How to consume a Lawson Web Service from a Personalized Script in Smart Office

Thibaud Lopez Schneider
Lawson Software
June 15, 2010

In this paper I illustrate a new solution to consume a Lawson Web Service from a Personalized Script in JScript.NET in Lawson Smart Office. This new solution complements the other three known solutions. This one is interesting because it minimizes code source surface while still ensuring SOAP validation. And the solution does not involve any C# coding, nor does it require Microsoft Visual Studio. For this new solution, we will use the free Microsoft Web Services Description Language Tool (wsdl.exe) to generate a proxy class in C# that we'll use from JScript.NET.

Table of contents

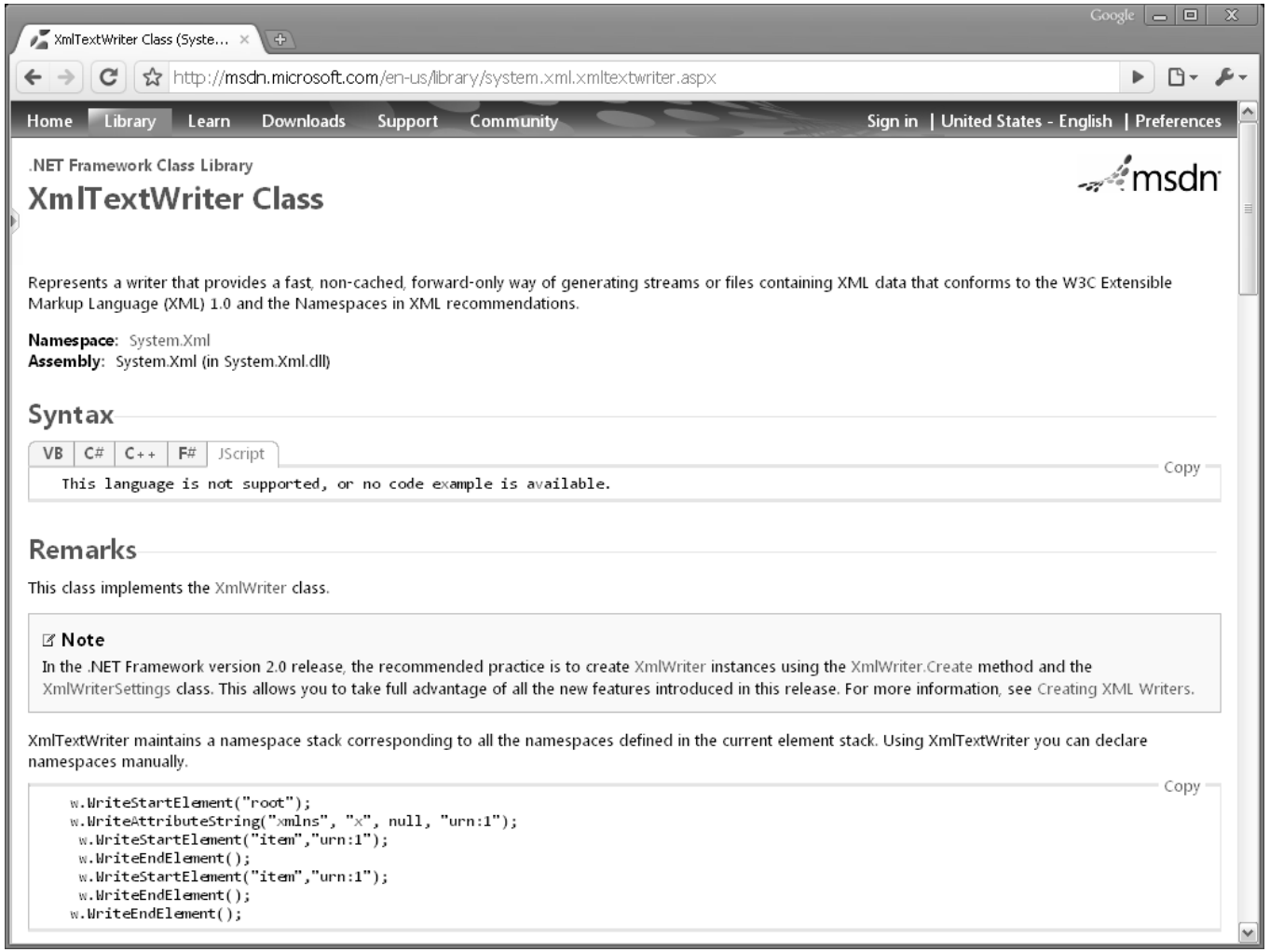
Background

As of today, there are three known solutions to call a Lawson Web Service from a Personalized Script in Lawson Smart Office: 1) the “Big string”, 2) the XML writer, and 3) the C# proxy written with Microsoft Visual Studio. Each of these solutions has its advantages and disadvantages.

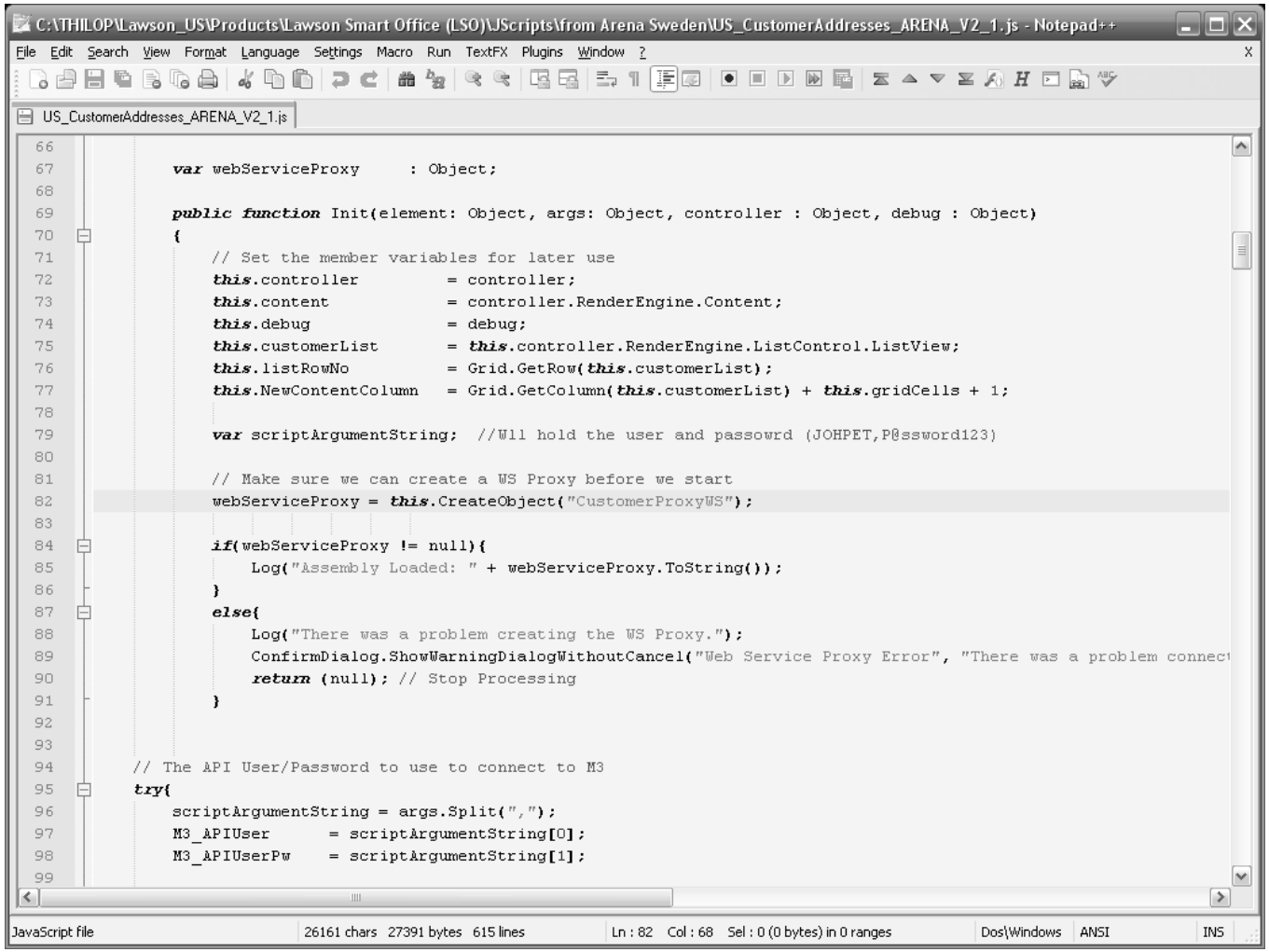
Known Solution 1: The “Big String” solution creates the SOAP Request in a long String or StringBuilder in JScript.NET, and sends it to the SOAP server with XmlHttpRequest or WebRequest. This solution is easy to implement because it doesn’t require any special skills nor tool, but it doesn’t provide SOAP validation, it also requires handling the Request/Response, and it’s hard to maintain.

```
xml += '<?xml version="1.0" encoding="utf-8"?>';
xml += '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"';
xml += ' xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"';
xml += ' xmlns:xsd="http://www.w3.org/2001/XMLSchema">';
xml += '<soap:Header>';
xml += '<mws xmlns="http://mws.intentia.net/mws2">';
xml += '<user>' + user + '</user>';
xml += '<password>' + password + '</password>';
xml += '</mws>';
xml += '</soap:Header>';
xml += '<soap:Body>';
xml += '<List xmlns="http://sales.lawson.com/MMSO60MI/List">';
xml += '<ListItem>';
xml += '<Warehouse>' + warehouse + '</Warehouse>';
xml += '<ItemNumber>' + itemNumber + '</ItemNumber>';
xml += '</ListItem>';
xml += '</List>';
xml += '</soap:Body>';
xml += '</soap:Envelope>';
```

Known Solution 2: The XML writer solution uses any XML writer class such as System.Xml.XmlTextWriter to create the SOAP Request. I have not tested this solution myself. It's similar to the "Big String" solution, in better because it adds XML validation, but it requires more skills in the programming.

