

EMPLOYEESE EXTEND IMPLEMENTATION GUIDE



Version: 1.0
Date: 10/4/2007

Overview	3
Introduction	3
Audience	3
Document Status	3
Conventions.....	3
Building a Simple Web Service Call Using .NET	4
Tools.....	4
Set-Up	4
Create a new project	4
Add a reference to WSE 3.0	4
Add a web reference using WSDL	5
Building a simple web service request	7
Initializing the web proxy client object	7
Authentication	7
Building the request payload	8
Parsing the response payload.....	9
.NET Sample Code.....	10
Building a Simple Web Service Call Using Java	12
Tools.....	12
Set-Up	12
Create a new project	12
Add Apache Axis to your Project.....	12
Add Apache wss4j to your Project	12
Project Classpath	13
Generate Web Service Proxy using WSDL2Java.....	13
Setting up the classpath	13
Using WSDL2Java	13
Cleaning Up Errors.....	14
Building a simple web service request.....	14
Initializing the web proxy client object	14
Authentication	15
Building the request payload	16
Parsing the response payload.....	16
Java Sample Code	18
Applying Filters to Get Services	20
Common Filters.....	20
Employee Key Filter.....	20
Company Filter.....	20
Last Name Filter	21
Combination Filters.....	21
GetEmployeeStatus Example	21
GetEmployeeCorpGroup Example.....	21
Appendix.....	23
Related Documents	23
Example Payloads	23
GetEmployeeCorpGroup Request	23
GetEmployeeCorpGroup Response.....	24
HireEmployee Request.....	25
HireEmployee Response	26

Overview

Introduction

Employeease Extend is a collection of flexible web services that enable provisioning of data, consumption of data and execution of processes within the Employeease Network. This separates the presentation layer from the application layer so that organizations can integrate with any other system or entity across the network regardless of programming language, platform, device, or web browser.

Audience

This tutorial is intended for software developers who are looking to implement web services using Employeease Extend.

Document Status

Version 1.0 – 10/04/2007 - Initial release.

Conventions

Code snippets will appear using a mono-spaced font and a blue color scheme.

Example

```
for (int i = 0; i < 10; i++)  
{  
    System.Console.Out.WriteLine(i);  
}
```

Useful tips and best practices will appear using the following bullet style.

Example

- Work closely with your HR administrator to accurately model your company's business processes.

Building a Simple Web Service Call Using .NET

Tools

The following tool sets are used during this tutorial. Some steps may need to be modified when working with versions other than what is listed below.

- Microsoft Visual C# 2005 Express Edition
- Microsoft .NET Framework 2.0
- Web Services Enhancements (WSE) 3.0 for Microsoft .NET
- XML Editor (Optional, but recommended)

