Creating a .NET Client for a SAP Java Web Service Using Visual Studio .NET 2003, WSE 2.0, and WS-Security

Release 640

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Table of contents

Creating a .NET Client Without Security

Task

This tutorial takes you through the development steps required for developing a .NET Web service client with no security implications.

Objectives

By the end of this tutorial, you will be able to:

✓ Create a simple .NET Client for an existing Web service.

Prerequisites

☐ *QuickCarRental* application with Web services is installed on your J2EE Engine.

Systems, Installations, and Authorizations

☐ You have installed the J2EE Engine.

☐ You can access the J2EE Engine using the Visual Administrator.

☐ You have installed Microsoft Visual Studio .NET

Knowledge

☐ You can create and deploy a Web service based on a session bean. For more information, see Web Service Toolset.

☐ You have basic knowledge in Microsoft Visual Studio .NET 2003 and the C# programming language.

Next Step:

Creating a Microsoft Visual Studio Project

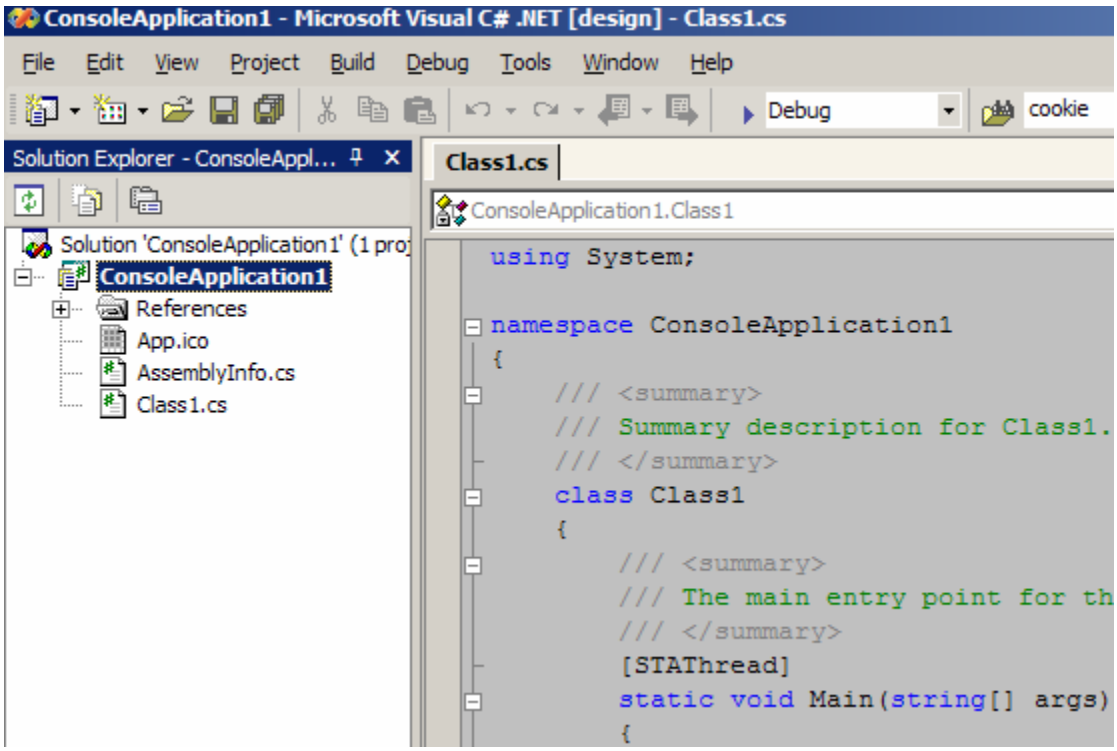
Creating a Microsoft Visual Studio Project

Procedure

1. Start the Microsoft Visual Studio.
   Make sure your J2EE Engine is running.
2. Choose *File* → *New* → *Project*.
3. In the *New Projects* dialog box that appears, select *Visual C# Projects* on the *Project Types* pane and *Console Application* on the *Templates* pane.
4. Enter a name and location for your project.
5. Choose *OK*.

