

Pulling and Updating Data from A SQL Database

Introduction

This guide will provide a basic example of pulling data from a SQL database, processing the data through a Service Objects RESTful web service and then uploading the validated data to a SQL database.

Requirements

- Visual Studio – *VS 2015 was used for this example; but the process should be relatively similar* for other versions of Visual Studio.
- Working knowledge of C# and Object Oriented Programming
- A SQL Database – SQL Server 2008 was used for this example but connecting to many databases should be relatively similar
- A working License Key(trial or production) from Service Objects. Click here to obtain a free trial key for the service you are most interested in.

Contents

Setting up your Visual Studio Project	2
Adding A Response Class for Serialization	3
Method Definitions and App.Config Files.	5
Pulling Information from A SQL Database	6
Calling a DOTS Web Service and Processing the Response	8
Updating a SQL Database with Validated Information	9

Pulling and Updating Data from A SQL Database

Setting up your Visual Studio Project

To begin, launch Visual Studio and create a new project. Under the templates section, select “Visual C#” and then select “Windows” and then “console Application” as shown in the figure below.

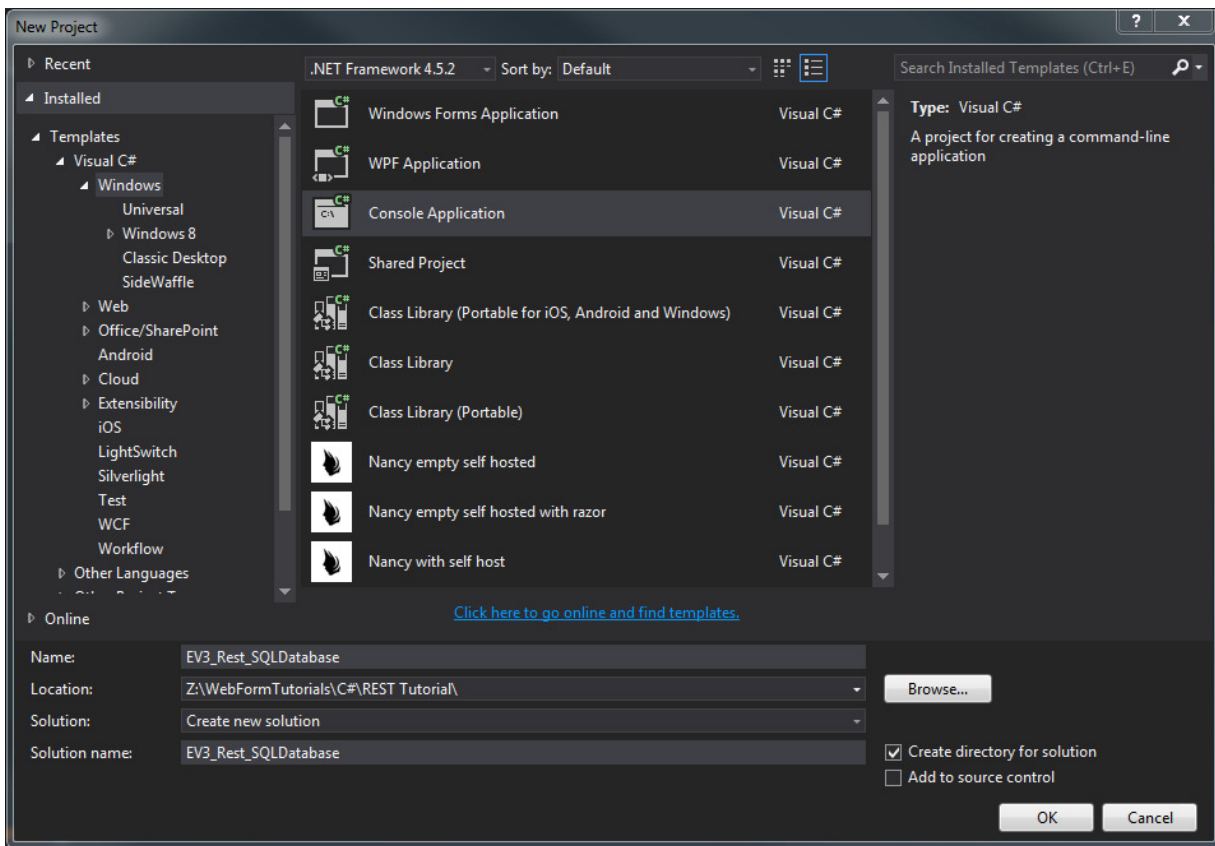


Figure 1

Choose a name and location for your project and select OK.

