[Phone])VALUES

(@EmpID,@Name,@Location,@Salary,@Gender,@Phone)";

cmd = new SqlCommand(Query, con);

cmd.Parameters.AddWithValue("@EmpID", emp.EmployeeId);

cmd.Parameters.AddWithValue("@Name", emp.EmployeeName);

cmd.Parameters.AddWithValue("@Location", emp.Location);

cmd.Parameters.AddWithValue("@Salary", emp.Salary);

cmd.Parameters.AddWithValue("@Gender", emp.Gender);

cmd.Parameters.AddWithValue("@Phone", emp.Phone);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

result = "Record Added Successfully !";

}

catch (FaultException fex)

{

result = "Error";

}

return result;

}

public List<Employee> GetEmployees()

{

List<Employee> lst = new List<Employee>();

try

{

SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["SQLDBConnectionString"].ToString());

string Query = "SELECT \* FROM Employees";

DataSet ds = new DataSet();

SqlDataAdapter sda = new SqlDataAdapter(Query, con);

sda.Fill(ds);

lst = ds.Tables[0].DataTableToList<Employee>();

}

catch (FaultException fex)

{

throw new FaultException<string>("Error: " + fex);

}

return lst;

}

public string DeleteEmployeeRecord(int Id)

{

string result = "";

SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["SQLDBConnectionString"].ToString());

SqlCommand cmd = new SqlCommand();

string Query = "DELETE FROM Employees Where EmployeeId=@EmpID";

cmd = new SqlCommand(Query, con);

cmd.Parameters.AddWithValue("@EmpID", Id);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

result = "Record Deleted Successfully!";

return result;

}

public Employee SearchEmployeeRecord(int Id)

{

DataSet ds = new DataSet();

Employee e = new Employee();

try

{

SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["SQLDBConnectionString"].ToString());

string Query = "SELECT \* FROM Employees WHERE EmployeeId=@EmpID";

SqlDataAdapter sda = new SqlDataAdapter(Query, con);

sda.SelectCommand.Parameters.AddWithValue("@EmpID", Id);

sda.Fill(ds);

e = ds.Tables[0].DataTableToList<Employee>().ToList().FirstOrDefault();

}

catch (FaultException fex)

{

throw new FaultException<string>("Error: " + fex);

}

return e;

}

public string UpdateEmployeeRecord(Employee emp)

{

string result = "";

SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["SQLDBConnectionString"].ToString());

SqlCommand cmd = new SqlCommand();

string Query = "UPDATE Employees SET EmployeeName=@Name,Location=@Location,Salary=@Salary,Gender=@Gender,Phone=@Phone WHERE EmployeeId=@EmpID";

cmd = new SqlCommand(Query, con);

cmd.Parameters.AddWithValue("@EmpID", emp.EmployeeId);

cmd.Parameters.AddWithValue("@Name", emp.EmployeeName);

cmd.Parameters.AddWithValue("@Location", emp.Location);

cmd.Parameters.AddWithValue("@Salary", emp.Salary);

cmd.Parameters.AddWithValue("@Gender", emp.Gender);

cmd.Parameters.AddWithValue("@Phone", emp.Phone);

con.Open();

cmd.ExecuteNonQuery();

result = "Record Updated Successfully !";

con.Close();

return result;

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Reflection;

using System.Web;

namespace EmployeeWCF

{

public static class Helper

{

public static List<T> DataTableToList<T>(this DataTable table) where T : class, new()

{

try

{

List<T> list = new List<T>();

foreach (var row in table.AsEnumerable())

{

T obj = new T();

foreach (var prop in obj.GetType().GetProperties())

{

try

{

PropertyInfo propertyInfo = obj.GetType().GetProperty(prop.Name);

propertyInfo.SetValue(obj, Convert.ChangeType(row[prop.Name], propertyInfo.PropertyType), null);

}

catch

{

continue;

}

}

list.Add(obj);

}

return list;

}

catch

{

return null;

}

}

}

}

Hosts Code

<?xml version="1.0"?>

<configuration>

<appSettings/>

<connectionStrings>

<add name="SQLDBConnectionString" connectionString="Data Source=NA03OSDVP00679;Initial Catalog=FSE; integrated security=SSPI;Persist Security Info=True;Trusted\_Connection=Yes" providerName="System.Data.SqlClient"/>

</connectionStrings>

<system.web>

<compilation debug="true" targetFramework="4.0"/>

<!--

The <authentication> section enables configuration

of the security authentication mode used by

ASP.NET to identify an incoming user.

-->

<authentication mode="Windows"/>

<!--

The <customErrors> section enables configuration

of what to do if/when an unhandled error occurs

during the execution of a request. Specifically,

it enables developers to configure html error pages

to be displayed in place of a error stack trace.

<customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">

<error statusCode="403" redirect="NoAccess.htm" />

<error statusCode="404" redirect="FileNotFound.htm" />

</customErrors>

-->

<pages controlRenderingCompatibilityVersion="3.5" clientIDMode="AutoID"/>

</system.web>

<!--

The system.webServer section is required for running ASP.NET AJAX under Internet

Information Services 7.0. It is not necessary for previous version of IIS.

-->

<system.webServer>

<!--

To browse web app root directory during debugging, set the value below to true.

Set to false before deployment to avoid disclosing web app folder information.

-->

<directoryBrowse enabled="true"/>

</system.webServer>

<system.serviceModel>

<services>

<service behaviorConfiguration="EmployeeWCF.Service1Behavior" name="EmployeeWCF.Service1">

<endpoint address="" binding="basicHttpBinding" contract="EmployeeWCF.IEmployeeService">

</endpoint>

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

<host>

<baseAddresses>

<add baseAddress="http://localhost:8080/EmployeeService"/>

</baseAddresses>

</host>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="EmployeeWCF.Service1Behavior">

<!-- To avoid disclosing metadata information, set the value below to false before deployment -->

<serviceMetadata httpGetEnabled="true"/>

<!-- To receive exception details in faults for debugging purposes, set the value below to true. Set to false before deployment to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="false"/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

<startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/></startup></configuration>

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.ServiceModel;

using System.Text;

namespace WCFHost

{

class Program

{

static void Main(string[] args)

{

using (ServiceHost host = new ServiceHost(typeof(EmployeeWCF.Service1)))

{

host.Open();

Console.WriteLine("Service Hosted : " + DateTime.Now.ToString());

Console.ReadLine

();

}

}

}

}

Client Code

HomeController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace EmployeeMVCClient.Controllers

{

public class HomeController : Controller

{

//

// GET: /Home/

public ActionResult Index()

{

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

List<EmployeeClient.Employee> Employees = client.GetEmployees().ToList();

return View(Employees);

}

[HttpGet]

public ActionResult EditEmployee(int Id)

{

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

EmployeeClient.Employee Employee = client.SearchEmployeeRecord(Id);

return View(Employee);

}

[HttpPost]

[ActionName("EditEmployee")]

public ActionResult EditEmployee\_Post()

{

if (ModelState.IsValid)

{

EmployeeClient.Employee e = new EmployeeClient.Employee();

TryUpdateModel(e);

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

client.UpdateEmployeeRecord(e);

RedirectToAction("Index");

}

return View();

}

[HttpGet]

public ActionResult DeleteEmployee(int Id)

{

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

client.DeleteEmployeeRecord(Id);

return View("Index");

}

[HttpGet]

public ActionResult DetailsEmployee(int Id)

{

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

EmployeeClient.Employee Employee = client.SearchEmployeeRecord(Id);

return View(Employee);

}

[HttpGet]

public ActionResult AddEmployee()

{

return View();

}

[HttpPost]

[ActionName("AddEmployee")]

public ActionResult AddEmployee\_Post()

{

if (ModelState.IsValid)

{

EmployeeClient.Employee e = new EmployeeClient.Employee();

UpdateModel(e);

EmployeeClient.EmployeeServiceClient client = new EmployeeClient.EmployeeServiceClient();

string Message = client.AddEmployeeRecord(e);

RedirectToAction("Index");

}

return View();

}

}

}

AddEmployee.cshtml

@model EmployeeMVCClient.EmployeeClient.Employee

@{

ViewBag.Title = "AddEmployee";

}

<h2>AddEmployee</h2>

@using (Html.BeginForm("AddEmployee", "Home", FormMethod.Post))

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Employee</legend>

<table>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.EmployeeId)

</div>

</td>

<td>

<div class="editor-field">

@Html.EditorFor(model => model.EmployeeId)

@Html.ValidationMessageFor(model => model.EmployeeId)

</div>

</td>

</tr>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.EmployeeName)

</div>

</td>

<td>

<div class="editor-field">

@Html.EditorFor(model => model.EmployeeName)

@Html.ValidationMessageFor(model => model.EmployeeName)

</div>

</td>

</tr>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.Gender)

</div>

</td>

<td>

<div class="editor-field">

@Html.RadioButtonFor(model => model.Gender, false, "Male") Male

@Html.RadioButtonFor(model => model.Gender, true, "Female") Female

@Html.ValidationMessageFor(model => model.Gender)

</div>

</td>

</tr>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.Location)

</div>

</td>

<td>

<div class="editor-field">

@Html.EditorFor(model => model.Location)

@Html.ValidationMessageFor(model => model.Location)

</div>

</td>

</tr>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.Phone)

