# A WCF Service Application with an XML DataStore Serialize/Deserialize & a “Tester”

## About

This project presents two separate WCF Service Applications and a client "tester" console application in the same Visual Studio solution. The WCF Services were hosted using IIS Express. A simple console "test" application connects to both of the services through separate proxy references. There are two services: school service (the main demo) and a basic math service.

## School Service

### Operational Contracts

The School Service allows for the following operational contracts:

/// <summary>

/// AddStudent

/// Accepts all necessary values to create a new Student object

/// </summary>

/// <param name="lastName">(string) last name of the Person</param>

/// <param name="firstName">(string) first name of the Person</param>

/// <param name="dob">(DateTime) date of birth of the Person</param>

/// <param name="gender">(GenderEnum) gender of the Person</param>

/// <param name="id">(string) identifier of the Person</param>

/// <param name="major">(string) major of the Student</param>

/// <param name="units">(float) completed units of the Student</param>

/// <param name="gpa">(float) GPA of the student</param>

/// <returns>Student object</returns>

[OperationContract]

Student AddStudent(string lastName, string firstName, DateTime dob, GenderEnum gender, string id, string major, float units, float gpa);

/// <summary>

/// DeleteStudent

/// Finds a Student by the Person ID and deletes the object

/// </summary>

/// <param name="id">(string) identifier of the Person</param>

[OperationContract]

void DeleteStudent(string id);

/// <summary>

/// GetStudent

/// Finds and returns the Student by the Person ID

/// </summary>

/// <param name="id">(string) identifier of the Person</param>

/// <returns>(Student) student object</returns>

[OperationContract]

Student GetStudent(string id);

/// <summary>

/// GetStudents

/// Finds all the Student objects and

/// returns them as an enumerable object

/// holding all the students

/// </summary>

/// <returns>List<Student> list of student objects</returns>

[OperationContract]

List<Student> GetStudents();

/// <summary>

/// UpdateStudent

/// Accepts values corresponding to all Student properties

/// First Deletes the student and then adds a new student with

/// the given parameters. Only affects the First of Default

/// Student ID object that matches the input parameter.

/// </summary>

/// <param name="lastName">(string) last name of the Person</param>

/// <param name="firstName">(string) first name of the Person</param>

/// <param name="dob">(DateTime) date of birth of the Person</param>

/// <param name="gender">(GenderEnum) gender of the Person</param>

/// <param name="id">(string) identifier of the Person</param>

/// <param name="major">(string) major of the Student</param>

/// <param name="units">(float) completed units of the Student</param>

/// <param name="gpa">(float) GPA of the student</param>

/// <returns>(Student) student that was deleted and then added (updated)</returns>

[OperationContract]

Student UpdateStudent(string lastName, string firstName, DateTime dob, GenderEnum gender, string id, string major, float units, float gpa);

/// <summary>

/// AddTeacher

/// Accepts all necessary values to create a new Teacher object

/// </summary>

/// <param name="lastName">(string) last name of the Person</param>

/// <param name="firstName">(string) first name of the Person</param>

/// <param name="dob">(DateTime) date of birth of the Person</param>

/// <param name="gender">(GenderEnum) gender of the Person</param>

/// <param name="id">(int) identifier of the Teacher</param>

/// <param name="dateofhire">(DateTine) date of hire of the Teacher</param>

/// <param name="salary">(int) salary of the Teacher</param>

/// <returns>(Teacher) teacher object</returns>

[OperationContract]

Teacher AddTeacher(string lastName, string firstName, DateTime dob, GenderEnum gender, int idTeacher, DateTime dateofhire, int salary);

/// <summary>

/// DeleteTeacher

/// Deletes a Teacher object by its Teacher ID

/// </summary>

/// <param name="id">(int) identifier of the Teacher</param>

[OperationContract]

void DeleteTeacher(int id);

/// <summary>

/// GetTeacher

/// Gets a Teacher object by its Teacher ID

/// </summary>

/// <param name="id">(int) identifier of the Teacher</param>

/// <returns>(Teacher) teacher object</returns>

[OperationContract]

Teacher GetTeacher(int id);

/// <summary>

/// GetTeachers()

/// Returns a list of Teacher objects

/// </summary>

/// <returns>(List<Teacher>) teachers</returns>

[OperationContract]

List<Teacher> GetTeachers();

/// <summary>

/// UpdateTeacher

/// First deletes, then adds a new Teacher

/// object with a unique Teacher ID that is first

/// or default in the data store

/// </summary>

/// <param name="lastName">(string) last name of the Person</param>

/// <param name="firstName">(string) first name of the Person</param>

/// <param name="dob">(DateTime) date of birth of the Person</param>

/// <param name="gender">(GenderEnum) gender of the Person</param>

/// <param name="id">(int) identifier of the Teacher</param>

/// <param name="dateofhire">(DateTine) date of hire of the Teacher</param>

/// <param name="salary">(int) salary of the Teacher</param>

/// <returns>(Teacher) teacher object</returns>

[OperationContract]

