

ETFOMM Application Programming Interface Report

Version 1.3

Prepared by:

New Global Systems for Intelligent Transportation Management
75 Cavalier Blvd Suite 221
Florence, KY 41042

May 2017

Contents

1. ETFOMM COMPONENT INTERFACE ARCHITECTURE	3
2. MAJOR COMPONENTS	5
2.1 WCF Service Library	5
2.2 etRunner	7
2.3 etfomm.dll	8
2.4 API Client	9
3. API USER MANUAL	11
3.1 API Console Client	11
3.1.1 Console Client Development Guide	11
3.1.2 Console Client Usage Guide	16
3.2 API Web Client.....	20
3.2.1 API Web Client Architecture	20
3.2.2 Web Client Development Guide	21
3.2.3 Web Client Usage Guide.....	31
4. TEST RESULTS.....	37
4.1 Objectives of Test	37
4.2 Testing Plans.....	37
4.3 Test Results.....	37

1. ETFOMM COMPONENT INTERFACE ARCHITECTURE

The ETFOMM component interface is developed with Service-Oriented Architecture (SOA). By using SOA, component configurations are defined as services, which can be invoked by end users through various protocols. Specifically, the ETFOMM interface uses Windows Communication Foundation (WCF) to achieve communications between end users and ETFOMM simulations. WCF is a runtime and a set of APIs in the .NET Framework for building connected, service-oriented applications. The ETFOMM component interface architecture is shown in Figure 1.

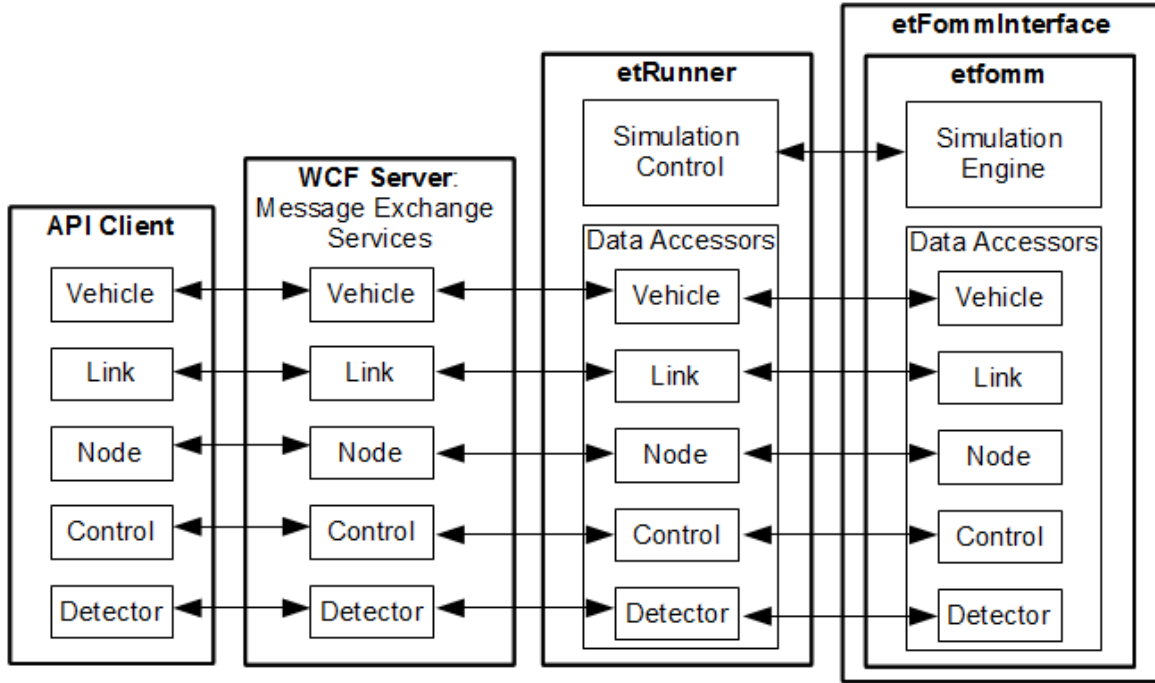


Figure 1: ETFOMM Component Interface Architecture

In this architecture, etfomm is the ETFOMM simulation engine developed in FORTRAN, and compiled into a C++ dynamic-link library. etfomm defines a set of data accessors for setting and getting component configurations. In order to call the etfomm data accessors inside the dynamic-link library, another set of data accessors are defined in etRunner with a one-to-one mapping relationship with the data accessors in etfomm library. The functions in etfomm are invoked through C++ function pointers, which are uniformly managed in etFomMInterface.

The component configuration services are hosted in WCF Server. End users define an API client program, assign values for components, and invoke the WCF services through certain protocol to upload the component data to WCF Server. On the other side, etRunner acts as the controller of the entire system. etRunner invokes the WCF services through the same protocol as that in the API client program to download the component data from WCF Server, and then passes the component data to ETFOMM. When component configurations are finished, etRunner starts the simulation engine inside ETFOMM. At a user-defined interval, the simulation engine can pause and wait for further input from the API client program. At this moment, etRunner fetches data from ETFOMM, and again invokes the WCF services to upload the updated component data to WCF Server. Correspondingly, the API client program invokes the WCF services to download the updated component data, and performs subsequent tasks upon the instructions from the end users. By transferring component data back and forth between the API client program and

ETFOMM simulation engine through WCF Server, the end users can performs tasks including initializing components, inspecting intermediate simulation status, updating component configurations in fly, and controlling the simulation steps. Figure 2 demonstrates the control flow chart of using the ETFOMM component interface.

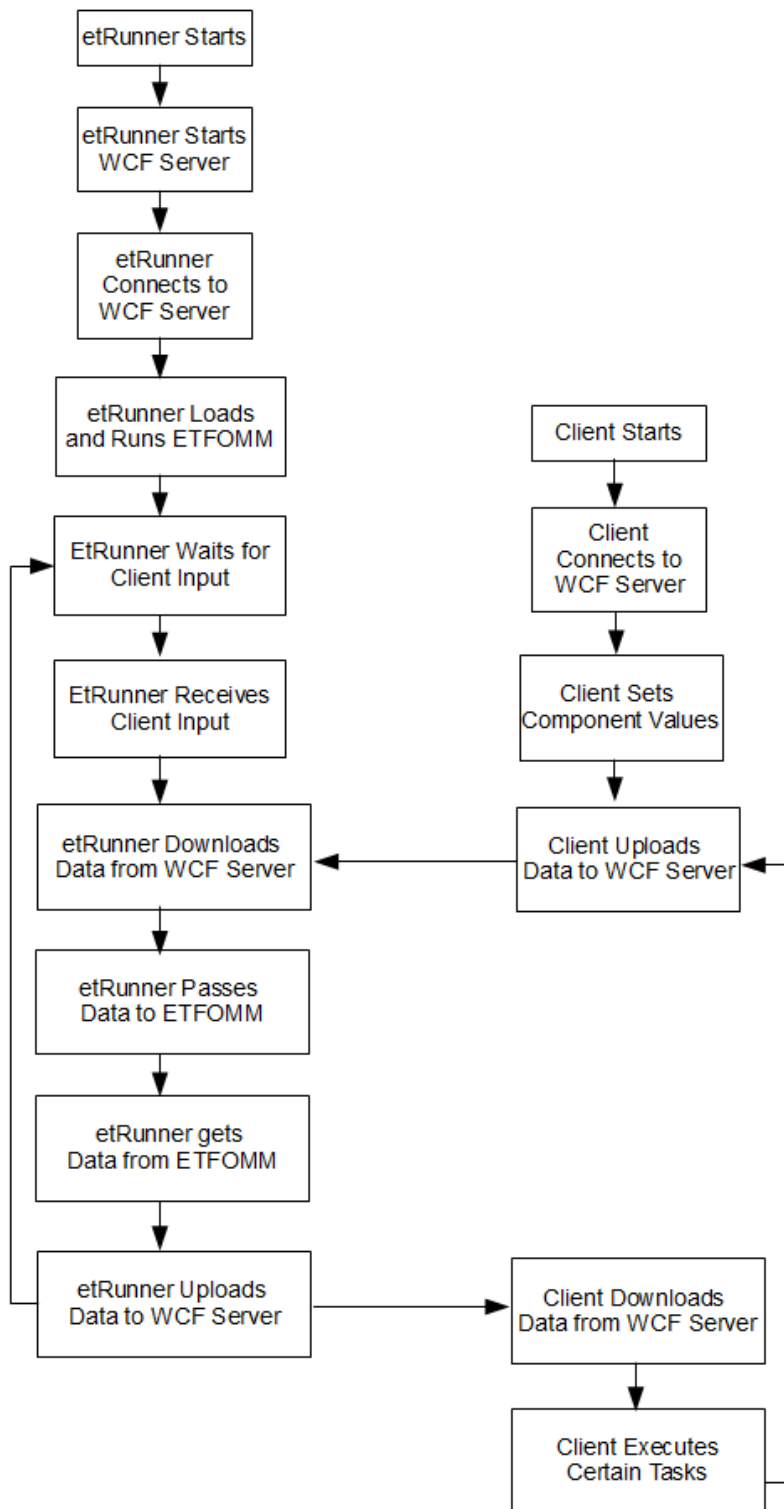


Figure 2: ETFOMM Component Interface Control Flow Chart

2. MAJOR COMPONENTS

2.1 WCF Service Library

WCF Service Library is a DLL that contains exchange data types and functions which can be remotely invoked by WCF clients. In the ETFOMM interface, WCF Service Library is named WCFServer. It includes a header file (**WCFServer.h**) and an implementation file (**WCFServer.CPP**).

2.1.1 WCFServer.h file is used to declare and define the data types, which will be exchanged between WCF Server and Clients, and functions which will be called by WCF Host and each client.

2.1.1.1 Data Type

The data types are declared as **value struct type**. In the WCF Service Library, it is not allowed to declare **struct type**. The user should declare **ref/value/enum struct type**.

The data type should be declared as:

```
//data type of entry node in WCF
public value struct WCF_ENTRYNODES_DATA {
    int Node_ID;
    int flowrate;
    int hov_violators_per10000;
    float carpool_pct;
    float truck_pct;
    array<int>^ lane_pct;
};
```

This is an example of entry node data struct. It includes three INT variables: Node_ID, flowrate, hov_violators_per10000; two FLOAT variables: carpool_pct, truck_pct; and one ARRAY of INT variables of lane_pct.

Hint: Because WCF is based on .Net Framework, WCF is coded in C++/CLI which is supported in .Net Framework. Therefore, int[] doesn't work in the project, programmer must use array<int>^ to define an array of elements.

2.1.1.2 Remote Functions

WCF Service Library should define remote functions for communications between WCF Server Host/Clients. The definition of remote functions is similar as making a protocol.

The function definition should be under Service Contract which is the interface of WCF Service. Definition of each function should use Operation Contract key word.

For example,

```
[ServiceContract]
public interface class IService1
{
    [OperationContract]
    void SetServerEntryNodeData(array <WCF_ENTRYNODES_DATA> ^ wcf_entry_nodes);

    [OperationContract]
    array <WCF_ENTRYNODES_DATA>^ GetServerEntryNodeData();
}
```

```
}
```

These are two function definitions in WCF Service. Each function only has method signature. In the header file, programmer only need to declare the method signature. The implementation should be in the .cpp file.

Programmer should use [ServiceContract] key word before the WCF Service Interface, and [OperationContract] key word before method signature, in order to follow the WCF coding rules.

2.1.2 WCFServer.cpp is the implementation file of WCF Service Library. It is used to implement the class of WCF Service Interface, which contains the data structure of exchange and method implementations of remote functions.

2.1.2.1 Data Structure

In the WCF Service, programmer must create a ref class to implement WCF interface. The code is like:

```
public ref class Service1 : IService1
{
    ...
}
```

The data structure must be in the ref class.

For example,

```
static array<WCF_ENTRYNODES_DATA>^ server_entry_nodes = gcnew
array<WCF_ENTRYNODES_DATA>(1);
```

The above sample defines and initializes one array of vehicles in the WCF Service. The host and clients therefore can exchange vehicle data through WCF server.

The array should be a static variable which will create an application level variable in WCF. Furthermore, the array construction should use **gcnew** instead of **new**.

2.1.2.2 Method Implementations

The example of method implementation is below:

```
virtual void SetServerEntryNodeData(array <WCF_ENTRYNODES_DATA> ^ wcf_entry_nodes)
{
    System::Array::Resize(server_entry_nodes, wcf_entry_nodes->Length);
    for (int i = 0; i < wcf_entry_nodes->Length; i++)
    {
        server_entry_nodes[i].carpool_pct = wcf_entry_nodes[i].carpool_pct;
        server_entry_nodes[i].flowrate = wcf_entry_nodes[i].flowrate;
        server_entry_nodes[i].hov_violators_per10000 =
wcf_entry_nodes[i].hov_violators_per10000;
        server_entry_nodes[i].lane_pct = wcf_entry_nodes[i].lane_pct;
        server_entry_nodes[i].Node_ID = wcf_entry_nodes[i].Node_ID;
        server_entry_nodes[i].truck_pct = wcf_entry_nodes[i].truck_pct;
    }
}

virtual array <WCF_ENTRYNODES_DATA> ^ GetServerEntryNodeData()
```

```

{
    return server_entry_nodes;
}

```

These functions set and get entry node data in WCF Server, correspondingly.

2.1.3 WCF Service Library DLL

After the compilation of WCFServer project, there is a WCFServer.DLL file in the debug folder. WCFServer.DLL should be configured as a reference for etRunner and WCF clients, so that they can use the remote functions defined in WCFServer project. Figure 3 shows the internal structure of the WCF Service Library.

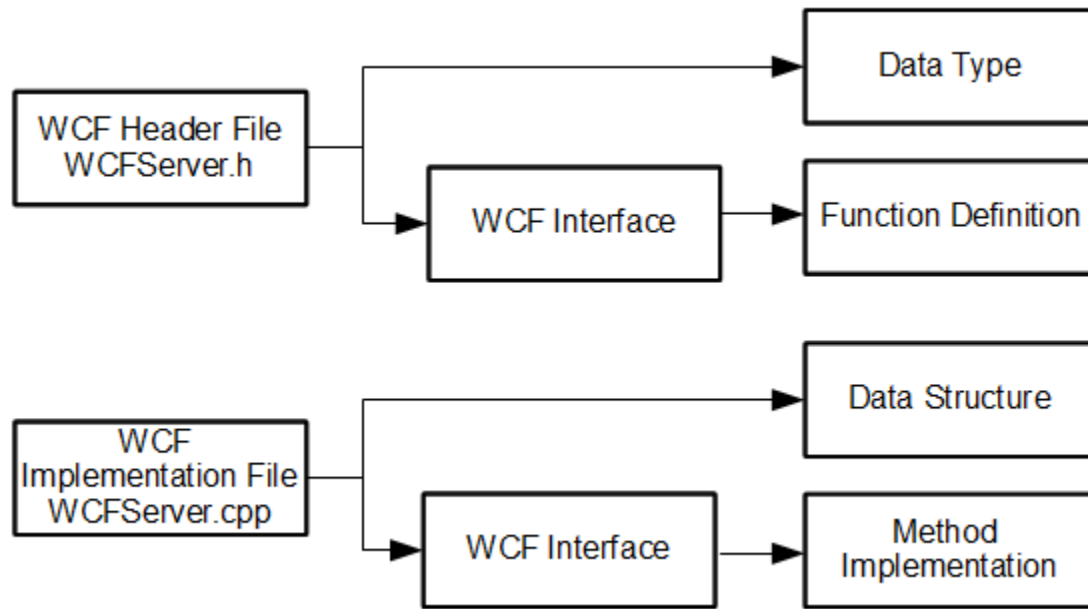


Figure 3: WCF Service Library DLL Internal Structure

2.2 etRunner

etRunner is the control program between WCF Server Host and ETFOMM. It is main program in etFomm API. etRunner starts host of WCF Service and exchanges data between ETFOMM and WCF Server Host.