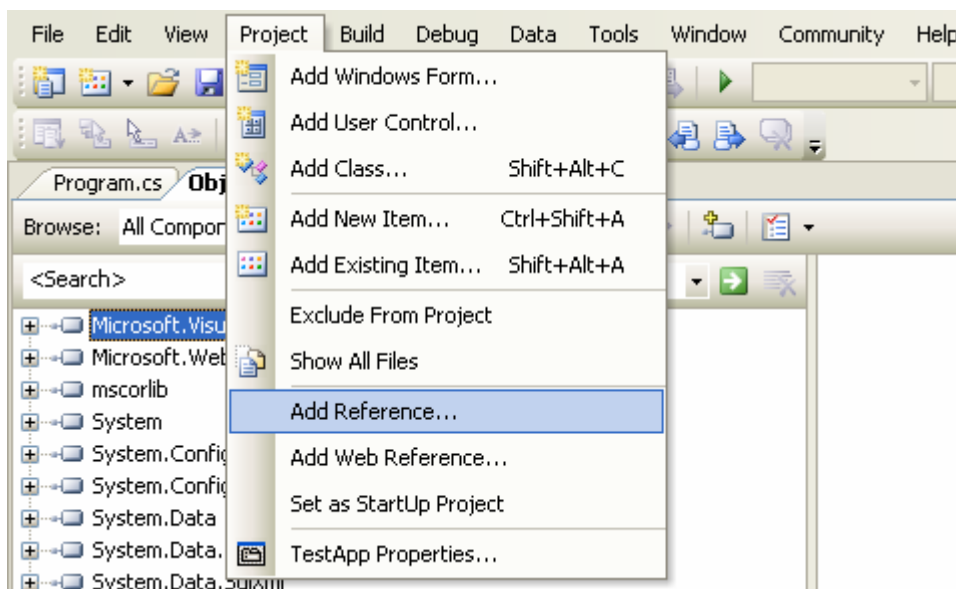
Set-Up

Create a new project

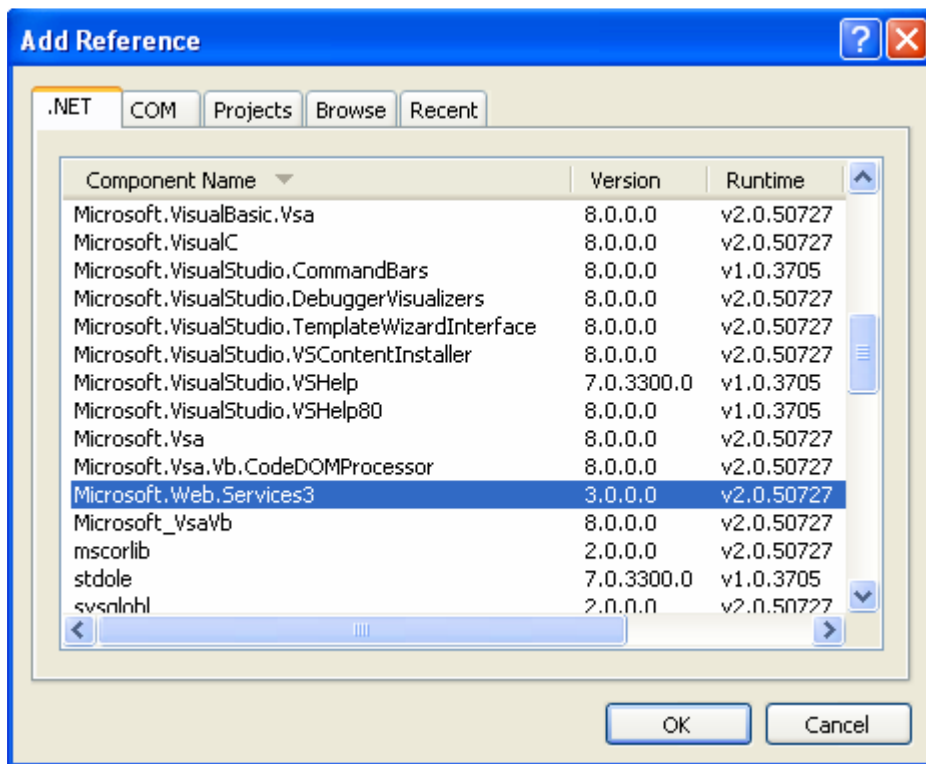
To simplify the code, we will be using creating a console application using the “Console Application” template in the New Project wizard. To remain consistent throughout the tutorial, please name this project “TestApp”.

Add a reference to WSE 3.0

In order to instantiate a UserToken conforming to the WS-I basic security profile, we will need to add a reference to the WSE 3.0 package. This can be done by using the main menu items Project->Add Reference.

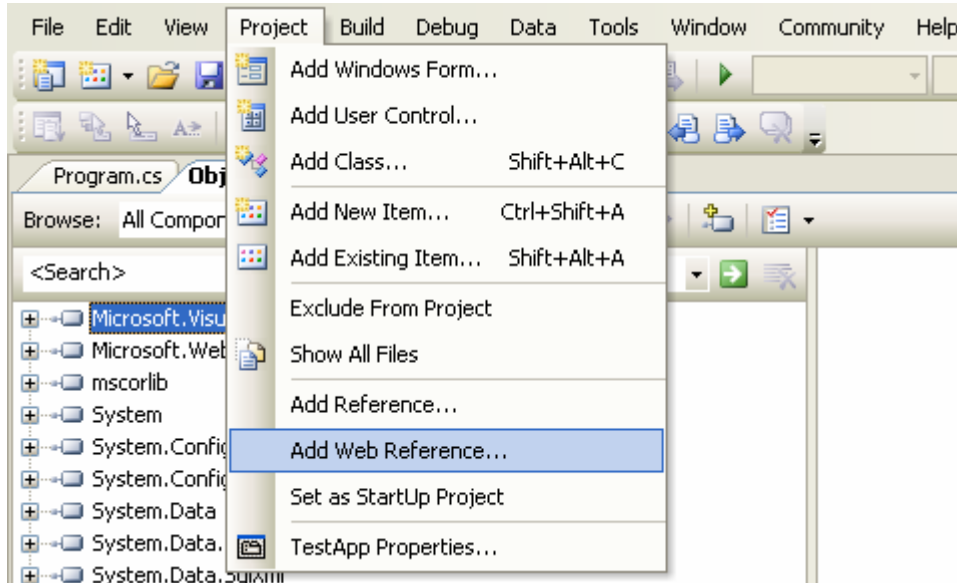


From the “Add Reference” pop-up, stay on the .NET tab and select the *Microsoft.Web.Services3* package. Click OK.

