Adabas SOA Gateway

Tutorials

Version 2012-12-17

December 2012

Adabas SOA Gateway

This document applies to Adabas SOA Gateway Version 2012-12-17.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ASG-AASGTUTORIALS-TBD-20121217

Table of Contents

Tutorials

This section contains tutorials which will show you to access your Adabas database using SOA Gateway in an exemplary way, based on a number of programming languages and standard applications which have built-in SOAP facilities or extensions.

It also contains tutorials detailing how to set up and use the Usage Governace capability offered by SOA Gateway.

There is a tutorial demonstrating how EntireX can be used as a transport for SOAP messages to SOA Gateway.

- **Adabas Tutorials**
- **Usage Governance Tutorials**
- **EntireX Tutorial**

1 Adabas Tutorials

The Adabas tutorials are based on the well-known Adabas 'Employees' file. The following sample Resource definitions are put in place by the SOA Gateway server install process:

- adabas_Employees
- adabas_Employees_9
- adabas_Employees_special
- adabas_EmployeesMini
- adabas_Vehicles

The default Adabas Database ID preset for the demo files is 212, which may not reflect the actual Database ID in your environment, so these will need to be adjusted before trying to run the sample programs. Please familiarize yourself with the SOA Gateway Control Center functions required to achieve this before continuing the tutorials trail.

Basic Tutorials

The basic tutorials cover the following areas:

- **Using Java (Axis2) wrapper classes**
- **Using the PHP SOAP Extension**
- **Using Ruby**
- **Using C#**
- **Using soapUi**
- **Using Microsoft Office : Infopath**

Advanced Tutorials

Dealing with LOBs (Large OBjects)

- **Preparing the LOB file**
- **Using LOBs with soapUi**
- **Using LOBs with PHP**
- **Using LOBs with your favourite browser**

Dealing with Transactions

- **Transactions using soapUi**

Samples

This section also includes the following demo files and code samples that you can try out:

- **Tut_02_List.java**
- **ex01_SoaGatewayFirst.php**
- **ex02_SoaGatewayEmpList.php**
- **ex02a_SoaGatewayEmpListDescending.php**
- **ex02a_SoaGatewayEmpListSorted.php**
- **ex06_SoaGatewaySpecial_SubDescriptor.php**
- **ex06_SoaGatewaySpecial_SuperDescriptor.php**
- **ex03_SoaGatewayEmpAdd.php**
- **ex04_SoaGatewayEmpGet.php**
- **ex05_SoaGatewayEmpDel.php**
- **ex10_SoaGatewaySimpleForm.php**
- **ex15_SoaGatewayUpdateForm.php**
- **empMiniList.rb**
- **ASGDemo.cs**

2 Using Java (Axis2) wrapper classes

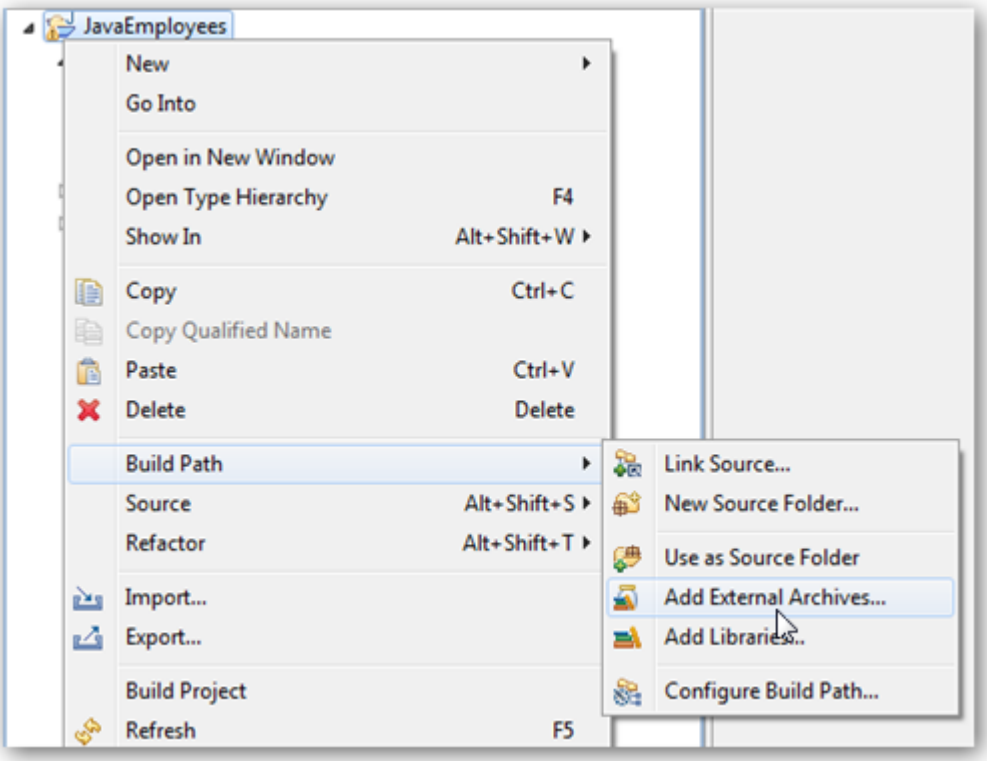
Tutorial: Generate a Java wrapper for the "Employees" file