Result

Once you have created the project into the Microsoft Visual Studio, the following structure is displayed:



Next Step:

Creating a Web Service Proxy

Creating a Web Service Proxy

Use

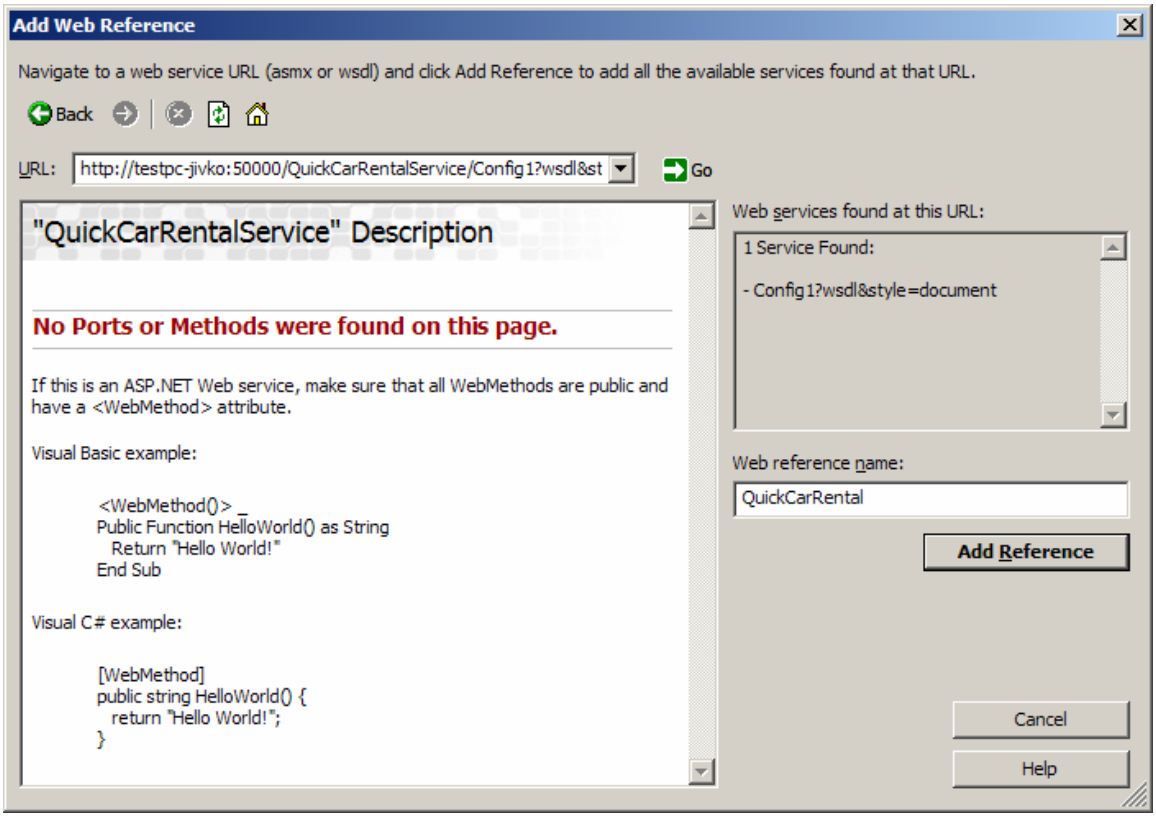
For Web service clients, the proxy class handles the work of mapping parameters to XML elements and then sending the SOAP message over the network. As long as a service description exists, a proxy class can be generated if the service description conforms to the Web Services Description Language (WSDL).

Prerequisites

Have the correct WSDL URL of the Web service.

Procedure

1. From the *Solution Explorer,* choose *References* and select *Add Web Reference* from the context menu.
2. In the *Add Web Reference* dialog box, enter the WSDL URL and choose *Go*.

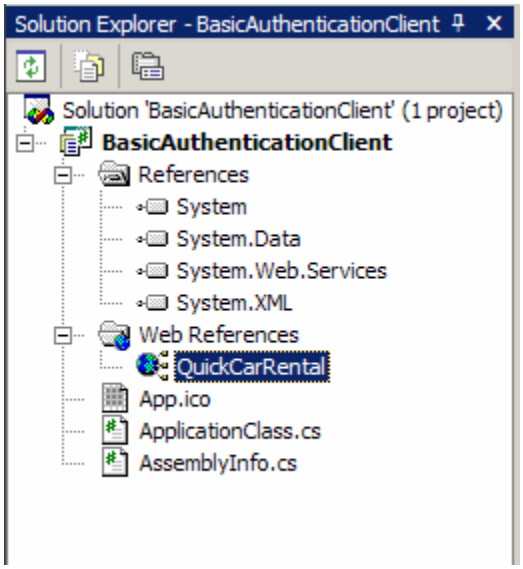


If the WSDL is correct and accessible, you will see the Web service description.