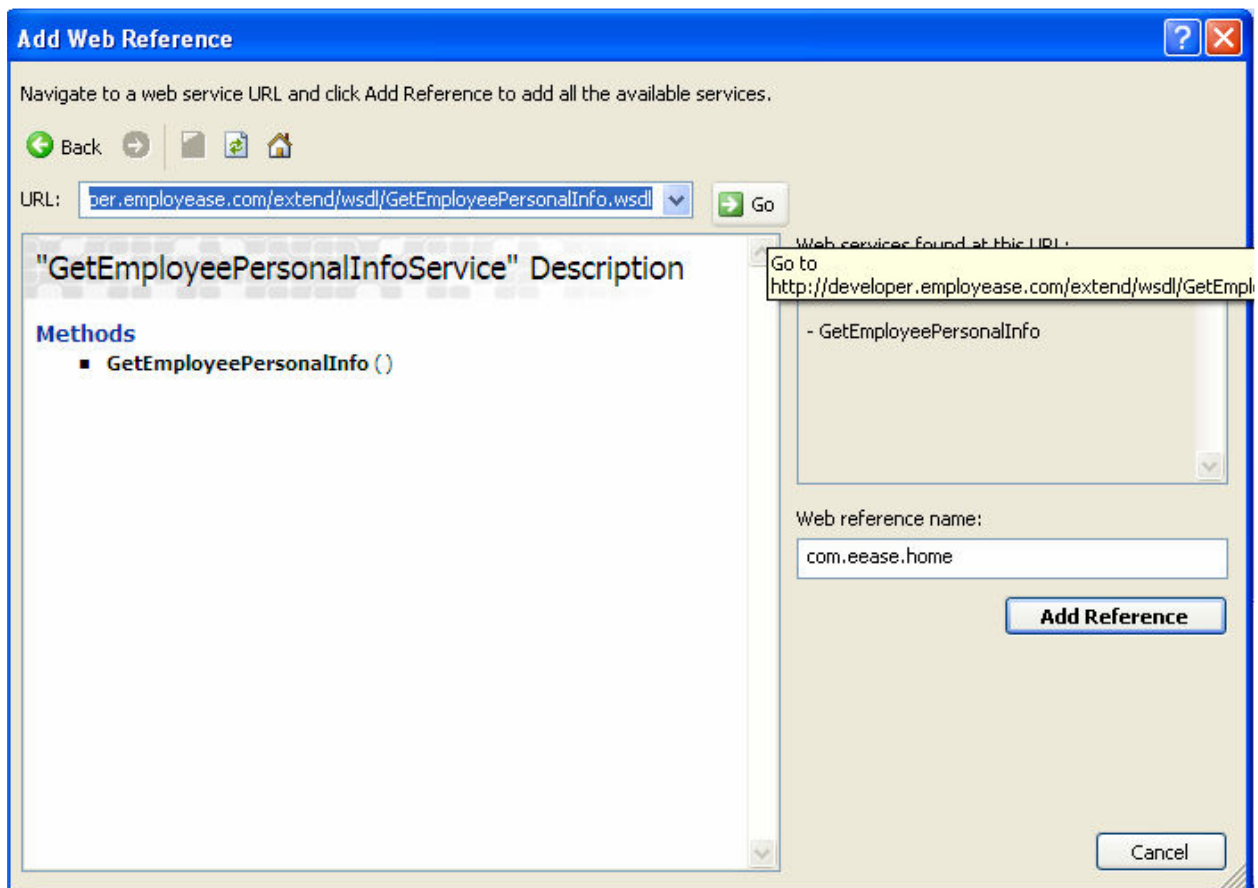


Add a web reference using WSDL

Adding a web reference allows the .NET framework to auto-generate most of the functionality necessary to make a web service call. This can be done using the main menu items Project->Add Reference.



This should bring up the “Add Web Reference” pop-up, which is shown below.



For this tutorial, we will be using the **GetEmployeePersonalInfo** web service. Enter the URL into the pop-up and click the *Go* button. The URL for the **GetEmployeePersonalInfo** WSDL is:

<http://developer.employease.com/wsdl/GetEmployeePersonalInfo.wsdl>

Once the pop-up is finished fetching the WSDL and accompanying schema definitions, it should display that it has found the Method: `GetEmployeePersonalInfo()`. Click the “Add Reference” button on the right hand side of the pop-up to finish adding this web reference.

Once the reference has been added, all of the objects required to send a **GetEmployeePersonalInfo** request have been created. We are now ready to begin writing code.

Building a simple web service request

Initializing the web proxy client object

To begin using the web service proxy client object that was generated by adding a web reference to the **GetEmployeePersonalInfo** WSDL, we need to add a reference to the it's namespace using project namespaces. This can be done by adding the following line to the Program.cs source file.

```
using TestApp.com.eease.home;
```

Now that we have visibility into this namespace, we have access to the `GetEmployeePersonalInfoService` object which will act as a web service proxy client. To initialize this object, use the following line.

```
GetEmployeePersonalInfoService oService = new GetEmployeePersonalInfoService();
```

To simplify the code used in this tutorial, we will not be following object oriented design in the example code. Go ahead and add the above line to the Main method in the Program class.

Authentication

In order to authenticate with the Employee Network, we will need to add a `UsernameToken` element to our SOAP header that conforms to the WS-I basic security profile. Luckily, we can take advantage of the functionality in the WSE 3.0 add-on to generate this for us.

In order to build a `UsernameToken` you must have created an ECXML user, a company identifier, and a shared secret password provided by Employee. For this tutorial, we will use the following credentials:

User: openapi.ecxml

Company Identifier: openapi

Password: VfmHcgz1QXMG2

- To test a service other than the **GetEmployeePersonalInfo** service, or to begin working with your own company's information, you will need to obtain these credentials from your primary user (generally your HR Director). They will need to work with Employeease Integration Services to grant the appropriate service rights and provide the shared secret.

Go ahead and declare the following static member variables within the Program object.

```
private static string USERNAME = "openapi.ecxml@openapi";  
private static string SHARED_SECRET = "VfmHcgz1QXMG2";  
private static string COMPANY_IDENTIFIER = "openapi";
```

Import the namespace containing the UsernameToken with the following line:

```
using Microsoft.Web.Services3.Security.Tokens;
```

The WS-I basic security profile Username token will be formed using a combination of the ECXML username and the company identifier (e.g. "openapi.ecxml@openapi", "soa.ecxml@test.com", etc). The Password element will be a hashed combination of the shared secret, nonce, and a timestamp. The UsernameToken object constructor will handle all of this for us.

```
UsernameToken oUsernameToken = new UsernameToken(USERNAME, SHARED_SECRET);
```

The next steps are to create a SecurityHeaderType object and add the serialized UsernameToken to it's list of headers.

```
System.Xml.XmlElement oSecurityHeaderXml = oUsernameToken.GetXml(new  
System.Xml.XmlDocument());  
SecurityHeaderType oSecurityHeader = new SecurityHeaderType();  
oSecurityHeader.Any = new System.Xml.XmlElement[] { oSecurityHeaderXml };
```

The final step is to associate the security header with the web service proxy client object.

```
oService.Security = oSecurityHeader;
```

Building the request payload

Now it is time to populate the request payload. Because we are using a "Get" service, we will be sending a filter in our request payload. If we were using an "Add" or a "Change" web service to modify employee data, we would send an EmployeeKey element and a data element used to modify the existing data (e.g. user account info, corp group).