2.2.1 Start WCF Server Host

The WCF Server Host is one endpoint of an IP Address and a Port. The code to configure the WCF Server Host is below:

```

WSHttpBinding^ sevBinding = gcnw WSHttpBinding();
sevBinding->MaxBufferPoolSize = 2147483647;
sevBinding->MaxReceivedMessageSize = 2147483647;
host->AddServiceEndpoint(IService1::typeid, sevBinding,
L"http://localhost:6000/service");

host->Open();

```

Firstly, programmer should create one Networking Binding, and set its message quota to maximum of 32-bit integer, so the endpoint can handle message exchange of large data struct.

Secondly, programmer should add one service endpoint for service host. The service endpoint takes parameters of WCF Service Interface, Networking Binding type, and IP Address.

Finally, programmer should use Open() method to start WCF Server Host.

2.2.2 Connect WCF Server Host

After starting WCF Server Host, the etRunner should connect to this host. The code to configure etRunner for connection is shown as following:

```
WSHttpBinding^ binding = gcnew WSHttpBinding();
binding->MaxBufferPoolSize = 2147483647;
binding->MaxReceivedMessageSize = 2147483647;
EndpointAddress^ address = gcnew
EndpointAddress(String::Format(L"http://localhost:6000/service",
Environment::MachineName));
ChannelFactory <IService1^>^ factory = gcnew ChannelFactory <IService1^> (binding,
address);
IService1^ proxy;
proxy = factory->CreateChannel();
```

Firstly, programmer should create one Networking Binding, and set its message quota to maximum of 32-bit integer, so the endpoint can handle message exchange of large data struct.

Then, programmer should configure endpoint address which are same as WCF Server Host.

After that, programmer should create a channel factory with WCF Service Interface, endpoint address, and network binding.

Finally, programmer should define a WCF Service Interface, and use CreateChannel() functions to connect WCF Server Host.

Hint: Connect WCF Server Host must use same endpoint address, network binding, path and WCF Service Interface. In the etRunner, connect WCF Server Host is similar as Client. etRunner program could be seen as a mixture of WCF Host Handler and one Client

2.3 etfomm.dll

etFomm.dll is a C++ DLL, which can be called and in a native C++ program. etRunner calls and runs etfomm.dll via another interface class, etFommInterface. etFommInterface provides one-to-one wrapper functions for etfomm.dll functions.

There are three steps to load etFomm.dll in etFommInterface.

1. Redefine etfomm's data types.
2. Define function pointers pointing to etfomm's functions.
3. Load etfomm's functions.

2.3.1 Redefine etFomm's data types.

etFommInterface redefines the data types which should be same as data types in etFomm.

The below code is one sample to display that:


```

struct ENTRYNODES_DATA
{
    int Node_ID;
    int flowrate;
    int hov_violators_per10000;
    float carpool_pct;
    float truck_pct;
    int lane_pct[MAX_LANE_PCT]; //5
};

```

Hint: In the WCF Service Header file, there is the same step to redefine data types. *etFommInterface* and *etRunner* share the same set of data types.

2.3.2. Define function pointers pointing to etfomm's functions.

After redefine the data types, *etFommInterface* defines function pointers pointing to *etfomm*'s functions which will be called in *etRunner*.

The below code is one sample:

```

typedef int (__stdcall *FPTR_ENTRYNODES)(int, int, float, ENTRYNODES_DATA*);

FPTR_ENTRYNODES DEFINE_ENTRYNODES;

```

FPTR_ENTRYNODES is a type of function pointer pointing to a function that have 4 parameters with types of (int, int, float, ENTRYNODES_DATA*) and 1 return value with int type. DEFINE_ENTRYNODES is the function pointer will point to the *etfomm*'s function which defines the values for entry nodes.

2.3.3. Load etFomm's functions.

etFommInterface associates function pointers with *etfomm*'s functions. The below code is one sample:

```

DEFINE_ENTRYNODES = (FPTR_ENTRYNODES)GetProcAddress(hDLL, "define_entrynodes");

```

Hint: The *etRunner* is control program of WCF Server Host and *etFomm.dll*, also it is a bridge between WCF Server Host and *etFomm.dll*.

2.4 API Client

API Client program is a user-defined program to perform certain simulation tasks through invoking functions defined in *etfomm.dll*, such as initializing ETFOMM components, running simulation through *etRunner* or modifying values of ETFOMM components.

Because the API Client is a WCF Client, the programmer should also configure WCF Connection in API Client Program.

The code to configure WCF Connection in API Client Program is shown as following:

```

WSHttpBinding^ binding = gcnew WSHttpBinding();
binding->MaxBufferPoolSize = 2147483647;
binding->MaxReceivedMessageSize = 2147483647;
EndpointAddress^ address = gcnew
EndpointAddress(String::Format(L"http://localhost:6000/service",
Environment::MachineName));

```

```

ChannelFactory<IService1^>^ factory = gcnew ChannelFactory<IService1^>(binding,
address);
IService1^ proxy = factory->CreateChannel();

```

Firstly, programmer should create one Networking Binding, and set its message quota to maximum of 32-bit integer, so the endpoint can handle message exchange of large data struct.

Then, programmer should configure endpoint address which are same as WCF Server Host.

After that, programmer should create a channel factory with WCF Service Interface, endpoint address, and network binding.

Finally, programmer should define a WCF Service Interface, and use CreateChannel() functions to connect WCF Server Host.

After that, programmer can use the following code to access data in WCF Server Host:

```

array <WCF_ENTRYNODES_DATA> ^ wcf_entry_node = gcnew array<WCF_ENTRYNODES_DATA>(1);

proxy->SetServerEntryNodeData(wcf_entry_node);

wcf_entry_node = proxy->GetServerEntryNodeData();

```

WCF Service Library includes accessor functions for each ETFOMM component to allow user get/set data easily and fast.

3. API USER MANUAL

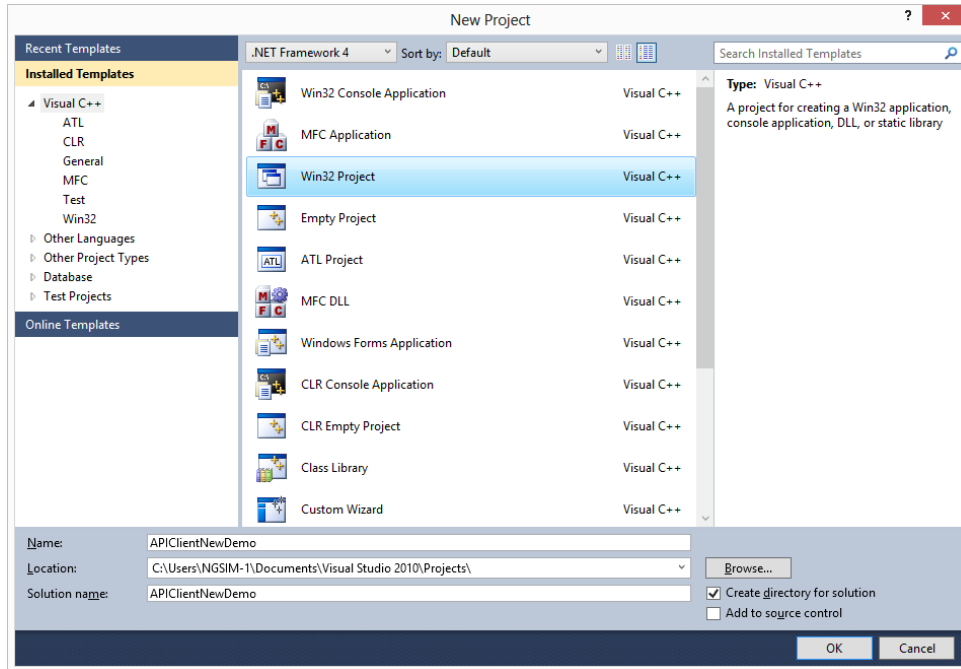
3.1 API Console Client

3.1.1 Console Client Development Guide

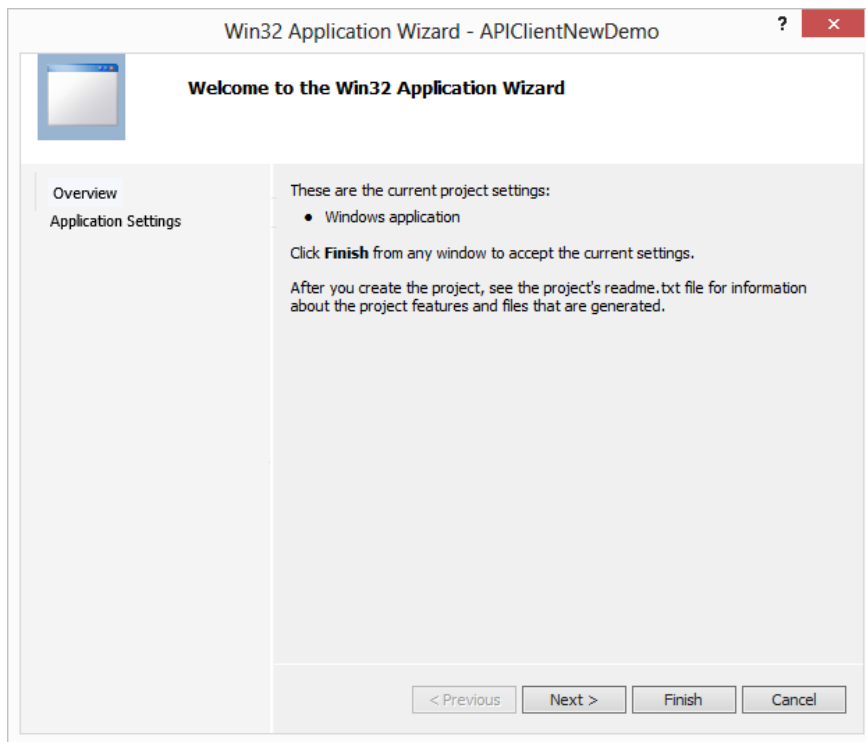
The user have to get a version of precompiled WCFServer.dll and etRunner64.exe at first.

Steps:

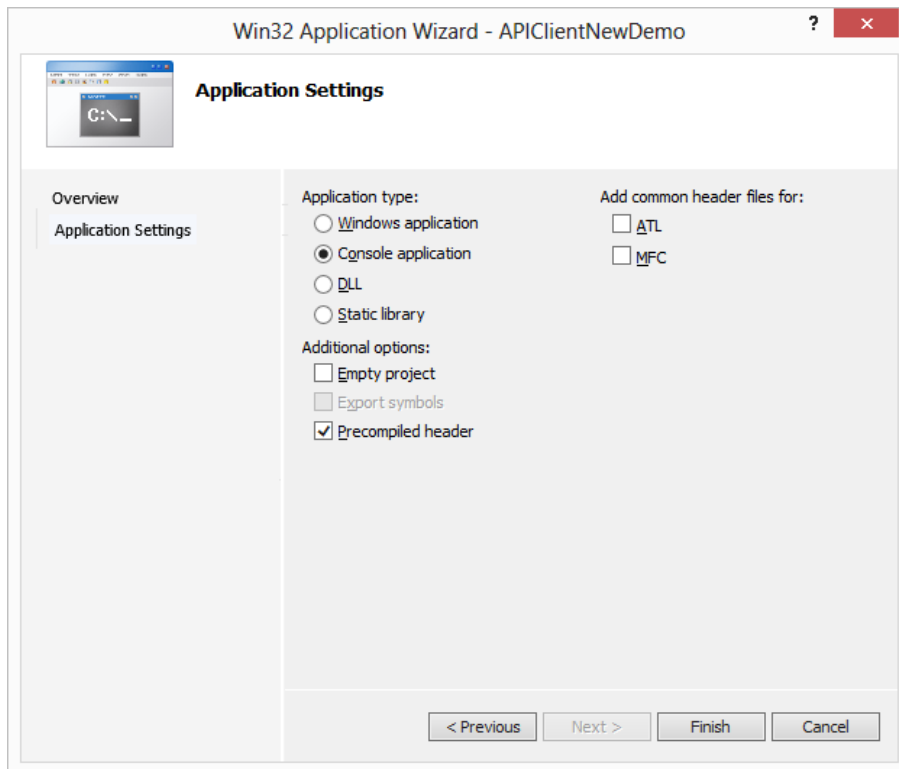
1. Create a new project named APIClientNewDemo. Under the menu [File – New – Project]



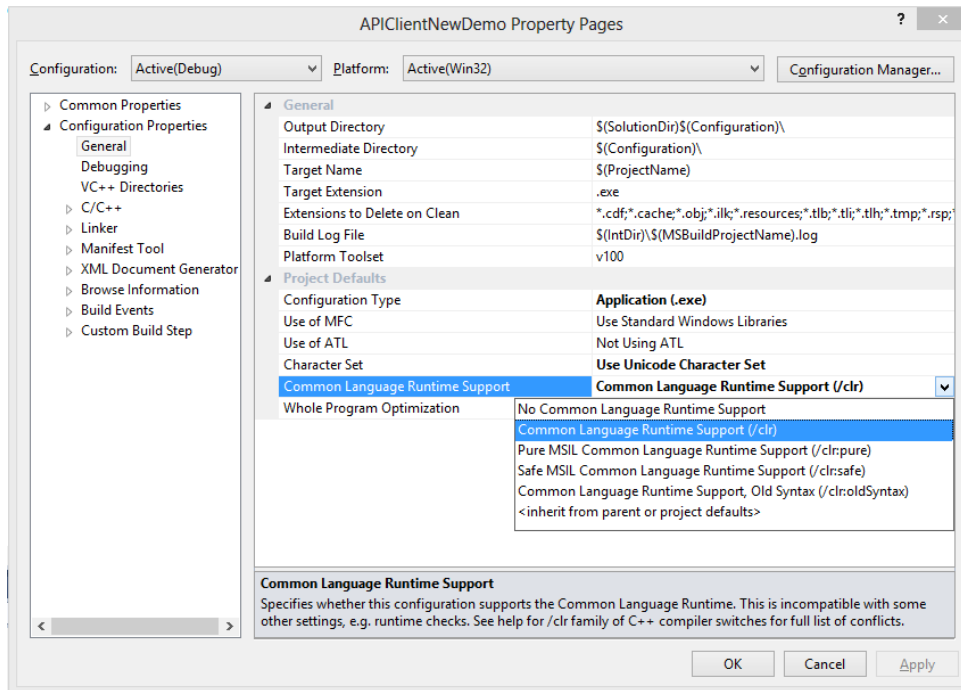
2. Click <Next>



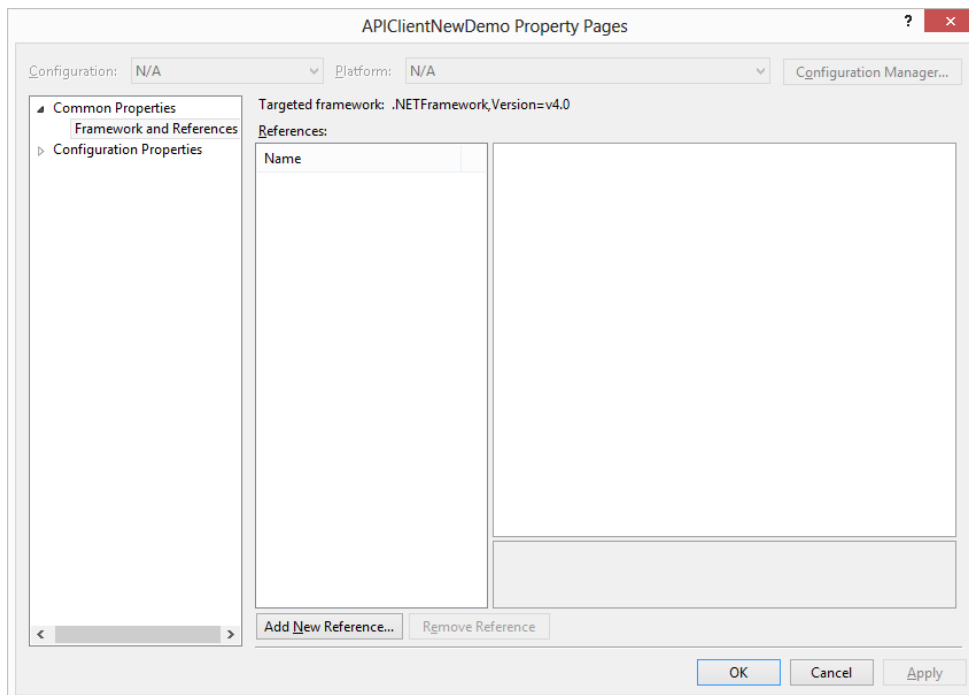
3. Select [Console application] as “Application type” and Click <Finish>