3. Enter a Web reference name and choose *Add Reference*.

Result

Once you have created the Web service proxy, you will see the following Web reference added in the *Solution Explorer*:



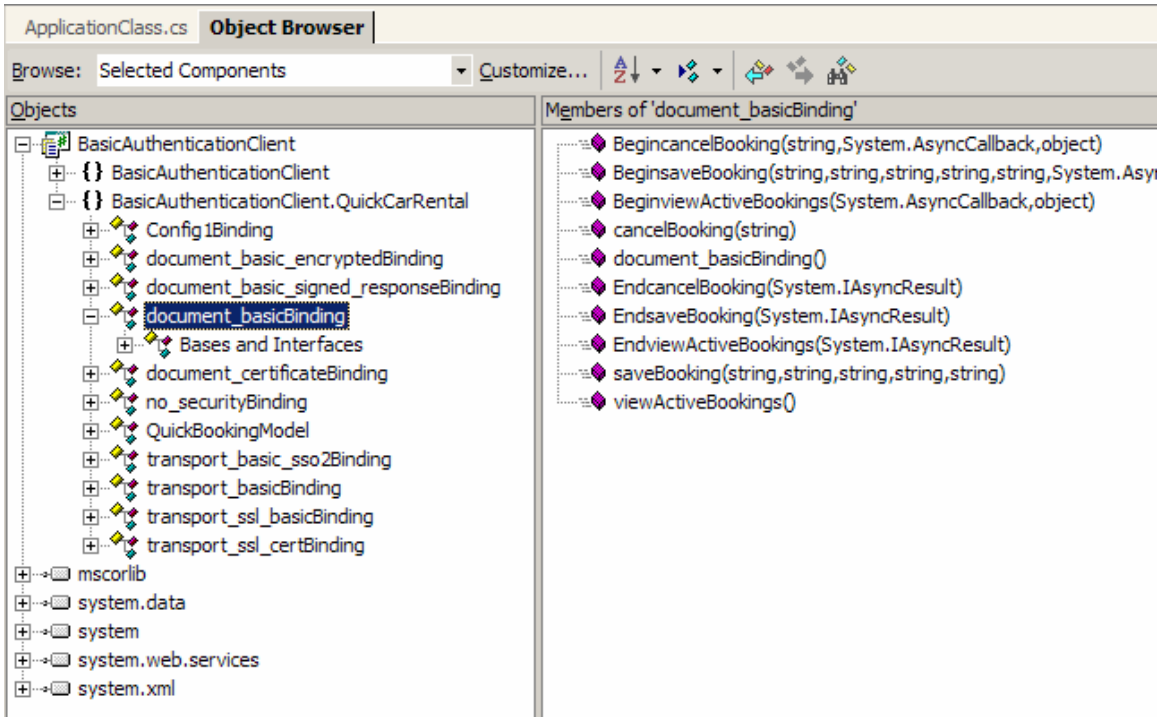
Next Step:

Using the Web Service Proxy to Call the Web Service

Using the Web Service Proxy to Call the Web Service

Use

The Web service proxy generation creates a C# source code file that contains the Web service proxy class in a separate namespace. You can view the class methods and/or open the source code using the *Object Browser*.



Procedure

1. Open the source file for the class that contains your `Main` method and add the Web service namespace at the top so that you can use the Web service proxy class definition.

   You will find the appropriate namespace in the generated class of the Web service proxy.

   ```
   using System;
   using WebServiceClient.QuickCarRental;
   ```

2. Create a new Web service proxy and call its methods.

   You will find the name of the Web service proxy in the generated source code file for the proxy.

```
[STAThread]
static void Main(string[] args)
{
    document_basicBinding proxy = new document_basicBinding();

    try
    {
        QuickBookingModel[] Bookings = proxy.viewActiveBookings();

        QuickBookingModel NewModel = proxy.saveBooking("Truck",
                                                       DateTime.Now.ToString("dd.MM.yyyy"),
                                                       DateTime.Now.ToString("dd.MM.yyyy"),
                                                       "Sofia",
                                                       "Walldorf");

        proxy.cancelBooking(NewModel.bookingId);
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    Console.WriteLine("Press any key to continue...");
    Console.Read();
}
```

3. Build and run the application.

Result

You may add print out code to see the returned data from the Web service.

Creating a .NET Client with a User Name Token

Task

To authenticate against a Web service with a user name and password, you have to use Web Service Enhancements 2.0 (WSE 2.0) to attach user credentials in a standardized way. The WSE uses security tokens internally to represent security claims from Web service methods. For example, a `UsernameToken` is used to attach information about the user to the SOAP message. With this token, the WSE 2.0 can authenticate the user, validate the password, and check if the user has all rights to execute the Web service method.