In this example we will be using the Email Validation 3 service and we will be calling service via HTTP GET. The steps and process of calling a Service Objects web service via REST will be very similar for all the other services; you can simply follow along and with the service you prefer to use for your application.

Pulling and Updating Data from A SQL Database

Adding A Response Class for Serialization

To start, we are going to create a response class that will allow us to serialize the XML returned from the Email Validation 3 service into a C# class. This will allow us to access the values returned from the service easily and in a much more readable way instead of trying to parse the XML response.

To do this, right click the project in the solution explorer, go to “Add” and then select “Add Class” On the pop-up window that make sure the class item is selected and then give it a name. We are using “EV3Response” for this project. Select “OK” to create the code file.

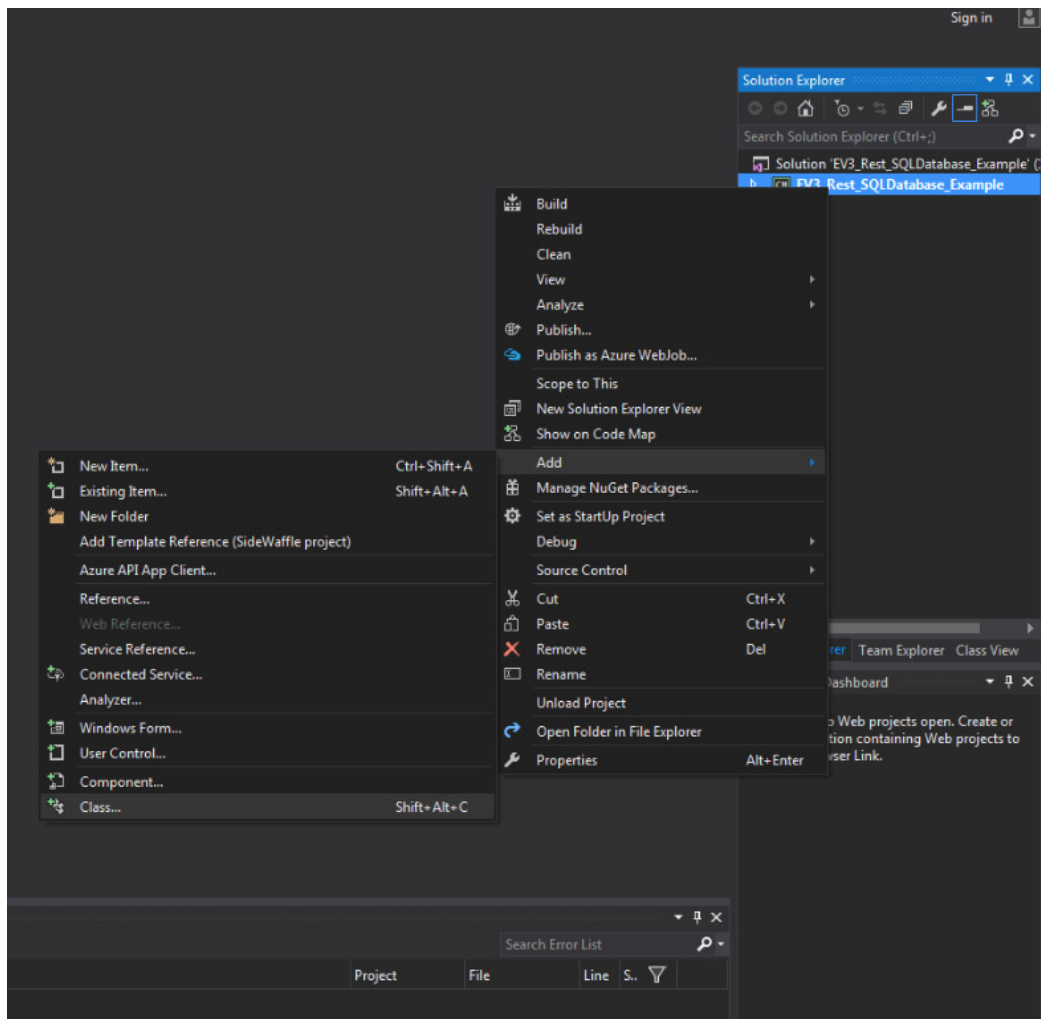


Figure 2

Pulling and Updating Data from A SQL Database

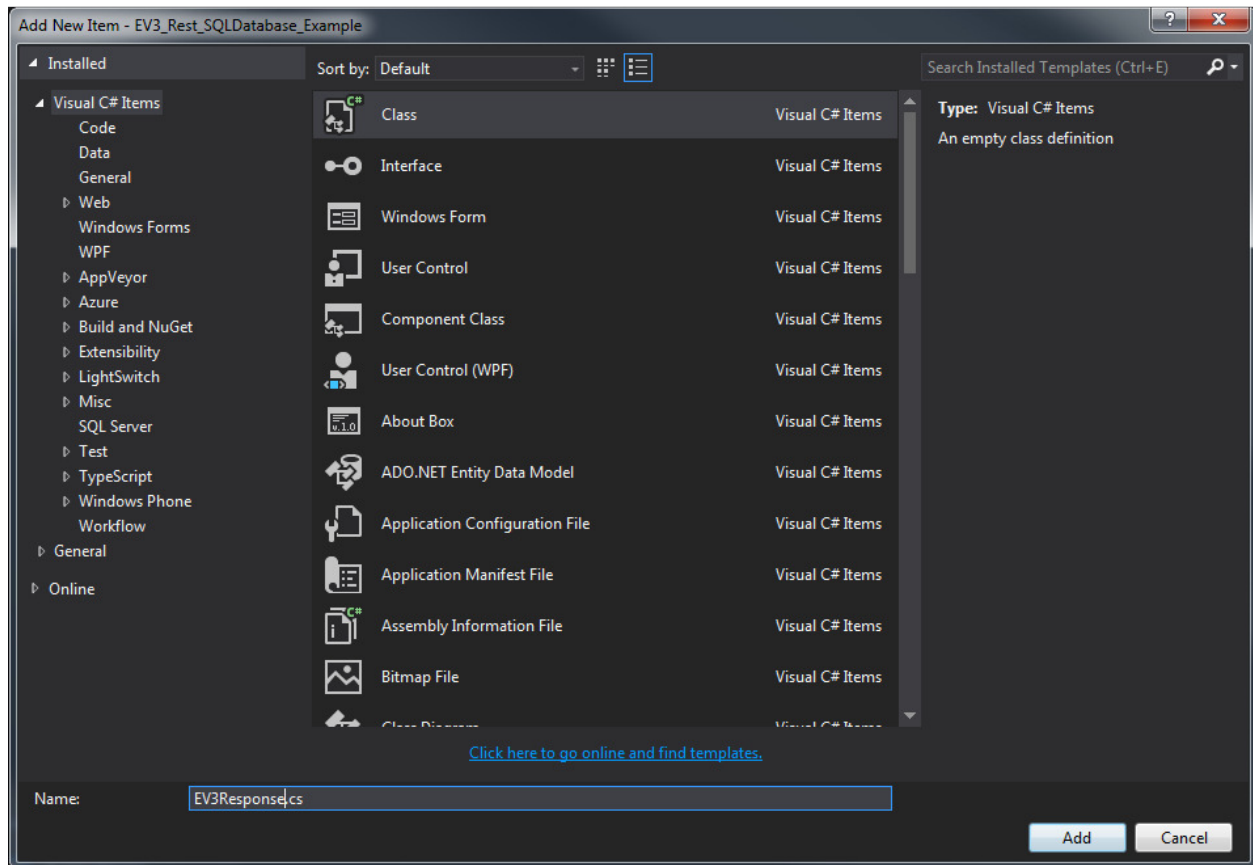


Figure 3

Your new code file will be empty and ready for code to define the response. We will need to define for the program what the structure of the XML response from the EV3 service will look like so that we can properly serialize it into our C# class. Copy and paste the contents from the **EV3Response_Contents.txt** into the EV3Response file in Visual studio.