For this tutorial, we will be using the Companies filter. In .NET, this will be represented as an array of company identifiers (strings).

```
GetEmployeePersonalInfoRequestFilter oFilter =  
    new GetEmployeePersonalInfoRequestFilter();  
oFilter.Companies = new string[] { COMPANY_IDENTIFIER };
```

- The Companies filter for “Get” services is the least efficient filter offered. In order to keep response times down for all customers, please request as little information as necessary per request.

Next, we need to create a Request object and apply the filter that we created.

```
GetEmployeePersonalInfoRequest oRequest =  
    new GetEmployeePersonalInfoRequest();  
oRequest.Filter = oFilter;
```

We now have enough information to make our web service call. For the GetEmployeePersonalInfo web service, the result will be an array of EmployeePersonalInfoType objects.

```
EmployeePersonalInfoType[] arPersonalInfo =  
    oService.GetEmployeePersonalInfo(oRequest);
```

Parsing the response payload

Parsing the response is as simple as iterating through the resulting array (in this case an array of EmployeePersonalInfoTypes) and picking out the necessary data.

```
// Loop through each employee in the response payload.  
for (int i = 0; i < arPersonalInfo.Length; i++)  
{  
    String stLastName = arPersonalInfo[i].PersonalInfo.PersonName.LastName;  
    String stFirstName = arPersonalInfo[i].PersonalInfo.PersonName.FirstName;  
}
```

- For the employee based “Get” services (GetUser, GetEmployeeCorpGroup, etc), each record in the response will include an EmployeeKey element which contains either a National ID, Employee ID, or both. This depends on the company set-up.
- For an “Add” or a “Change web service, each record in the response will include a disposition element which indicates whether the transaction was successful and contains any exception messages that may have been generated during the transaction.

.NET Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using TestApp.com.eease.home;
using Microsoft.Web.Services3.Security.Tokens;

namespace TestApp
{
    class Program
    {
        private static string USERNAME = "openapi.ecxml@openapi";
        private static string SECRET = "VfmHcgz1QXMG2";
        private static string COMPANY_IDENTIFIER = "openapi";

        static void Main(string[] args)
        {
            // Create the web service proxy client.
            GetEmployeePersonalInfoService oService =
                new GetEmployeePersonalInfoService();

            // Create the UsernameToken as defined in the WS-I secure profile.
            UsernameToken oUsernameToken = new UsernameToken(USERNAME, SECRET);
            // Serialize the UsernameToken into XML.
            System.Xml.XmlElement oSecurityHeaderXml =
                oUsernameToken.GetXml(new System.Xml.XmlDocument());
            // Initialize the com.eease.home.SecurityHeaderType.
            SecurityHeaderType oSecurityHeader = new SecurityHeaderType();
            // Initialize the security header type's list of headers to include the
            // serialized UsernameToken.
            oSecurityHeader.Any = new System.Xml.XmlElement[] { oSecurityHeaderXml };
            // Associate the security header with the web service proxy client.
            oService.Security = oSecurityHeader;

            // Use the company filter to return all employees in a company.
            GetEmployeePersonalInfoRequestFilter oFilter =
                new GetEmployeePersonalInfoRequestFilter();
            oFilter.Companies = new string[] { COMPANY_IDENTIFIER };
            // Build the GetEmployeePersonalInfo request and set the filter.
            GetEmployeePersonalInfoRequest oRequest =
                new GetEmployeePersonalInfoRequest();
            oRequest.Filter = oFilter;

            // Call the web service and store the result in an array of
            // EmployeePersonalInfoType.
            EmployeePersonalInfoType[] arPersonalInfo =
                oService.GetEmployeePersonalInfo(oRequest);

            // Loop through each employee in the response payload.
            for (int i = 0; i < arPersonalInfo.Length; i++)
            {
                String stLastName =
                    arPersonalInfo[i].PersonalInfo.PersonName.LastName;
                String stFirstName =
                    arPersonalInfo[i].PersonalInfo.PersonName.FirstName;
                String stBirthDate =
                    arPersonalInfo[i].PersonalInfo.BirthDate.ToShortDateString();
                String stFullName = stLastName + ", " + stFirstName;

                // Write the employees name and birthdate to the console.
            }
        }
    }
}
```

```
        System.Console.Out.WriteLine(stFullName + " - " + stBirthDate);
    }