</div>

</td>

<td>

<div class="editor-field">

@Html.EditorFor(model => model.Phone)

@Html.ValidationMessageFor(model => model.Phone)

</div>

</td>

</tr>

<tr>

<td>

<div class="editor-label">

@Html.LabelFor(model => model.Salary)

</div>

</td>

<td>

<div class="editor-field">

@Html.EditorFor(model => model.Salary)

@Html.ValidationMessageFor(model => model.Salary)

</div>

</td>

</tr>

<tr>

<td>

<p>

<input type="submit" value="Create" />

</p>

</td>

<td> </td>

</tr>

</table>

</fieldset>

@ViewBag.Message;

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

DetailsEmployee.cshtml

@model EmployeeMVCClient.EmployeeClient.Employee

@{

ViewBag.Title = "DetailsEmployee";

}

<h2>DetailsEmployee</h2>

<fieldset>

<legend>Employee</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.EmployeeName)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.EmployeeName)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Gender)

</div>

<div class="display-field">

@Html.RadioButtonFor(model => model.Gender, false, "Male") Male

@Html.RadioButtonFor(model => model.Gender, true, "Female") Female

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Location)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Location)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Phone)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Phone)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.Salary)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.Salary)

</div>

</fieldset>

<p>

@Html.ActionLink("Edit", "Edit", new { id=Model.EmployeeId }) |

@Html.ActionLink("Back to List", "Index")

</p>

EditEmployee.cshtml

@model EmployeeMVCClient.EmployeeClient.Employee

@{

ViewBag.Title = "EditEmployee";

}

<h2>EditEmployee</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Employee</legend>

@Html.HiddenFor(model => model.EmployeeId)

<div class="editor-label">

@Html.LabelFor(model => model.EmployeeName)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.EmployeeName)

@Html.ValidationMessageFor(model => model.EmployeeName)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Gender)

</div>

<div class="editor-field">

@Html.RadioButtonFor(model => model.Gender, false, "Male") Male

@Html.RadioButtonFor(model => model.Gender, true, "Female") Female

@Html.ValidationMessageFor(model => model.Gender)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Location)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Location)

@Html.ValidationMessageFor(model => model.Location)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Phone)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Phone)

@Html.ValidationMessageFor(model => model.Phone)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.Salary)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.Salary)

@Html.ValidationMessageFor(model => model.Salary)

</div>

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

Index.cshtml

@model IEnumerable<EmployeeMVCClient.EmployeeClient.Employee>

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Create New", "AddEmployee")

</p>

<table>

<tr>

<th>

@Html.DisplayNameFor(model => model.EmployeeName)

</th>

<th>

@Html.DisplayNameFor(model => model.Gender)

</th>

<th>

@Html.DisplayNameFor(model => model.Location)

</th>

<th>

@Html.DisplayNameFor(model => model.Phone)

</th>

<th>

@Html.DisplayNameFor(model => model.Salary)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.EmployeeName)

</td>

<td>

@(item.Gender ? "Female" :"Male" )

</td>

<td>

@Html.DisplayFor(modelItem => item.Location)

</td>

<td>

@Html.DisplayFor(modelItem => item.Phone)

</td>

<td>

@Html.DisplayFor(modelItem => item.Salary)

</td>

<td>

@Html.ActionLink("Edit", "EditEmployee", new { id=item.EmployeeId }) |

@Html.ActionLink("Details", "DetailsEmployee", new { id=item.EmployeeId }) |

@Html.ActionLink("Delete", "DeleteEmployee", new { id=item.EmployeeId })

</td>

</tr>

}

</table>

2.WCF Service Task, Host in Windows Service

Create a WCF Weather service to implement

a.

Double celciustofarenheit(double temp)

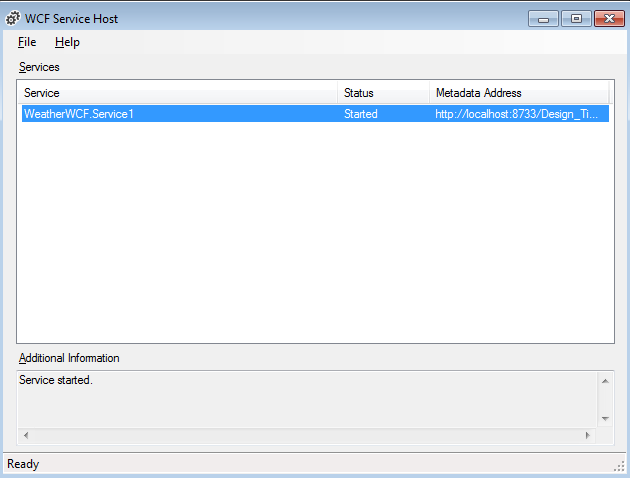
b.Double farenheittocelcius (double temp)