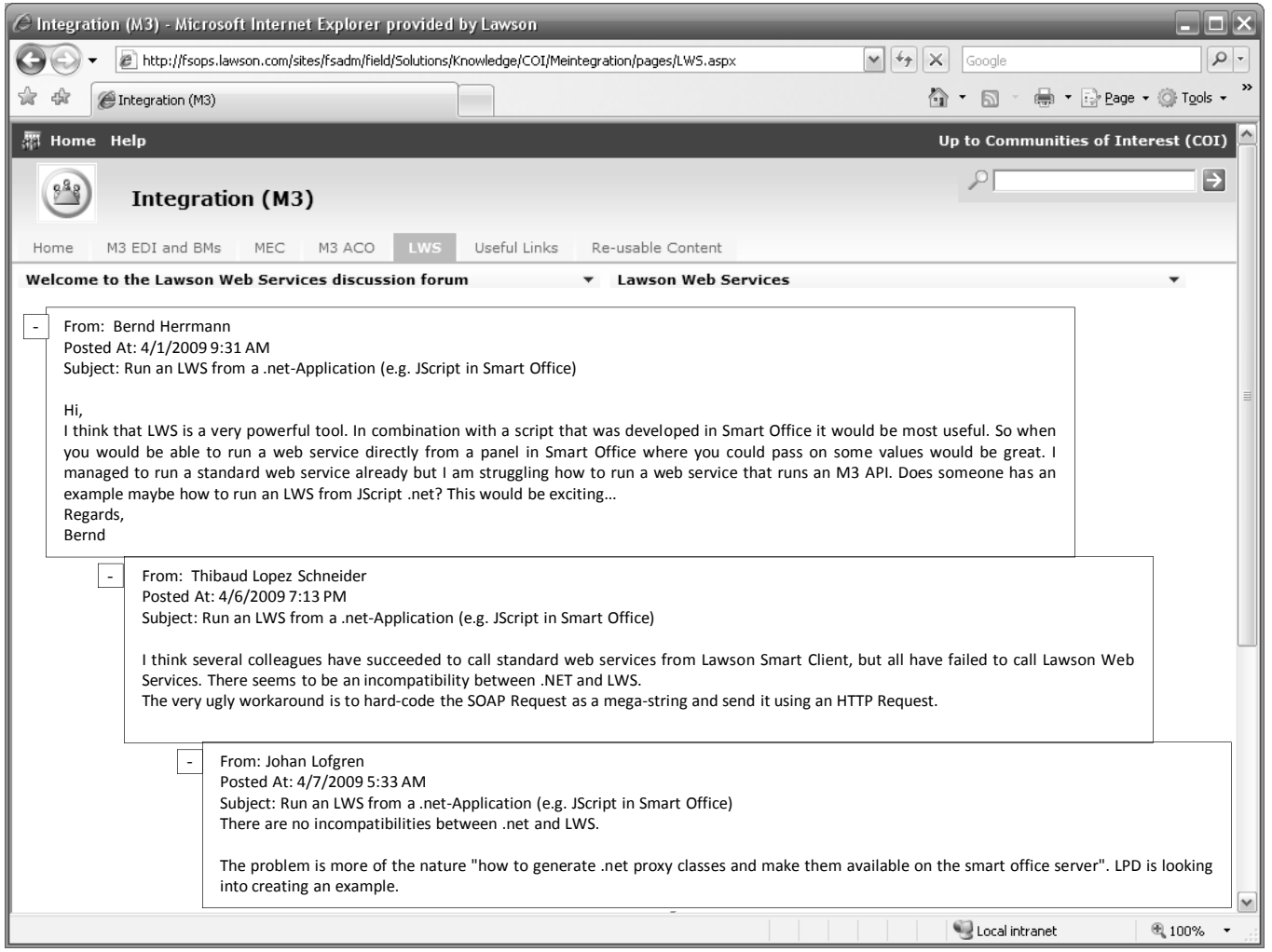


Known Solution 3: Peter A Johansson found a great solution by creating a C# proxy in Microsoft Visual Studio and generating a DLL that hides all the complexity. (Also, Peter suggests storing the DLL in the instance cache to minimize network traffic.) Visual Studio is easy to learn and generates most of the coding, but this solution requires C# programming skills, and Visual Studio skills.



Here is an excerpt of the discussions from the Lawson Web Services Forum in the Community Of Interest (COI) illustrating that there is still no official solution to call a Lawson Web Service from a Personalized Script in Lawson Smart Office.

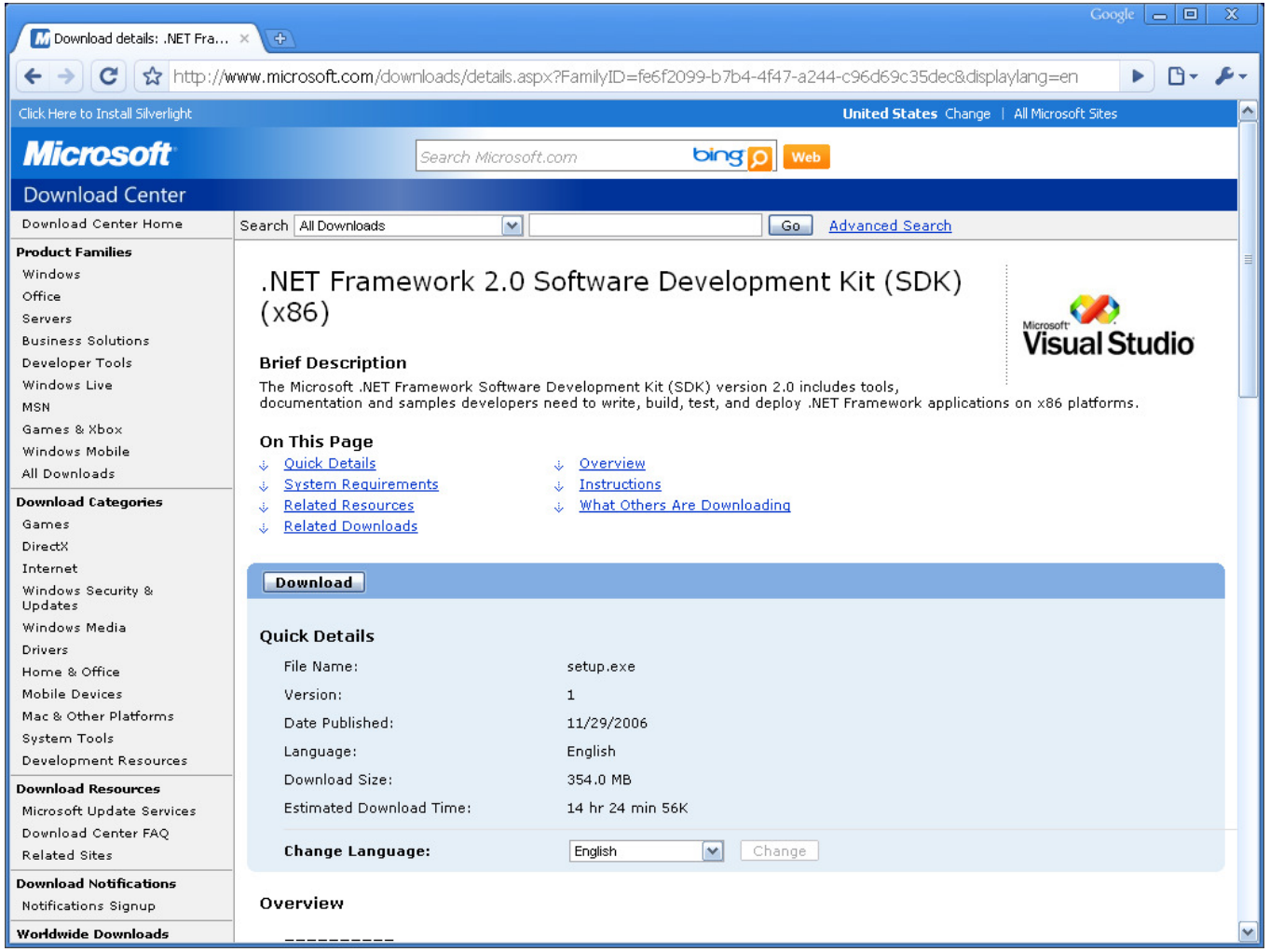
http://fsops.lawson.com/sites/fsadm/field/Solutions/Knowledge/COI/Meintegration/Lists/General%20Discussion10/DispForm.aspx?ID=1&Source=http%3A%2F%2Ffsops%2Elawson%2Ecom%2Fsites%2Ffsadm%2Ffield%2FSolutions%2FKnowledge%2FCOI%2FMeintegration%2Fpages%2FLWS%2Easpx



The fourth solution

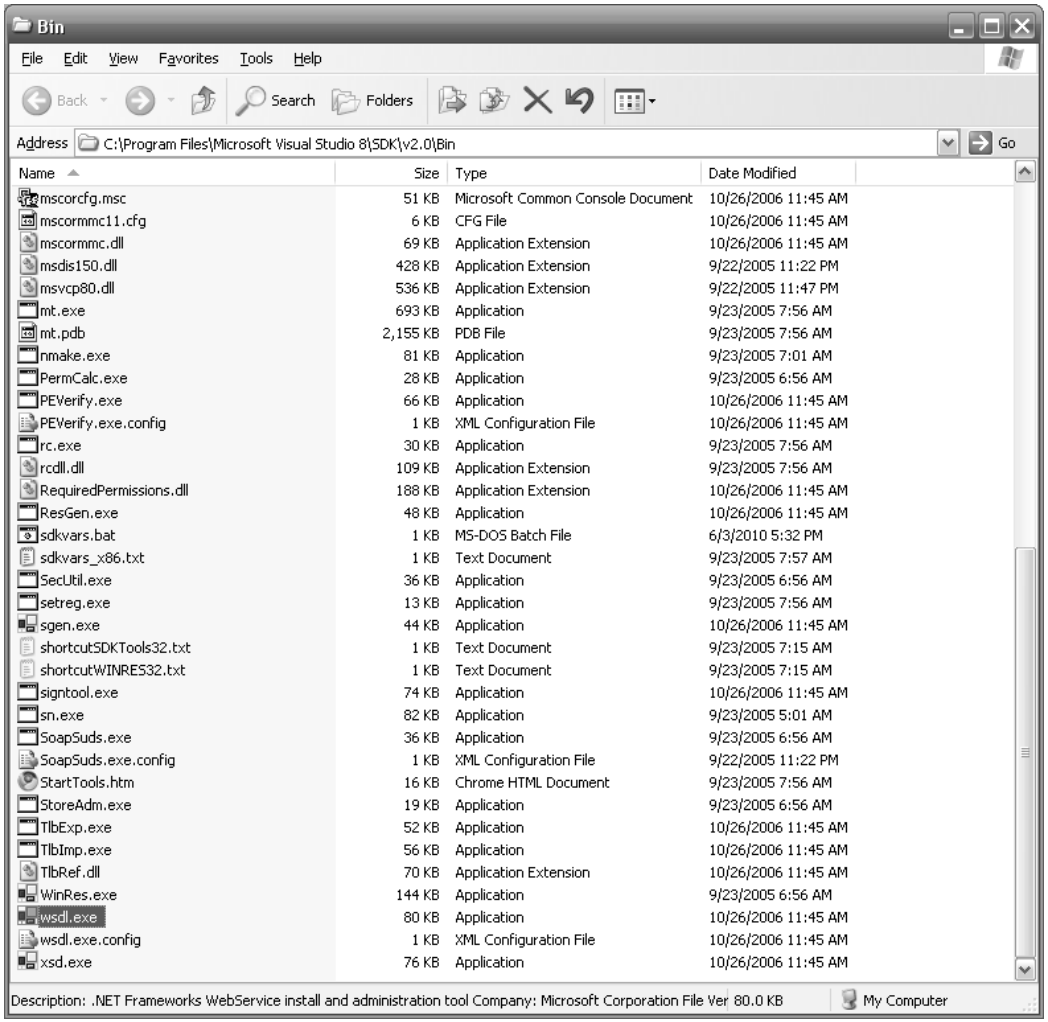
Step 1. Install the free Microsoft .NET SDK. In the SDK, we're particularly interested in the Web Services Description Language Tool (wsdl.exe) which is a proxy class generator for the command line.