Objectives

By the end of this tutorial, you will be able to:

- ✓ Create a simple .NET client for an existing Web service that will authenticate with username and password.

Prerequisites

- ☐ You have WSE 2.0 installed.
- ☐ You have built the Web service proxy using Creating a .NET Client Without Security

Systems, Installations, and Authorizations

- ☐ You have installed the J2EE Engine.
- ☐ You can access the J2EE Engine using the Visual Administrator.
- ☐ You have installed Microsoft Visual Studio .NET

Knowledge

- ☐ You can create and deploy a Web service based on a session bean. For more information, see Web Service Toolset.
- ☐ You have basic knowledge in Microsoft Visual Studio .NET and C#.

Next Step:

Creating a Client with Basic Authentication

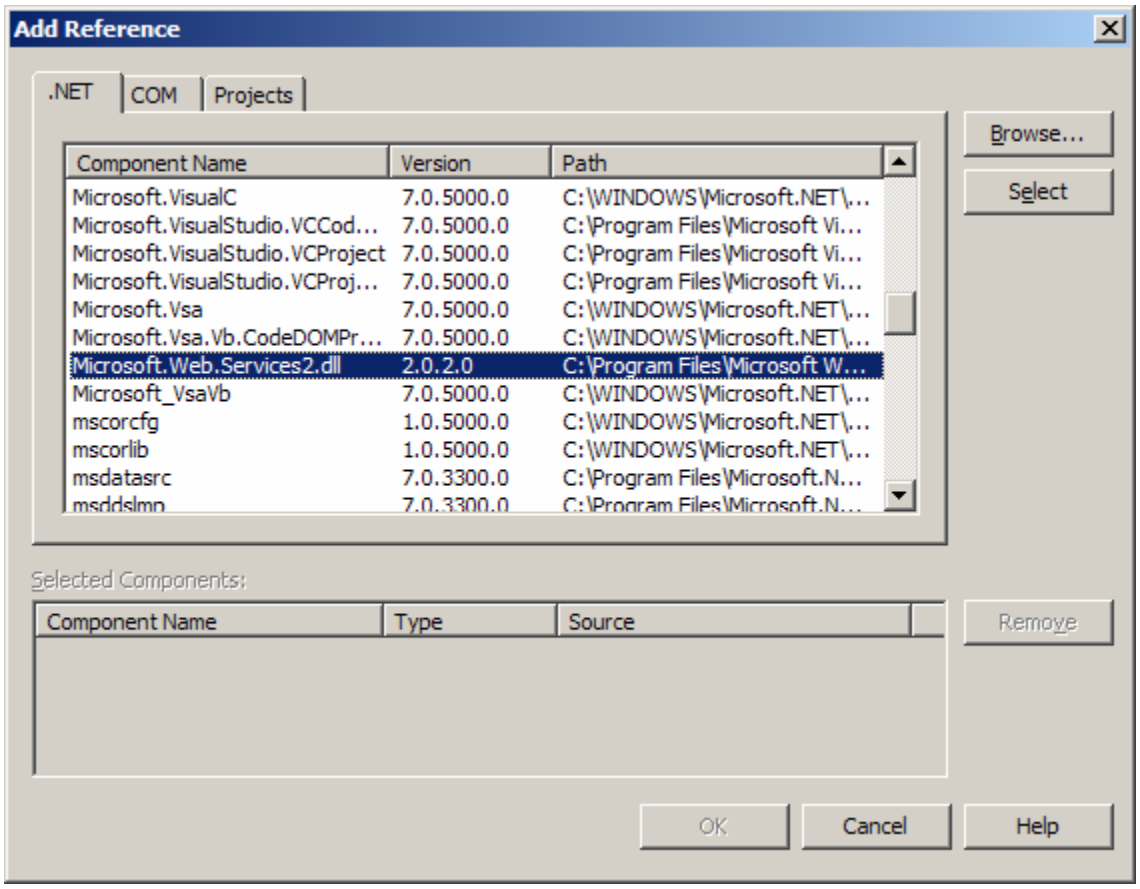
Creating a Client with Basic Authentication