Java wrapper/stub classes are generated using the **Apache Axis2** feature **WSDL2Java**.

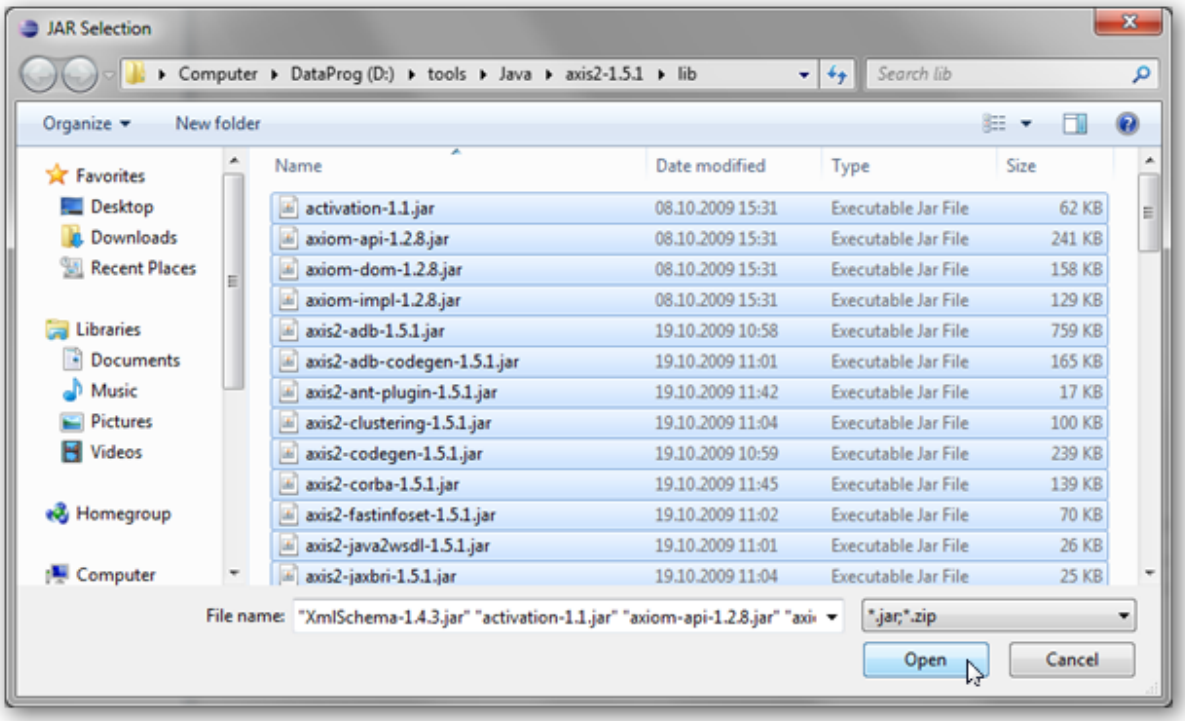
If you do not have it already, download and install the latest Axis2 kit.