http://www.microsoft.com/downloads/details.aspx?FamilyID=fe6f2099-b7b4-4f47-a244-c96d69c35dec&displaylang=en



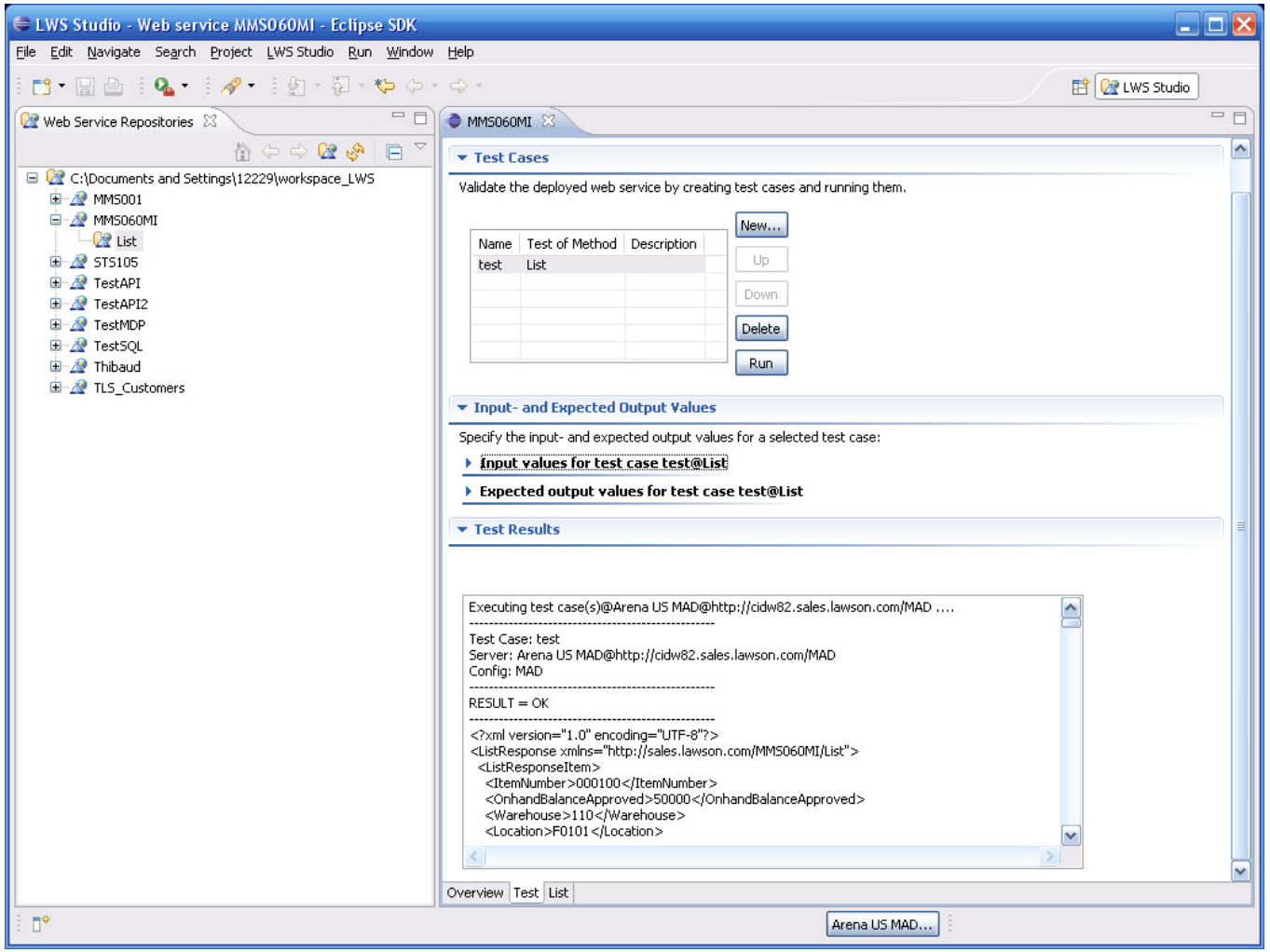
Verify that the SDK correctly installed the Web Services Description Language Tool (wsdl.exe).

C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\

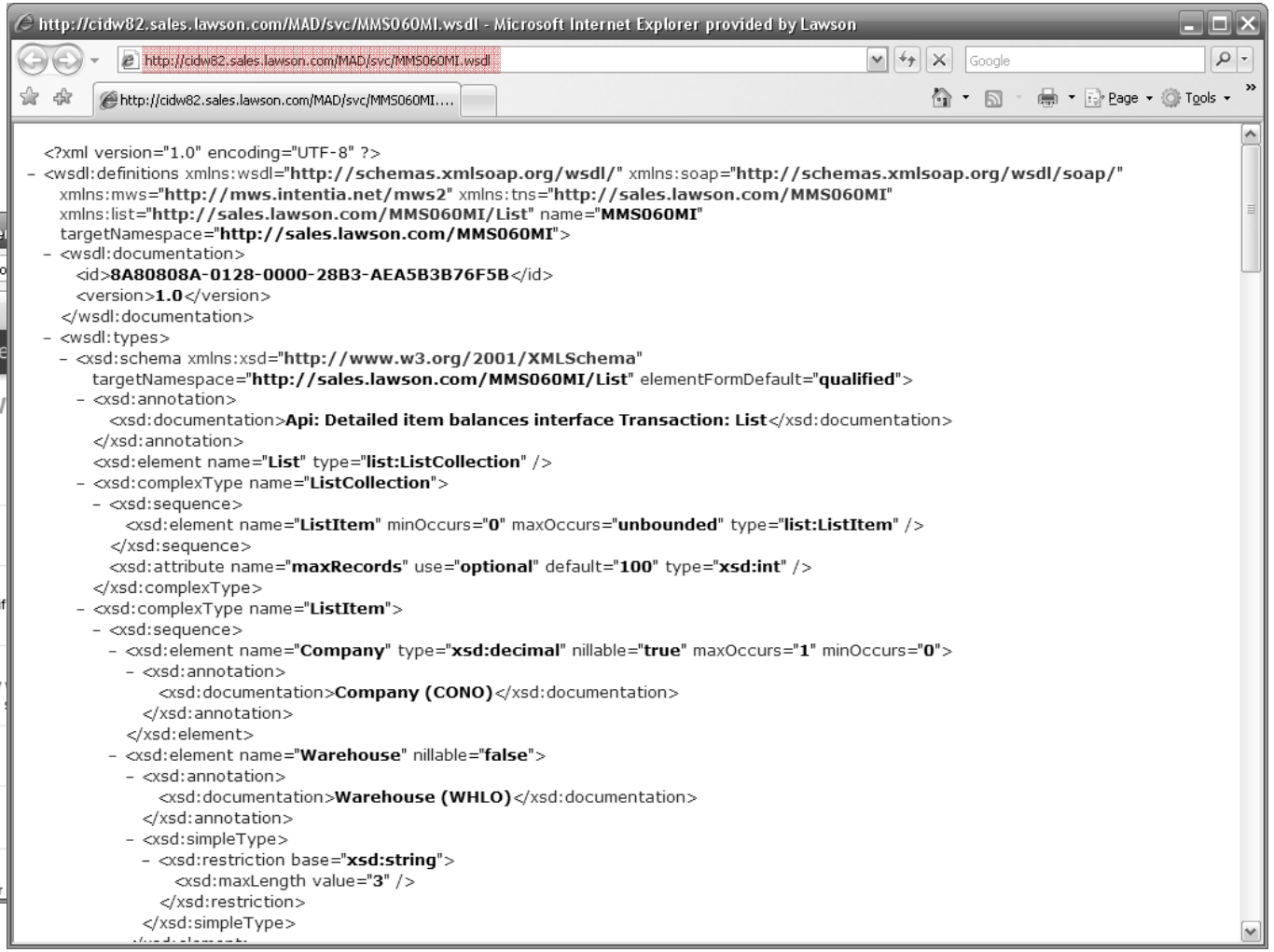


Step 2. Test your web service with Lawson Web Services Studio.

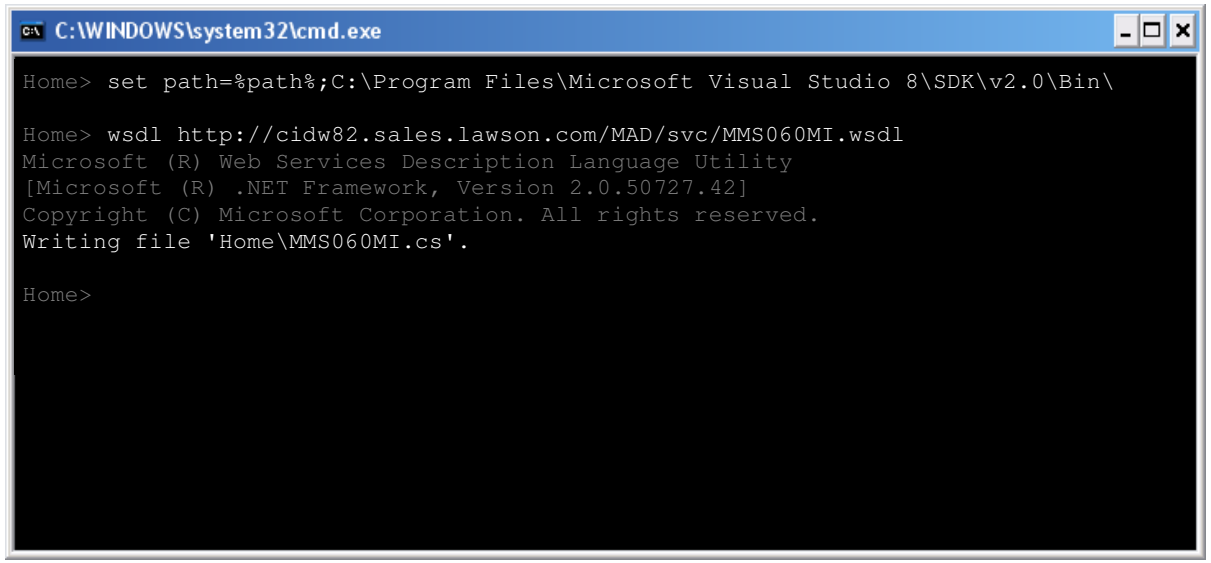
Make sure that it works and returns RESULT = OK.