Use

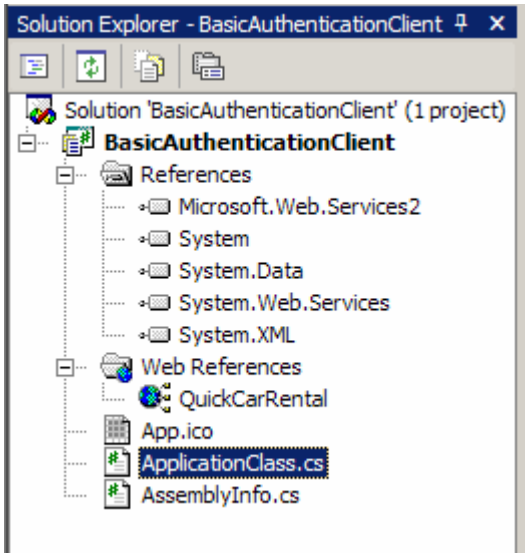
To use WSE 2.0 functionality on the client side, you have to add a reference to the assembly `Microsoft.Web.Services` file, which is installed in the global assembly cache (GAC) of the local physical machine. Furthermore, you have to change the base class of the Web service proxy to `Microsoft.Web.Services.WebServicesClientProtocol`. In this way, the client has more properties (such as `SoapContext`) available for manipulating the SOAP message.

Adding a Reference

1. From the *Solution Explorer,* choose *References* and select *Add Reference* from the context menu.
2. On the *Add Reference* dialog box that appears, select the *.NET* tab and then the *Microsoft.Web.Services2.dll* component.



3. Choose *Select* on the right and then choose *OK*.
   You can see the new reference added in the *Solution Explorer*.



Switching to WSE Proxy Class

1. Open the proxy class generated from the WSDL.
2. Change the base class for each binding of your proxy from `System.Web.Services.Protocols.SoapHttpClientProtocol` to `Microsoft.Web.Services2.WebServicesClientProtocol`
3. Save the file.

Adding the Code for Authentication with a User Name Token

1. Open the main source file.
2. Import the new WSE namespace and the security namespaces.

   using Microsoft.Web.Services2;

   using Microsoft.Web.Services2.Security;

   using Microsoft.Web.Services2.Security.Tokens;
3. Create a proxy from the basic authentication binding.

```
document_basicBinding proxy = new document_basicBinding();
```

4. Create a `UsernameToken`, set its properties and add it to the security tokens collection of the request.

```
UsernameToken token =
    new UsernameToken( "Administrator", "sapsap", PasswordOption.SendPlainText );

proxy.RequestSoapContext.Security.Tokens.Add(token);
```

5. Build and run the sample.

Result

You may add some print out code, to see the returned data from the Web service.

```
XML created by the proxy:

<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <soap:Header>
            <wsa:Action></wsa:Action>
            <wsa:MessageID>uuid:7981b457-7ffb-4c2a-9951-
      95958a71c2a4</wsa:MessageID>
            <wsa:ReplyTo>
                  <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/ad
            dressing/role/anonymous</wsa:Address>
            </wsa:ReplyTo>
            <wsa:To>http://localhost:55000/QuickCarRentalService/Config
      1?wsdl&amp;style=document
            </wsa:To>
            <wsse:Security soap:mustUnderstand="1">
                  <wsu:Timestamp wsu:Id="Timestamp-6c029b91-3f6a-42b9-
                  bbc1-d3ec1056d457">
                        <wsu:Created>2005-05-19T08:34:36Z</wsu:Created>
                        <wsu:Expires>2005-05-19T08:39:36Z</wsu:Expires>
                  </wsu:Timestamp>
                  <wsse:UsernameToken xmlns:wsu="http://docs.oasis-
            open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
            1.0.xsd" wsu:Id="SecurityToken-6b2dab2b-6f44-4bd8-a1e0-
            0c87ee20d0cc">
                        <wsse:Username>Administrator</wsse:Username>
                        <wsse:Password Type="http://docs.oasis-
                  open.org/wss/2004/01/oasis-200401-wss-username-token-
                  profile-1.0#PasswordText">sapsap</wsse:Password>
                        <wsse:Nonce>gJ1KqmPh4j4/OV55KqEeFw==</wsse:Nonc
                  e>
                        <wsu:Created>2005-05-19T08:34:36Z</wsu:Created>
                  </wsse:UsernameToken>
            </wsse:Security>
      </soap:Header>
<soap:Body>
      <viewActiveBookings xmlns="urn:QuickCarRentalServiceVi" />
</soap:Body>
</soap:Envelope>
```

Creating a Client with Document Encryption

Web Services Enhancements for Microsoft .NET (WSE) enables .NET Framework clients and Web services created using ASP.NET to encrypt and decrypt SOAP messages used to communicate with Web services. Encrypting and decrypting SOAP messages is a key feature for securing a Web application because the default SOAP messages are plain text and thus they can be read by any recipient or intermediary. An encrypted SOAP message is cryptographically encoded, so that only the owner of a private key or a symmetric key can read the contents of the message.