    // Display the console until a newline has been entered.
    System.Console.In.ReadLine();
}
}
```

Building a Simple Web Service Call Using Java

Tools

The following tool sets are used during this tutorial. Some steps may need to be modified when working with versions other than what is listed below.

- Eclipse IDE for Java Developers 3.3 (<http://www.eclipse.org>)
- Java Development Kit 1.5.0_06 (<http://java.sun.com>)
- Apache Axis 1.4 Binary Distribution (<http://ws.apache.org/axis/>)
- Apache wss4j 1.5.2 Jar (<http://ws.apache.org/wss4j/>)
- XML Editor (Optional, but recommended)

Set-Up

Create a new project

In Eclipse, create a new Java Project using File->New Project. To remain consistent throughout the tutorial, please name this project “TestApp”.

Under the Project layout, select the option to create separate folders for sources and class files. Click the Next button.

The default set-up should list the *src* directory as your source file directory and the *bin* directory as your class output directory. If this is not your default, please adjust to match these settings. Click the Finish button.

Add Apache Axis to your Project

Go ahead and unpack the Apache Axis distribution if you haven’t already. Make a note of where the files have been stored.

Back in Eclipse, right click on your project and select New->Folder. Name this folder “lib”. Click the Finish button.

Right click on your newly created lib folder and select Import->File System. Navigate to the directory containing Apache Axis and go to the lib folder (i.e. C:\Axis\lib). Select all of the jar files in the directory and click the Finish button.

All of the jar files in the Axis lib folder should have been copied over to the lib folder in your Project directory.

Add Apache wss4j to your Project

To add the Apache wss4j library to your project, follow the same directions as above while navigating to the location where the wss4j jar file was downloaded.

Project Classpath

Now we need to add the jar files that were added to the lib folder to the Project classpath. This can be done by right clicking on your project in the Package Explorer and selecting the Properties option.

In the popup, select Java Build Path and click the Libraries sub-tab. Click the Add Jars button, and select all of the jars in the lib/ directory.

Generate Web Service Proxy using WSDL2Java

Because we downloaded the binary distribution of Apache Axis, we can use the WSDL2Java tool from the command line to generate a web service proxy object.

Setting up the classpath

The classpath set-up for the JVM should consist of all the jar files stored in the Apache Axis lib directory. The directory listing as of this writing is indicated below. The included jar files (and version numbers) may differ depending on the version of Apache Axis that was downloaded.

axis-ant.jar	jaxrpc.jar
axis.jar	log4j-1.2.8.jar
commons-discovery-0.2.jar	saaj.jar
commons-logging-1.0.4.jar	wsdl4j-1.5.1.jar

By adding all of the jar files in the directory to the classpath, we could now call the WSDL2Java tool using the following command:

```
java -cp /Axis/lib/axis-ant.jar:/Axis/lib/axis.jar:/Axis/lib/commons-discovery-0.2.jar:/Axis/lib/commons-logging-1.0.4.jar:/Axis/lib/jaxrpc.jar:/Axis/lib/log4j-1.2.8.jar:/Axis/lib/saaj.jar:/Axis/lib/wsdl4j-1.5.1.jar org.apache.axis.wsdl.WSDL2Java
```

Using WSDL2Java

The `-o` option is used to tell the WSDL2Java tool where to store the generated source files.

For this tutorial, we will be using the **GetEmployeePersonalInfo** web service. The URL for the **GetEmployeePersonalInfo** WSDL is:

<http://developer.employease.com/wsdl/GetEmployeePersonalInfo.wsdl>

Using the classpath you set-up above, the final command entry should resemble something like this:

```
java -cp /Axis/lib/axis-ant.jar:/Axis/lib/axis.jar:/Axis/lib/commons-discovery-0.2.jar:/Axis/lib/commons-logging-1.0.4.jar:/Axis/lib/jaxrpc.jar:/Axis/lib/log4j-
```

```
1.2.8.jar:/Axis/lib/saaj.jar:/Axis/lib/wsdl4j-1.5.1.jar org.apache.axis.wsdl.WSDL2Java -o
/TestApp/src http://developer.employeease.com/wsdl/GetEmployeePersonalInfo.wsdl
```

- If you are working behind a web proxy, adjust the following properties and add the result to the above command before the classpath option (-cp).

```
-DproxySet=true -DproxyHost=proxy.test.com -DproxyPort=8888
```

Cleaning Up Errors

In a perfect world, the auto-generated code would not contain any errors. Luckily, the errors that exist are easy to correct.

After refreshing the TestApp project in Eclipse, you should now see 5 packages in your source directory. All of the classes in the default package belong in the com.eease.home package. Go ahead and select all of the classes and drag them into the com.eease.home package. This should leave us with only one error unresolved.

The remaining error should be contained in a constructor for the NationalIdType class. Because the default constructor is adequate, go ahead and change the invalid constructor so that it ignores the passed in parameter.

```
// Simple Types must have a String constructor
public NationalIdType(java.lang.String _value)
{
    super();
}
```

We should now have generated all of the objects required to make a call to the **GetEmployeePersonalInfo** service.

Building a simple web service request

Initializing the web proxy client object

Go ahead and create a new class in the com.eease.home package called TestService with a main() method.

The GetEmployeePersonalInfoPortType object will act as a web service proxy client. To initialize this object, use the following line.

```
GetEmployeePersonalInfoPortType oProxy =
new GetEmployeePersonalInfoServiceLocator().getGetEmployeePersonalInfoPort();
```

To simplify this tutorial, we will not be using object oriented design in the example code. Go ahead and add the above line to the Main method in the TestService class.

Authentication

In order to authenticate with the Employee Network, we will need to add a UsernameToken element to our SOAP header that conforms to the WS-I basic security profile. Luckily, we can take advantage of the functionality in the WSSJ library to generate this for us.

In order to build a UsernameToken you must have created an ECXML user, a company identifier, and a shared secret password provided by Employee. For this tutorial, we will use the following credentials:

User: openapi.ecxml

Company Identifier: openapi

Password: VfmHcgz1QXMG2

- To test a service other than the **GetEmployeePersonalInfo** service, or to begin working with your own company's information, you will need to obtain these credentials from your primary user (generally your HR Director). They will need to work with Employee to grant the appropriate service rights and provide the shared secret.

Go ahead and declare the following static member variables within the Program object.