Copy the URL of your web service's WSDL. (For that, open the Lawson Web Services Administration page, click on *List Web Services*, locate your web service, click on the *wsdl* link on its right, and copy that URL to the clipboard.) For example: http://cidw82.sales.lawson.com/MAD/svc/MMS060MI.wsdl

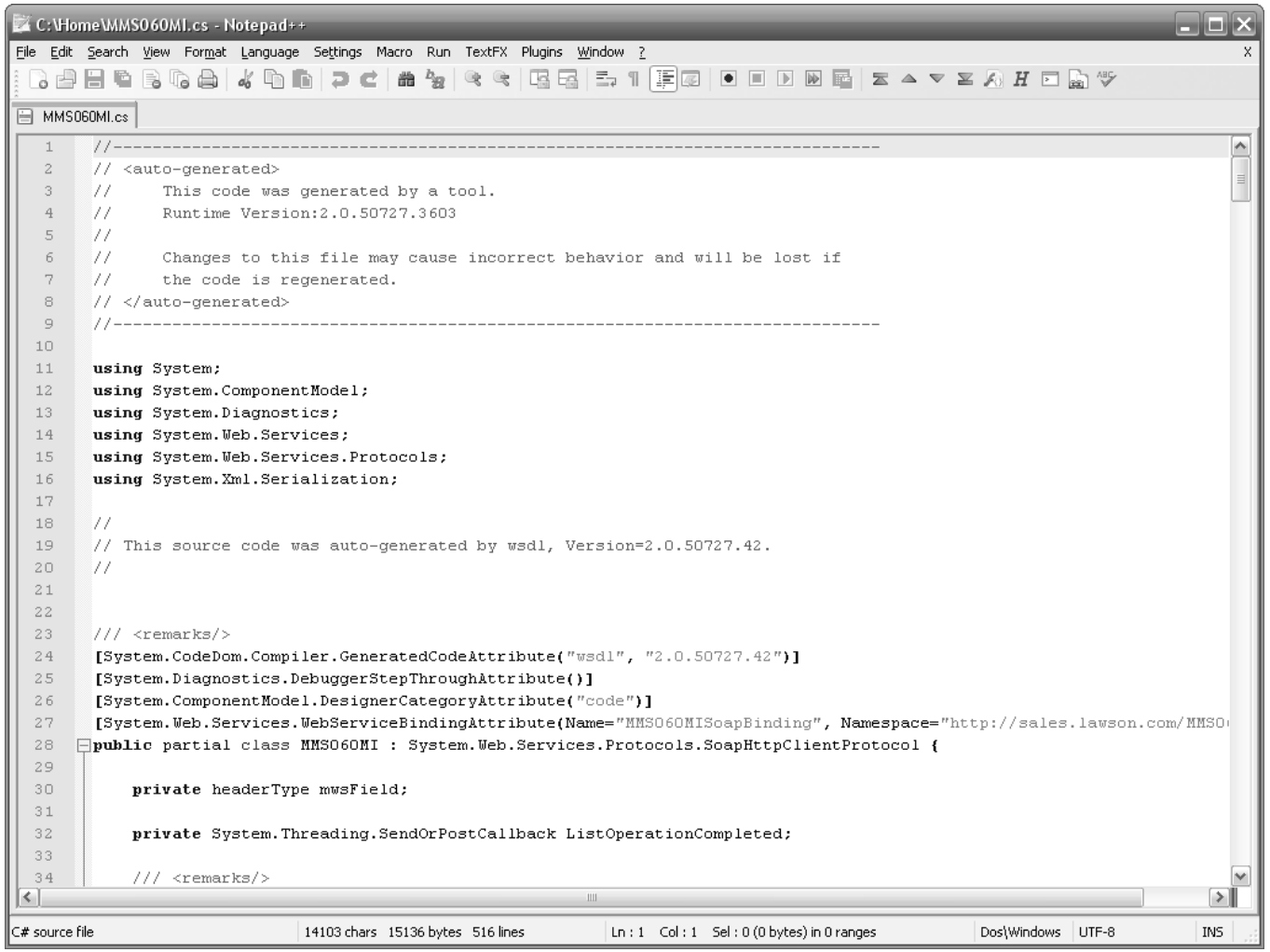


Step 3. Generate a C# proxy class for your web service with the following command: wsdl *MyWSDL*.



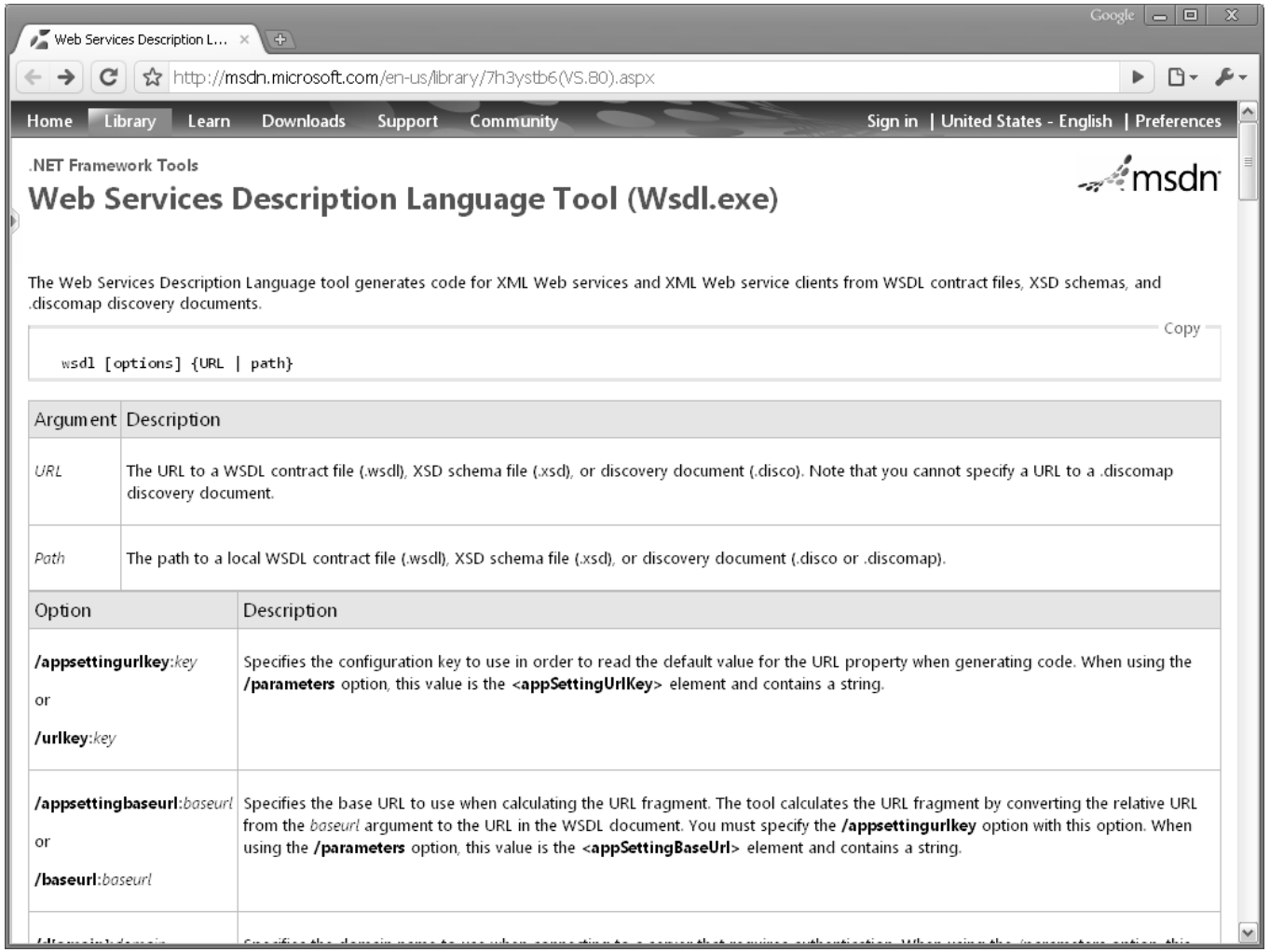
Note 1: You can also try to generate a proxy in JScript.NET, but I tried the three types of Lawson Web Services (API, MDP, and SQL) and I always got the error *"Custom attributes on parameter declarations are not supported by JScriptCodeProvider."* Note 2: You can also save the WSDL as a file in your computer and use a filepath instead of a URL.

Verify that the wsdl.exe tool correctly generated the C# proxy class.

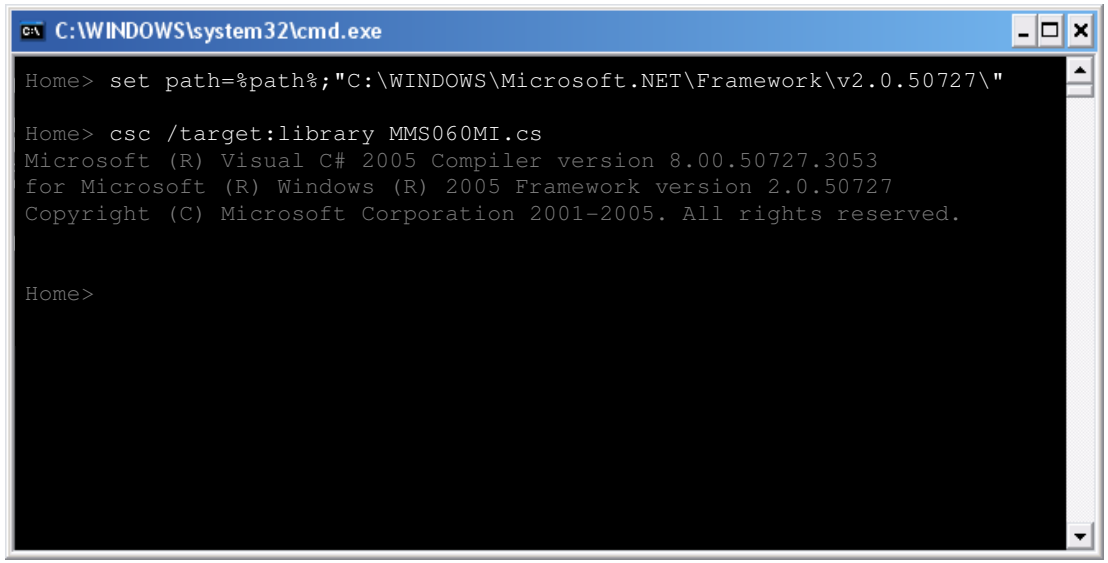


For more information on the wsdl.exe tool, refer to its documentation.