WSE supports both asymmetric and symmetric encryption. Asymmetric encryption allows a Web service client to encrypt the message using the public key of an X.509 certificate, such that only the owner of the private key of the X.509 certificate can decrypt the SOAP message. Symmetric encryption requires that a Web service and client share a secret key outside the SOAP message communication. Then, a client encrypts SOAP messages using that shared key and a Web service decrypts the SOAP messages using the same secret key.

Task

Use message level encryption on the request.

Objectives

By the end of this tutorial, you will be able to:

- ✓ Create a simple .NET Client for an existing Web service that will encrypt the message body with X.509 client certificate.

Prerequisites

- ☐ You have WSE 2.0 installed.
- ☐ You have built the Web service proxy using Creating a Client with Basic Authentication

Systems, Installations, and Authorizations

- ☐ You have installed the J2EE Engine.
- ☐ You can access the J2EE Engine using the Visual Administrator.
- ☐ You have installed Microsoft Visual Studio .NET

Knowledge

- ☐ You can create and deploy a Web service based on a session bean. For more information, see Web Service Toolset.
- ☐ You have basic knowledge in Microsoft Visual Studio .NET and C#

Next Step:

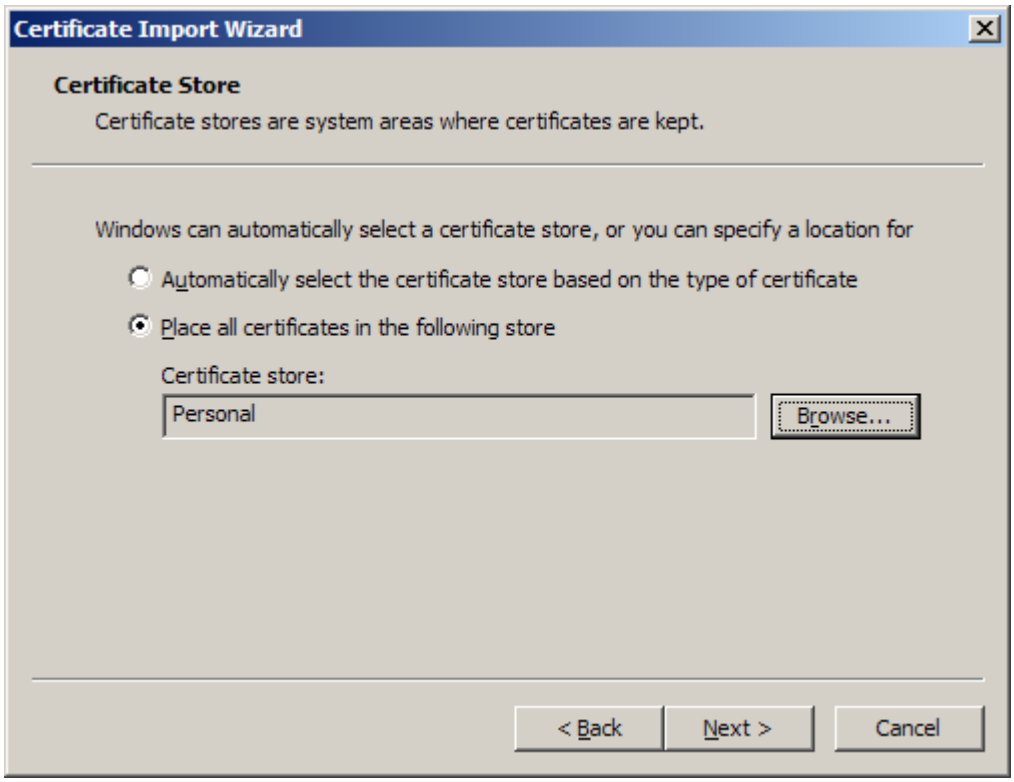
Creating a Client with Document Encryption

Creating a Client with Document Encryption

The WSE provides the functionality to encrypt both the request to the Web service and the response back to the client. When you use a X.509 certificate, the sender encrypts the message with the public key of the sender's X.509 certificate. Then the receiver uses his own private key of the X.509 certificate to decrypt the message. To encrypt a SOAP message, WSE 2.0 provides the class `EncryptedData`.

Installing X.509 Certificate on your Personal Store Using the Certificate Import Wizard

1. Locate your certificate file.
2. Select it and choose *Install* from the context menu.
   Use the *Certificate Import Wizard* to guide you through the import.
3. Select your *Personal Store* as certificate store.