These are the steps required to generate the Java wrapper classes for the "adabas_EmployeesMini" DataView supplied with SOA Gateway:

1. Create a new Java-project (refer to Getting started with Eclipse), name it "JavaEmployees"
2. Right-click the "JavaEmployees" project folder, select "Build Path", then "Add External Archives..:"

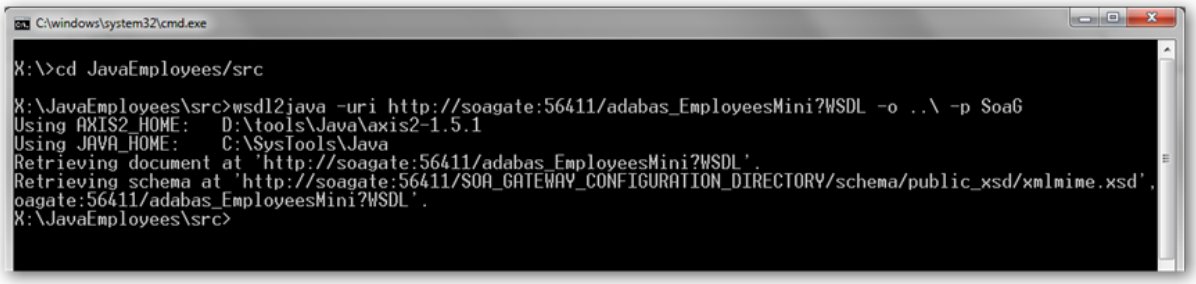


Add all .jar files from the axis2 "lib" directory to the project's Build-Path.



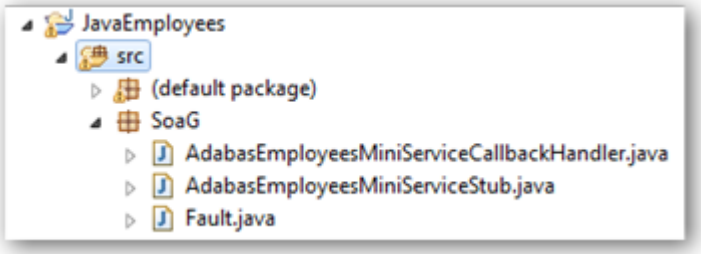
3. Open a command prompt (aka "DOS box"), change to the "JavaEmployees\src" directory and run the following command

```
wsdl2java -uri http://<yourserver>:<yourport>/adabas_EmployeesMini?WSDL -o ..\
-p SoaG
```

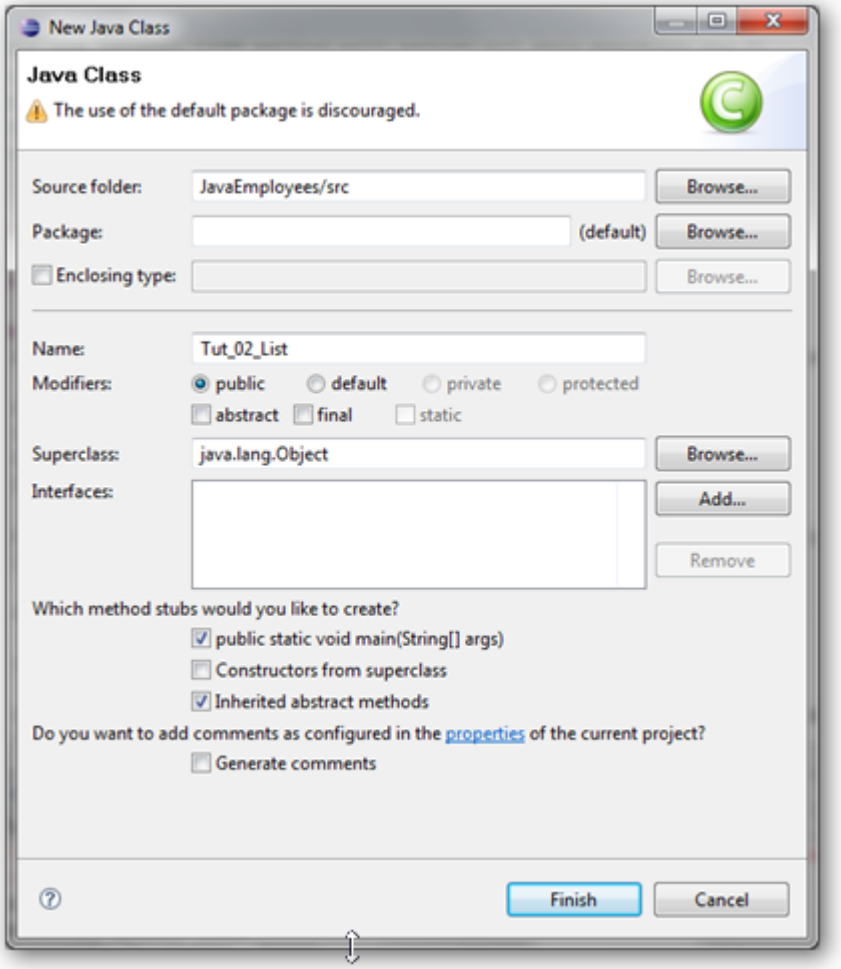


The following items are generated from a SOA Gateway WSDL:

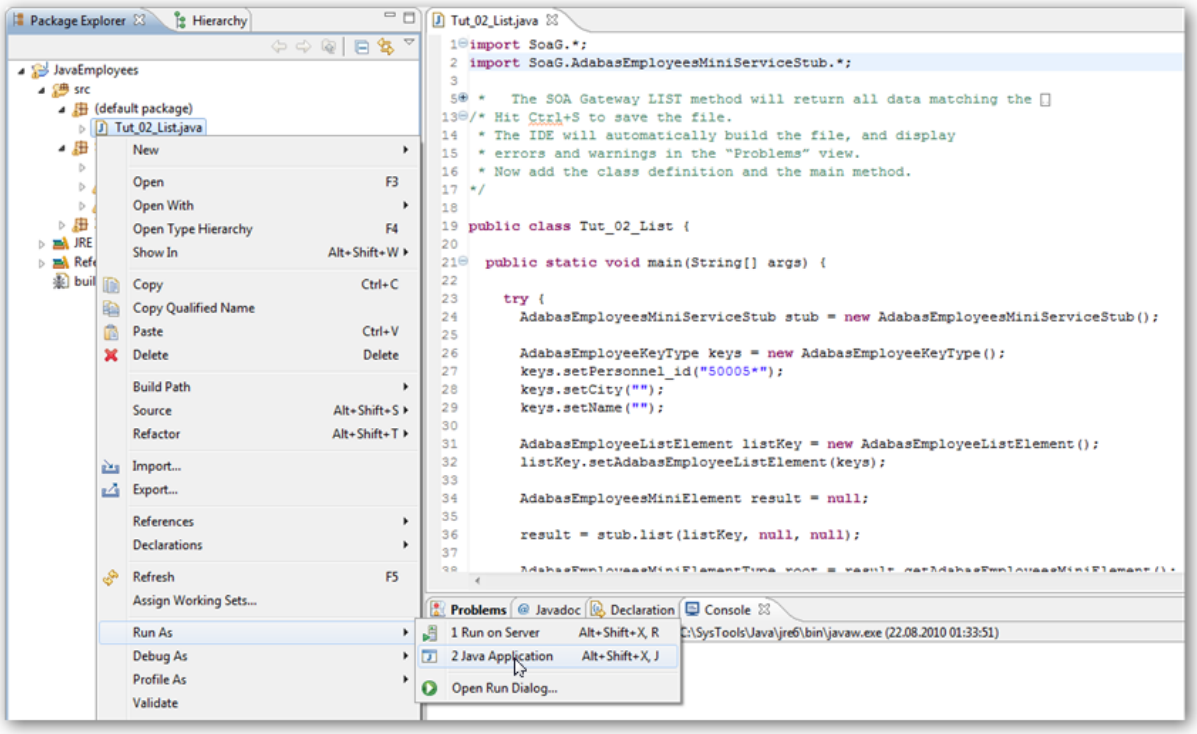
- A "Stub" class implementing all types and operations (ports / bindings)
- A CallbackHandler - a stub class (not used in this tutorial) providing hooks for client-side extensions to the generated result- and error handlers.
- A Fault class