Teacher UpdateTeacher(string lastName, string firstName, DateTime dob, GenderEnum gender, int id, DateTime dateofhire, int salary);

/// <summary>

/// GetPeople

/// Returns a collection object representing a List<Person>

/// that is a list of Person objects with the last name

/// and first name provided

/// </summary>

/// <param name="lastName">(string) last name of the Person</param>

/// <param name="firstName">(string) first name of the Person</param>

/// <returns>(PersonList) collection of people with the first and last name specified

/// as an input parameter</returns>

[OperationContract]

PersonList GetPeople(string lastName, string firstName);

### DataContract Modeling (Data Model)

It is important to understand the details of how the person, student, teacher, and PersonList (List<Person>) objects are modeled in this solution. The PersonList is just a CollectionDataContract that derives from List<Person>. This just models a list of Person objects which is just a single DataContract model. We use the Collection of PersonList as the in-memory datastore and object to be serialized/deserialized. The Person object is the base model for both the derived Student and Teacher classes. In the modeling contract for this base object we must remember to explicitly identify the polymorphic relationships to the base parent like this:

[DataContract]

[KnownType(typeof(Student))]

[KnownType(typeof(Teacher))]

public class Person ....

Then when we inherit from this base parent other objects like Student and Teacher...

[DataContract]

public class Student : Person

We can continue with the standard DataContract and DataMember pattern modeling of our data class.

Note: Only the state data like field/properties are used for serialization/deserialization and transport across the wire via our service. We can put any additional methods in our models for the service to utilize, but the clients connecting to our service only care about the state data flowing back to them.

Check out the WCF DataContract modeling for each of the types of objects:

PersonList: []

Person: []

Student: []

Teacher: []

### DataStore (In Memory) Serialization and Deserialization

The school service has a backend DataStore where it serializes and deserializes to/from an XML file each person, student, or teacher object (*polymorphism is used here Person => Student & Person => Teacher)* into a collection of PersonList (List<Person>) with Person objects. The XML attribute in the collection’s Person object tag holds the polymorphic type (*example: i:type="Student").* It's best to just look at the XML example of the DataStore to see how the polymorphic objects are serialized into XML.

My serialization demo creates an XML file called "PersonData.xml" in a temporary directory (*C:\Users\YourUserName\AppData\Local\Temp*) on the user hard drive (this is a simple demo program, not production). If desired, a different data storage location can be specified in the code.

To see a sample of what the XML looks like when it serializes the DataContract models, please reference this file here:

[*insert demo XML file*]

The DataStore utility class holds a reference to the data store object that is deserialized into memory upon loading/starting of the service. If the temporary file does not exist, the service creates one. The service calls upon the DataStore utility class to retrieve the now in-memory database and do basic crud operations by reference to this database object in memory. After each create, update, or delete operation by the service, the service calls upon the utility DataStore class to save the in-memory database object to file by serialization.

Basically, the DataStore utility class in the service holds the in-memory database and is responsible for serialization/deserialization. The service performs work by calling this DataStore database by reference into its own class and performs the create, read, update, and delete. Then when the service requests "Save" the DataStore object serializes the in-memory object to a file, in this case, it is XML.

Note: In this particular demo, the temporary datastore file on the hard drive is wiped clean (if it already exists) when the service executes. This was done to facilitate testing of my class project.

## HTTP Client Console Tester

This is a separate project in the Visual Studio solution that tests both the School (see above) and Math (see below) Services.

Remember what gets sent over the wire is just state data. So how does the client know about/access the PersonList, Person, Student, or Teacher DataContract models if they are stored on the service? It's simple. When you build your client to connect through a proxy it will download/translate the DataContract models for you in your "Service Reference". Make sure the service host is exposing its metadata exchange or WSDL data. In this Visual Studio demo, I used the *Add Service Reference Utility (SVCUTIL.EXE)* to make the process easy.

Note/Warning: If you UPDATE the DataContract models on your service host application, YOU MUST, your client application MUST, refresh/update the service reference. It's as easy as right-clicking on the service reference and clicking update while the service host is running and sharing its WSDL.

All that my “tester” client console application did was connect to my service and do some simple CRUD applications and tested each OperationContract "method" on the service. See some of the screen captures in the DEMO.

Note: You must run your service host instance in "without debugging mode" when creating/updating your client tester program in the same solution because VS has its hooks. Furthermore, after you get your client tester application running you may wish to DISABLE the pesky WCF Tester via the project properties that is default in Visual Studio or just specify your solution to load first your service then client projects in the run order.