```
private static string USERNAME = "openapi.ecxml@openapi";  
private static string SHARED_SECRET = "VfmHcgz1QXMG2";  
private static string COMPANY_IDENTIFIER = "openapi";
```

Import the package containing the UsernameToken with the following line:

```
import org.apache.ws.security.message.token.UsernameToken;
```

When working with objects that generate XML, it is often necessary to initialize a Document object to be used as the root of the document tree. This can be done by utilizing the DocumentBuilderFactory class.

```
DocumentBuilder oDocBuilder =  
DocumentBuilderFactory.newInstance().newDocumentBuilder();  
Document oDocument = oDocBuilder.newDocument();
```

The WS-I basic security profile Username element will be formed using a combination of the ECXML username and the company identifier (e.g. "openapi.ecxml@openapi", "soa.ecxml@test.com", etc). The Password element will be a hashed combination of the shared secret, nonce, and a timestamp. The UsernameToken object will handle all of this for us. We just need to apply it with a username and password.

```
UsernameToken oUsernameToken = new UsernameToken(true, oDocument);  
oUsernameToken.setName(USERNAME);  
oUsernameToken.setPassword(SHARED_SECRET);
```

The next steps are to create a SecurityHeaderType object and add the serialized UsernameToken to it's list of headers.

```
MessageElement[] oHeaderElements =
    new MessageElement[] { new MessageElement(oUsernameToken.getElement()) };
SecurityHeaderType oSecurityHeader = new SecurityHeaderType();
oSecurityHeader.set_any(oHeaderElements);
```

Building the request payload

Now it is time to populate the request payload. Because we are using a “Get” service, we will be setting a filter in our request payload. If we were using an “Add” or a “Change” web service to modify employee data, we would send an EmployeeKey element and a data element used to modify the existing data (e.g. user account info, corp group).

For this tutorial, we will be using the Companies filter. In Java, this will be represented as an array of company identifiers (Strings).

```
GetEmployeePersonalInfoRequestFilter oFilter =
    new GetEmployeePersonalInfoRequestFilter();
oFilter.setCompanies(new String[] { COMPANY_IDENTIFIER });
```

- The Companies filter for “Get” services is the least efficient filter offered. In order to keep response times down for all customers, please request as little information as necessary per request.

Next, we need to create a Request object and apply the filter that we created.

```
GetEmployeePersonalInfoRequest oRequest =
    new GetEmployeePersonalInfoRequest();
oRequest.setFilter(oFilter);
```

We now have enough information to make our web service call. For the GetEmployeePersonalInfo web service, the result will be an array of EmployeePersonalInfoType objects.

```
EmployeePersonalInfoType[] arPersonalInfo =
    oProxy.getEmployeePersonalInfo(oSecurityHeader, oRequest);
```

Parsing the response payload

Parsing the response is as simple as iterating through the resulting array (in this case an array of EmployeePersonalInfoTypes) and picking out the necessary data.

```
for (int i = 0; i < arPersonalInfo.Length; i++)
{
    String stLastName =
        arPersonalInfo[i].getPersonalInfo().getPersonName().getLastName();
    String stFirstName =
        arPersonalInfo[i].getPersonalInfo().getPersonName().getFirstName();
}
```

- For the employee based “Get” services (GetUser, GetEmployeeCorpGroup, etc), each record in the response will include an EmployeeKey element which contains either a National ID, Employee ID, or both. This depends on the company set-up.
- For an “Add” or a “Change web service, each record in the response will include a disposition element which indicates whether the transaction was successful and contains any exception messages that may have been generated during the transaction.

Java Sample Code

```

package home.eease.com;

import java.util.Date;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.rpc.ServiceException;

import org.apache.axis.message.MessageElement;
import org.apache.ws.security.message.token.UsernameToken;
import
org.oasis_open.docs.wss._2004._01.oasis_200401_wss_wssecurity_secext_1_0_xsd.SecurityHeaderType;
import
org.oasis_open.docs.wss._2004._01.oasis_200401_wss_wssecurity_secext_1_0_xsd.UsernameTokenType;
import org.w3c.dom.Document;

import com.eease.home.EmployeePersonalInfoType;
import com.eease.home.GetEmployeePersonalInfoPortType;
import com.eease.home.GetEmployeePersonalInfoRequest;
import com.eease.home.GetEmployeePersonalInfoRequestFilter;
import com.eease.home.GetEmployeePersonalInfoServiceLocator;

public class TestService
{
    private final static String USERNAME = "openapi.ecxml@openapi";
    private final static String SECRET = "VfmHcgz1QXMG2";
    private final static String COMPANY_IDENTIFIER = "openapi";

    /**
     * @param args
     * @throws ServiceException
     * @throws ParserConfigurationException
     */
    public static void main(String[] args) throws Exception
    {
        // Create the web service proxy client.
        GetEmployeePersonalInfoPortType oProxy =
new GetEmployeePersonalInfoServiceLocator().getGetEmployeePersonalInfoPort();

