Partner Interoperability Development Guide

Overview

The Partner Interoperability Development Guide describes the secure web service interface that Juvare provides to enhance system interoperability for its clients and partners. With this web service, data is communicated using XML, which complies with data standards represented in XML schema documents.

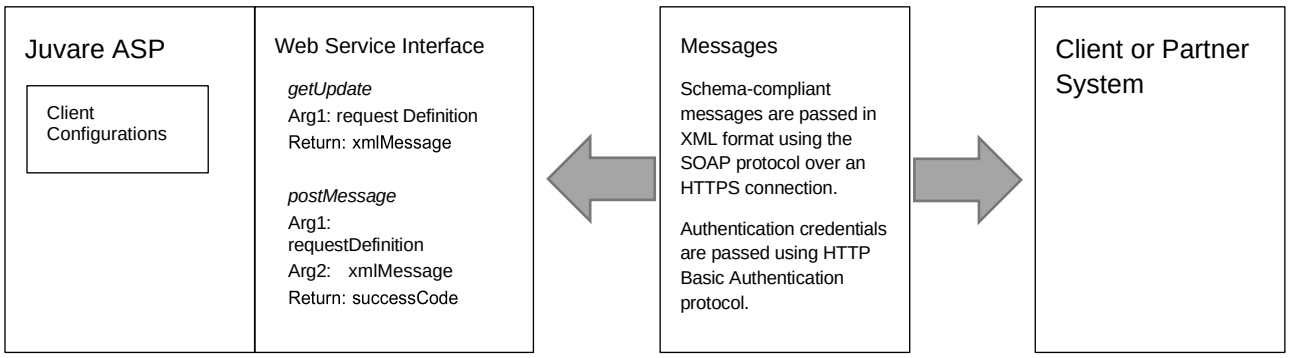
These schemas include:

- Hospital Availability Exchange (EDXL-HAVE), standard for hospital resource communication
- Tracking of Emergency Patients (EDXL-TEP), defines pre-hospital patient records
- Common Alert Protocol (EDXL-CAP), standard for emergency alert communication

Important: It is assumed that a business agreement between Juvare and the business partner is in effect and the specifics of the data to be shared are understood by all parties.

Technical Overview

The following diagram shows the technologies used to communicate data between Juvare and a partner system.



To Configure the Web Service

1. Work with Juvare to obtain a system ID and password, and request definition arguments.
2. Follow the development example outlined in this document to connect to a web service in Juvare's public test environment.
3. In the test environment, take these actions.
    a. Enhance the service client code
    b. Define the data dictionary of expected values
    c. Identify the appropriate XML schema
4. Develop the transformation code between XML messages and system data structures.

Web Service Interface

Juvare provides a simple web service interface that is appropriate for most interoperability needs. Connections are made over the TLS-enabled HTTP protocol (HTTPS). This provides security in the form of encrypted communication and a certificate authority-verified connection to the hosting Juvare server.

XML messages are packaged inside of SOAP envelopes. The web service is described using a document or literal style WSDL. The WSDL file encompasses everything that a .NET or Java generator utility needs to create custom stub classes that hide most of the web service code.

The partner system provides a username and password using the HTTP Basic Authentication interface provided by their platform. This is conveniently supported in both .NET and Enterprise Java.

The interface consists of two methods:

Interface Name	Arguments	Return Value
getUpdate	requestDefinition	xmlMessage
postMessage	requestDefinition xmlMessage	successCode

The **getUpdate** interface can be used to poll for data, either as needed or on an ongoing basis. A request definition XML string is specified as an argument, which resolves to a query for the specific data of interest. An XML message is returned, which must comply with the data dictionary and the XML schema determined by the developers.

The **postMessage** interface is used to upload data to Juvare. A request definition XML string is specified along with the XML message as arguments that resolve to a customized display for Juvare users. The XML message must comply with the data dictionary and the XML schema determined by the developers.