http://msdn.microsoft.com/en-us/library/7h3ystb6(VS.80).aspx

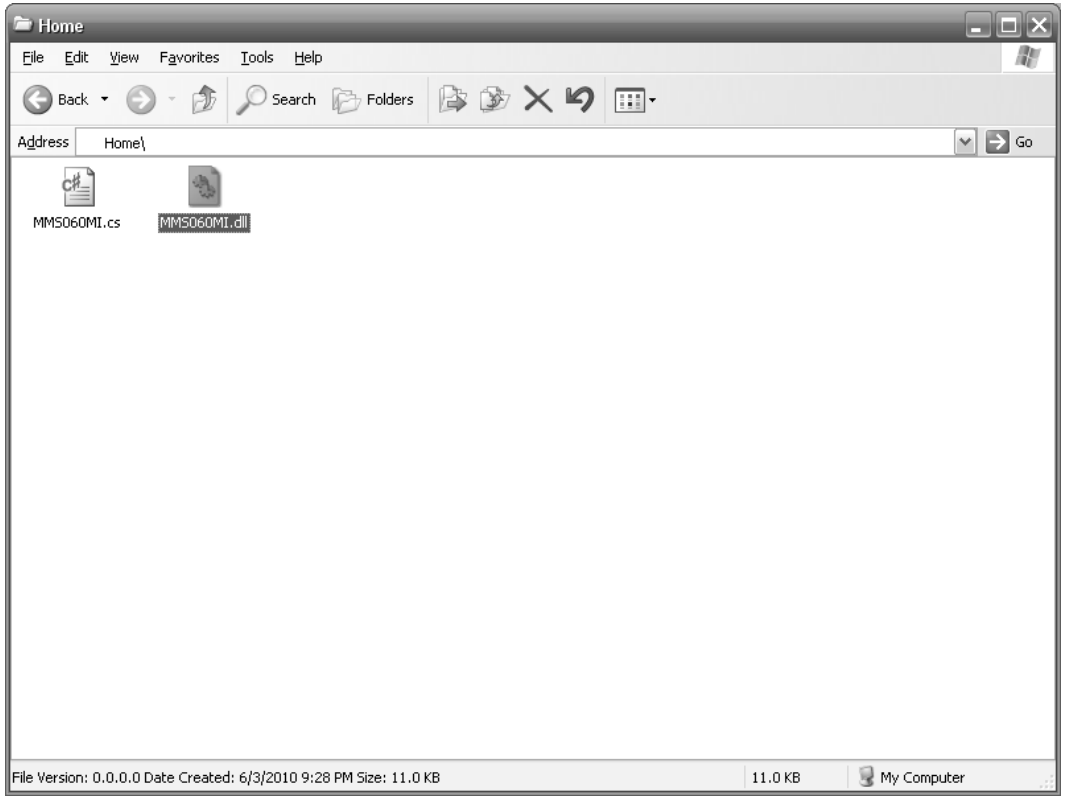


Step 4. Compile the C# proxy class with the following command line: csc /target:library *MyProxy.cs*

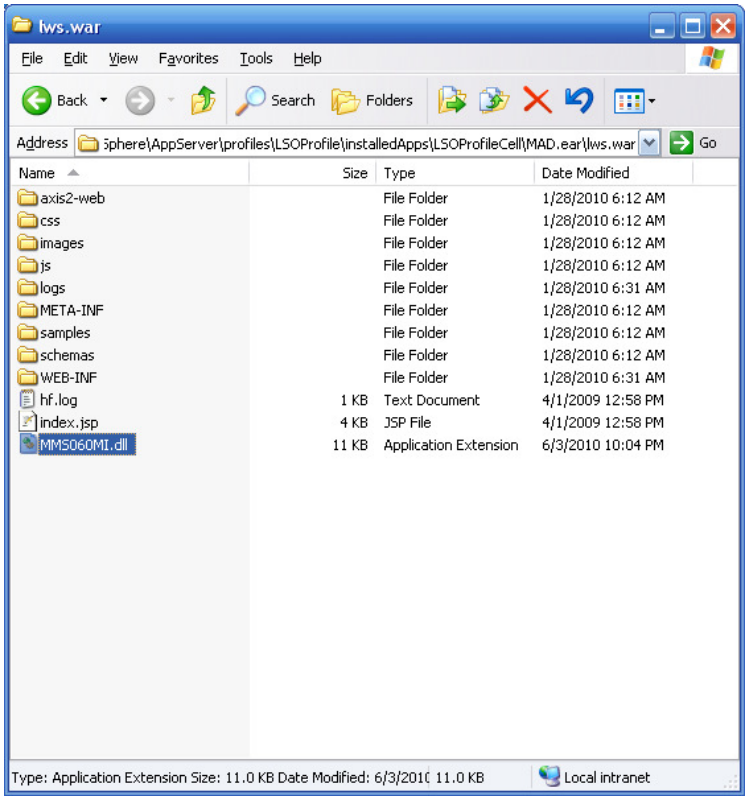


Note 1: The C# compiler (csc.exe) is provided with the .NET Framework so you should have already had it before installing the SDK. Note 2: I'm using the .NET Framework 2.0.x because that's the same one used by the wsdl.exe tool that generated the proxy class, but it also compiled with the .NET Framework 4.0.x.

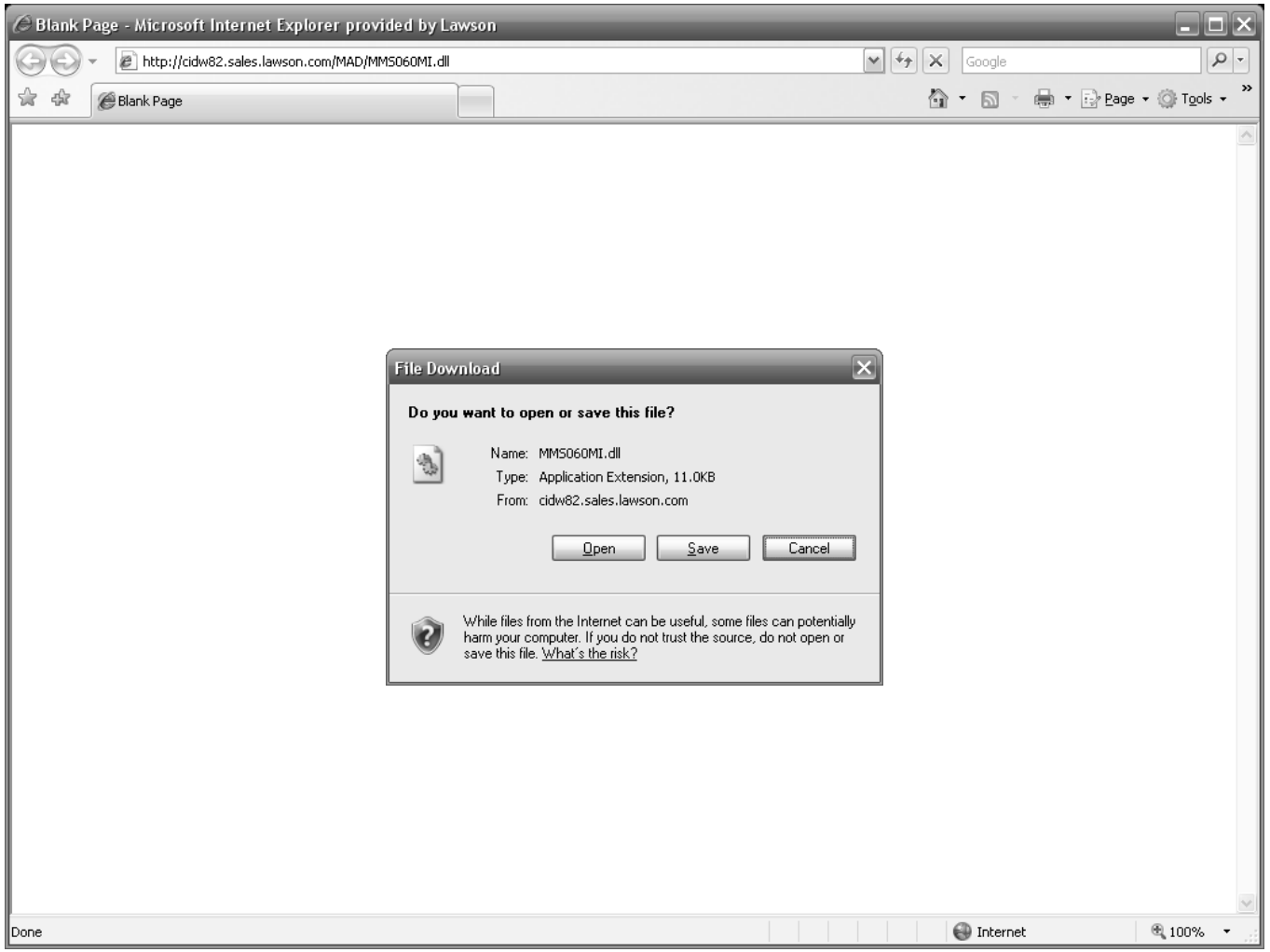
Verify that the compiler correctly created the DLL.



Step 5. Publish the DLL into a location that is accessible to your users, for example in the lws.war folder in WebSphere.



Verify that you have access to the DLL as a user.



Note: You can use a URL like in the screenshot, or shared folder, or a UNC path like \\host\c$\path\file.dll, etc.

