Using Microsoft Visual Studio .NET with Avid Interplay WS

Revision 1.3 – December 5, 2007

Overview **2**
Microsoft Visual Studio .NET Versions 2
General Conventions Used in this Document 2
Locating the WSDL and XSD files 2

Creating a Visual Studio .NET Client Step-by-Step **3**
Create a New Console Project 3
Add a Web Reference to the Interplay WS Assets Service 4
Write Code to Use the Interplay WS Web Reference 5

Advanced Topics **6**
Enabling MTOM Support (Visual Studio 2005) 6
Changing the Endpoint URL 8
Enabling Secure Transmission over HTTPS 9
Changing the Timeout Length for SOAP Requests 11
Logging Outgoing and Incoming Message Traffic 12

Overview

The Interplay WS API provides SOAP web services for interacting with Avid Interplay. The first release provides operations for exchanging and modifying metadata in the Avid Interplay Engine. This document demonstrates how to create a C# client for Interplay WS using Microsoft Visual Studio .NET.

The Interplay WS distribution also comes with *screencast* demonstrating how to use Visual Studio with Interplay WS. You may want to watch the screencast for a more visual and dynamic demonstration.

Microsoft Visual Studio .NET Versions

You can create a client for Interplay WS using any version of Visual Studio that supports the .NET framework. This document was written using Visual Studio 2005 with .NET 2.0, but we have noted where differences may occur for Visual Studio .NET 2003.

General Conventions Used in this Document

Server name and port
The server name and port of the Interplay WS services will be different for each site. For the purposes of this documentation, we will be using the *localhost* server and port *80* (the default port). You should modify your work to reflect the actual hostname and port of your Interplay WS installation.

For simplicity, this documentation will mainly use *http*. Instructions for using *https* can be found near the end of this document. For security purposes, we recommend you use *https* in production environments.

Workgroup name and server mapping
The Interplay WS application must be configured with workgroup name and server mappings. This can be done via the Avid Interplay Framework Configuration client. For the purposes of this documentation, we will assume that the workgroup name *WGA* is mapped to our Interplay Engine server.

Locating the WSDL and XSD files

Since Interplay WS is a SOAP web service, it uses a Web Services Description Language (WSDL) file to describe its interface. This WSDL also references an XML Schema Document (XSD) to define the XML types used in the messages. In development, your SOAP toolkit will use the WSDL file to generate client code to access the service.

The WSDL for Interplay WS Assets can be found at:
http://iws-srv/services/Assets?wsdl
(Remember to change the server name and port to match your environment)

This WSDL will also reference the XSD, which can be found at:
http://iws-srv/services/Assets?xsd=assets.xsd