Both interfaces expect the request definition XML string to follow the **PartnerEndpoint.xsd** schema indicated below.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://gateway.juvare.com/types"
targetNamespace="http://gateway.juvare.com/types" version="1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="getUpdate" type="tns:getUpdate" />
<xs:element name="getUpdateResponse" type="tns:getUpdateResponse" />
<xs:element name="postMessage" type="tns:postMessage" />
<xs:element name="postMessageResponse" type="tns:postMessageResponse" />
<xs:complexType name="postMessage">
                                    <xs:sequence>
<xs:element minOccurs="0" name="String_1" type="xs:string" />
<xs:element minOccurs="0" name="String_2" type="xs:string" />
                                    </xs:sequence>
</xs:complexType>
<xs:complexType name="postMessageResponse">
                                    <xs:sequence>
<xs:element minOccurs="0" name="result" nillable="true" type="xs:string" />
```

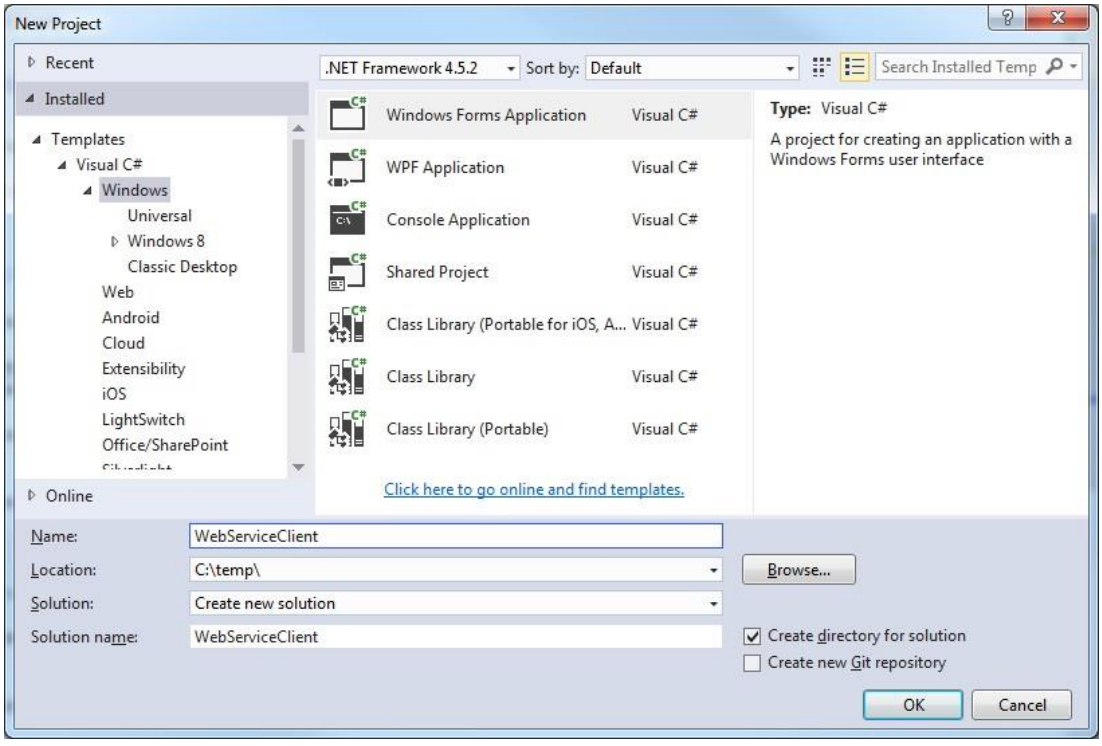