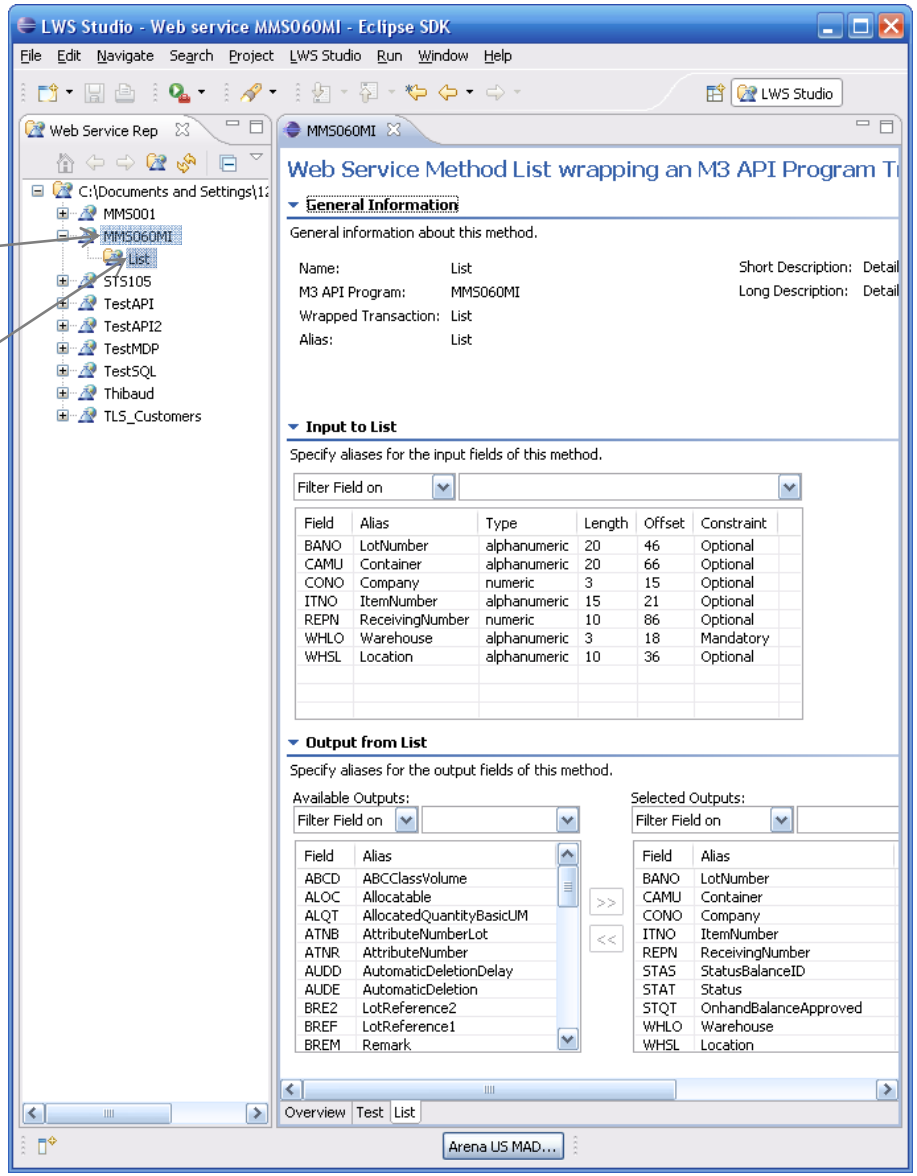
Step 6. Create a Personalized Script in JScript.NET.

```
import System;
import System.Reflection;

package MForms.JScript {
   class TestMyWebService {
      public function Init(element: Object, args: Object, controller : Object, debug : Object) {

        // load the DLL
        var assembly = Assembly.LoadFrom("http://cidw82.sales.lawson.com/MAD/MMS060MI.dll");

        // create a proxy
        var proxy = assembly.CreateInstance("MMS060MI");

        // SOAP Request Header (user and password)
        var header = assembly.CreateInstance("headerType");
        header.user = "secret";
        header.password = "secret";
        proxy.mws = header;

        // SOAP Request Body (input parameters)
        var body = assembly.CreateInstance("ListItem");
        body.Company = new Decimal(701);
        body.Warehouse = "110";

        var collection = assembly.CreateInstance("ListCollection");
        collection.maxRecords = 100;
        var ListItem = body.GetType();
        collection.ListItem = new ListItem[1];
        collection.ListItem[0] = body;

        // call the web service
        var response = proxy.List(collection);

        // SOAP Response
        debug.WriteLine(response.length + " records returned");
        for (var i in response) {
            debug.WriteLine(response[i].Location + "=" + response[i].StatusBalanceID);
        }

      }
   }
}
```

```
// create a proxy
var proxy = assembly.CreateInstance("MMS060MI");

// SOAP Request Header (user and password)
var header = assembly.CreateInstance("headerType");
header.user = "secret";
header.password = "secret";
proxy.mws = header;

// SOAP Request Body (input parameters)
var body = assembly.CreateInstance("ListItem");
body.Company = new Decimal(701);
body.Warehouse = "110";

var collection = assembly.CreateInstance("ListCollection");
collection.maxRecords = 100;
var ListItem = body.GetType();
collection.ListItem = new ListItem[1];
collection.ListItem[0] = body;

// call the web service
var response = proxy.List(collection);

// SOAP Response
debug.WriteLine(response.length + " records returned");
for (var i in response) {
    debug.WriteLine(
        response[i].Location + "=" +
        response[i].StatusBalanceID);
}
```

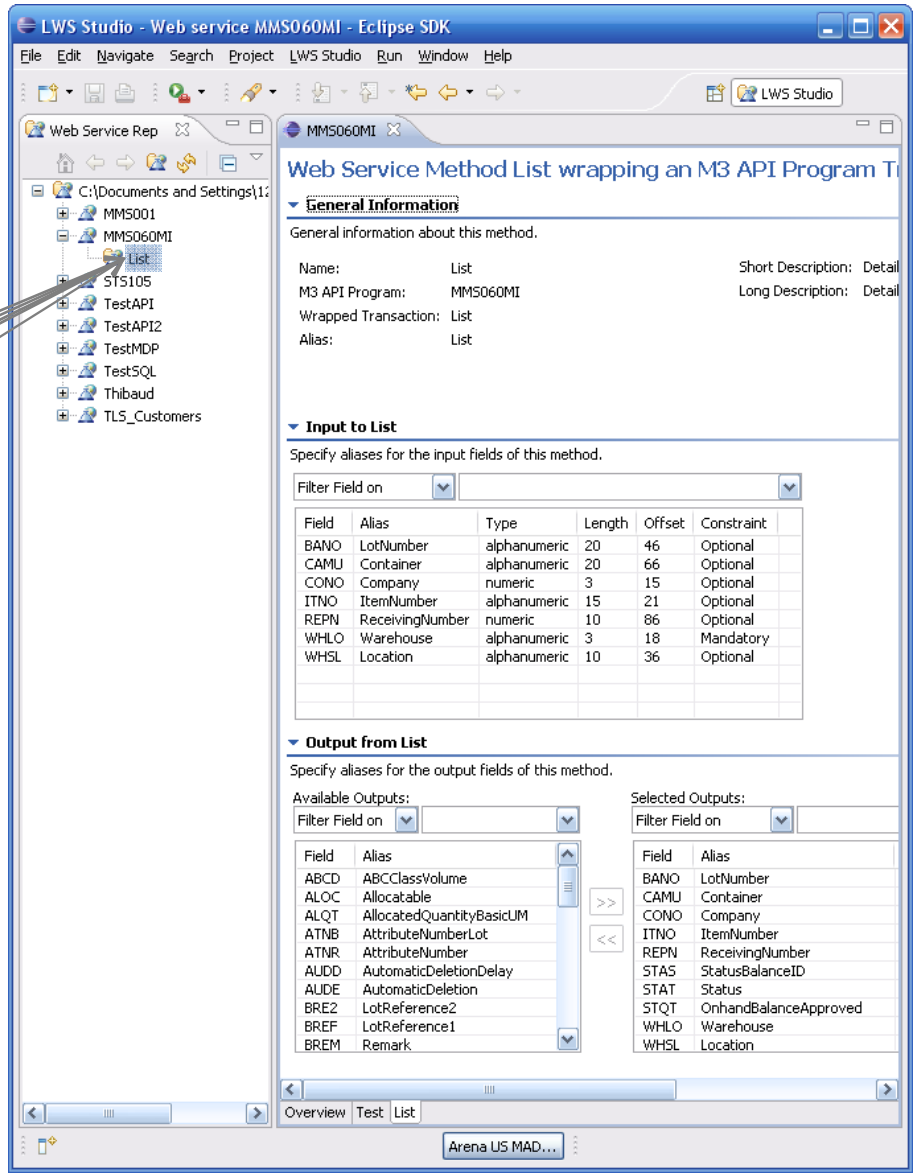
Web Service name

Method name



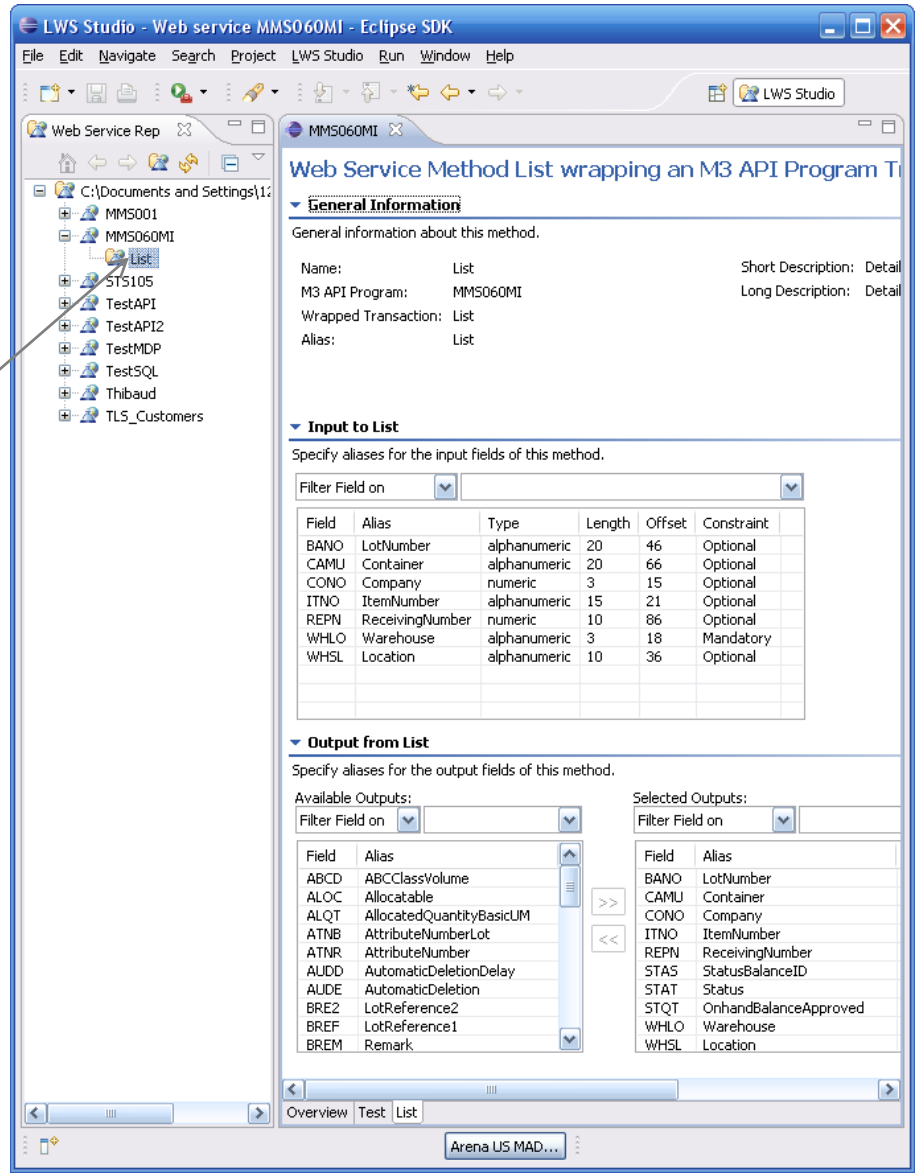
```
// create a proxy
var proxy = assembly.CreateInstance("MMS060MI");

// SOAP Request Header (user and password)
var header = assembly.CreateInstance("headerType");
header.user = "secret";
header.password = "secret";
proxy.mws = header;

// SOAP Request Body (input parameters)
var body = assembly.CreateInstance("ListItem");
body.Company = new Decimal(701);
body.Warehouse = "110";

var collection = assembly.CreateInstance("ListCollection");
collection.maxRecords = 100;
var ListItem = body.GetType();
collection.ListItem = new ListItem[1];
collection.ListItem[0] = body;

// call the web service
var response = proxy.List(collection);

// SOAP Response
debug.WriteLine(response.length + " records returned");
for (var i in response) {
    debug.WriteLine(
        response[i].Location + "=" +
        response[i].StatusBalanceID);
}
```