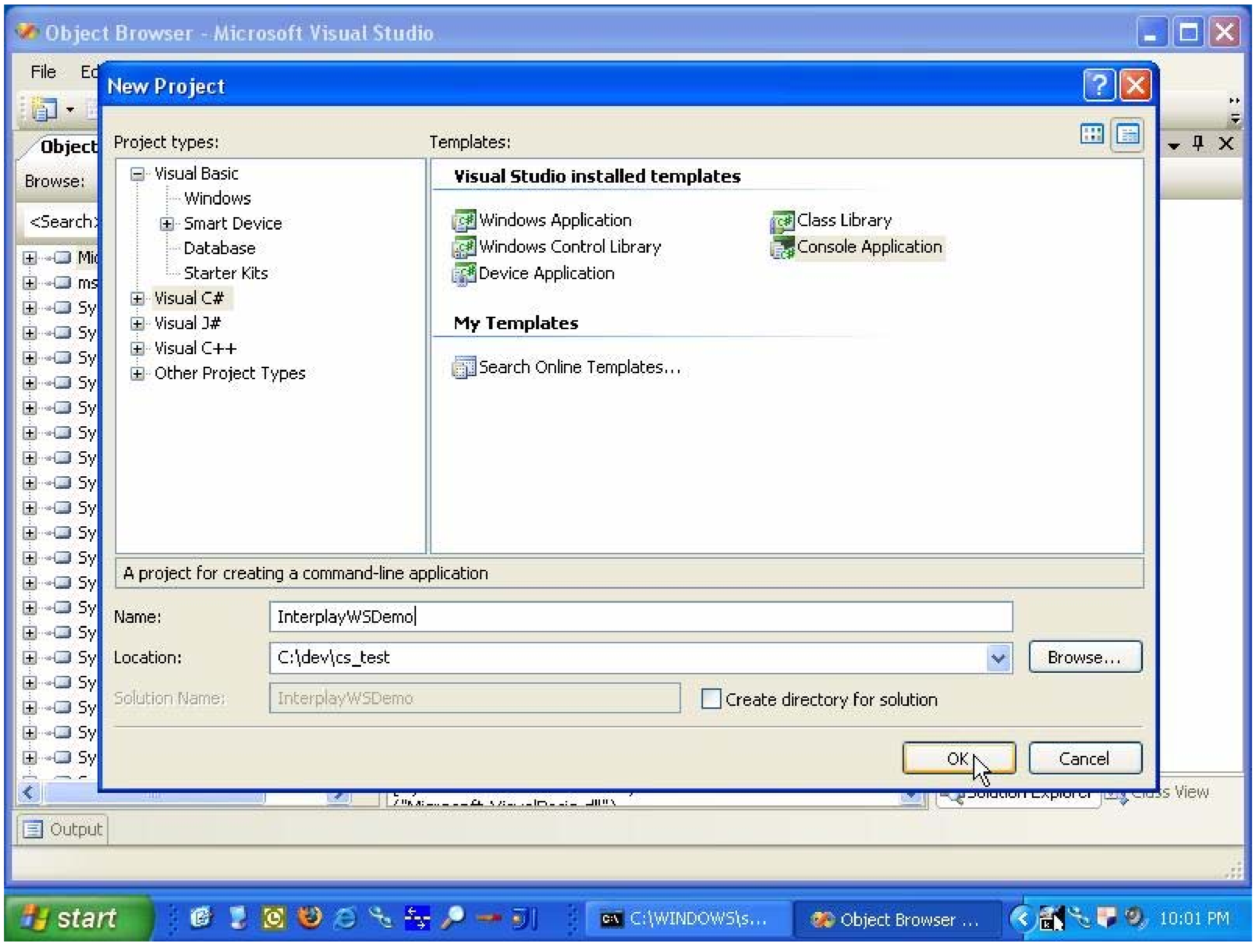
Creating a Visual Studio .NET Client Step-by-Step

For this tutorial, we will create a simple C# Console application that gets the children of the *Projects/Rainforest* folder.