        // Create a document object used to generate the XML.
        DocumentBuilder oDocBuilder =
            DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document oDocument = oDocBuilder.newDocument();
        // Create the UsernameToken as defined in the WS-I secure profile.
        UsernameToken oUsernameToken = new UsernameToken(true, oDocument);
        oUsernameToken.setName(USERNAME);
        oUsernameToken.setPassword(SECRET);
        // Initialize an array of message elements containing the UsernameToken.
        MessageElement[] oHeaderElements =
            new MessageElement[]
            { new MessageElement(oUsernameToken.getElement()) };
        // Initialize the security header.
        SecurityHeaderType oSecurityHeader = new SecurityHeaderType();
        oSecurityHeader.set_any(oHeaderElements);

        // Use the company filter to return all employees in a company.
    }
}

```

```
GetEmployeePersonalInfoRequestFilter oFilter =
    new GetEmployeePersonalInfoRequestFilter();
oFilter.setCompanies(new String[] { COMPANY_IDENTIFIER });
// Build the GetEmployeePersonalInfo request and set the filter.
GetEmployeePersonalInfoRequest oRequest =
    new GetEmployeePersonalInfoRequest();
oRequest.setFilter(oFilter);

// Call the web service and store the result in an array of
// GetEmployeePersonalInfoType.
EmployeePersonalInfoType[] arPersonalInfo =
    oProxy.getEmployeePersonalInfo(oSecurityHeader, oRequest);

// Loop through each employee in the response payload.
for(int i = 0; i < arPersonalInfo.length; i++)
{
    String stLastName =
arPersonalInfo[i].getPersonalInfo().getPersonName().getLastName();
    String stFirstName =
arPersonalInfo[i].getPersonalInfo().getPersonName().getFirstName();
    String stBirthDate =
arPersonalInfo[i].getPersonalInfo().getBirthDate().toString();
    String stFullName = stLastName + ", " + stFirstName;

    // Write the employee's name and birth date to the console.
    System.out.println(stFullName + " - " + stBirthDate);
}
}
}
```

Applying Filters to Get Services

Common Filters

Employee Key Filter

The employee key filter is used to for a specific set of one or more employees. An EmployeeKey element contains a unique identifier and a company identifier. The unique identifier consists of a national identifier (e.g. SSN) or an employee identifier.

- The identifier must be both required and unique. Please work with your HR administrator to determine the appropriate identifier that should be sent according to your company set-up.

The example below would return any employees that have an SSN number of 111-11-1111 or an Employee ID of 11111.

```
<Filter>
  <Employees>
    <EmployeeKey>
      <Identifier>
        <NationalId idOwner="US">111-11-1111</NationalId>
      </Identifier>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </EmployeeKey>
    <EmployeeKey>
      <Identifier>
        <EmployeeId>11111</EmployeeId>
      </Identifier>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </EmployeeKey>
  </Employees>
</Filter>
```

Company Filter

The company filter is used to retrieve all employees within a set of one or more companies.

- The company filter is the most resource intensive of all the filters. To avoid timeouts and slower response times, please consider using a more restrictive filter if at all possible. Examples include using the EmployeeKey filter to only request relevant employees or using a combination filter when available.

The example below would return all employees within the test.com company.

```
<Filter>
  <Companies>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Companies>
</Filter>
```

Last Name Filter

The last name filter is used to retrieve all employees whose last name matches the search criteria. This emulates the functionality of the last name search in the Employee Management Center.

- Two wildcards are supported by this filter, ? and *.

The example below would return all employees whose last name begins with John.

```
<Filter>
  <LastNames>
    <LastNameKey>
      <LastName>John*</LastName>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </LastNameKey>
  </LastNames>
</Filter>
```

Combination Filters

Combination filters are additional filters that can be applied to a Get web service call. A few examples have been included below.

GetEmployeeStatus Example

This example request would return all employees within the test.com company that had a terminated status as of 1/1/2007.

```
<GetEmployeeStatusRequest>
  <EffectiveOn>2007-01-01</EffectiveOn>
  <Filter>
    <Companies>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </Companies>
    <Statuses>
      <StatusCode>Terminated</StatusCode>
    </Statuses>
  </Filter>
</GetEmployeeStatusRequest>
```

GetEmployeeCorpGroup Example

This example request would return the Division and Location corporate groups for all employees within the test.com company as of 3/25/2006.

```
<GetEmployeeCorpRequest>
  <EffectiveOn>2006-03-25</EffectiveOn>
  <Filter>
    <Companies>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </Companies>
    <Types>
```

```
<CorpGroupType>Division</CorpGroupType>  
  <CorpGroupType>Location</CorpGroupType>  
</Types>  
</Filter>  
</GetEmployeeCorpRequest>
```

Appendix

Related Documents

WS-I Basic Security Profile

<http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>

XML Schema

<http://www.w3.org/2001/XMLSchema>

SOAP

<http://www.w3.org/TR/soap/>

WSDL

<http://www.w3.org/TR/wsdl>

Example Payloads

GetEmployeeCorpGroup Request

```
POST /wsi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client
Protocol 2.0.50727.42)
Content-Type: text/xml; charset=utf-8
SOAPAction: "GetEmployeeCorpGroup"
Host: home.eease.com
Content-Length: 1328
Expect: 100-continue
Connection: Keep-Alive
HTTP/1.1 100 Continue

<?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">
  <soap:Header>
    <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="SecurityToken-c07ac111-f0a3-48ec-81d6-7879005c6f02"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
        <wsse:Username>test.ecxml@test.com</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-
1.0#PasswordDigest">YTSB3aEwvqOcL/QhIiABC2LHjfk=</wsse:Password>
        <wsse:Nonce>df3dL6uauTGzs5T2PUjSURA==</wsse:Nonce>
        <wsu:Created>2007-08-02T20:58:07Z</wsu:Created>
      </wsse:UsernameToken>
    </Security>
```