## Basic Math Service

This is the second available service in my demo service application. It's here because my student lab required it to be here, to show I can do two services running at the same time. If you would like to learn more about it, please reference the preceding project blog article here: <https://portfolio.katiegirl.net/2019/03/14/basic-wcf-service-application-client-console-tester/>

The basic Math service provides basic mathematical operations contracts including:

[OperationContract]

double Add(double value1, double value2);

[OperationContract]

double Subtract(double value1, double value2);

[OperationContract]

double Multiply(double value1, double value2);

[OperationContract]

double Divide(double value1, double value2);

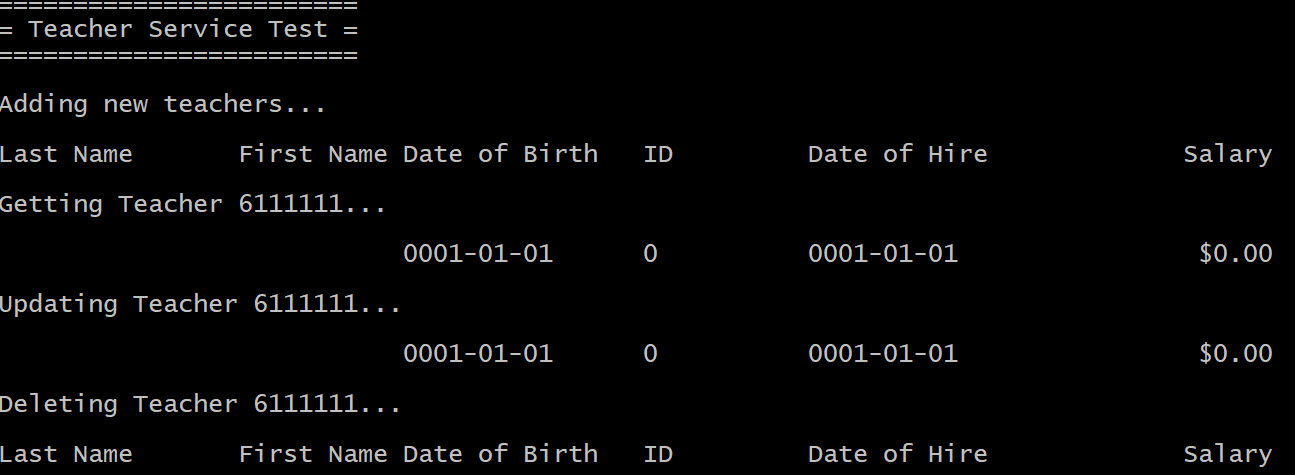
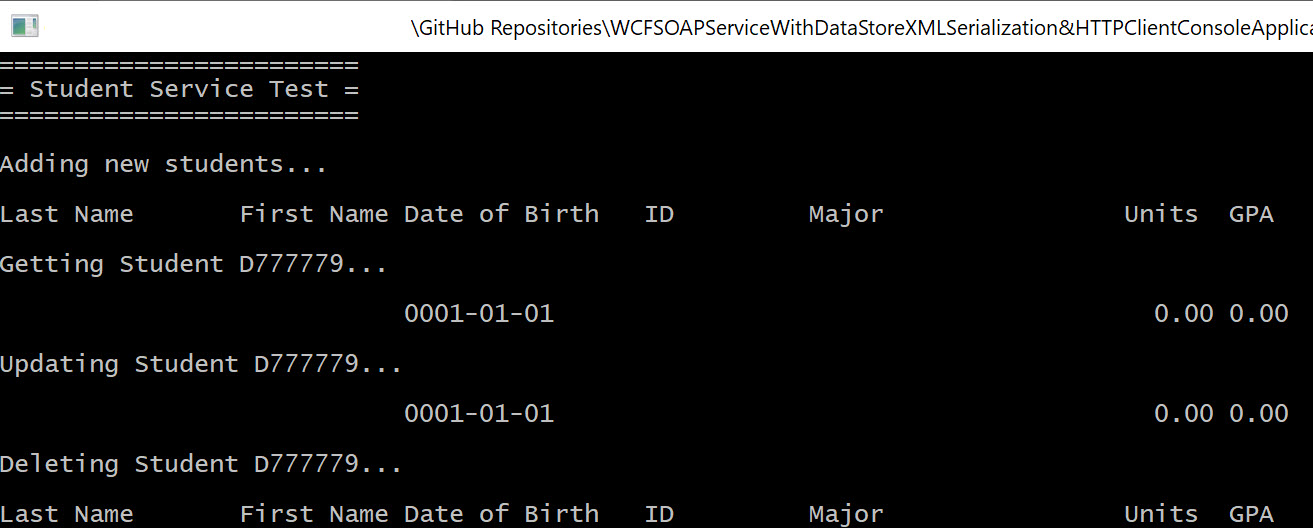