Create a New Console Project

1. Create a new project by choosing *File -> New -> Project…*

2. Choose a Visual C# Console Application, and enter the name and location of the project.



A project for creating a command-line application

Name: InterplayWSDemo

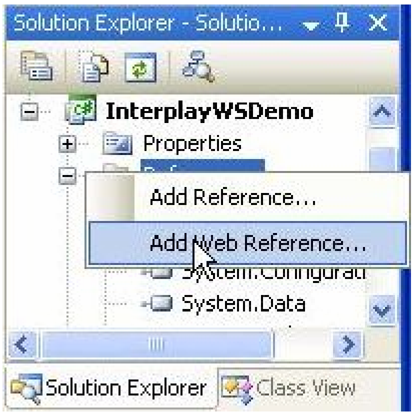
Location: C:\dev\cs_test

Solution Name: InterplayWSDemo

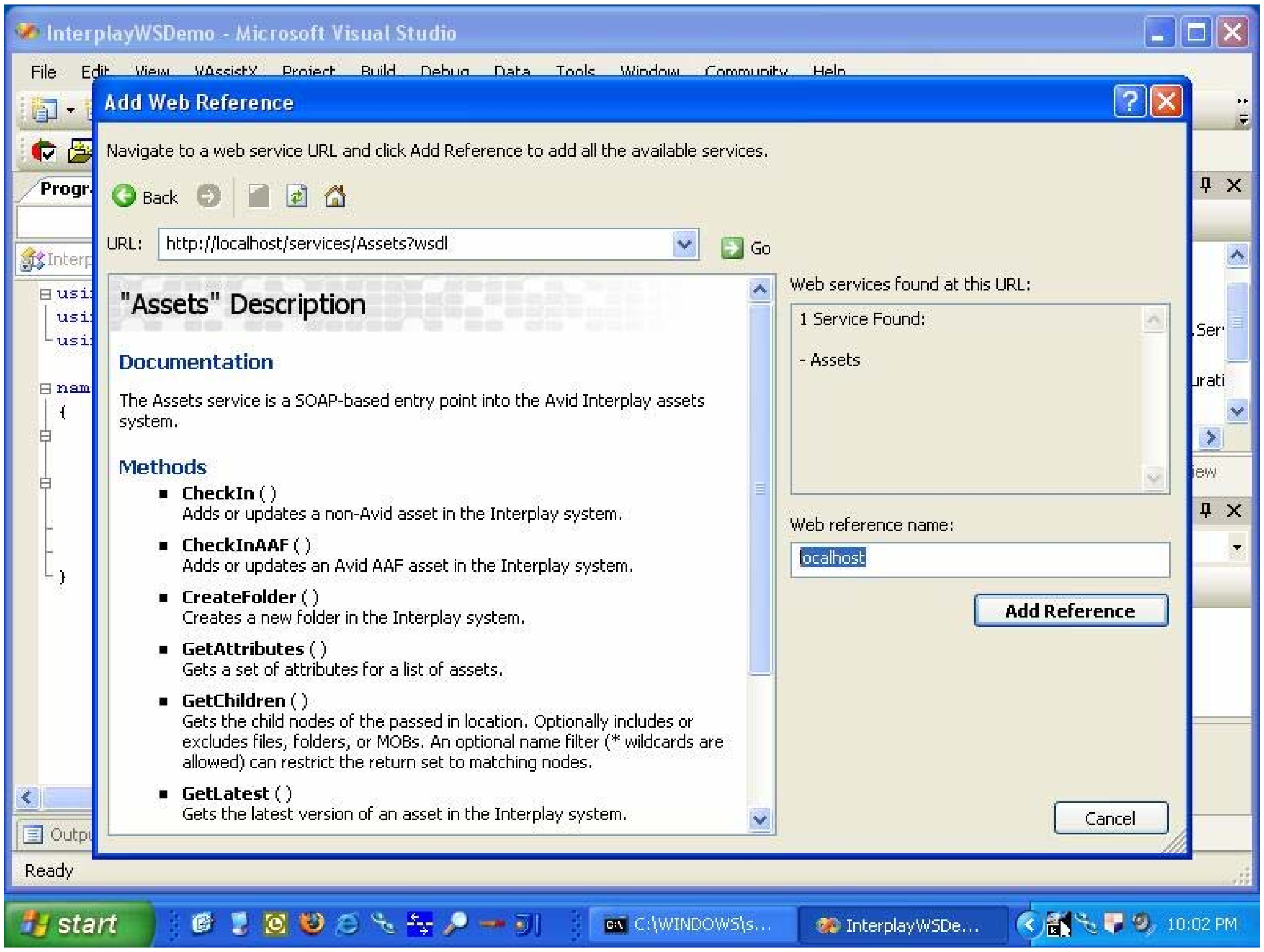
Create directory for solution

Add a Web Reference to the Interplay WS Assets Service

1. Right-click on *References* in the Solution Explorer and choose *Add Web Reference…*



2. Type in the URL to the WSDL file (see *Locating the WSDL and XSD File* above) and click *Go*.



Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://localhost/services/Assets?wsdl

"Assets" Description

Documentation

The Assets service is a SOAP-based entry point into the Avid Interplay assets system.

Methods

- **CheckIn** ( )
  Adds or updates a non-Avid asset in the Interplay system.
- **CheckInAAF** ( )
  Adds or updates an Avid AAF asset in the Interplay system.
- **CreateFolder** ( )
  Creates a new folder in the Interplay system.
- **GetAttributes** ( )
  Gets a set of attributes for a list of assets.
- **GetChildren** ( )
  Gets the child nodes of the passed in location. Optionally includes or excludes files, folders, or MOBs. An optional name filter (* wildcards are allowed) can restrict the return set to matching nodes.
- **GetLatest** ( )
  Gets the latest version of an asset in the Interplay system.

Web services found at this URL:

1 Service Found:

- Assets

Web reference name: localhost

3. Change the *Web reference name* to the name you wish to use for referencing the Interplay WS Assets service and then click *Add Reference*. For these examples, we will use *InterplayWS*.