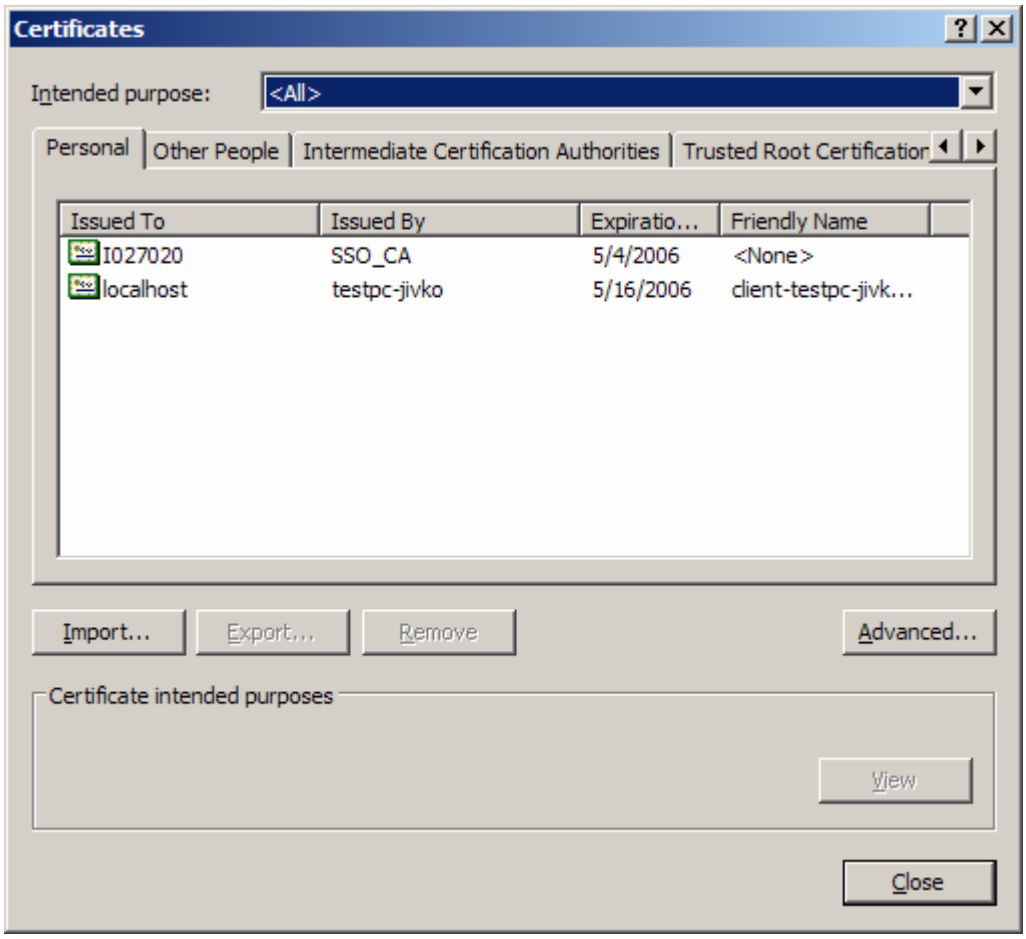
4. Finish the import.

Installing X.509 Certificate on your Personal Store Using the Command Line

Execute the following command to add your certificate to the current user personal store:

```
certmgr.exe -add -r LocalMachine -s My -c -n %CLIENT_NAME% -r
CurrentUser -s Personal
```

You can see the newly installed certificate in your *Personal Store* using the Internet Explorer browser. Go to *Internet Options* → *Content* → *Certificates* and select the *Personal* tab.



Use the X.509 Certificate to Encrypt the Message Data

1. Open your solution from *Creating a Client with Basic Authentication*
2. Add the following method to get the X.509 certificate.

```
static Microsoft.Web.Services2.Security.X509.X509Certificate GetCertificate(string CertificateName)
{
    Microsoft.Web.Services2.Security.X509.X509Certificate Certificate = null;

    X509CertificateStore CertStore =
        X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore);

    if (CertStore.OpenRead())
    {
        foreach (Microsoft.Web.Services2.Security.X509.X509Certificate cert in CertStore.Certificates)
        {
            Console.WriteLine(cert.GetName());
            if (CertificateName == cert.GetName())
            {
                Certificate = cert;
                break;
            }
        }

        //Microsoft.Web.Services2.Security.X509.X509CertificateCollection certs =
        //  CertStore.FindCertificateBySubjectName(certName);

        CertStore.Close();
    }

    return Certificate;
}
```