```

</soap:Header>
<soap:Body>
  <GetEmployeeCorpGroupRequest>
    <EffectiveOn>2007-08-01</EffectiveOn>
    <Filter>
      <Employees>
        <EmployeeKey>
          <Identifier>
            <NationalId idOwner="US">111-11-1111</NationalId>
          </Identifier>
          <CompanyIdentifier>test.com</CompanyIdentifier>
        </EmployeeKey>
      </Employees>
      <Types>
        <CorpGroupType>Division</CorpGroupType>
      </Types>
    </Filter>
  </GetEmployeeCorpGroupRequest>
</soap:Body>
</soap:Envelope>

```

GetEmployeeCorpGroup Response

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept: application/soap+xml, text/html, image/gif, image/jpeg, *; q=.2, */*;
q=.2
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 584
Date: Thu, 02 Aug 2007 20:52:47 GMT

```

```

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <GetEmployeeCorpGroupResponse>
      <EmployeeCorpGroup>
        <EmployeeKey>
          <NationalId idOwner="US">111-11-1111</NationalId>
          <EmployeeId>11111</EmployeeId>
          <CompanyIdentifier>test.com</CompanyIdentifier>
        </EmployeeKey>
        <CorpGroupKey>
          <CorpGroupType>Division</CorpGroupType>
          <CorpGroupName>Test Division</CorpGroupName>
          <CompanyIdentifier>test.com</CompanyIdentifier>
        </CorpGroupKey>
        <EffectiveInfo>
          <StartDate>2006-01-01</StartDate>
        </EffectiveInfo>
      </EmployeeCorpGroup>
    </GetEmployeeCorpGroupResponse>
  </env:Body>
</env:Envelope>

```

HireEmployee Request

```
POST /wsi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client
Protocol 2.0.50727.832)
Content-Type: text/xml; charset=utf-8
SOAPAction: "HireEmployee"
Host: home.eease.com
Content-Length: 2532
Expect: 100-continue
Connection: Keep-Alive
```

```
HTTP/1.1 100 Continue
```

```
<?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">
  <soap:Header>
    <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="SecurityToken-2546a2b8-64ce-4bad-9493-f7c4b3478630"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
        <wsse:Username>test.ecxml@test.com</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-
1.0#PasswordDigest">J8+a3Jszjr34sXyZd5SkWneNrWY194=</wsse:Password>
        <wsse:Nonce>res8w6vaYmWyE32auqdd5+A==</wsse:Nonce>
        <wsu:Created>2007-08-21T20:11:16Z</wsu:Created>
      </wsse:UsernameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <HireEmployeeRequest>
      <Employees>
        <Employee>
          <EmployeeKey>
            <Identifier>
              <NationalId idOwner="US">111-11-1111</NationalId>
            </Identifier>
            <CompanyIdentifier>test.com</CompanyIdentifier>
          </EmployeeKey>
          <HireDate>2007-08-01</HireDate>
          <Reason>Import created action</Reason>
          <StatusClassification>Regular</StatusClassification>
          <FullPartTime>Full-time</FullPartTime>
          <PersonalInfo>
            <PersonName>
              <FirstName>Joe</FirstName>
              <LastName>Johnson</LastName>
            </PersonName>
            <Gender>Male</Gender>
            <BirthDate>1950-01-01</BirthDate>
```

```

</PersonalInfo>
<WorkInfo>
  <EmployeeID>111111111</EmployeeID>
  <EmailAddress>jjohnson@test.com</EmailAddress>
</WorkInfo>
<CorpGroups>
  <Division>
    <CorpGroupType>Division</CorpGroupType>
    <CorpGroupName>Corporate</CorpGroupName>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Division>
  <Location>
    <CorpGroupType>Location</CorpGroupType>
    <CorpGroupName>Baltimore</CorpGroupName>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Location>
  <Class>
    <CorpGroupType>Class</CorpGroupType>
    <CorpGroupName>Benefits Eligible</CorpGroupName>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Class>
  <Department>
    <CorpGroupType>Department</CorpGroupType>
    <CorpGroupName>Sales</CorpGroupName>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Department>
  <Union>
    <CorpGroupType>Union</CorpGroupType>
    <CorpGroupName>N/A</CorpGroupName>
    <CompanyIdentifier>test.com</CompanyIdentifier>
  </Union>
</CorpGroups>
<PayGroup>
  <PayGroupName>Weekly</PayGroupName>
  <CompanyIdentifier>test.com</CompanyIdentifier>
</PayGroup>
</Employee>
</Employees>
</HireEmployeeRequest>
</soap:Body>
</soap:Envelope>

```

HireEmployee Response

```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept: application/soap+xml, text/html, image/gif, image/jpeg, *; q=.2, */*;
q=.2
Content-Type: application/soap+xml;charset=utf-8
Content-Length: 401
Date: Tue, 21 Aug 2007 20:06:01 GMT

```

```

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>

```

```
<HireEmployeeResponse>
  <EmployeeStatus>
    <EmployeeKey>
      <Identifier>
        <NationalId idOwner="US">111-11-1111</NationalId>
      </Identifier>
      <CompanyIdentifier>test.com</CompanyIdentifier>
    </EmployeeKey>
    <EmployeeDisposition>
      <Successful>true</Successful>
    </EmployeeDisposition>
  </EmployeeStatus>
</HireEmployeeResponse>
</env:Body>
</env:Envelope>
```