Write Code to Use the Interplay WS Web Reference

1. Add a *using* statement to the C# file so we use the new web reference namespace. Note that this will differ depending on your project name and the name you gave the web reference.

```
using InterplayWSDemo.InterplayWS;
```

2. Get an implementation of the *Assets* port. For this example, we are writing this and all other code in the *Main* method of our program.

```
Assets port = new Assets();
```

3. Set the *UserCredentials* used by the *Assets* port.

```
port.UserCredentials = new UserCredentialsType();
port.UserCredentials.Username = "jsmith";
port.UserCredentials.Password = "secret";
```

4. Create and populate the parameters for the *GetChildren* operation. See the Interplay WS Reference Guide for more information on Interplay URIs.

```
GetChildrenType param = new GetChildrenType();
param.InterplayURI = "interplay://WGA/Projects/Rainforest";
param.IncludeFiles = true; // Note that bools default to false if not specified
param.IncludeFolders = true;
param.IncludeMOBs = true;
```

5. Execute the *GetChildren* operation via the *Assets* port.

```
GetChildrenResponseType response = port.GetChildren(param);
```

6. Handle the results. For this example, we'll just output the Interplay URI and the returned attributes. *See the Interplay WS Reference Guide for more info on attributes.*

```
foreach (AssetDescriptionType ad in response.Results) {
    Console.WriteLine(ad.InterplayURI);
    foreach (AttributeType att in ad.Attributes) {
        Console.WriteLine(att.Group + "." + att.Name + " = " + att.Value);
    }
    Console.WriteLine();
}
```

7. Save and run the program. You should see the results in a console window.

Note: If an optional parameter is mapped to a primitive, Visual Studio uses a special boolean to indicate if the parameter is specified. For example, it generates *MaxResults* and *MaxResultsSpecified* properties for SearchType. If *MaxResultsSpecified* is *false*, it will not include a *MaxResults* element in the request.

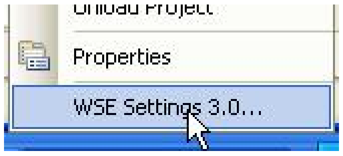
Advanced Topics

Enabling MTOM Support (Visual Studio 2005)

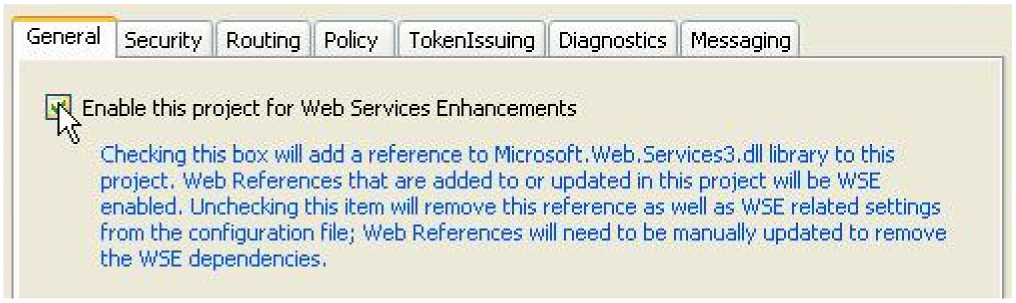
MTOM is the recommended best practice for efficiently transporting binary files via SOAP. Visual Studio .NET 2003 does *not* support MTOM. Visual Studio 2005 supports MTOM via the Web Services Enhancements 3.0 add-on. If you wish to use MTOM (highly recommended), then follow the instructions below.

1. Download and install the Web Services Enhancements 3.0 add-on via the following URL: http://msdn2.microsoft.com/en-us/webservices/Aa740663.aspx

2. Open your Interplay WS client project and right-click the project in Solutions Explorer and choose *WSE Settings 3.0…*



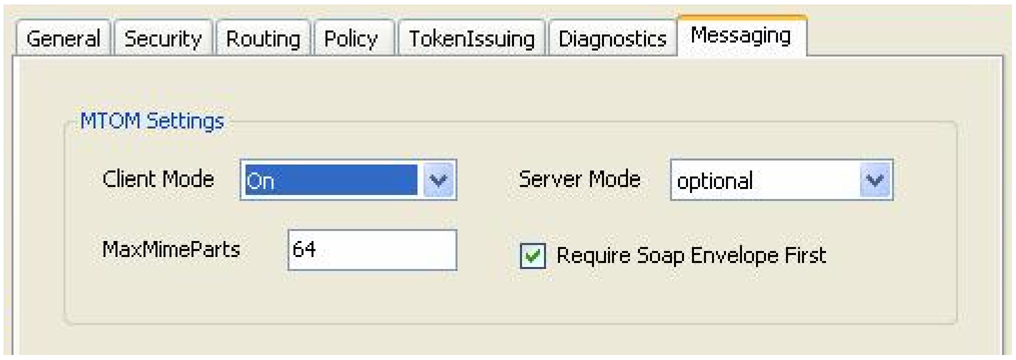
3. In the resulting dialog, check *Enable this project for Web Services Enhancements*.