Method name + Item suffix



```
// create a proxy
var proxy = assembly.CreateInstance("MMS060MI");

// SOAP Request Header (user and password)
var header = assembly.CreateInstance("headerType");
header.user = "secret";
header.password = "secret";
proxy.mws = header;

// SOAP Request Body (input parameters)
var body = assembly.CreateInstance("ListItem");
body.Company = new Decimal(701);
body.Warehouse = "110";

var collection = assembly.CreateInstance("ListCollection");
collection.maxRecords = 100;
var ListItem = body.GetType();
collection.ListItem = new ListItem[1];
collection.ListItem[0] = body;

// call the web service
var response = proxy.List(collection);

// SOAP Response
debug.WriteLine(response.length + " records returned");
for (var i in response) {
    debug.WriteLine(
        response[i].Location + "=" +
        response[i].StatusBalanceID);
}
```

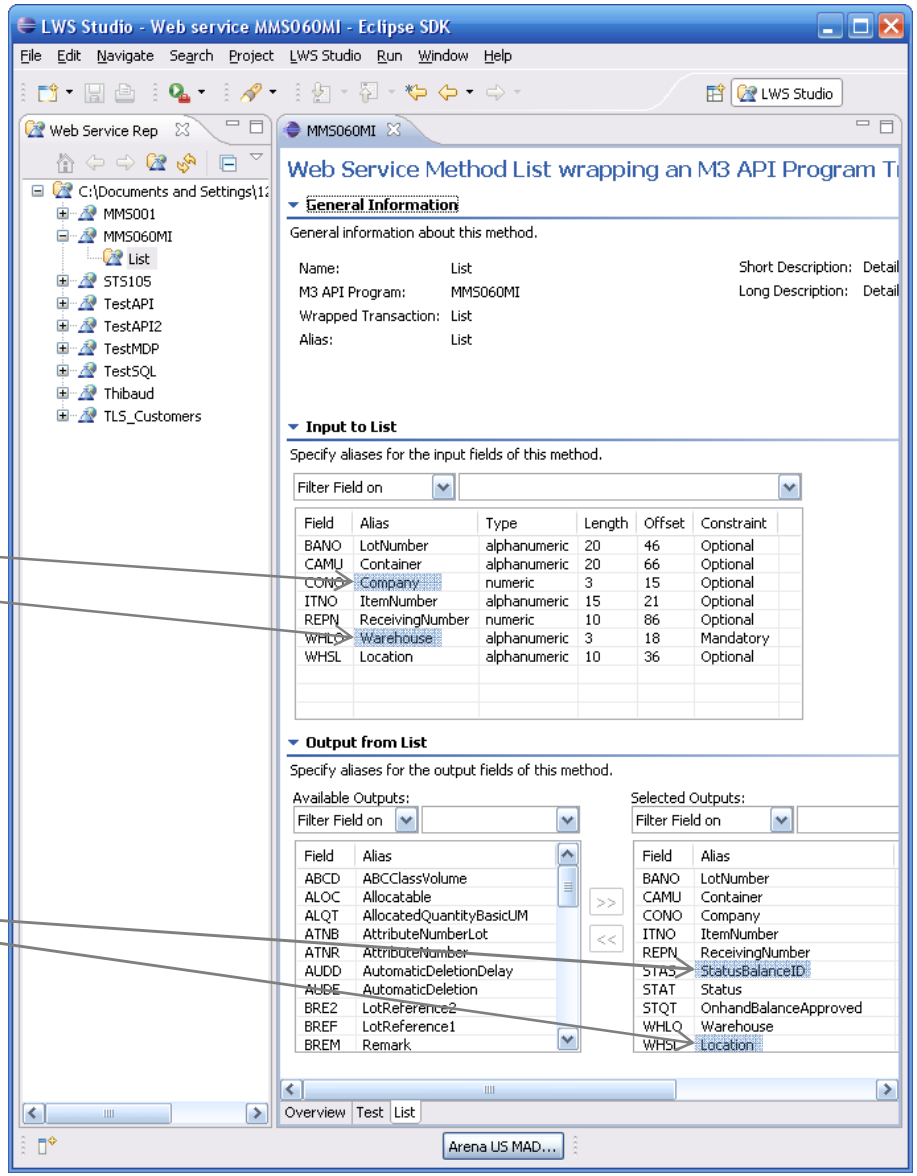
Method name
+ *Collection* suffix



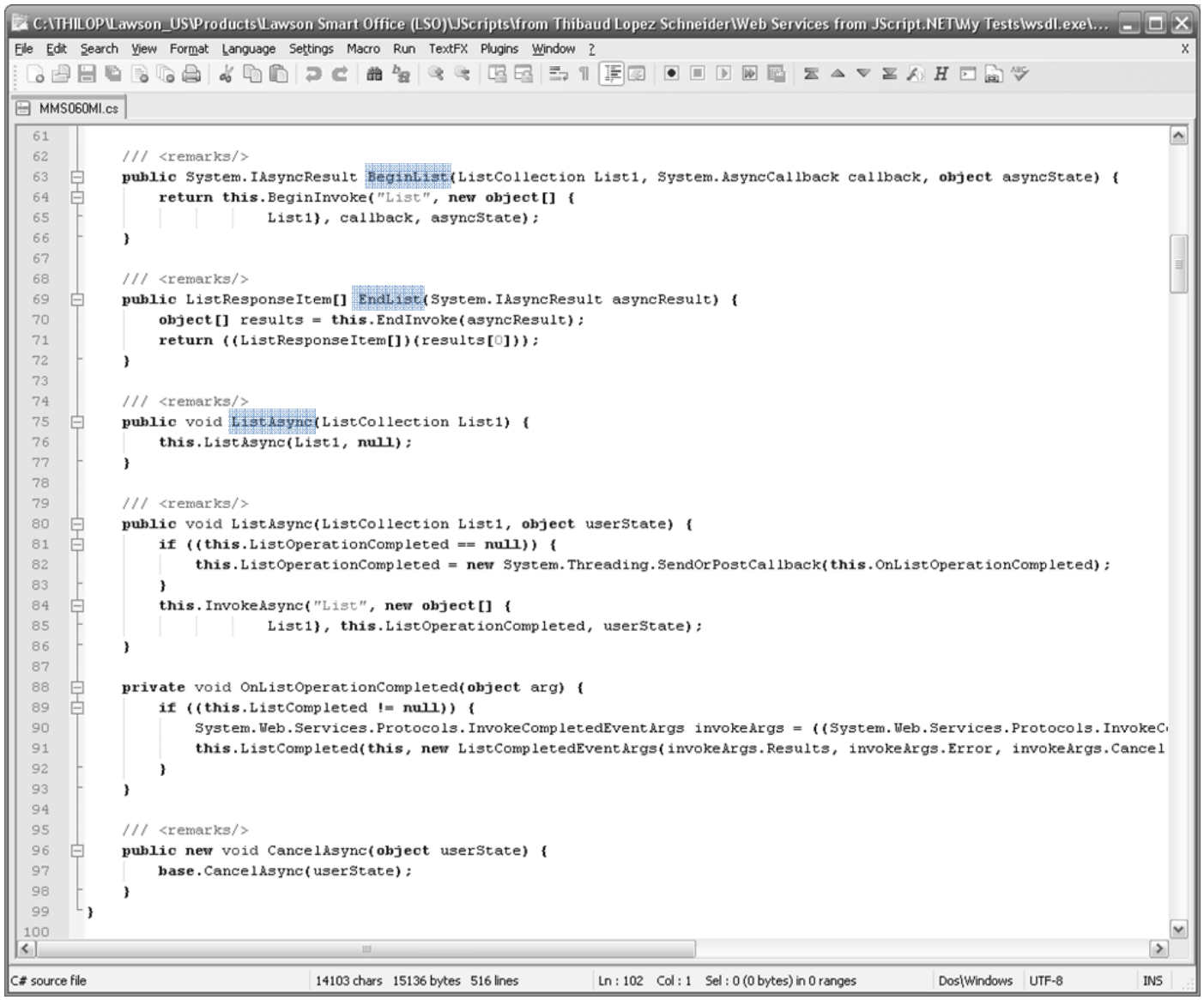
```
// create a proxy
var proxy = assembly.CreateInstance("MMS060MI");

// SOAP Request Header (user and password)
var header = assembly.CreateInstance("headerType");
header.user = "secret";
header.password = "secret";
proxy.mws = header;

// SOAP Request Body (input parameters)
var body = assembly.CreateInstance("ListItem");
body.Company = new Decimal(701);
body.Warehouse = "110";

var collection = assembly.CreateInstance("ListCollection");
collection.maxRecords = 100;
var ListItem = body.GetType();
collection.ListItem = new ListItem[1];
collection.ListItem[0] = body;

// call the web service
var response = proxy.List(collection);

// SOAP Response
debug.WriteLine(response.length + " records returned");
for (var i in response) {
    debug.WriteLine(
        response[i].Location + "=" +
        response[i].StatusBalanceID);
}
```

Input parameters

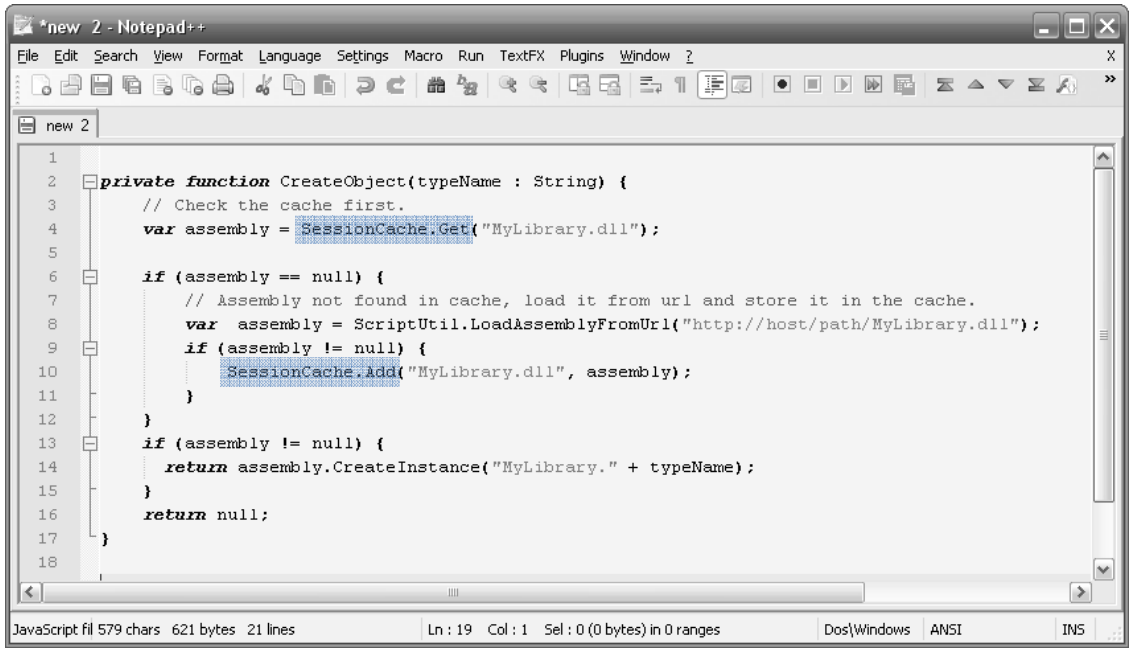
Output parameters



Note 1: the example I wrote makes a synchronous call, meaning Smart Office will freeze until it receives the response. That's bad. Instead, we should use asynchronous calls. The generated proxy class contains the methods ListAsync, BeginList, and EndList. I might provide an example some day.