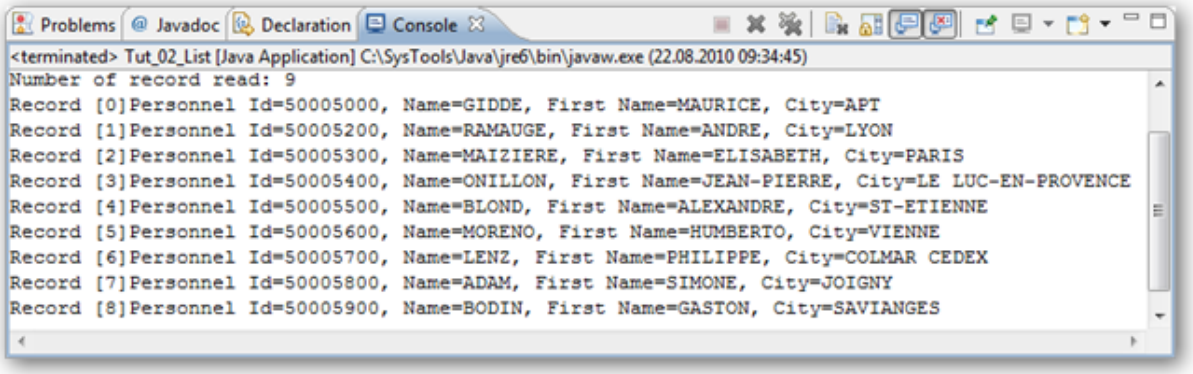
4. Add a new Java class named "Tut_02_List" to the project (**File** -> **New** -> **Class**), opt to create a "main" method, click "**Finish**".



5. Remove the generated code from the newly added class entirely, use (paste) the code from **Tut_02_List.java** to create your first test program accessing ADABAS data via SOA Gateway.
6. Run the program as a "Java Application"



7. The output appears in the "Console" window:



8. This sample selects all "Employees" records with a personnel-id of 50005nnn, you may want to experiment varying the key data, this is easily done by modifying the properties passed to the generated classes. E.g. try the following to list all records for "Employees" whose names start "SMI", living in cities with names starting "D".

```
keys.setPersonnel_id("");

keys.setName("SMI*");

keys.setCity("D*");
```

The following Java/Axis2 tutorial programs are available:

Tutorial	What it does
Tut_01a_Get.java	GET a single record by Personnel Id
Tut_01b_GetByISN.java	GET a single record by ISN
Tut_02_List.java	LIST some records
Tut_03a_SelectSimple.java	SELECT by a range of Personnel IDs
Tut_03b_SelectSorted.java	SELECT by a range of Personnel IDs and sort by Name
Tut_03c_SelectConversational.java	SELECT by multiple ranges of Personnel IDs, returned in "chunks" of 20 records each
Tut_04a_AddUpdateDelete.java	ADD + UPDATE + DELETE in "autocommit" mode
Tut_04b_AddTransactional.java	ADD in a transactional context

3 Accessing Adabas using the PHP SOAP extension

This tutorial

- Demonstrates how to access SOA Gateway from a PHP script
- Provides a number of PHP examples.

What is PHP ?

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

The PHP interpreter is available as source code or as pre-compiled binaries for major platforms, including most Linux™ distributions, Windows®, Mac OS X, and iSeries™.

The latest release is PHP 5 and is seeing increasing adoption. PHP 5 introduces improvements to the object model; also, the underlying memory management has been redesigned with multi-threading and performance in mind.

For more information about PHP, or to download the software, please refer to the **PHP homepage**.

New in PHP 5 is a built-in SOAP extension. It is supplied as part of PHP.

For this tutorial to work, you should have PHP 5 up and running in your Web server, see the install.txt document in the PHP distribution library for details.