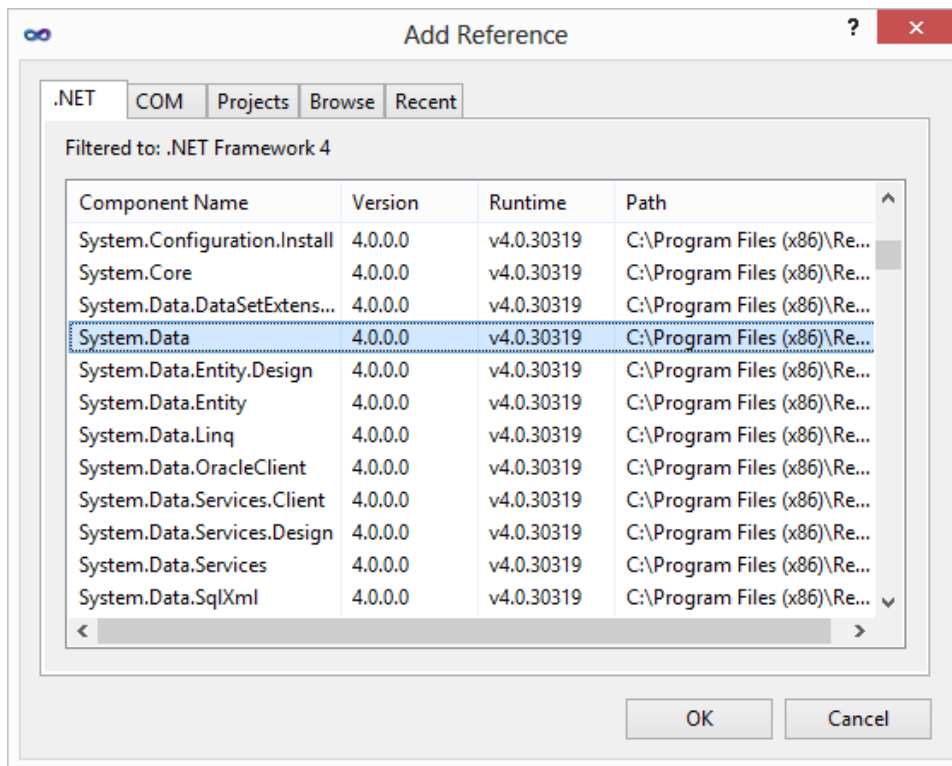
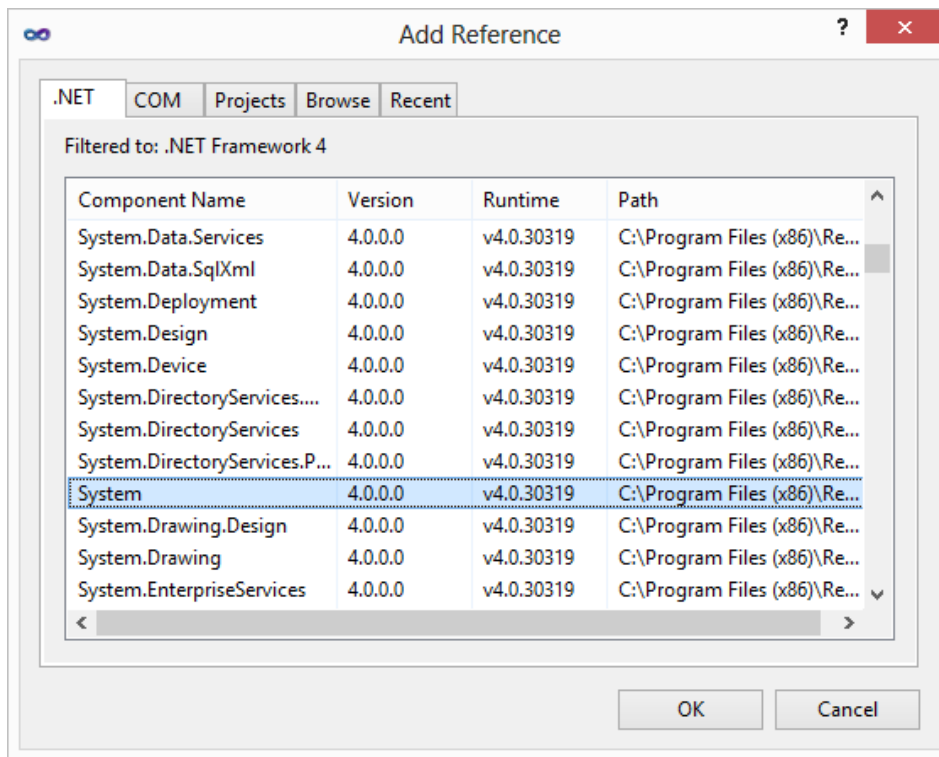
4. Under the menu [Project – Properties], select [Configuration Properties - General] and select [Common Language Runtime Support (/clr)] in [Common Language Runtime Support] tab.

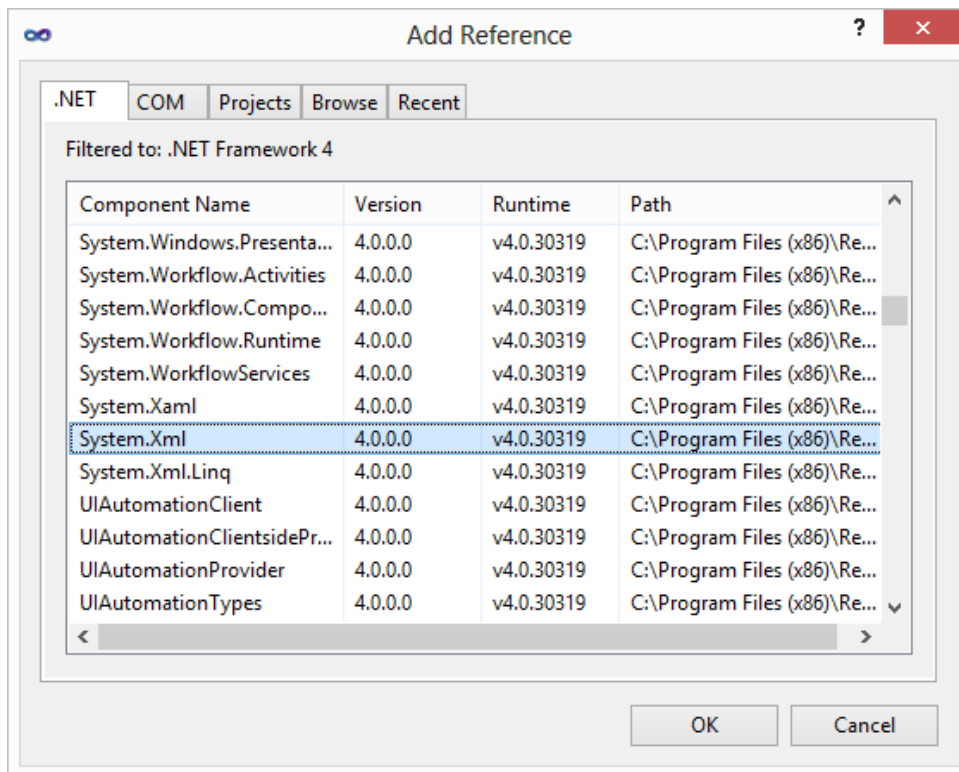


5. Under the menu [Project – Properties], select [Common Properties – Framework and References] and click <Add New Reference...>

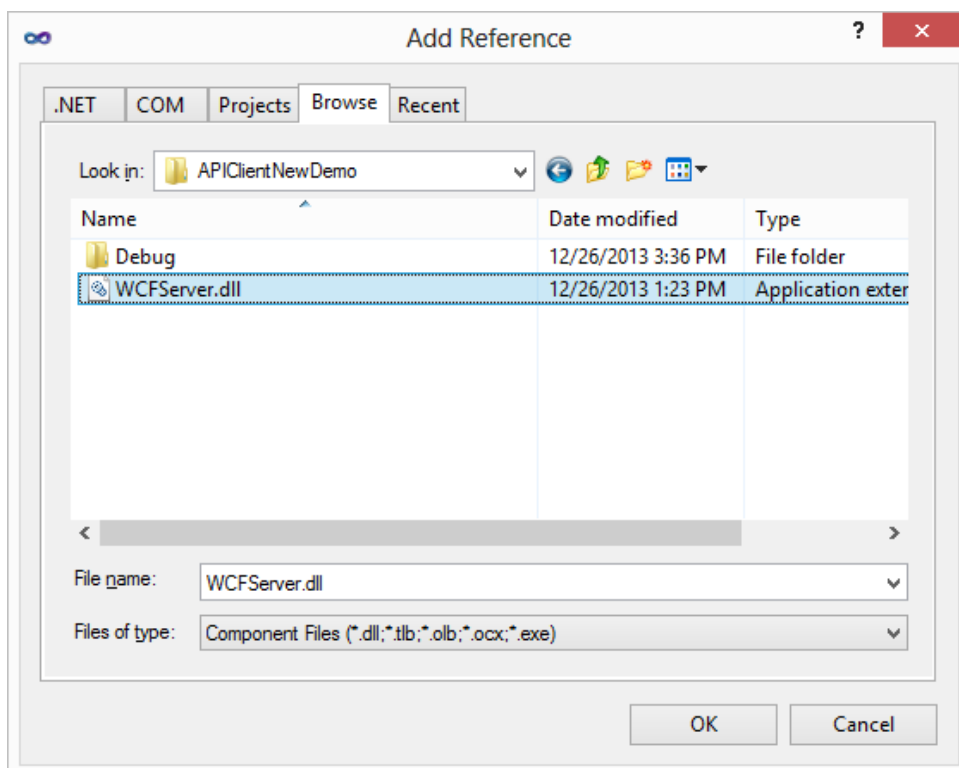


6. Select <.NET> tab and add three component: {System}, {System.Data} and {System.Xml}

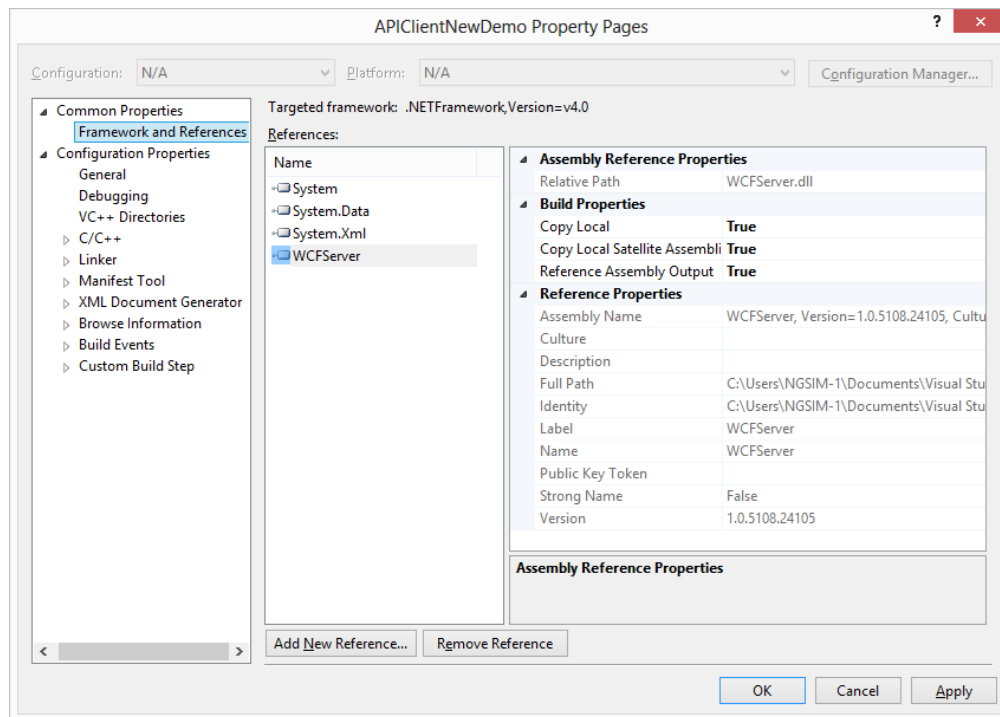




7. Select <Browse> Tab and browse and select WCFServer.dll



8. The [Common Properties – Framework and References] page will look like this



9. Click <Apply> and then click <OK>

10. Add following code at the beginning of API Client program source code:

```
#using <System.dll>
#using <System.ServiceModel.dll>
using namespace System;
using namespace System::ServiceModel;
using namespace WCFServer;
```

3.1.2 Console Client Usage Guide

The sample API Client program shows how to use default values to initialize ETFOMM components, run simulation through etRunner and modify values of ETFOMM components. Operation steps:

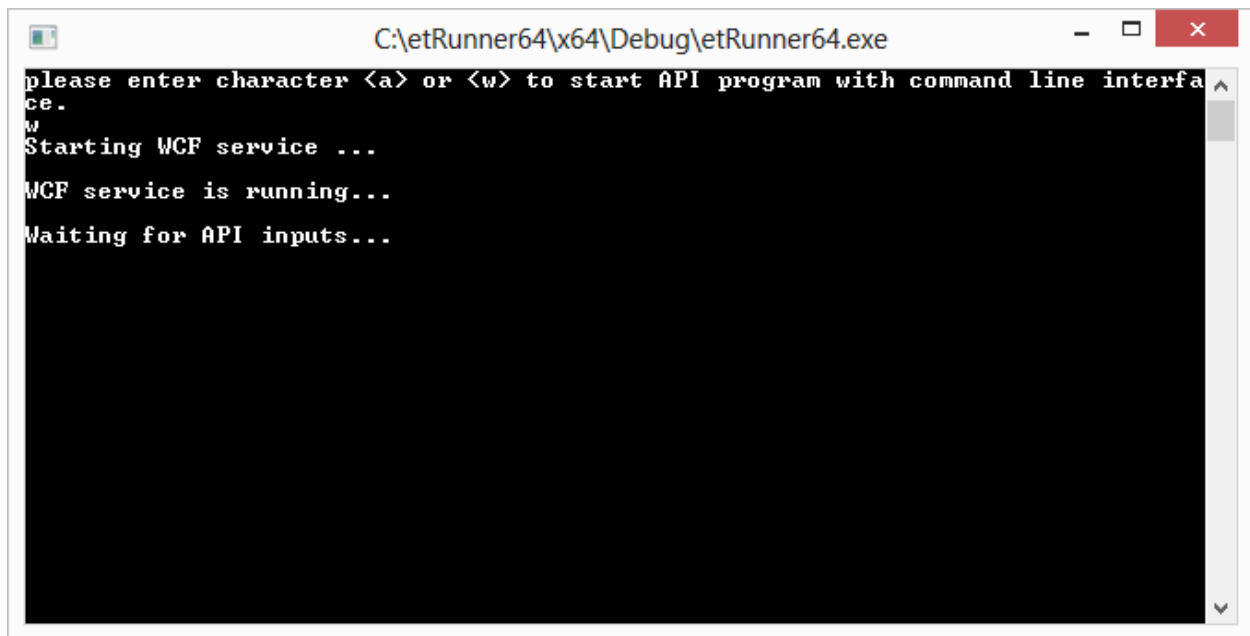
1. place the following files to C:\:

etfomm library: etfomm64.dll

random seeds file: rnsfile.dat

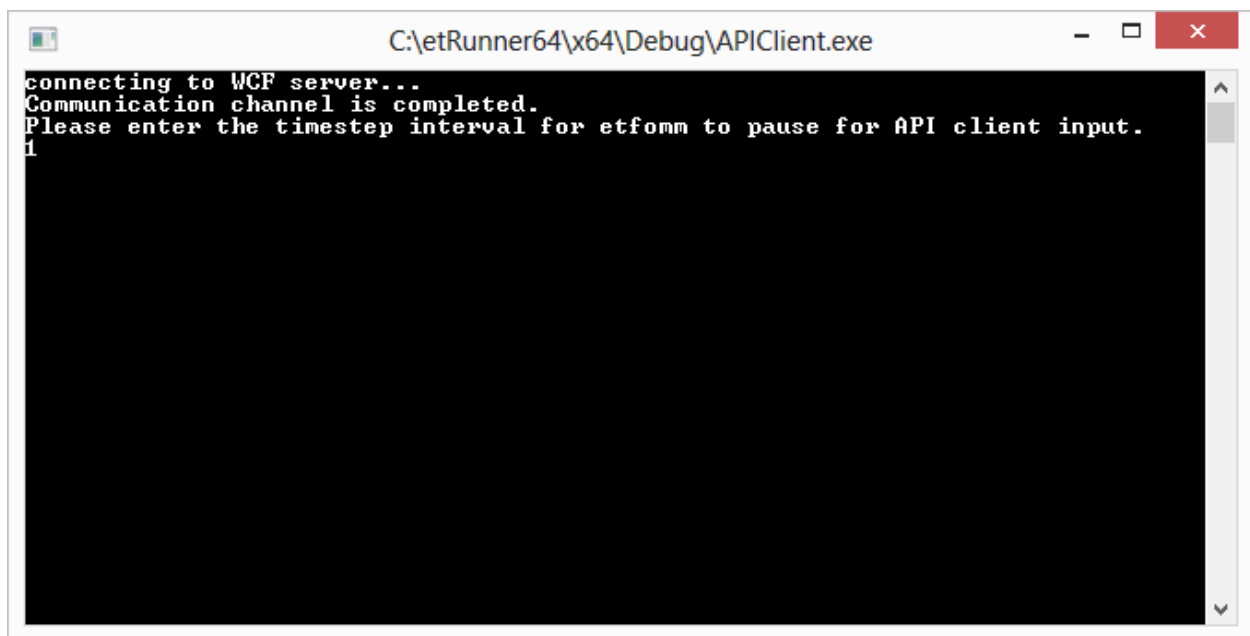
input files: API_Test5.trf and API_Test5.pat.

2. Run etRunner64.exe as Administrator, and enter “w” to start WCF service. etRunner begins to accept inputs from API Client:

A screenshot of a Windows command prompt window titled "C:\etRunner64\x64\Debug\etRunner64.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The text inside the window is as follows:

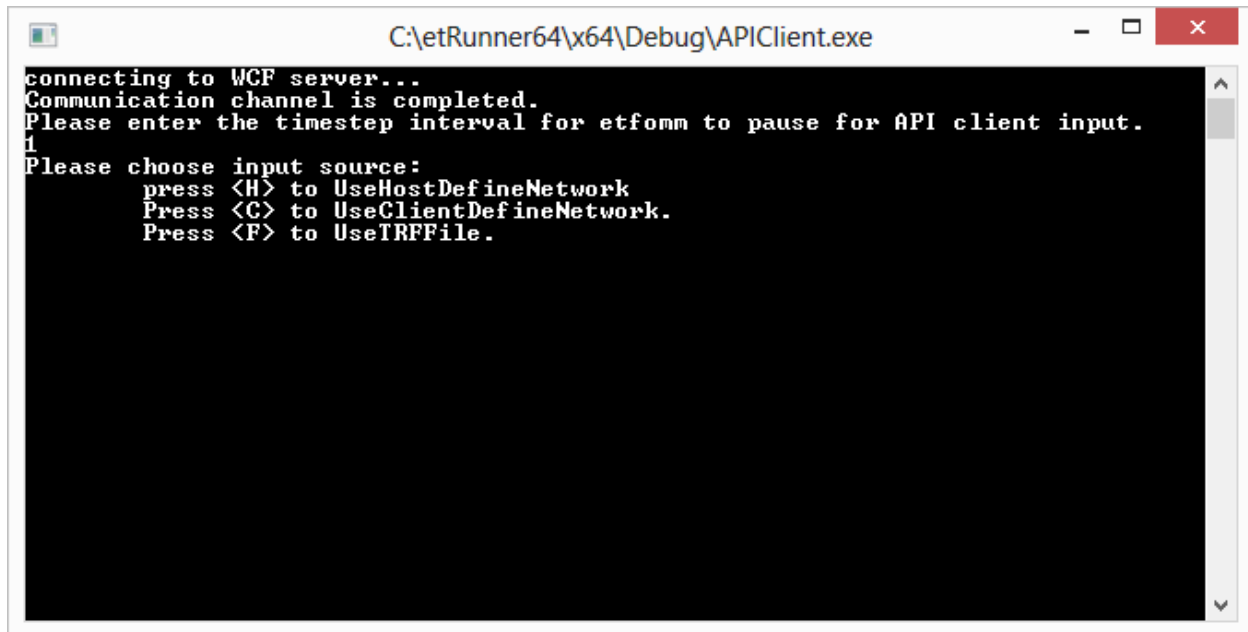
```
please enter character <a> or <w> to start API program with command line interface.  
w  
Starting WCF service ...  
WCF service is running...  
Waiting for API inputs...
```

3. Run APIClient.exe as Administrator, and enter “1” for timestep interval for etfomm to pause for API input.

A screenshot of a Windows command prompt window titled "C:\etRunner64\x64\Debug\APIClient.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The text inside the window is as follows:

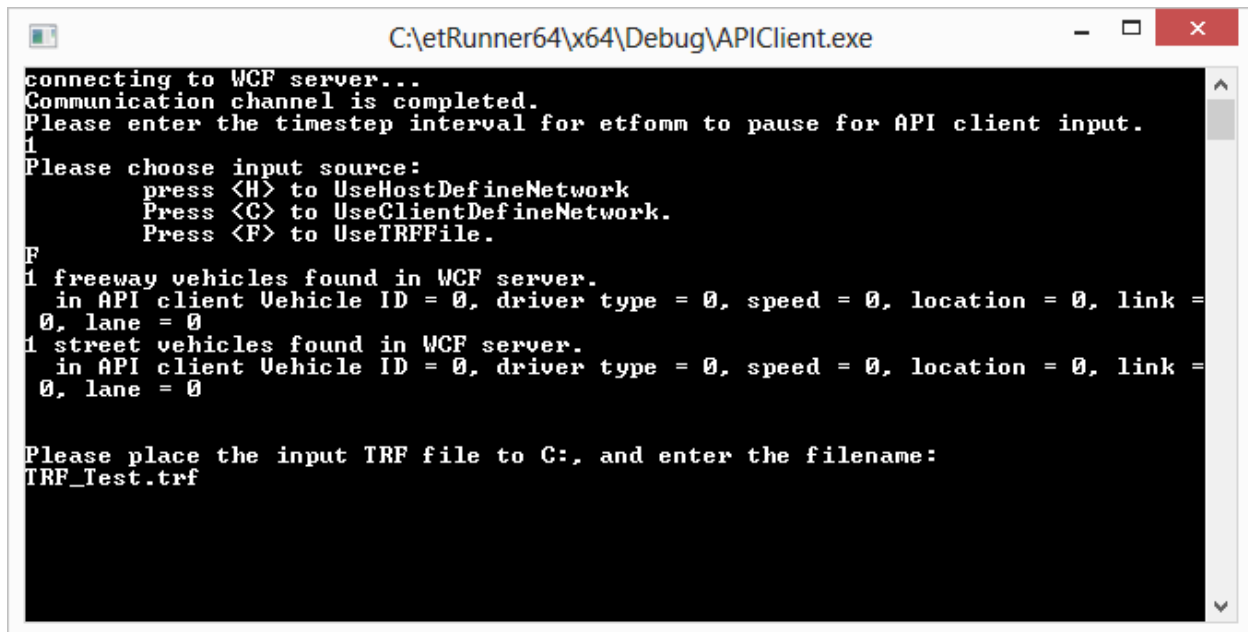
```
connecting to WCF server...  
Communication channel is completed.  
Please enter the timestep interval for etfomm to pause for API client input.  
1
```

4. In APIClient.exe, choose input source:



```
C:\etRunner64\x64\Debug\APIClient.exe
connecting to WCF server...
Communication channel is completed.
Please enter the timestep interval for etfomm to pause for API client input.
1
Please choose input source:
    press <H> to UseHostDefineNetwork
    Press <C> to UseClientDefineNetwork.
    Press <F> to UseTRFFFile.
```

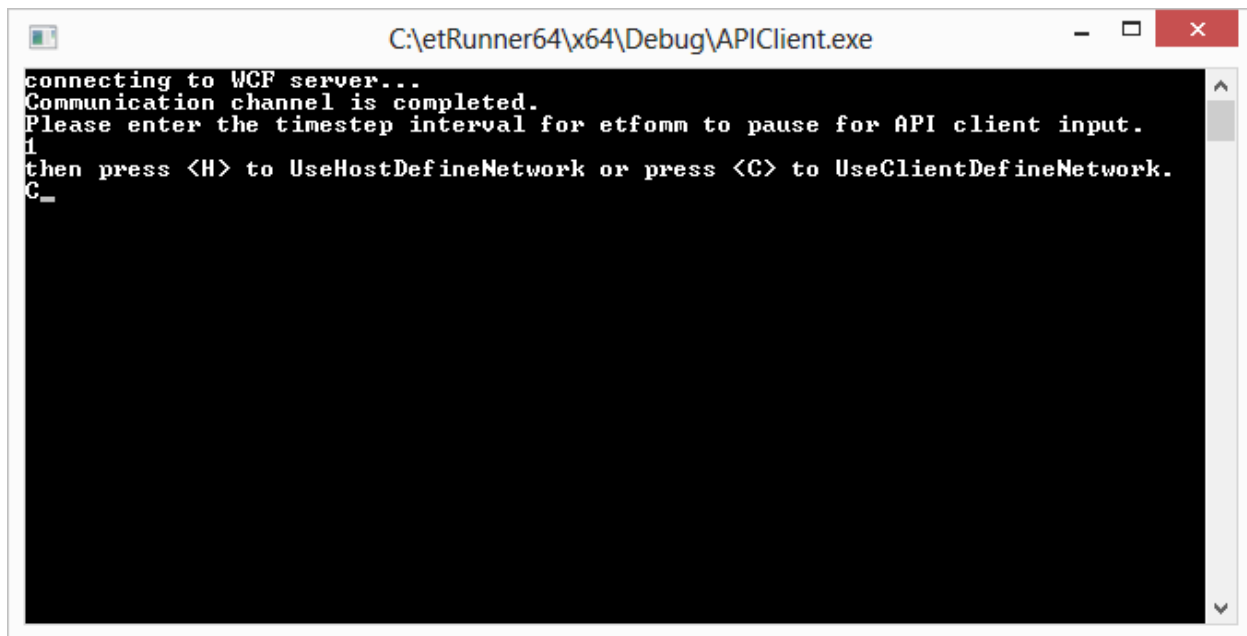
4.1 If enter “F” to use a TRF file as input, please place the TRF file to C:\ and enter the file name. For example, if the TRF file name is “TRF_Test.trf”:



```
C:\etRunner64\x64\Debug\APIClient.exe
connecting to WCF server...
Communication channel is completed.
Please enter the timestep interval for etfomm to pause for API client input.
1
Please choose input source:
    press <H> to UseHostDefineNetwork
    Press <C> to UseClientDefineNetwork.
    Press <F> to UseTRFFFile.
F
1 freeway vehicles found in WCF server.
  in API client Vehicle ID = 0, driver type = 0, speed = 0, location = 0, link =
  0, lane = 0
1 street vehicles found in WCF server.
  in API client Vehicle ID = 0, driver type = 0, speed = 0, location = 0, link =
  0, lane = 0

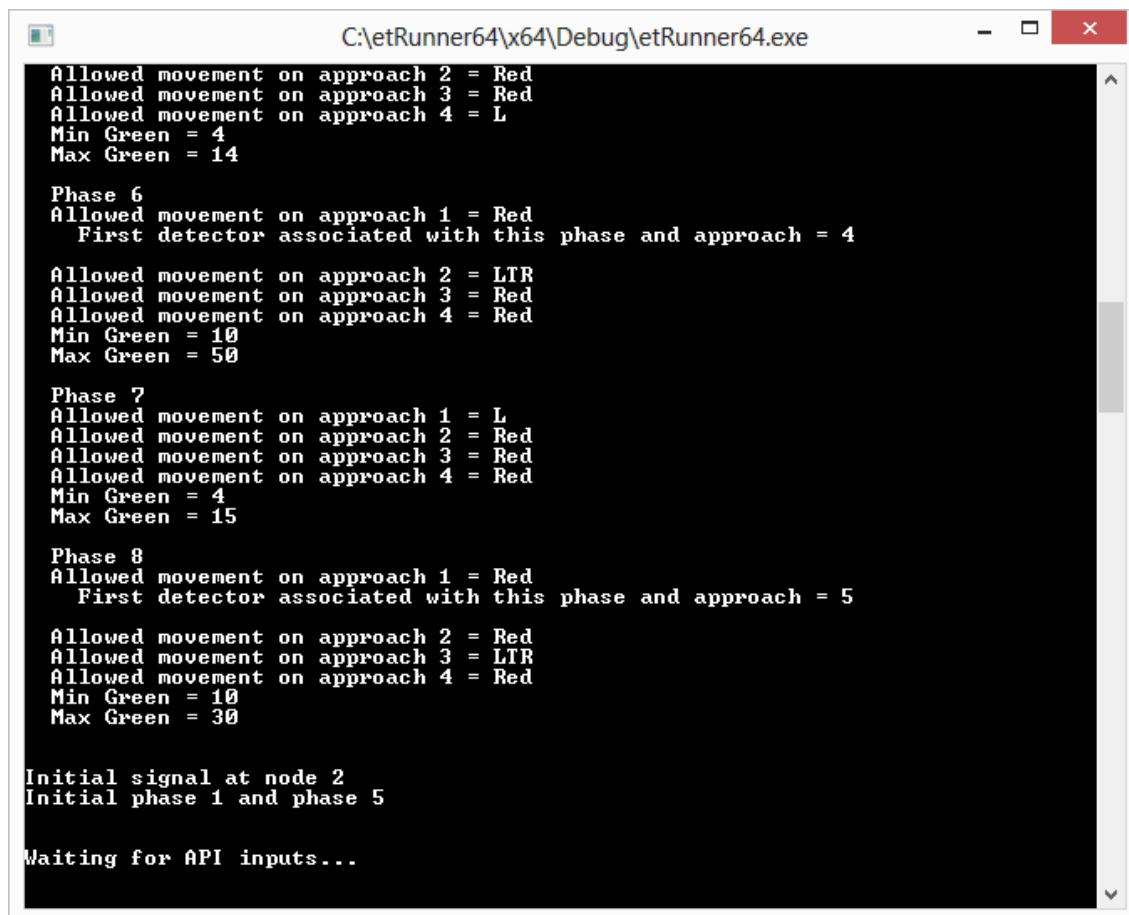
Please place the input TRF file to C:., and enter the filename:
TRF_Test.trf
```