Enable this project for Web Services Enhancements

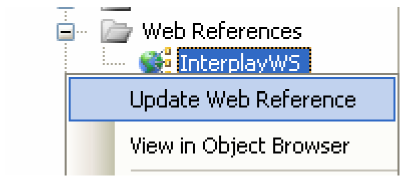
Checking this box will add a reference to Microsoft.Web.Services3.dll library to this project. Web References that are added to or updated in this project will be WSE enabled. Unchecking this item will remove this reference as well as WSE related settings from the configuration file; Web References will need to be manually updated to remove the WSE dependencies.

4. Select the *Messaging* tab and turn on *Client Mode* under *MTOM Settings*. Then click OK.



Client Mode: On
Server Mode: optional
MaxMimeParts: 64
✓ Require Soap Envelope First

5. Update the Web Reference by right-clicking it in Solution Explorer and choosing *Update Web Reference*



6. In your custom code, you must now specifically use the WSE enabled *AssetsWse* port.

```
AssetsWse port = new AssetsWse();
```

7. If you wish to disable MTOM temporarily, you can do so via code.

```
port.RequireMtom = false;   // Do this to disable MTOM
```

Changing the Endpoint URL

In some cases, you may want to change the endpoint that your client code uses. By default, the code will use whichever endpoint URL was specified in the WSDL to create the service. To make your client communicate with an Interplay WS service on a different host, change the endpoint URL.

```
port.Url = "http://iws-srv2/services/Assets";
```

Another common reason to change the endpoint is for debugging purposes. You can use a tool such as TCPMon to create a relay to the Interplay WS host. You would then configure your client code to send requests to TCPMon on your local machine. TCPMon could then intercept the sent and received messages so you can view their contents. If TCPMon is listening on port 81 on your local machine, your code should look like this:

```
port.Url = "http://localhost:81/services/Assets";
```

Enabling Secure Transmission over HTTPS

The Interplay WS service also allows for encrypted transmissions via HTTPS. The Interplay WS server uses a *self-signed certificate*, so Visual Studio .NET will not trust it by default. There are several ways to override this to allow .NET to trust the self-signed certificate.

Trusting a Self-Signed Certificate with Visual Studio 2005

The easiest way to configure .NET 2.0 and .NET 3.0 to trust the Interplay WS HTTPS service is by overriding the ServerCertificateValidationCallback.

The following steps show an example of how to do this in your Interplay WS client:

1. Add a delegate for the ServerCertificateValidationCallback before any calls to Interplay WS are made.*

```
ServicePointManager.ServerCertificateValidationCallback += delegate(
    Object obj,
    System.Security.Cryptography.X509Certificates.X509Certificate certificate,
    System.Security.Cryptography.X509Certificates.X509Chain chain,
    System.Net.Security.SslPolicyErrors errors)
{
    // Do any certificate validation logic here.
    // This example checks to make sure that the Organizational Unit is “Interplay”
    return certificate.Subject.Contains("OU=Interplay");
};
```

2. Change your *Assets* port to use HTTPS instead of HTTP.

```
port.Url = "https://iws-srv/services/Assets";
```

** System.Net.ServicePointManager.ServerCertificateValidationCallback is a system setting. Setting it could affect other Certificate aware code in your program.*

Trusting a Self-Signed Certificate with Visual Studio .NET 2003

The easiest way to configure .NET 1.0 to trust the Interplay WS HTTPS service is by setting up a custom Certificate Policy.

The following steps show an example of how to do this in your Interplay WS client:

1. Create a custom *InterplayWsTrustingCertificatePolicy*.

```
public class InterplayWsTrustingCertificatePolicy : System.Net.ICertificatePolicy {
   public InterplayWsTrustingCertificatePolicy() { }

   public bool CheckValidationResult(
         System.Net.ServicePoint sp,
         System.Security.Cryptography.X509Certificates.X509Certificate certificate,
         System.Net.WebRequest request,
         int problem)
   {
      // Do any certificate validation logic here.
      // This example checks to make sure that the Organizational Unit is “Interplay”
      return certificate.GetName().IndexOf("OU=Interplay") != -1;
   }
}
```

2. Set the System.Net.ServicePointManager.CertificatePolicy to your custom certificate policy. This should be done before any calls to Interplay WS are made and only needs to be called once.*

```
System.Net.ServicePointManager.CertificatePolicy = new InterplayWsTrustingCertificatePolicy();
```