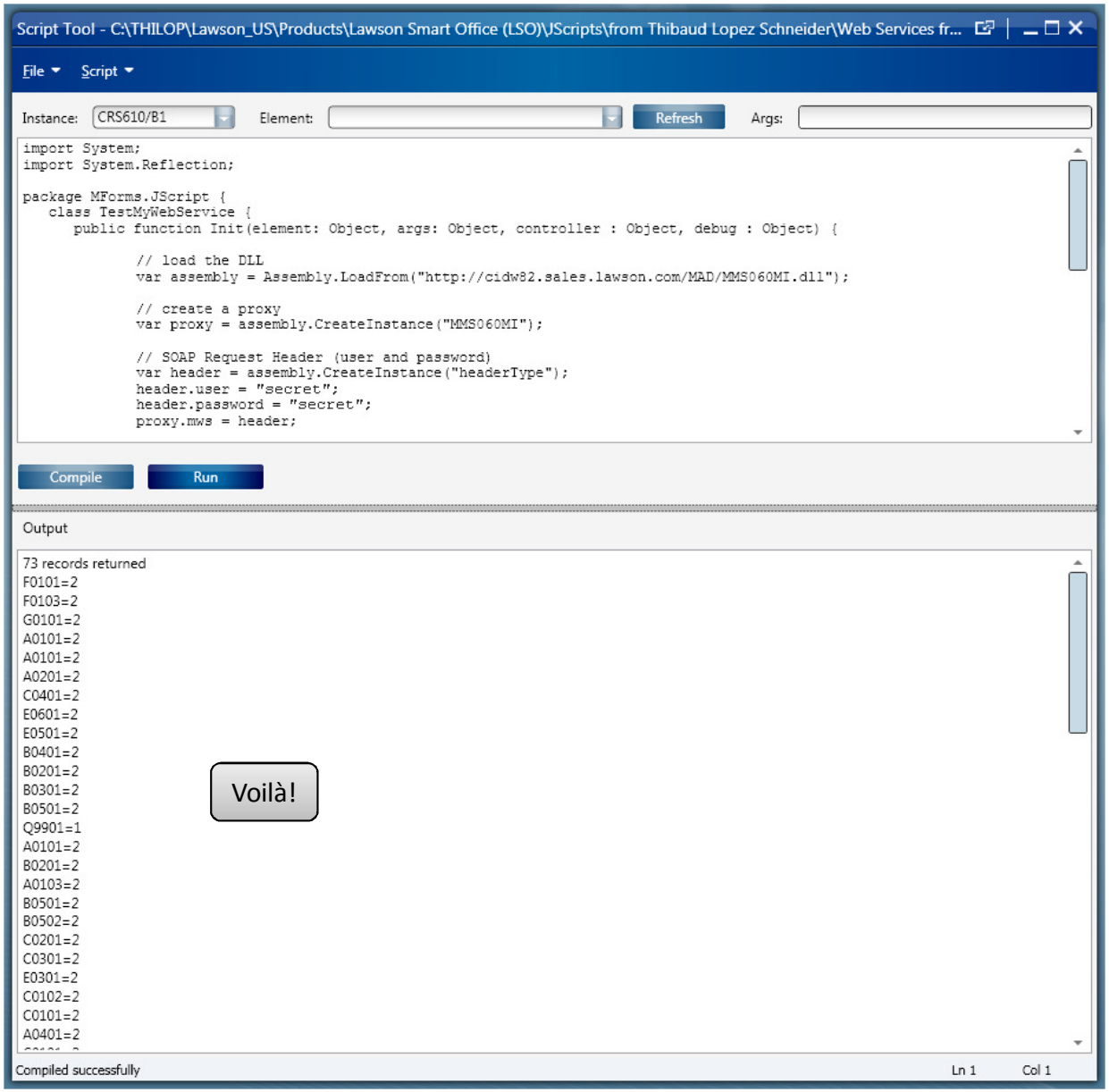


Note 2: Also, Peter A Johansson recommends storing the DLL in the Session cache so as to minimize network traffic.

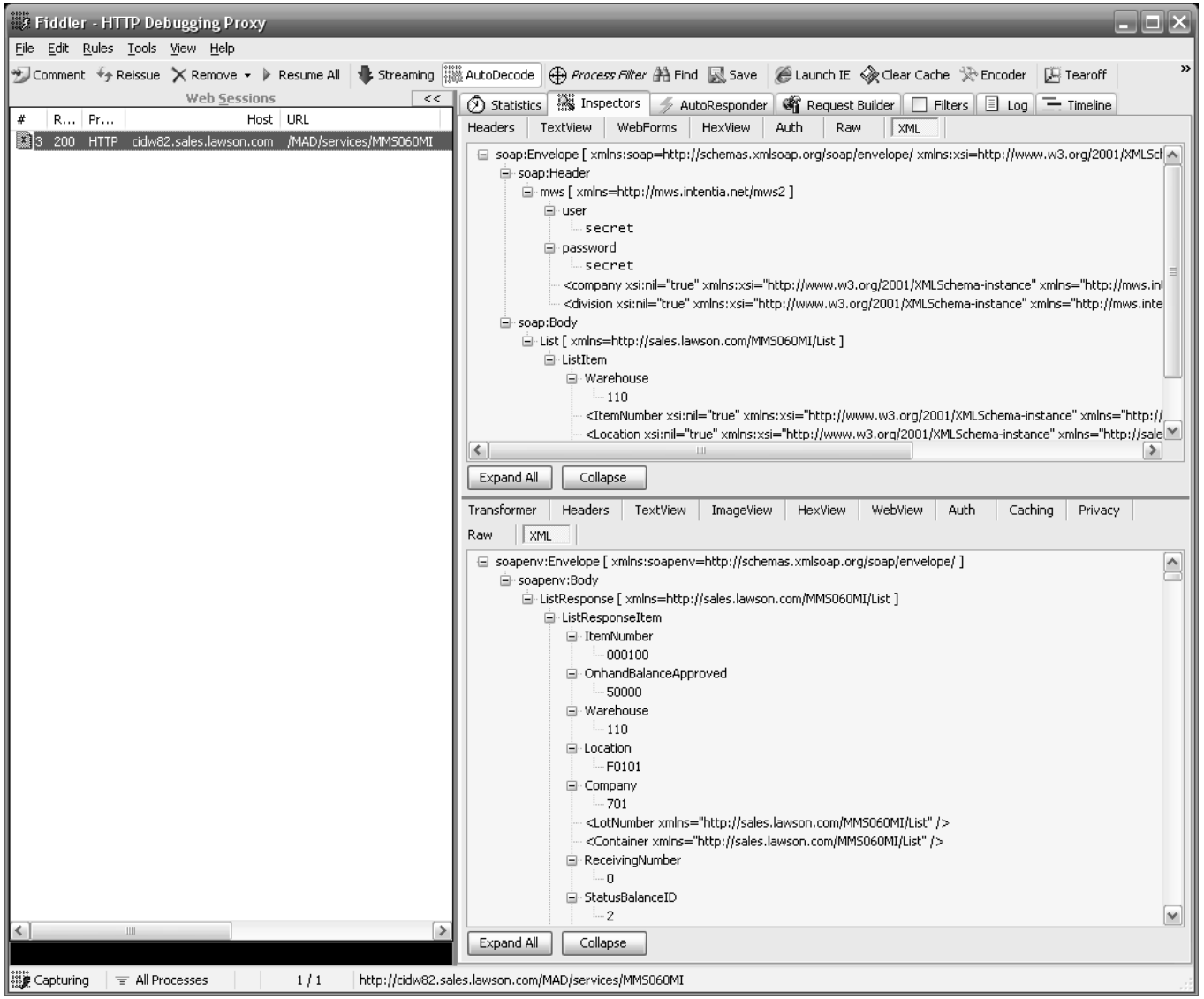


Note 3: Also, I recommend using exception handling in the code. It's not shown in my example.

Step 7. Compile and Run the script in Lawson Smart Office with the Script Tool (this example does not require any particular M3 Program Instance).



Optionally, you can check the SOAP Request in Fiddler.



	SOAP validation?	DLL free?	Notepad only?	JScript programming only?
"Big String"	No	Yes	Yes	Yes
Xml Writer	No	Yes	Yes	Yes
Proxy with Visual Studio C#	Yes	No	No	No
wsdl.exe *	Yes	No	No	Yes

Summary: This paper presented a new solution to call a Lawson Web Service from a Personalized Script in JScript.NET in Lawson Smart Office. This solution complements the other three known solutions. Now the developer has even more flexibility to choose a solution that best suit its needs. This fourth solution provides SOAP validation, and doesn't require C# programming, nor Microsoft Visual Studio. But it requires to install the free Microsoft .NET SDK, it requires following the proxy syntax, and produces a DLL that must be published to the users. *(*)* Peter A Johansson and myself recommend this fourth solution over the other three. Peter adds: "The first three are no longer 'best practice' […] are 'old and obsolete' and should not be used."

Discussion

From an architecture point of view, there are good and bad reasons to call a Lawson Web Service from a Personalized Script.

For example, one could call a Lawson Web Service from a standalone script that is executing in its own window in Smart Office, i.e not in an M3 Panel. In that case, using Lawson Web Services would be legitimate as there are no other alternatives (except direct SQL to M3 which is not recommended for writing to M3). Here, Lawson Web Services would have the same role as the IDSP connector had for IBrix.

However, when it comes to Personalized Scripts that live and run inside of an M3 Panel, it's better to access the data from within M3 itself, close to the source, i.e. from the M3 Business Engine with some M3 Java code in MAK, even though it means an M3 modification. There are several technical reasons why: performance, security, dependencies, maintenance, etc.

Special thanks to Peter A Johansson for the code of the DLL solution, and for the review of this paper.

Thibaud Lopez Schneider
thibaud.lopez.schneider@us.lawson.com