4.2 Or enter “C” to use client default data or “H” to use host default data.

A screenshot of a Windows application window titled "C:\etRunner64\x64\Debug\APIClient.exe". The window has a black background with white text. The text reads: "connecting to WCF server...", "Communication channel is completed.", "Please enter the timestep interval for etfomm to pause for API client input.", "1", "then press <H> to UseHostDefineNetwork or press <C> to UseClientDefineNetwork.", and "C_".

```
C:\etRunner64\x64\Debug\APIClient.exe
connecting to WCF server...
Communication channel is completed.
Please enter the timestep interval for etfomm to pause for API client input.
1
then press <H> to UseHostDefineNetwork or press <C> to UseClientDefineNetwork.
C_
```

5. In etRunner.exe, simulation runs for 1 time step and pauses for API input:

A screenshot of a Windows application window titled "C:\etRunner64\x64\Debug\etRunner64.exe". The window has a black background with white text. The text displays traffic light phase information for four approaches (1, 2, 3, 4), including allowed movements, first detectors, and green time ranges (Min Green, Max Green). It also shows initial signals at node 2 and a prompt to wait for API inputs.

```
C:\etRunner64\x64\Debug\etRunner64.exe
Allowed movement on approach 2 = Red
Allowed movement on approach 3 = Red
Allowed movement on approach 4 = L
Min Green = 4
Max Green = 14

Phase 6
Allowed movement on approach 1 = Red
First detector associated with this phase and approach = 4

Allowed movement on approach 2 = LTR
Allowed movement on approach 3 = Red
Allowed movement on approach 4 = Red
Min Green = 10
Max Green = 50

Phase 7
Allowed movement on approach 1 = L
Allowed movement on approach 2 = Red
Allowed movement on approach 3 = Red
Allowed movement on approach 4 = Red
Min Green = 4
Max Green = 15

Phase 8
Allowed movement on approach 1 = Red
First detector associated with this phase and approach = 5

Allowed movement on approach 2 = Red
Allowed movement on approach 3 = LTR
Allowed movement on approach 4 = Red
Min Green = 10
Max Green = 30

Initial signal at node 2
Initial phase 1 and phase 5

Waiting for API inputs...
```

6. APIClient.exe displays a list of command for user to choose from:

```

C:\etRunner64\x64\Debug\APIClient.exe
connecting to WCF server...
Communication channel is completed.
Please enter the timestep interval for etfomm to pause for API client input.
1
then press <H> to UseHostDefineNetwork or press <C> to UseClientDefineNetwork.
C
1 freeway vehicles found in WCF server.
  in API client Vehicle ID = 0, driver type = 0, speed = 0, location = 0, link =
  0, lane = 0
1 street vehicles found in WCF server.
  in API client Vehicle ID = 0, driver type = 0, speed = 0, location = 0, link =
  0, lane = 0
=====
*** Basic Commands ***
=   enter <1> to continue;
=   enter <2> to finish current run w/o pause;
*** Program-modify Commands ***
=   enter <100> to test the change of all data;
*** Display Commands ***
=   enter <303> to display freeway links data;
=   enter <304> to display street links data and AC signals data;
=   enter <305> to display FTC signals data;
=   enter <308> to display Node XY Coordinate;
*** Manual-modify Commands ***
=   enter <401> to change freeway vehicle information;
=   enter <402> to add a new vehicle;
=   enter <403> to add path;
=   enter <404> to change timestep interval;
=   enter <405> to change signal data;
=====
Please enter a command:

```

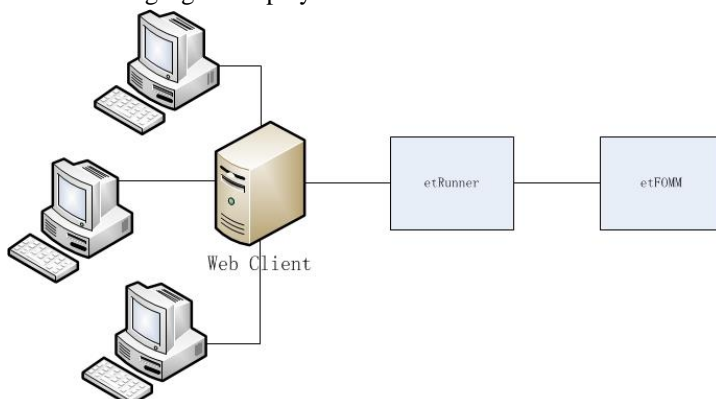
7. User can enter “1” to advance the simulation to next time step interval; enter “100” to modify all data structs; or enter “404” to change timestep interval.

Hint: command “100” to modify all data structs is only effective at the beginning of time period. User can change timestep interval to proceed to the desired time step.

3.2 API Web Client

3.2.1 API Web Client Architecture

The following figure displays the web client architecture.



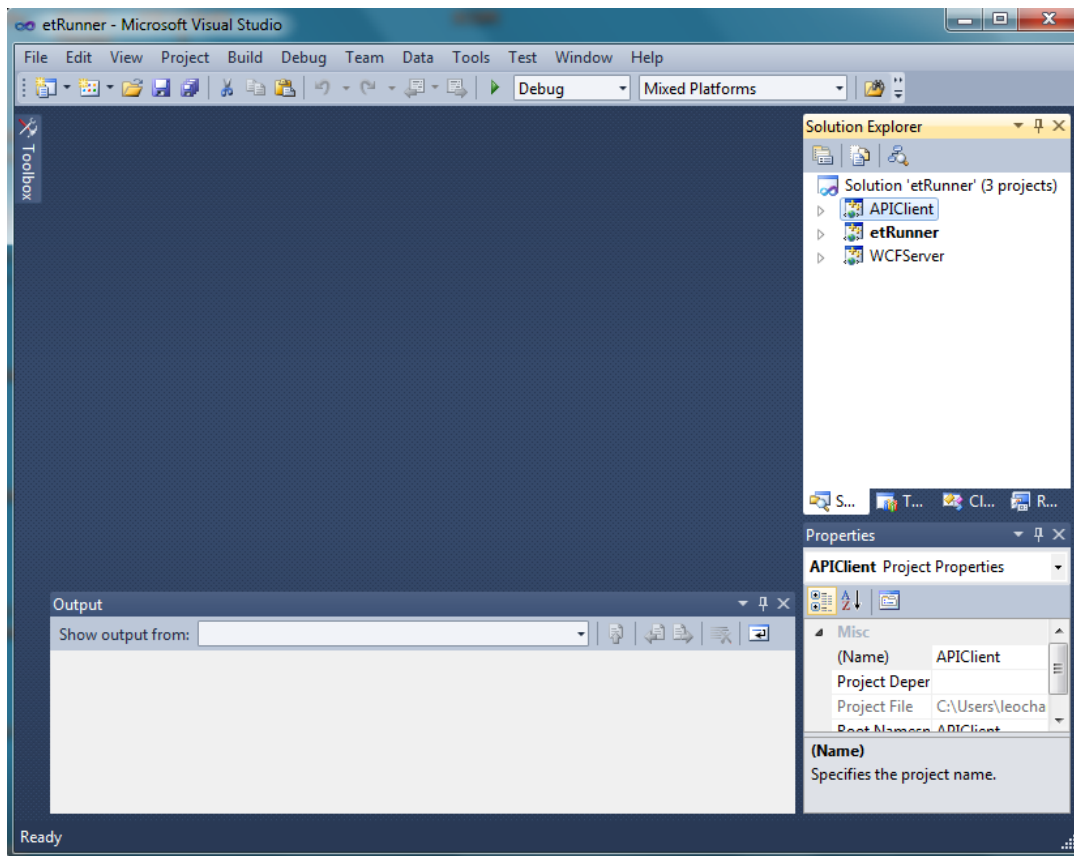
The website application is as similar as console application. The web client could be published to HTTP server. The end users could access web client to control etFOMM simulation 24/7 and “anywhere”.

3.2.2 Web Client Development Guide

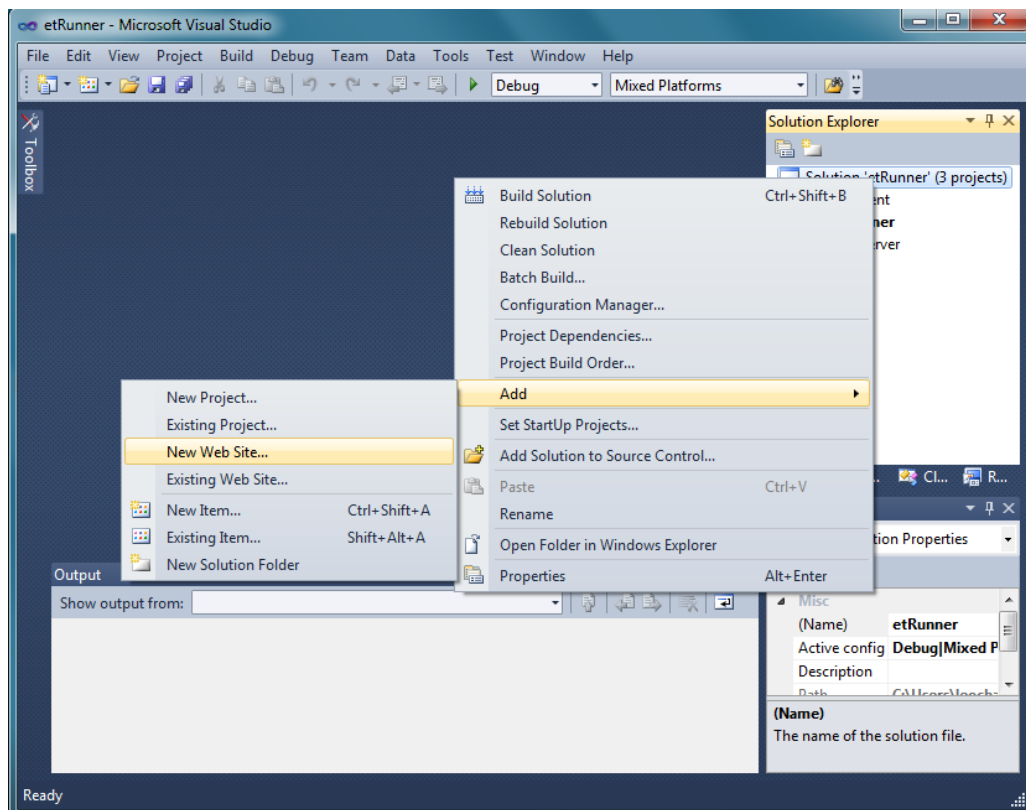
The API functions could be implemented to web client. The web client always is as same as console application. It could be developed by any programming language which could integrate with C++ CLR and .Net Framework, for example C#/C++, and deployed to a HTTP web server such as IIS, Apache, and Google Web Server. This is a sample process of building a web client in etFOMM.

3.2.2.1 Create/Import Web Client Website

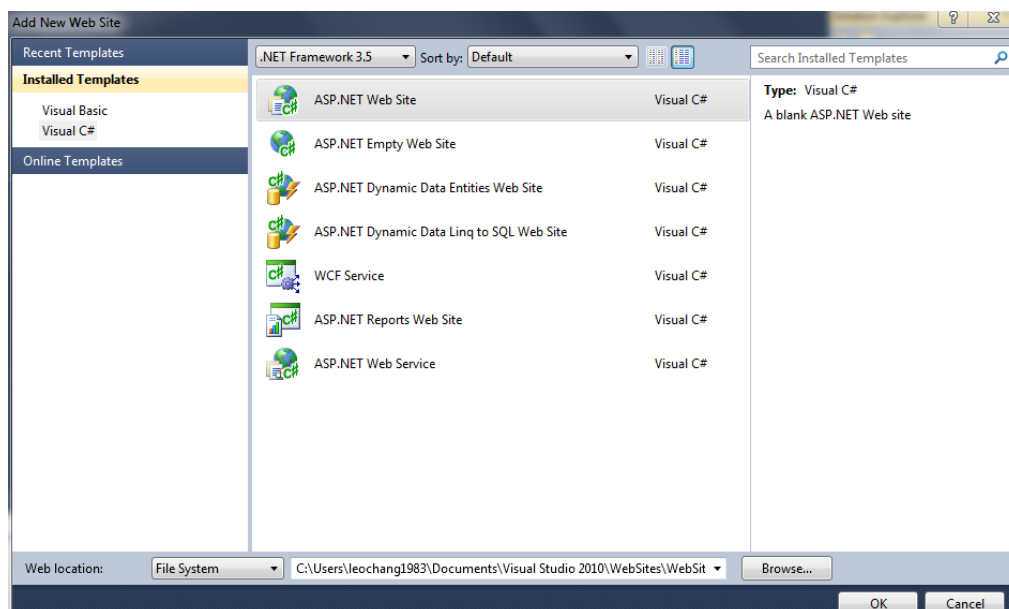
To build a web client and integrate with etFOMM, users must have etFOMM API which includes WCFServer interface, etRunner control program, and etFOMM DLL. All these three parts are in a etRunner project. Users could open the project by Visual Studio 2010 or higher version or any other C#/C++ IDE with .Net Framework. The figure displays the etRunner project in Visual Studio 2010:



The users could create a new etFOMM web client or import an existing web site to etRunner project. In the Visual Studio, users could right click on Solution and choose Add command, then select Add New Website or Existing Website. The figure displays this step in Visual Studio:

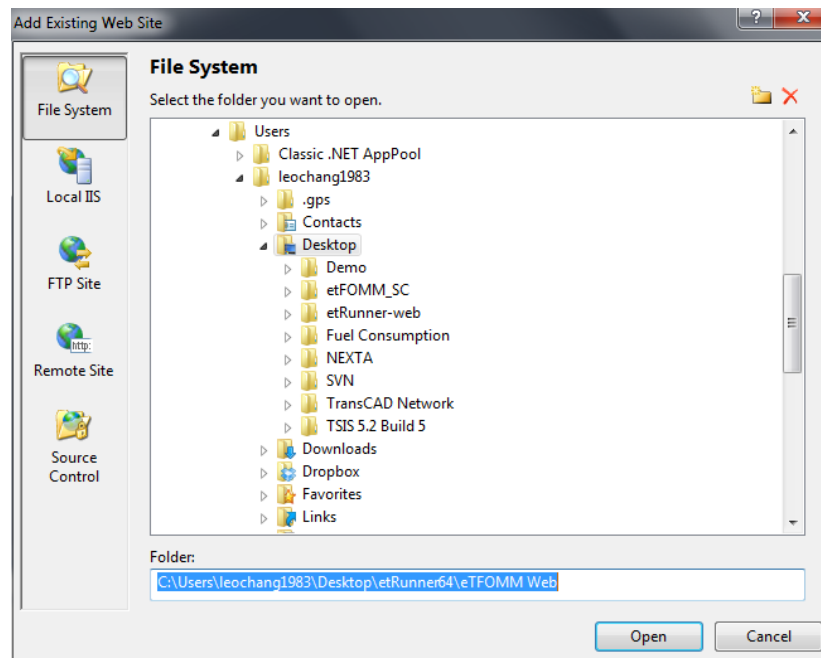


In this sample, one Visual C# website is created. In the window of add new website, users should choose Visual C# and ASP.NET Website. The figure displays the window of Add New Website in Visual Studio:



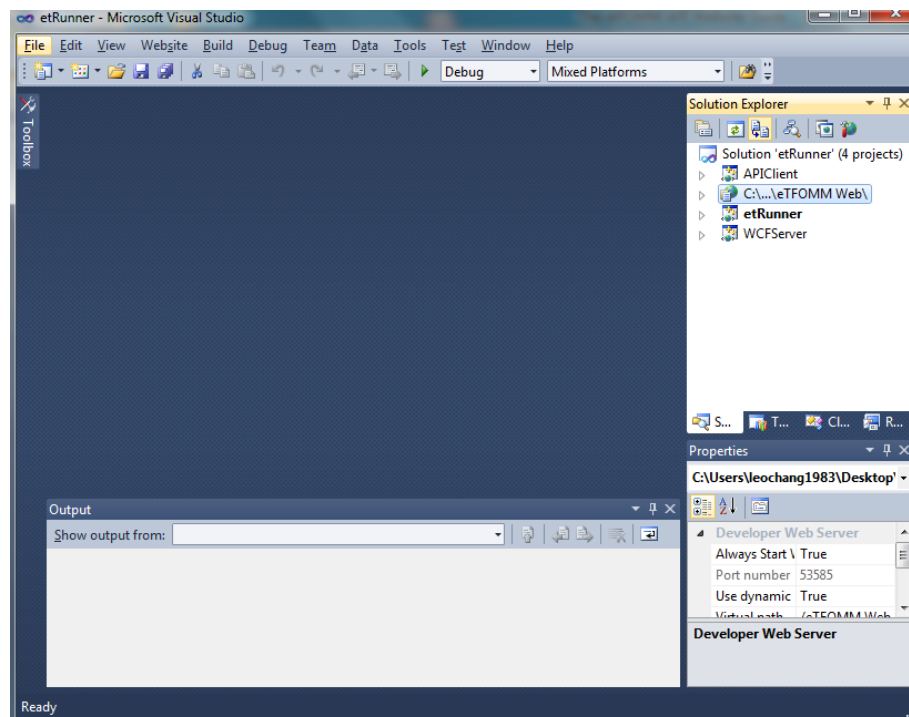
Because there are several version of etRunner, users may choose different .NET Framework version. The .NET Framework version selection is on the top of this window.

To import existing website, Visual Studio would display a window as below:



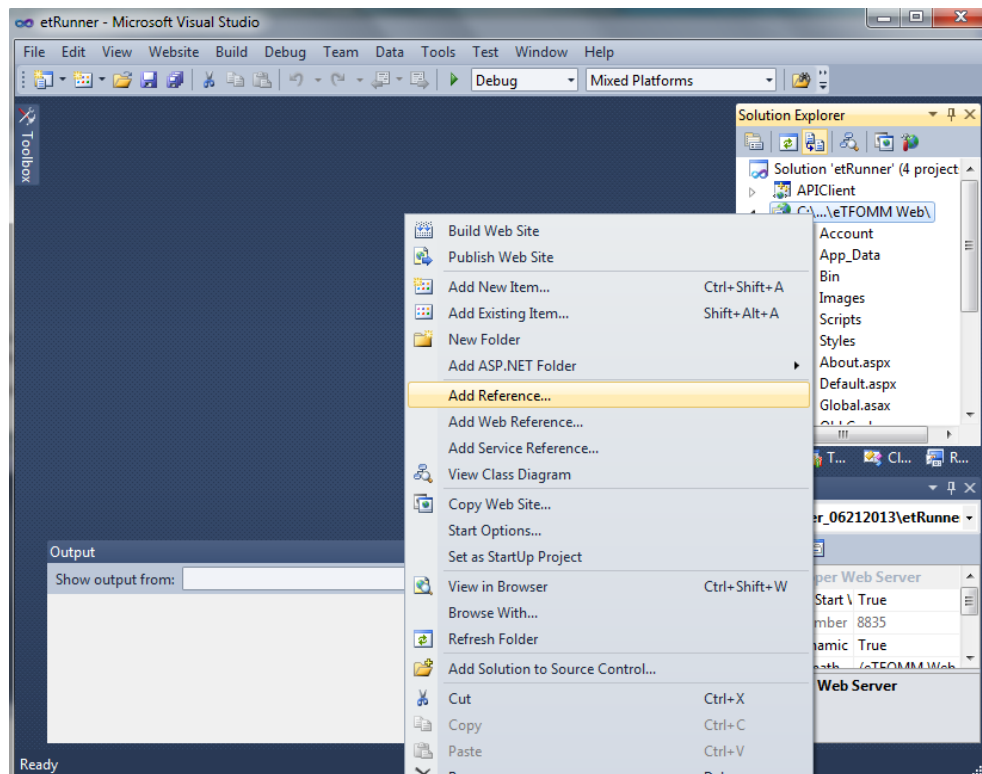
The users could import one existing website from different sources such as local computer, IIS, FTP site, and remote site. The functions are provided by Microsoft Visual Studio.

The figure displays the solution explorer after add a new website or import a existing website to etRunner.

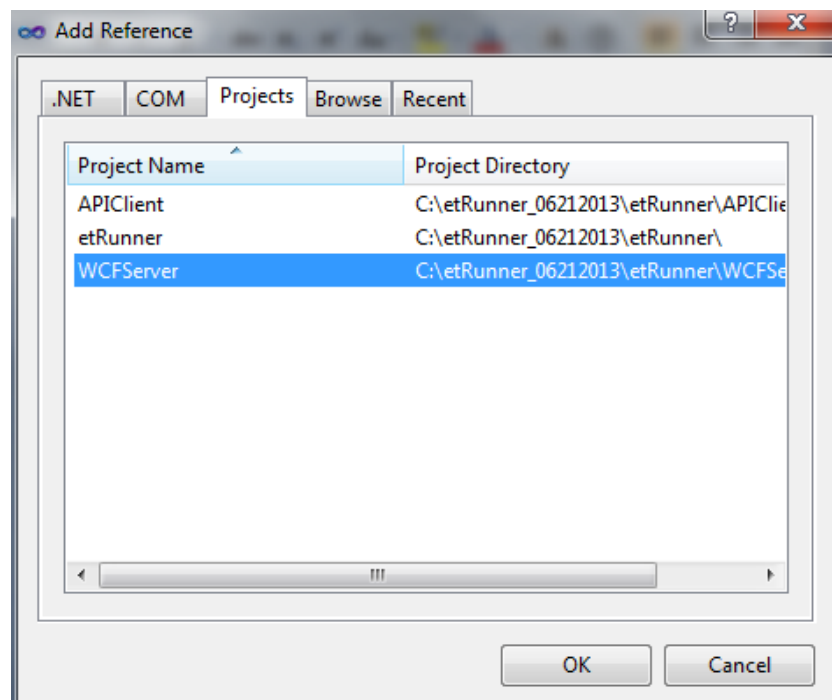


3.2.2.2 Load WCFServer DLL

After to creating a new web client website in etRunner project, users should integrate etFOMM API with website. In Visual Studio, users should add WCFServer.DLL as a reference to website.



Then users could select WCFServer from this solution or select WCFServer.DLL from a path.



3.2.2.3 Configuration of Web Client website

In the web client website, users should add below code to configure the connection with etRunner program and etFOMM.

One configuration code is:

```
using WCFServer;
```

This is define etFOMM API code. It should be added to the most pages which include connected functions with etFOMM API.

Another configuration codes are:

```
static EndpointAddress address;  
static WSHttpBinding binding;  
static ChannelFactory<IService1> factory;  
IService1 proxy;
```

These codes define WCF configuration parameters as Global Variables in web pages.

The below figure shows a sample of adding configuration codes:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.UI;  
using System.Web.UI.WebControls;  
using System.Data;  
using System.ServiceModel;  
using WCFServer;  
  
public partial class _Default : System.Web.UI.Page  
{  
    //Connect WCF Service  
  
    static EndpointAddress address;  
    static WSHttpBinding binding;  
    static ChannelFactory<IService1> factory;  
    IService1 proxy;
```

In addition, users should add below codes to initialization or page load function in the web page.