3. Change your *Assets* port to use HTTPS instead of HTTP.

```
port.Url = "https://iws-srv/services/Assets";
```

** System.Net.ServicePointManager.CertificatePolicy is a system setting. Setting it could affect other Certificate aware code in your program.*

Changing the Timeout Length for SOAP Requests

Some queries (particularly broad search queries) may take some time before they return with a response. By default, Microsoft .NET clients will timeout after 100 seconds. You can configure your client to timeout after a different length of time by setting the *Timeout* property on the generated *Assets* port. Timeout values are measured in milliseconds.

```
port.Timeout = 5*60*1000; // Time out after 5 minutes (5 min * 60 sec * 1000 ms)
```

If a timeout does occur in your client, a System.Net.WebException will be thrown with a status of System.Net.WebExceptionStatus.Timeout.

If you do not wish for timeouts to ever occur, you can also set your client to never timeout:

```
port.Timeout = System.Threading.Timeout.Infinite;  // Never time out
```

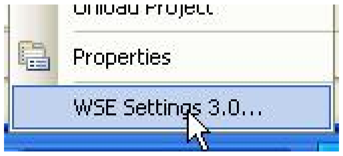
Logging Outgoing and Incoming Message Traffic

Visual Studio 2005 can log incoming and outgoing traffic via the Web Services Enhancements 3.0 add-on. This is not available for Visual Studio .NET 2003 users, but they should be able to log messages via a tool such as TCPMon, as discussed in the section *Changing the Endpoint URL*.

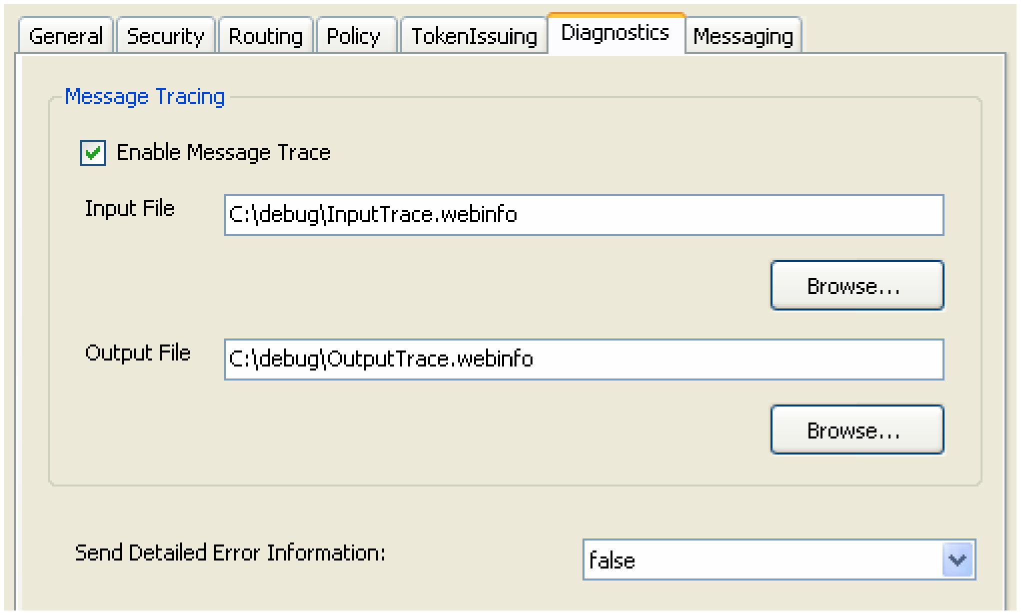
To log messages using Visual Studio 2005 Web Services Enhancements:

1. Download and install the Web Services Enhancements 3.0 add-on via the following URL: http://msdn2.microsoft.com/en-us/webservices/Aa740663.aspx

2. Open your Interplay WS client project and right-click the project in Solutions Explorer and choose *WSE Settings 3.0…*



3. In the resulting dialog, choose the *Diagnostics* tab, check *Enable Message Trace*, and configure where the logs should be written, and click OK.



Enable Message Trace [✓]

Input File: C:\debug\InputTrace.webinfo

Output File: C:\debug\OutputTrace.webinfo

Send Detailed Error Information: false

4. In your custom code, you must now specifically use the WSE enabled *AssetsWse* port.

```
AssetsWse port = new AssetsWse();
```