3. Add the following code to use the certificate for encrypting the whole body of the request.

```
Microsoft.Web.Services2.Security.X509.X509Certificate cert =
          GetCertificate("C=BG, O=***, CN=testpc-jivko");

if (cert == null)
{
 Console.WriteLine("Could not find the X.509 certificate.");
 return ;
}

if (!cert.SupportsDataEncryption)
{
 Console.WriteLine("Certificate does not support data
encryption");
 return ;
}

SecurityToken encryptionToken = new X509SecurityToken(cert);
EncryptedData enc = new EncryptedData(encryptionToken);
proxy.RequestSoapContext.Security.Elements.Add(enc);
proxy.RequestSoapContext.Security.Timestamp.TtlInSeconds = 60;
```

At the end, your `Main` method should look like this:

```
[STAThread]
static void Main(string[] args)
{
    document_basicBinding proxy = new document_basicBinding();

    UsernameToken token =
        new UsernameToken( "Administrator", "sapsap", PasswordOption.SendPlainText );

    proxy.RequestSoapContext.Security.Tokens.Add(token);


    Microsoft.Web.Services2.Security.X509.X509Certificate cert =
        GetCertificate("C=BG, O=***, CN=testpc-jivko");

    if (cert == null)
    {
        Console.WriteLine("Could not find the X.509 certificate.");
        return ;
    }

    if (!cert.SupportsDataEncryption)
    {
        Console.WriteLine("Certificate does not support data encryption");
        return ;
    }

    SecurityToken encryptionToken = new X509SecurityToken(cert);
    EncryptedData enc = new EncryptedData(encryptionToken);
    proxy.RequestSoapContext.Security.Elements.Add(enc);
    proxy.RequestSoapContext.Security.Timestamp.TtlInSeconds = 60;

    try
    {
        QuickBookingModel[] Bookings = proxy.viewActiveBookings();
```

5. Build and run the sample.

Result

You may add print out code to see the returned data from the Web service.

Creating a Client with a Document Signature

You can sign your SOAP message with an XML digital signature. The signature itself is unique for your message. If someone changes only a bit of your message then the signature changes and the receiver of your message knows that the message was changed during transport. The signature itself contains a hash, which is computed with the entire content of your SOAP message. When the message arrives at the other endpoint, then the receiver also computes a hash with the content of the incoming message. This hash is then compared with the hash contained in the sent SOAP message. If both hashes match, the receiver knows that no one else has changed the message during the transport.

Task

Use an XML digital signature to sign your SOAP message.

Objectives

By the end of this tutorial, you will be able to:

- ✓ Create a simple .NET client for an existing Web service that will sign the message body with X.509 certificate.

Prerequisites

- ☐ You have WSE 2.0 installed.
- ☐ You have set up the X.509 certificate as described in *Creating a Client with Document Encryption*

Systems, Installations, and Authorizations

- ☐ You have installed the J2EE Engine.
- ☐ You can access the J2EE Engine using the Visual Administrator.
- ☐ You have installed Microsoft Visual Studio .NET

Knowledge

- ☐ You can create and deploy a Web service based on a session bean. For more information, see Web Service Toolset.
- ☐ You have basic knowledge in Microsoft Visual Studio .NET and C#

Next Step:

Creating a Client with a Document Signature

Creating a Client with a Document Signature

Use

When you use an XML digital signature to sign your SOAP message, then the `<wsse:Security>` section of the SOAP header contains a new section called `<Signature>`. This new section contains all the information needed for the XML digital signature.

Procedure

1. Open your main source code file.
2. Add the following code to use XML signature:

```
Microsoft.Web.Services2.Security.X509.X509Certificate cert =
    GetCertificate("C=BG, O=***, CN=testpc-jivko");

if (cert == null)
{
    Console.WriteLine("Could not find the X.509 certificate.");
    return ;
}

if (!cert.SupportsDigitalSignature || cert.Key == null)
{
    Console.WriteLine("Certificate does not support digital sign
    return ;
}

SecurityToken signatureToken = new X509SecurityToken(cert);
proxy.RequestSoapContext.Security.Tokens.Add(signatureToken);

MessageSignature sig = new MessageSignature(signatureToken);
proxy.RequestSoapContext.Security.Elements.Add(sig);
proxy.RequestSoapContext.Security.Timestamp.TtlInSeconds = 60;
```

3. Build and run the sample.

Result

You may add print out code to see the returned data from the Web service.