```
address = new EndpointAddress("http://localhost:6000/service");  
binding = new WSHttpBinding();  
factory = new ChannelFactory<IService1>(binding, address);  
proxy = factory.CreateChannel();
```

The EndpointAddress and binding type must be as same as etRunner codes. In this sample, EndpointAddress is `http://localhost:6000/service` and binding type is `WSHttpBinding`. Other part of codes would not be modified.

3.2.2.4 Using etFOMM API Function

After configuration, web client has already connected with etFOMM simulation. The users could directly call etFOMM API functions in the web page. The proxy is a communication channel. The users could use below codes to call API functions:

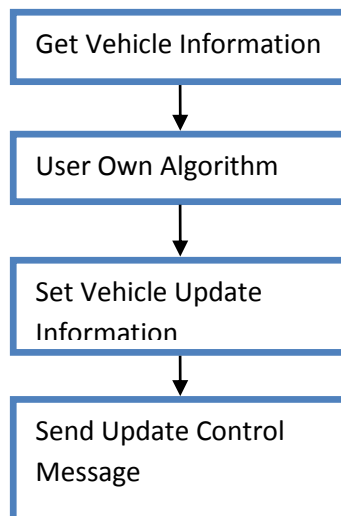
```
vehs = proxy.getServerVehicleData();  
proxy.setServerVehicleData(vehs);  
proxy.setClientState(4);
```

The **getServerVehicleData()** is API function to get current vehicle information from the etFOMM. Users could get one array of vehicles in WCF Server Host. Basing on vehicle information, users could design own algorithm in their website.

The **setServerVehicleData(vehs)** is API function to set update vehicle information to the etFOMM. Users could set one array of vehicles in WCF Server Host after the modification of web page activities.

The **setClientState(4)** is API function to send control message to simulation. Users could control the simulation by this function. In this case, client state 4 is updating vehicle information. To use this code, the etFOMM would know that client updated some information about vehicles.

The below figure displays the flow chart of web page coding about etFOMM API functions:

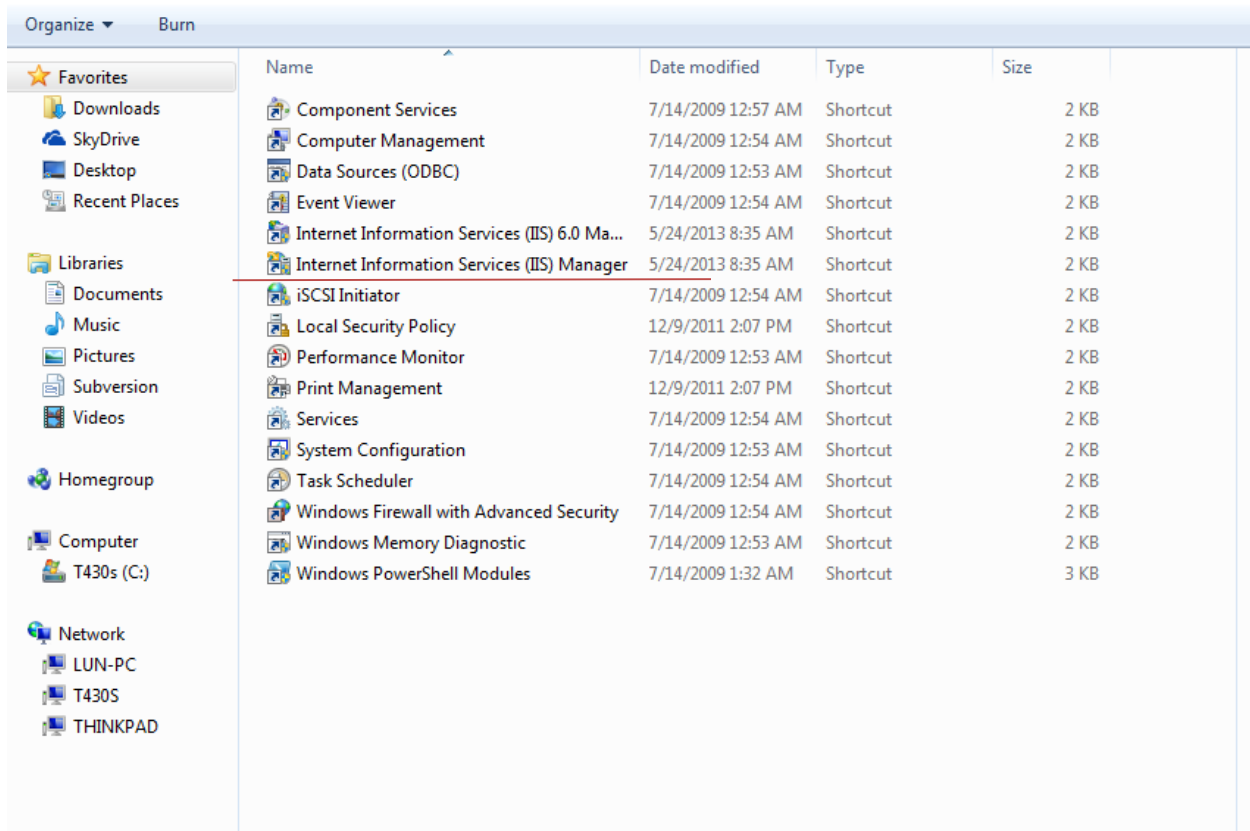


This is a simple sample of using etFOMM API to update vehicle information. WCF Service Library includes several functions to allow user get/set data easily and fast.

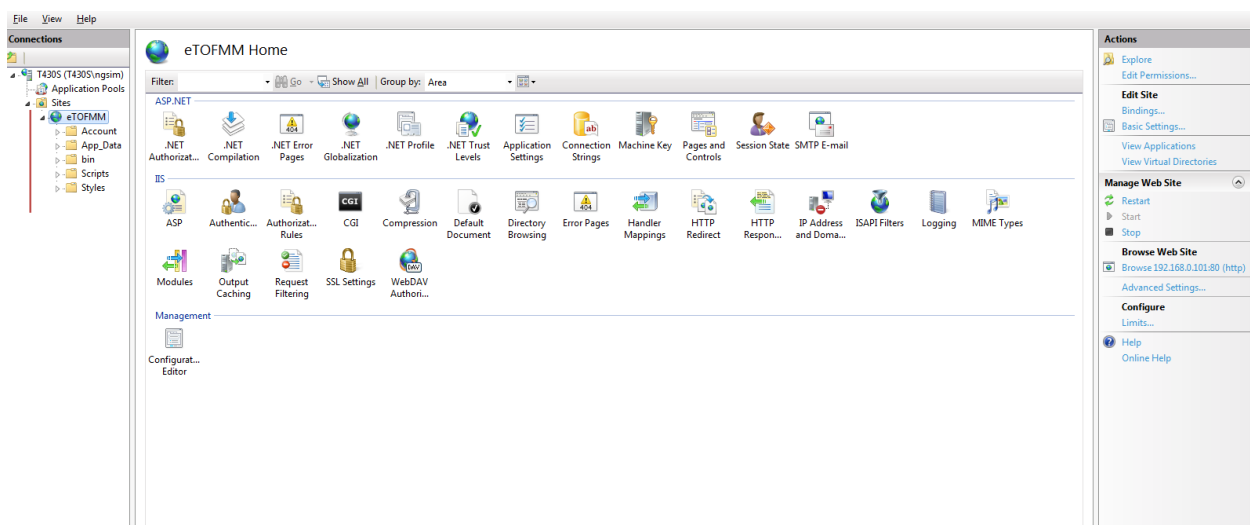
3.2.2.5 Deploy web client to Local IIS

The web client could be deployed to a HTTP web server. The below is a sample of deploying web client to Local IIS.

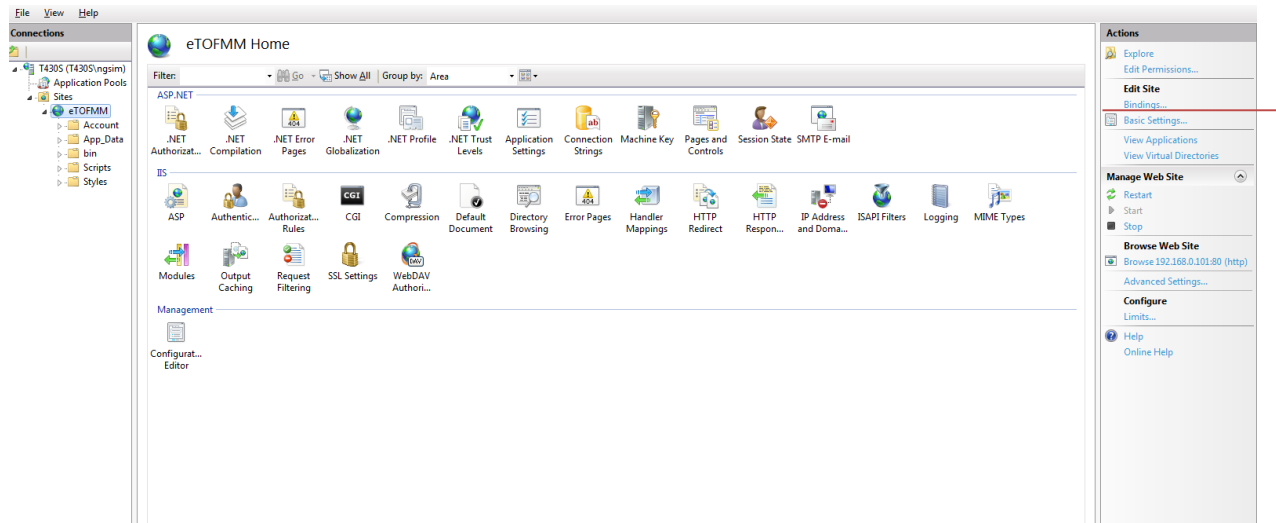
Open Administration Tool -> Internet Information Service (IIS) Manager



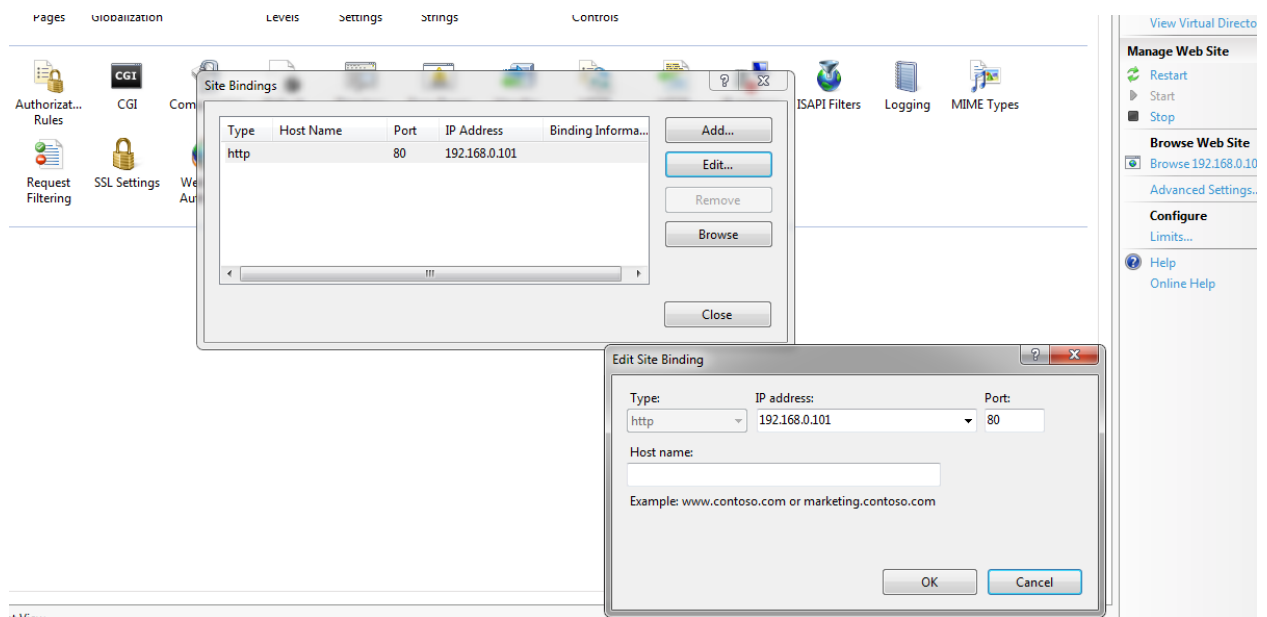
Find website from left connections panel. Left-click to go into website Home



Left click Bindings in right panel



Then configure IP address/port/host name of website

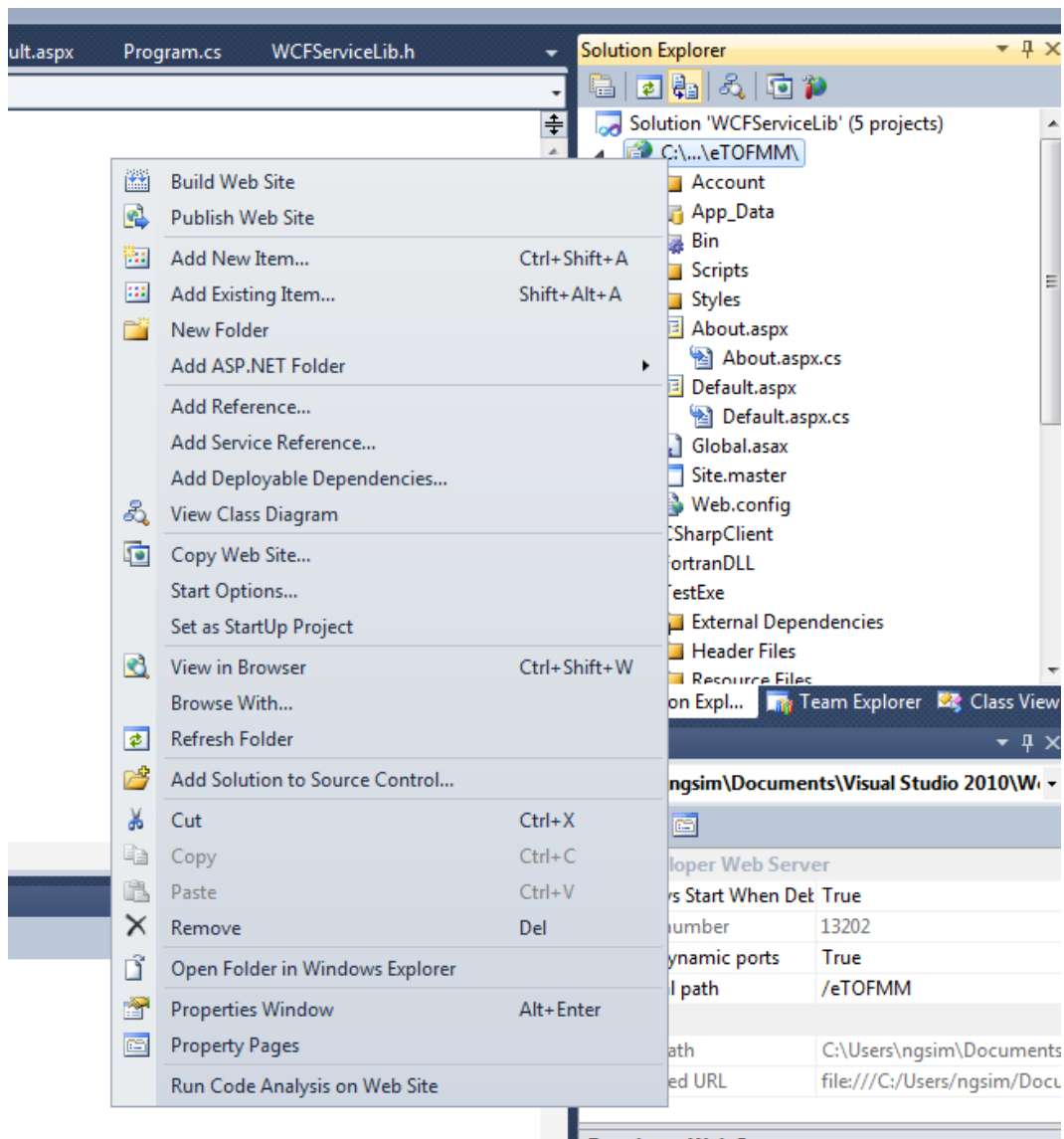


Then users could use Visual Studio to publish website solution to local IIS.

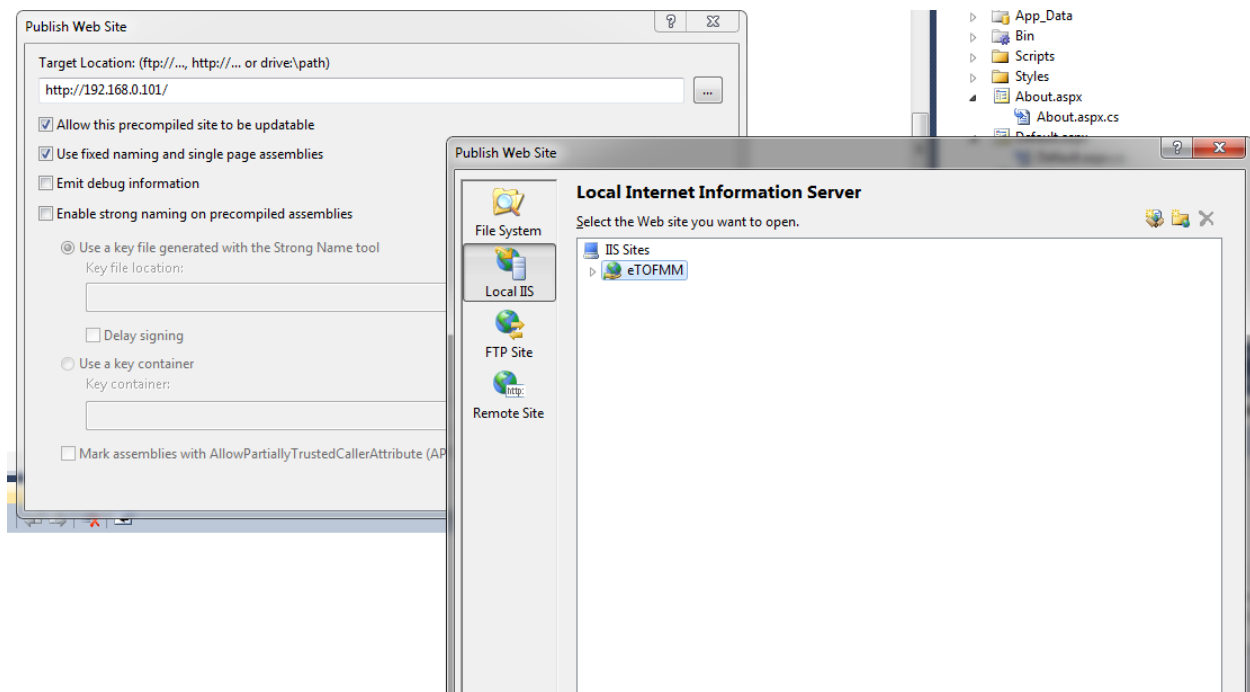
Open Visual Studio as Administrator

Open website project

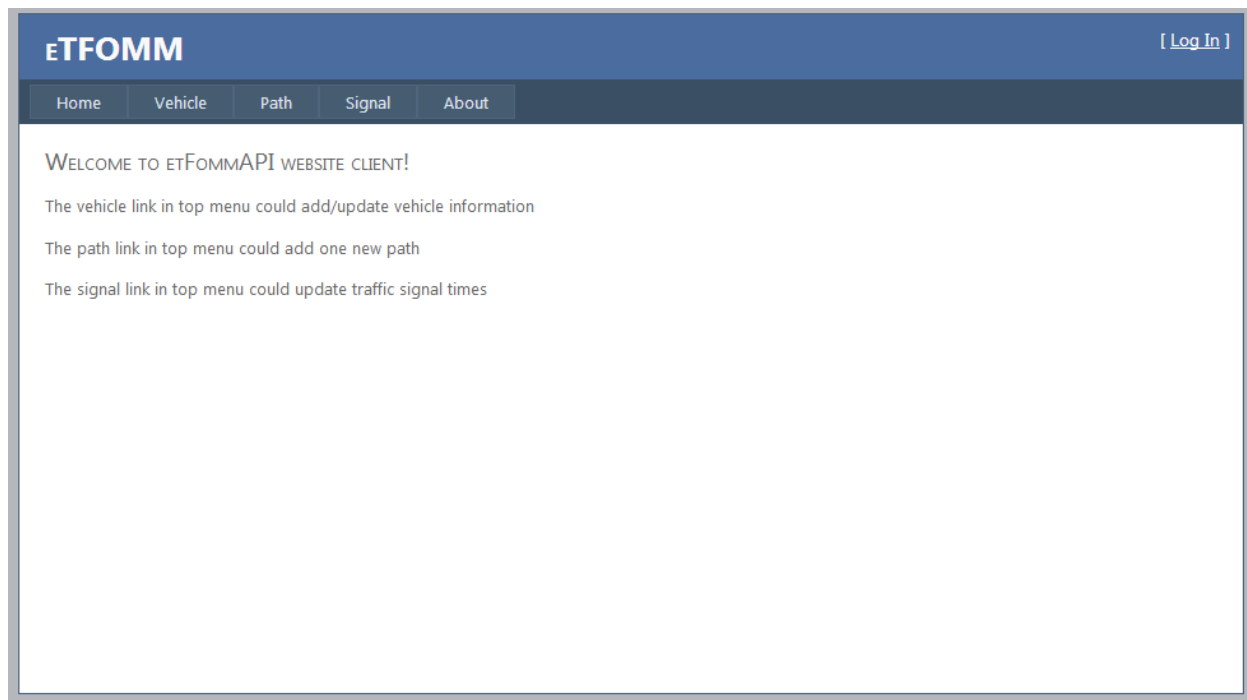
Right Click on website project



Choose Target Location and Find Local Internet Information Server



After Visual Studio compiler pass, the website could be accessed by Internet Browser.



The web client could provide all functions to users. This sample just includes a few functions in the etFOMM API. The etFOMM users could use their own programming experience to design useful and varied web client to meet their requirements.

3.2.3 Web Client Usage Guide

The below part introduces one web client sample. The figure 4 displays a web client main page.

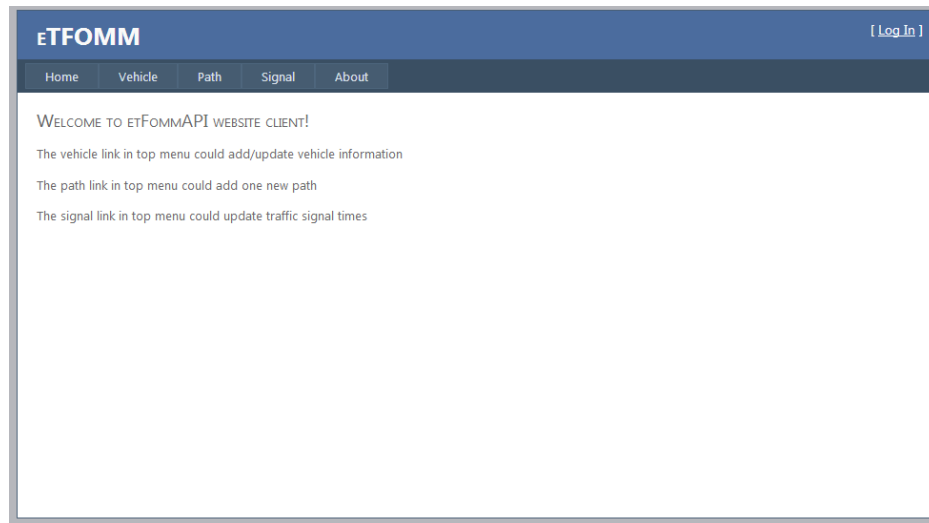


Figure 4 Index page of web client

The web client website was deployed to one IIS server. After that, the users could access the website by any Internet browser.

This version of web client has three main kinds of API functions: Vehicle, Path, and Signal. The users could choose different function pages by menu bar at the top of page.

Vehicle Functions

The vehicle page has API functions about simulation vehicle information. The figure 5 shows vehicle page.

		Index	Vehicle ID	DriverType	Speed	Location	Lane
Edit	Delete	2	0	0	0	0	0
Edit	Delete	6	0	0	0	0	0
Edit	Delete	7	0	0	0	0	0
Edit	Delete	8	0	0	0	0	0
Update Cancel		3	82	6	89.70856	1888.92	2
Edit	Delete	9	85	2	80.17306	1500.062	2
Edit	Delete	4	86	2	80.17306	1177.803	1
Edit	Delete	5	87	1	77.3124	912.2652	2
Edit	Delete	1	88	5	87.80145	680.5878	1

DriverType: Update ALL

Figure 7 Update vehicle information function

For updating function, users could update vehicle information individually or update one attribute of all vehicles. The figure 8 shows this function in the web client.

Edit	Delete	4	82	5	70	976.4652	1	1	5
Edit	Delete	5	85	2	70	0	2	1	5
Edit	Delete	6	86	2	70	0	1	1	5
Edit	Delete	7	87	1	70	0	2	1	5
Edit	Delete	8	88	5	70	1579.854	1	1	5
Edit	Delete	9	95	2	70	0.4474829	2	1	5

DriverType: 70 Update ALL
 ✓Speed
 Location
 Lane
 Link

Path ID: Driver:

Figure 8 update all vehicle information function

After updating vehicle information, users could use Submit Vehicle Update button to upload modification to simulation.

In addition, another function of vehicle is to add new vehicle which could be an emergency vehicle on roadway. The web client provides this function to users. The figure 9 shows this function in the web client.

Node ID: 1 Path ID: 0 Driver: 6
 Fleet: 5 Vehicle Type: 6 Allowed Speed: 30 Preemption Range: 1000
 Add New Vehicle

Figure 9 Add new vehicle function

Path Functions

Second function page of web client is path function page. In this page, users could add one new path in the simulation. The figure 10 displays path function page in this sample.

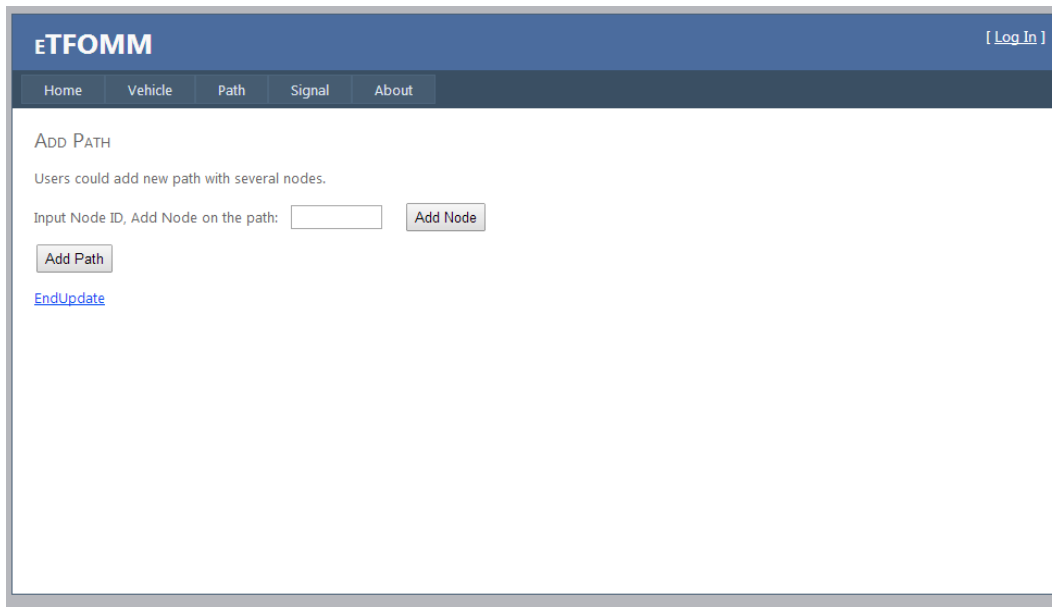


Figure 10 Path function page of web client website

In this page, there are two steps for add new path function. The first one is add node. The users should input node id into textbox then click Add Node button to add a node on the path. After all nodes are added, users should click Add Path button to create a new path into simulation. The figure 11 displays this function in the web client.

ADD PATH

Users could add new path with several nodes.

ID	NodeID
1	1
2	5
3	9
4	3
5	6

Input Node ID, Add Node on the path:

Figure 11 Add new path function

Signal Functions

The signal page of web client provides functions about get signal information or modify signal interval duration. The figure 12 displays signal page of this web client.

	Index	NodeID		
Update Cancel	1	2	15	20

Figure 14 Update signal function

After all updating of signals, the users could use Submit Signal Update button to modify signals in the simulation.

Update Frequency Functions

The web client provides Update Frequency function and Continue without Update function to users.

Update Frequency function is to set update frequency which gap duration of simulation between two update activities.

Continue without Update function is to run one update frequency simulation without any update.

These two functions are basic simulation control functions and be provided in vehicle page and signal page. The figure 15 shows Update Frequency function and Continue without Update function.

The signal view just displays Green Times of each intersection(node).