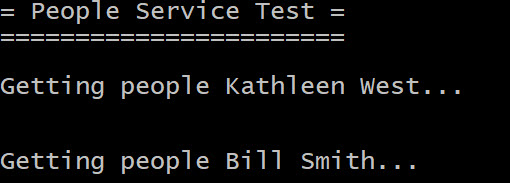
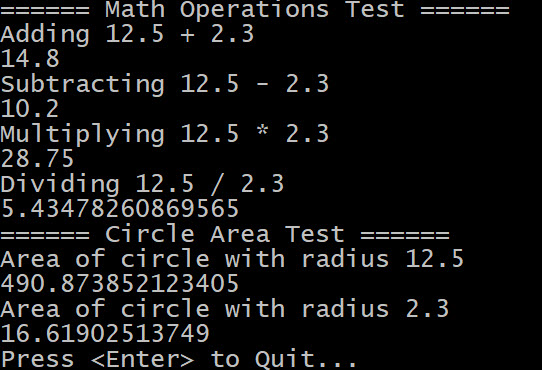
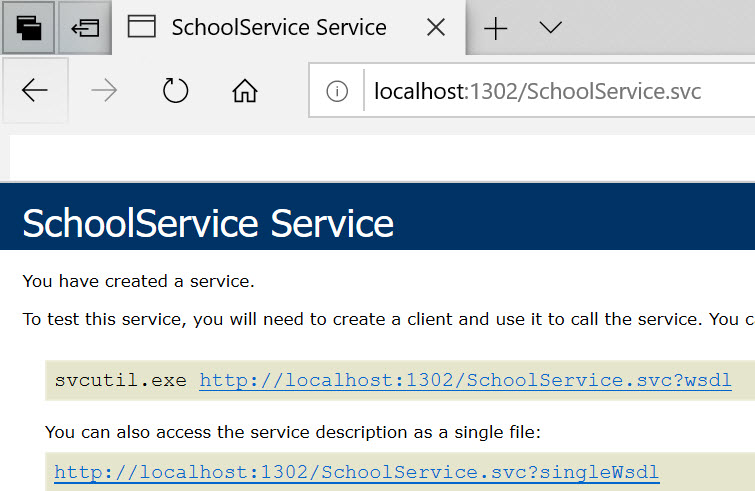
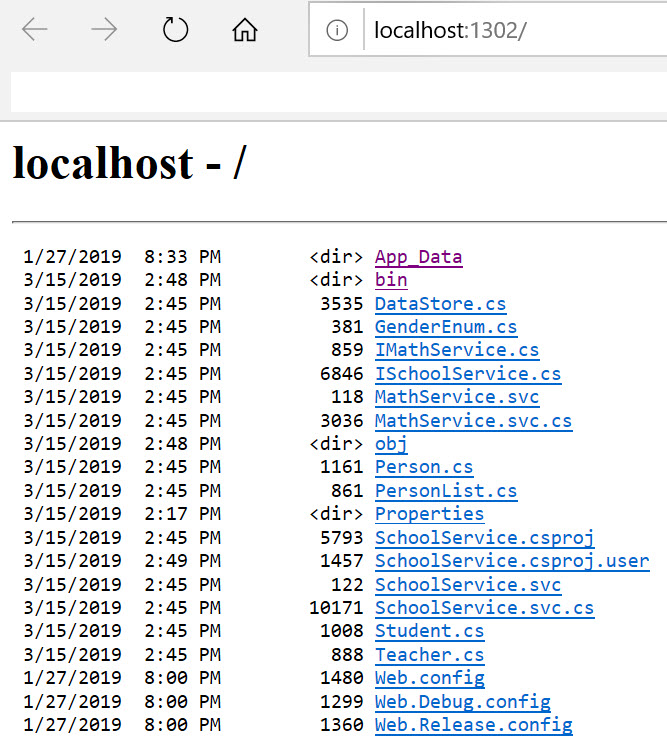
[OperationContract]

double CircleArea(double radius);

The client was expected to access the service using information available through the WSDL. I created my test client application by building a proxy using *svcutil.exe* [*http://localhost:portnumber/MathService.svcutil*](http://localhost:portnumber/MathService.svcutil) *.*

Note: You should update the client proxy or reference to the server before running this in your Visual Studio IDE by first going to *TestClient (Console Application) --> Connected Services --> MathServiceRef* and selecting "Update Service Reference".

## Demo



## Code

You may download a copy of the project by visiting the code repo here.