Now that the EV3 response is defined, let's talk a bit about what we have added. In this file there are 3 class definitions. There is the "ValidateEmailResponse" class which contains both the "ValidateEmailInfo" class and the "Error" class. These subsequent classes contains all the fields that the EV3 service will return. When defining a class like this in C# is important to pay attention to structure of the response that the service returns as well as the type of values that are returned. Both of these items can be found and verified by looking at the WSDL for web service and examining the response type and structure for the service.

Pulling and Updating Data from A SQL Database

Method Definitions and App.Config Files.

Open your “Program.cs” file so that we can begin to define some of the methods that we’ll be using in this project. Define the methods shown in this screenshot within the Program class (Main should already exist). These methods will have all the necessary contents to run the file but we’ll highlight some key aspects of the it in the rest of the tutorial. Be sure to include all the relevant using statements as well.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.IO;
using System.Net;
using System.Xml.Serialization;
using EV3Response;

namespace EV3_Rest_SQLDatabase_Example
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)...

        1 reference
        public static DataTable GetDataFromSQL(string ConnectionString, string sqlCommand)...
        2 references
        public static ValidateEmailResponse EV3RestClient(string URL)...
        1 reference
        public static DataTable FillTable(DataTable prevalidatedEmails)...
        1 reference
        public static void UpdateSQL(DataTable validatedData, string ConnectionString)...
    }
}
```

Figure 4

Now that we have our methods defined we will need to call these methods in the “Main” method of our program. In to make our connections to a SQL database more secure, it is a wise practice to put said connection strings and SQL commands for that matter in the app.config or web.config file of your project. Depending on what database system you are using, the syntax of your connection string may vary. Refer to the webpage www.connectionstrings.com and search for how to format your databases connection string.

In this section of code we pull the connection string from the config files so that we can pass them into the methods to make the connection to the SQL database.

Pulling and Updating Data from A SQL Database

Pulling Information from A SQL Database

Pulling data from a SQL database into a .NET application is a relatively easy process. Below is the screenshot that we will be using in our "GetDataFromSQL" method.

```
1 reference
public static DataTable GetDataFromSQL(string ConnectionString, string sqlCommand)
{
    Console.WriteLine("Pulling Table of Emails from SQL");
    DataTable DS = new DataTable();
    try
    {
        using (SqlConnection Conn = new SqlConnection(ConnectionString))
        {
            Conn.Open();
            using (SqlCommand Cmd = new SqlCommand(sqlCommand, Conn))
            {
                Cmd.Connection = Conn;
                Cmd.CommandType = CommandType.Text;

                Cmd.CommandTimeout = 360;
                SqlDataAdapter adapter = new SqlDataAdapter(Cmd);
                adapter.Fill(DS);
                Conn.Close();