Host the service in Windows Service

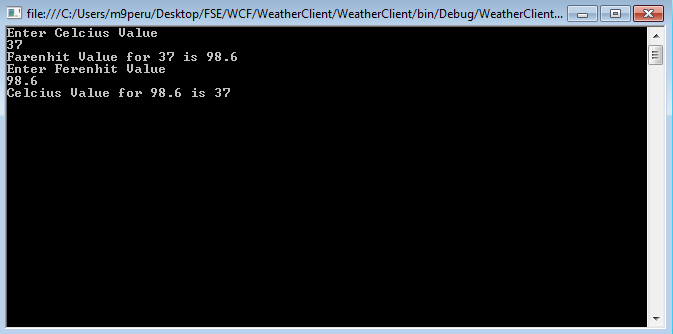
Consume the service in Console App

Design web form two perform crud operations

Output:



Client Output



Iservice1.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

namespace WeatherWCF

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IService1" in both code and config file together.

[ServiceContract]

public interface IService1

{

[OperationContract]

double Celcius(double f);

[OperationContract]

double Farenhit(double celsius);

}

}

Service1.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

namespace WeatherWCF

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "Service1" in both code and config file together.

public class Service1 : IService1

{

public double Celcius(double f)

{

double c = 5.0 / 9.0 \* (f - 32);

return c;

}

public double Farenhit(double celsius)

{

double faren = (celsius \* 9) / 5 + 32;

return faren;

}

}

}

Host

Weatherservices.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Linq;

using System.ServiceModel;

using System.ServiceProcess;

using System.Text;

namespace WCFHost

{

public partial class WeatherService : ServiceBase

{

public ServiceHost serviceHost = null;

public WeatherService()

{

InitializeComponent();

}

protected override void OnStart(string[] args)

{

if (serviceHost != null)

{

serviceHost.Close();

}

//Create a URI to serve as the base address

//Uri httpUrl = new Uri("http://localhost:8090/MyService/WeatherService");

serviceHost = new ServiceHost(typeof(WeatherWCF.Service1));

// Open the ServiceHostBase to create listeners and start

// listening for messages.

serviceHost.Open();

}

protected override void OnStop()

{

if (serviceHost != null)

{

serviceHost.Close();

serviceHost = null;

}

}

}

}

App.config

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.web>

<compilation debug="true" />

</system.web>

<!-- When deploying the service library project, the content of the config file must be added to the host's

app.config file. System.Configuration does not support config files for libraries. -->

<system.serviceModel>

<services>

<service name="WeatherWCF.Service1">

<host>

<baseAddresses>

<add baseAddress = "http://localhost:8733/Design\_Time\_Addresses/WeatherWCF/Service1/" />

</baseAddresses>

</host>

<!-- Service Endpoints -->

<!-- Unless fully qualified, address is relative to base address supplied above -->

<endpoint address="" binding="basicHttpBinding" contract="WeatherWCF.IService1">

<!--

Upon deployment, the following identity element should be removed or replaced to reflect the

identity under which the deployed service runs. If removed, WCF will infer an appropriate identity

automatically.

-->

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<!-- Metadata Endpoints -->

<!-- The Metadata Exchange endpoint is used by the service to describe itself to clients. -->

<!-- This endpoint does not use a secure binding and should be secured or removed before deployment -->

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior>

<!-- To avoid disclosing metadata information,

set the value below to false before deployment -->

<serviceMetadata httpGetEnabled="True"/>

<!-- To receive exception details in faults for debugging purposes,

set the value below to true. Set to false before deployment

to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="False" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

</configuration>

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.ServiceProcess;

using System.Text;

namespace WCFHost

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

static void Main()

{

#if(!Debug)

ServiceBase[] ServicesToRun;

ServicesToRun = new ServiceBase[]

{

new WeatherService()

};

ServiceBase.Run(ServicesToRun);

#else

WeatherService ws=new WeatherService();

ws.OnStart();

#endif

}

}

}

Client.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WeatherClient

{

class Program

{

static void Main(string[] args)

{

//creating the object of WCF service client

ServiceReference1.Service1Client weather = new ServiceReference1.Service1Client();

Double Celcius,Ferenhit;

//assigning the input values to the variables

Console.WriteLine("Enter Celcius Value");

Double.TryParse(Console.ReadLine(),out Celcius);

Console.WriteLine("Farenhit Value for {0} is {1}",Celcius,weather.Farenhit(Celcius));

Console.WriteLine("Enter Ferenhit Value");

Double.TryParse(Console.ReadLine(), out Ferenhit);

Console.WriteLine("Celcius Value for {0} is {1}", Ferenhit, weather.Celcius(Ferenhit));

Console.ReadLine();

}

}

}