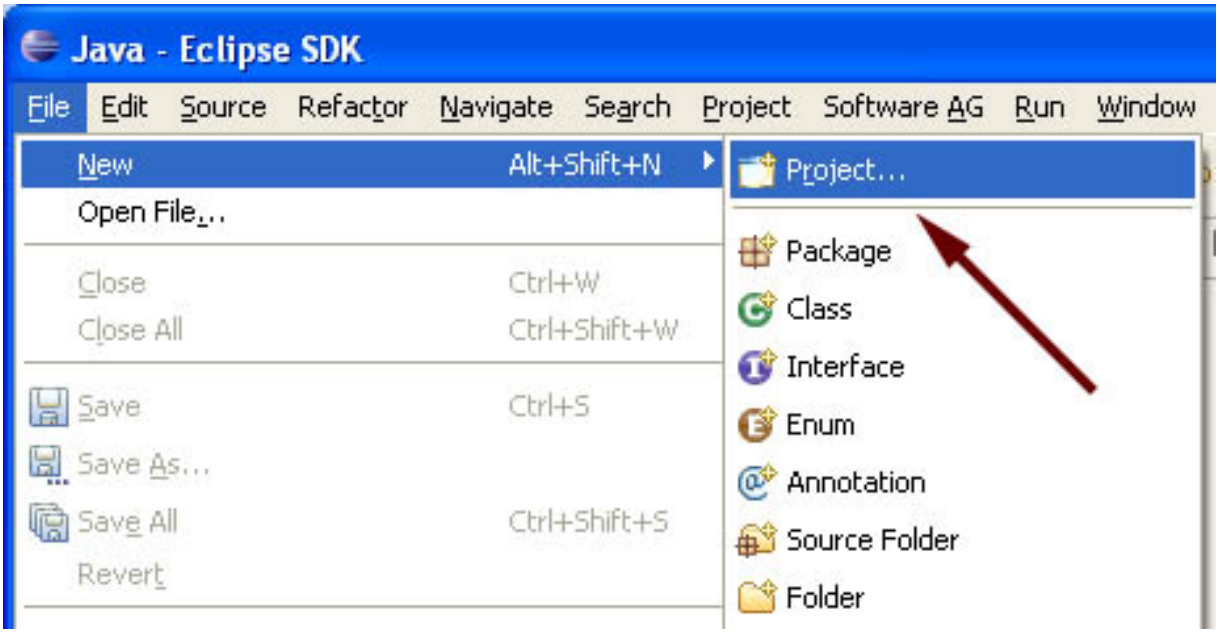
Installing the Eclipse PHP Development Tools

The Eclipse PHP Development Tools (PDT) are not absolutely necessary, but using Eclipse greatly simplifies the development process. This tutorial assumes the PDT to be installed.

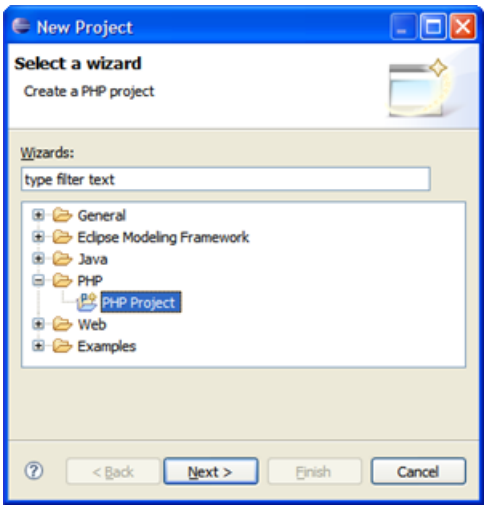
Please refer to the **Eclipse PDT project pages** for installation and configuration instructions.

Accessing Adabas from PHP

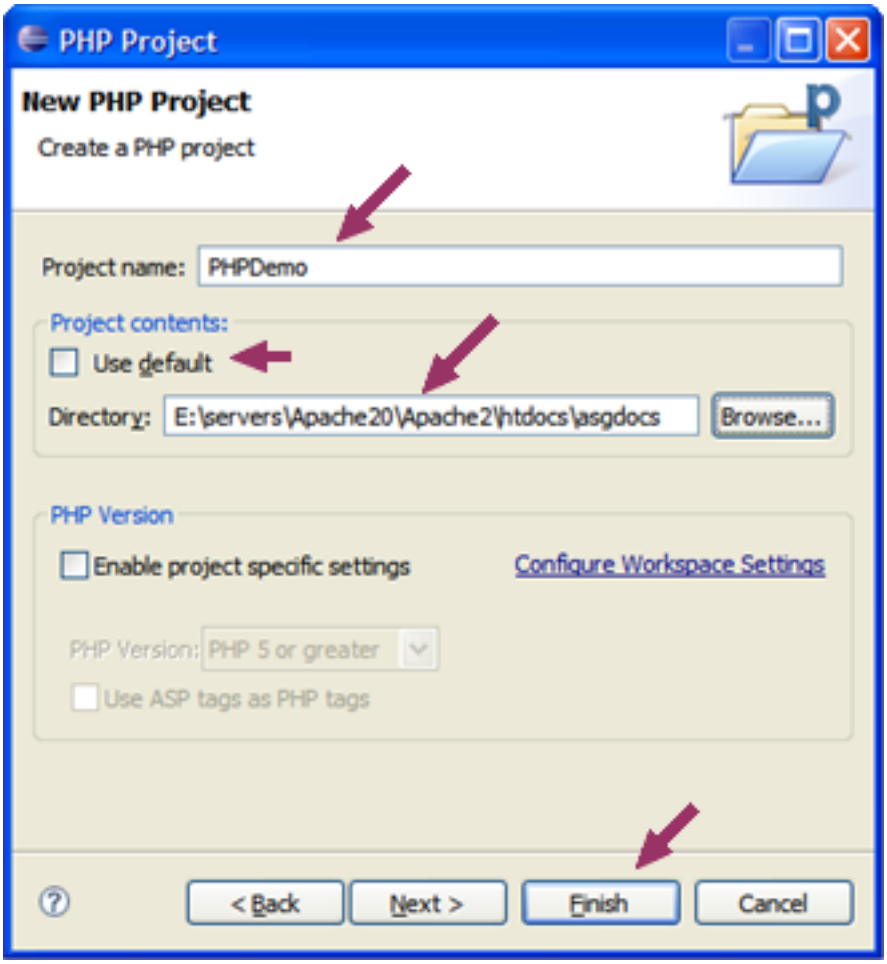
1. First, create a new project within your workspace.