                //Adds the necessary Columns for the EV3 Response to the DataTable
                DS.Columns.Add("Score", typeof(int));
                DS.Columns.Add("EmailAddressIn", typeof(string));
                DS.Columns.Add("EmailAddressOut", typeof(string));
                DS.Columns.Add("EmailCorrected", typeof(string));
                DS.Columns.Add("Box", typeof(string));
                DS.Columns.Add("Domain", typeof(string));
                DS.Columns.Add("TopLevelDomain", typeof(string));
                DS.Columns.Add("TopLevelDomainDescription", typeof(string));
                DS.Columns.Add("IsSMTPServerGood", typeof(string));
                DS.Columns.Add("IsCatchAllDomain", typeof(string));
                DS.Columns.Add("IsSMTPMailBoxGood", typeof(string));
                DS.Columns.Add("WarningCodes", typeof(string));
                DS.Columns.Add("WarningDescriptions", typeof(string));
                DS.Columns.Add("NotesCodes", typeof(string));
                DS.Columns.Add("NotesDescriptions", typeof(string));
                DS.Columns.Add("Error", typeof(string));
                return DS;
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.StackTrace);
        return null;
    }
}
```

Figure 5

This method takes two arguments, the connection string to the SQL database and the SQL command that we will use to retrieve our list of emails. The code will use these two strings to connect to the SQL database, and pull the data from our database. After the connection is made, the SqlDataAdapter will convert the data returned into a DataTable. After the Table is filled with the original

Pulling and Updating Data from A SQL Database

emails, we need to add columns the rest of the data that will be returned from the EV3 service. After all the necessary columns are added the method returns the table.

One thing to note is that the most secure option when pulling data from a SQL database is to use a stored procedure. In this example we simply use a “SELECT” statement which we store in our app.config file. Microsoft provides resources for [creating](#) and [using](#) a stored procedure in .NET framework. Refer to those two resources for more information.

Pulling and Updating Data from A SQL Database

Calling a DOTS Web Service and Processing the Response

The Fill method will call the EV3RestClient method; below are two screen shots of the method contents.

The FillTable method will loop through all the rows in the DataTable that is passed into it, make a call to the EV3 service with each email, and fill the resulting values in to the DataTable object. Depending on how many records are present in your database, it may take a long time to validate all the emails in your database. It may be wise to break the database up into smaller sets of data.

**The mainURL and backupURL will need to be changed from what they are noted as in this project when you move to a production key. In the code snippet above both the backup and primary URL point to the trial.serviceobjects.com server. Upon purchasing a production License Key, the primary and backup URLs should be changed to their respective service paths.*

A failover configuration for a call to a Service Objects is necessary to maintain uninterrupted operation in the unlikely event that the primary Service Objects server is un-operational. Below is a screen shot of the failover configuration.

```
//Failover to backup URL. Will display a note to the user if the backup URL is used.
if (validatedEmail == null || (validatedEmail.Error != null && validatedEmail.Error.TypeCode == "3"))
{
    Console.WriteLine("Backup URL used for email {0}", i);
    restCallURL = string.Empty;
    restCallURL = backupURL + Convert.ToString(row["EMAIL"]) + "&LicenseKey=" + licensekey;
    validatedEmail = EV3RestClient(restCallURL);
}
```

Figure 6

The "IF" statement in this configuration will check for two conditions, and will help ensure that your operation goes uninhibited in the event that the main Service Objects server is unavailable. The first condition the code will check for will be if the response that came back from the web service is null. If it is indeed null, then this would indicate that the web service is unable to respond and that the backup URL should be used. The other condition that the code will check for, is if it receives an error message from the service with a TypeCode of 3. This would indicate that service is returning a "Fatal Error" which means that the service is non-operational. In the event that you see a Fatal Error returned from the service, please contact Service Objects immediately.

After the failover configuration, the code performs another check to see if the Error object in the response is null; if there is no error detected, then the code will add the validated email data into the DataTable. If an error is detected, the Error type and Error description will be added to the Error field of our DataTable. This is not the full data returned by the error object but It will give us enough information to troubleshoot in the event that an error is returned.

Pulling and Updating Data from A SQL Database

Updating a SQL Database with Validated Information

Now that the emails have been validated and all the results from the web service have been saved, they need to be returned to the SQL database. In this example we will be using the `SqlBulkCopy` method and copying the validated email data into a new table. The `SqlBulkCopy` method is typically used for larger data sets and documentation for this method can be found by clicking [here](#). Is a screenshot with the contents of the `UpdateSQL` method.

```
1 reference
public static void UpdateSQL(DataTable validatedData, string connectionString)
{
    Console.WriteLine("Uploading DataTable to SQL");
    try
    {
        using (SqlConnection Conn = new SqlConnection(connectionString))
        using (SqlBulkCopy bulkCopy = new SqlBulkCopy(Conn))
        {
            //Copies the DataTable into the the "EmailTestTable_Validated" table
            Conn.Open();
            bulkCopy.DestinationTableName = "dbo.EmailTestTable_Validated";
            bulkCopy.WriteToServer(validatedData);
            Conn.Close();
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.StackTrace);
    }
}
```

Figure 7

This method is relatively simple and straightforward. The `SqlBulkCopy` method does all the heavy lifting, in this example we're writing the data to a new table that already has the corresponding columns and data types as the `DataTable` that we just filled out with the validated email data. With this being done, our validated Data is back in SQL for further processing.