```
                                    </xs:sequence>
</xs:complexType>
<xs:complexType name="getUpdate">
                                     <xs:sequence>
<xs:element minOccurs="0" name="String_1" type="xs:string" />
                                    </xs:sequence>
</xs:complexType>
<xs:complexType name="getUpdateResponse">
                                     <xs:sequence>
<xs:element minOccurs="0" name="result" nillable="true" type="xs:string" />
                                    </xs:sequence>
</xs:complexType>
</xs:schema>
```

Example: Connecting with a .NET Web Service Client

Step 1: Create a New Windows Application

1. Launch the Visual Studio application.
2. Open a new project by following the path: File > New > Project.
3. From the left side panel, follow the path: Visual C# > Windows.
4. From the .NET Framework center panel, select Windows Forms Application.
5. Create project name (Example: WebServiceClient) and insert it in the corresponding field.

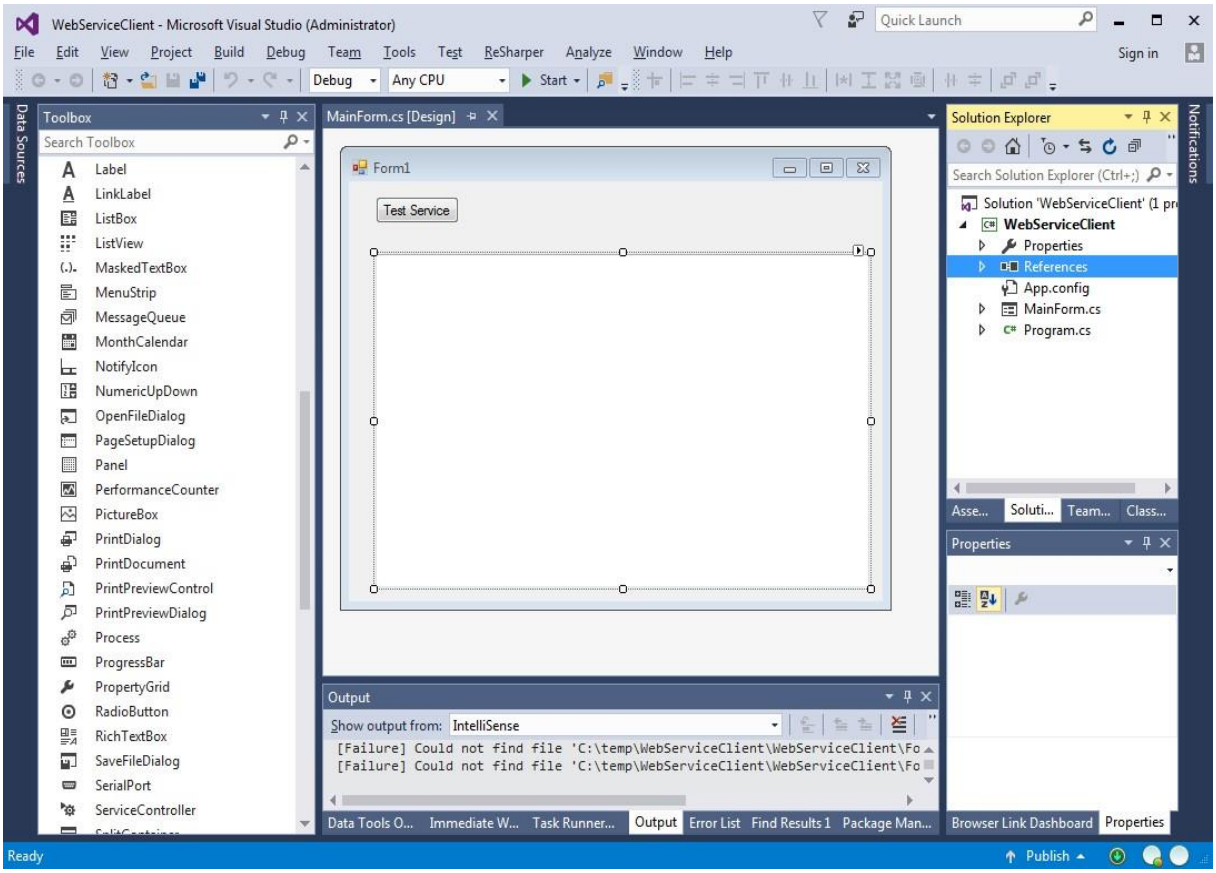


6. Click OK.

Step 2: Create a UI Form

Make your form look like the example below.

1. Add one button from the left sided *Toolbox* panel.
2. Add one text box from the left sided *Toolbox* panel.
3. Change the **TextBox Multiline** property to true.



Step 3: Rename the Output Text Box

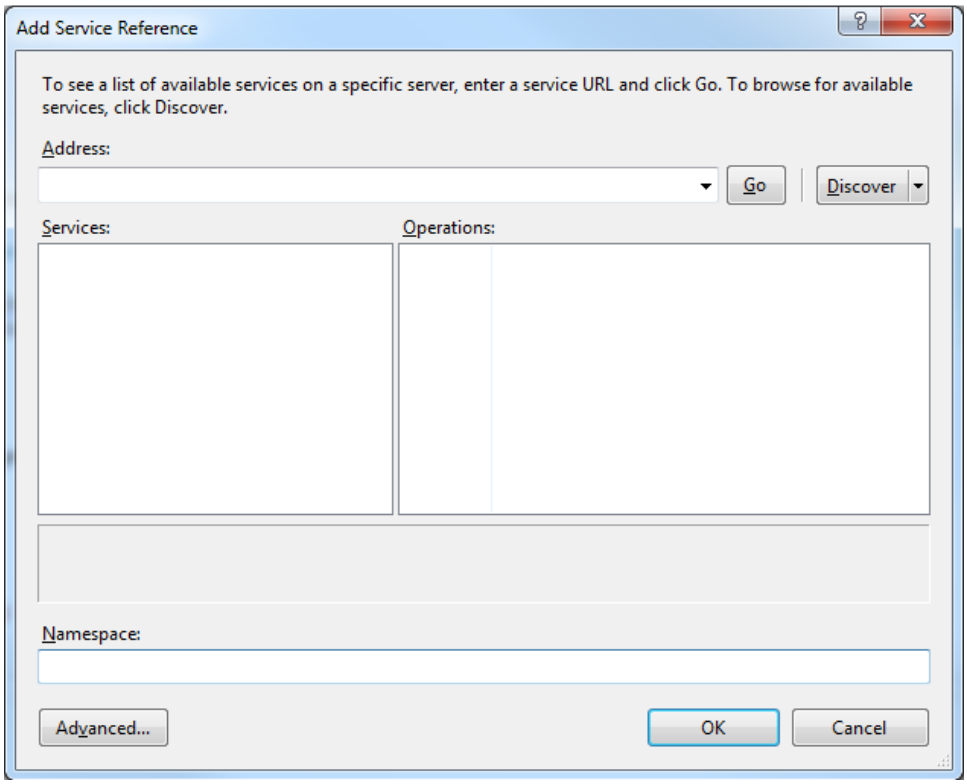
- Rename the multiline text box to **ResultTextBox**

Step 4: Generate the Stub Code

1. In *Solution Explorer*, in the *WebServiceClient* project, right click the **References** node and select **Add Service Reference** from the context menu. An *Add Service Reference* window opens.
2. Enter the following URL in the **Address** input field: https://emresource.lab.juvare.com/partnergateway/PartnerEndpoint?wsdl
3. Click **Go**.

   Note: The service proxy is generated and will appear in the *Services* section.

4. Security warnings may appear. Click **No** in each warning window.



5. Change the **Namespace** to **Juvare**.
6. Click **OK.**
7. Go to the application configuration file **App.config** and replace the xml element:

```
<security mode="Transport"/>
with the following:
<security mode="Transport"/>
          <transport clientCredentialType="Basic" />
</security>
```

Step 5: Add Event Handler Code

1. Double-click the button you previously added in **Step 2**. The *source code Editor* opens displaying the event handling code.
2. Insert the following code.

```
using (var partnerService = new PartnerEndpointClient())
{
var clientCredentials = partnerService.ClientCredentials;
clientCredentials.UserName.UserName = "testuser";
clientCredentials.UserName.Password = "pass1234";
var action = "GET_STATUS";
var systemName = "EMRESOURCE"; var divisionName = "TEST_REGION";
string response; try
```



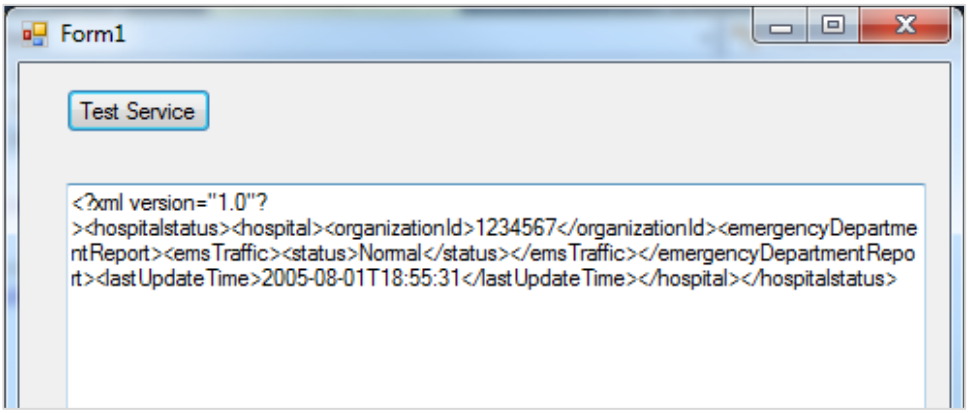
```
{
response = partnerService.getUpdate(
$@"<?xml version='1.0'?>
<requestDefinition actionName='{action}'
systemName='{systemName}' divisionName='{divisionName}'>
</requestDefinition>");
}
catch (Exception ex)
{
response = $"Error invoking service {ex.Message}";
}
ResultTextBox.Text = response;
```

3. Add the **Using** statements at the top of this file: using WebServiceClient.Juvare;.

   Note: If you named your project something other than **WebServiceClient**, you must use that name in the first **Imports** statement above (Example: using WebLabStudio.Juvare;).

Step 6: Run the Test Application

1. Compile and run the application.
2. Click the **Test Service** button. You should see the following.



Need Help?

For more information, contact the Juvare Support Center by sending an email to support@juvare.com or by calling 877-771-0911.