Users could update Green Times of different directions

Users could set frequency of pause etformm and update: 60 seconds

Figure 15 Update Frequency and Continue without Update functions

4. TEST RESULTS

4.1 Objectives of Test

The test on the ETFOMM component interface is to validate the following three implementations:

1. etRunner correctly calls etfomm functions via etFommInterface.
2. etfomm and API Client successfully exchange data via WCF Server.
3. Data modifications in API Client correctly reflect in etfomm.

4.2 Testing Plans

4.2.1 Test Data

The test on the ETFOMM component interface was conducted with the default value defined in APIClient::Client_Default_Network.cpp.

4.2.2 Rationale for Decision

1. etRunner correctly calls etfomm functions via etFommInterface:
 - 1.1. Compare the component values in etRunner before they are passed to etfomm, with those after they are retrieved from etfomm.
 - 1.2. Test result is “Passed” if the two sets of values in the above step are matched.
2. etfomm and API Client successfully exchange data via WCF Server.
 - 2.1. Compare the component values in API Client before they are set to WCF Server, with those in etRunner after they are retrieved from WCF Server.
 - 2.2. Compare the component values in before they are set to WCF Server, with those in API Client after they are retrieved from WCF Server.
 - 2.3. Test Result is “Passed” if the two sets of values in the above two steps are matched.
3. Data modifications in API Client correctly reflect in etfomm.
 - 3.1. Modified the component values in API Client.
 - 3.2. Compare 3 sets of component values: the component values in API Client before they are set to WCF Server, those in etRunner after they are retrieved from WCF Server, and those in etrunner after they are retrieved from etfomm.
 - 3.3. Test result is “Passed” if the three sets of values in step 2 are matched.

4.3 Test Results

4.2.1 Overview of Test Results

1. etRunner correctly calls etfomm functions via etFommInterface: all tests passed.
2. etfomm and API Client successfully exchange data via WCF Server: all tests passed.
3. Data modifications in API Client correctly reflect in etfomm: all tests passed.

4.2.2 Detailed Test Results

1. etRunner correctly calls etfomm functions via etFommInterface:

Component	Function Name in etfomm.dll	Result
Freeway Vehicle	set_fvehicle_struct	Passed

	get_fvehicle_struct_size	
	get_fvehicle_struct	
Street Vehicle	set_svehicle_struct	Passed
	get_svehicle_struct_size	
	get_svehicle_struct	
Freeway Link	set_number_of_freewaylinks	Passed
	define_freewaylinks	
	get_number_of_freewaylinks	
	get_freewaylinks	
Street Link	set_number_of_streetlinks	Passed
	define_streetlinks	
	get_number_of_streetlinks	
	get_streetlinks	
AC Signal	set_number_of_ac_signals	Passed
	define_ac_signals	
	get_number_of_ac_signals	
	get_ac_signals	
FTC Signal	set_number_of_ftc_signals	Passed
	define_ftc_signals	
	get_number_of_ftc_signals	
	get_ftc_signals	
Entry Node	set_number_of_entrynodes	Passed
	define_entrynodes	
	get_number_of_entrynodes	
	get_entrynodes	
Ramp Meter	set_number_of_rampmeters	Passed
	define_rampmeters	
	get_number_of_rampmeters	
	get_rampmeters	
Network	set_network_inputs	Passed
	get_network_inputs	
Freeway Network	set_freeway_network_inputs	Passed
	get_freeway_network_inputs	
Street Network	set_street_network_inputs	Passed
	get_street_network_inputs	
Vehicle Type	define_vehicle_types	Passed
	get_number_of_vehicle_types	
	get_vehicle_types	
Freeway Detector	set_number_of_freeway_detectors	Passed
	define_freeway_detectors	
	get_number_of_freeway_detectors	

	get_freeway_detector_data	
Street Detector	set_number_of_street_detectors	Passed
	define_street_detectors	
	get_number_of_street_detectors	
	get_street_detector_data	
Conditional Turn Percent	define_conditional_turnpcts	Passed
	get_conditional_turnpcts	
Bus Route	set_number_of_busroutes	Passed
	define_busroutes	
	get_number_of_busroutes	
	get_busroutes	
Bus Station	define_busstations	Passed
	get_busstations	
Incident	set_number_of_incidents	Passed
	define_incidents	
	get_number_of_incidents	
	get_incidents	
Parking Zone	set_number_of_parking_zones	Passed
	define_parking_zones	
	get_number_of_parking_zones	
	get_parking_zones	
Event	set_number_of_events	Passed
	define_events	
	get_number_of_events	
	get_events	
Diversion	set_number_of_diversions	Passed
	define_diversions	
	get_number_of_diversions	
	get_diversions	
Node Coordinate	define_node_coordinates	Passed
	get_node_coordinates	

2. etfomm and API Client successfully exchange data via WCF Server:

Component	Function Name in WCFServer.dll	Result
Freeway Vehicle	SetServerFVehicleData	Passed
	GetServerFVehicleData	Passed
Street Vehicle	SetServerSVehicleData	Passed
	GetServerSVehicleData	Passed
Freeway Link	SetServerFreewayData	Passed
	GetServerFreewayData	Passed

Street Link	SetServerStreetData	Passed
	GetServerStreetData	Passed
AC Signal	SetServerACData	Passed
	GetServerACData	Passed
FTC Signal	SetServerFTCSignalData	Passed
	GetServerFTCSignalData	Passed
Entry Node	SetServerEntryNodeData	Passed
	GetServerEntryNodeData	Passed
Ramp Meter	SetServerRampmeterInputs	Passed
	GetServerRampmeterInputs	Passed
Network	SetServerNetworkInput	Passed
	GetServerNetworkInput	Passed
Freeway Network	SetServerFreewayNetworkInput	Passed
	GetServerFreewayNetworkInput	Passed
Street Network	SetServerStreetNetworkInput	Passed
	GetServerStreetNetworkInput	Passed
Vehicle Type	SetServerVehicleTypeInputs	Passed
	GetServerVehicleTypeInputs	Passed
Freeway Detector	SetServerFreewayDetectorInputs	Passed
	GetServerFreewayDetectorInputs	Passed
Street Detector	SetServerStreetDetectorInputs	Passed
	GetServerStreetDetectorInputs	Passed
Conditional Turn Percent	SetServerCondTurnpctData	Passed
	GetServerCondTurnpctData	Passed
Bus Route	SetServerBusRouteInputs	Passed
	GetServerBusRouteInputs	Passed
Bus Station	SetServerBusStationInputs	Passed
	GetServerBusStationInputs	Passed
Incident	SetServerIncidentData_Inputs	Passed
	GetServerIncidentData_Inputs	Passed
Parking Zone	SetServerParkingData	Passed
	GetServerParkingData	Passed
Event	SetServerEventData	Passed
	GetServerEventData	Passed
Diversion	SetServerDiversionData	Passed
	GetServerDiversionData	Passed
Node Coordinate	SetServerXYCoordInputs	Passed
	GetServerXYCoordInputs	Passed

3. Data modifications in API Client correctly reflect in etfomm: all tests passed.

Component	Function Name in etRunner	Result
Freeway Vehicle	UpdateFVehicles	Passed
	ProcessFVehicleData	Passed
Street Vehicle	UpdateSVehicles	Passed
	ProcessSVehicleData	Passed
Freeway Link	UpdateFreewayLinkData	Passed
	ProcessFreewayLinksData	Passed
Street Link	UpdateStreetLinkData	Passed
	ProcessStreetLinksData	Passed
AC Signal	UpdateACSignals	Passed
	(inside ProcessStreetLinksData)	Passed
FTC Signal	UpdateFTCSignals	Passed
	ProcessFTCSignals	Passed
Entry Node	UpdateEntryNodes	Passed
	ProcessEntryNodes	Passed
Ramp Meter	UpdateRampMeters	Passed
	ProcessRampMeter	Passed