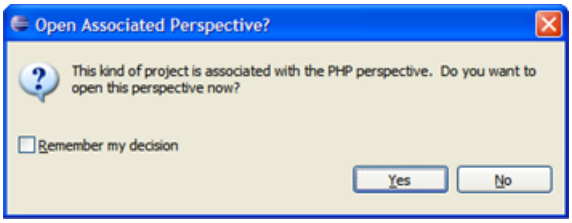
2. Opt to create a PHP Project, click **Next**



Give the project a name and unselect the "Use default" box. Browse to the default location of your html documents, in this case an Apache server document folder called "ASGdocs", click **Finish**

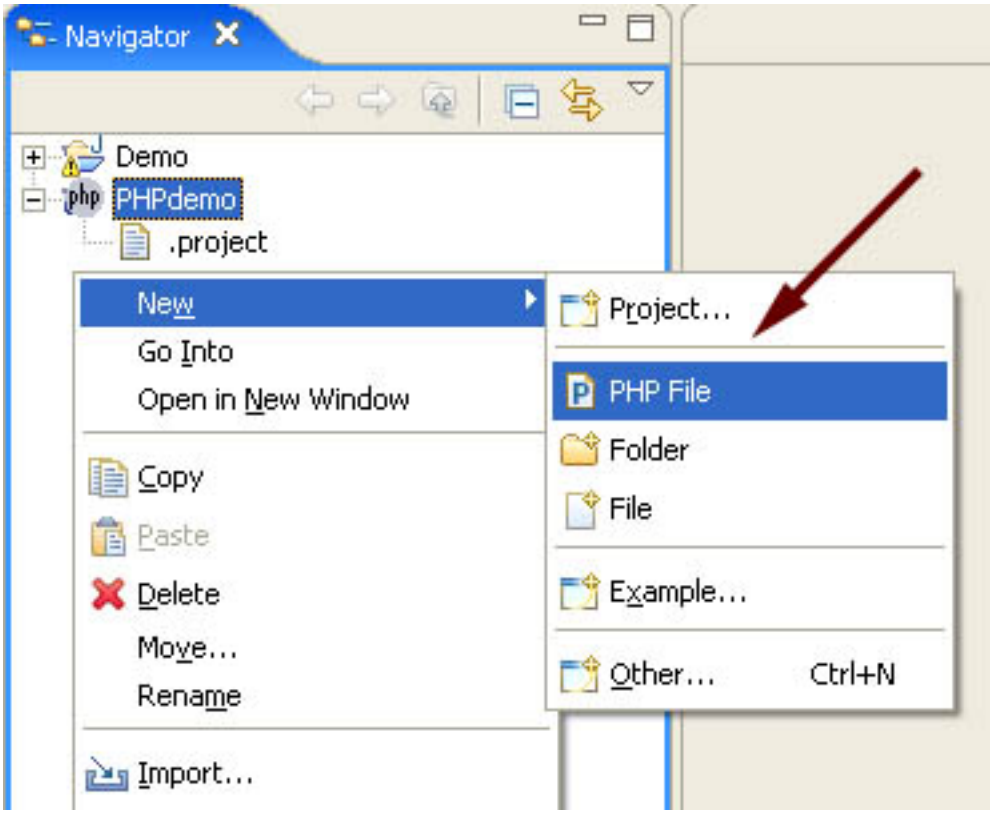


If you are asked to switch to the PHP perspective, opt to do so

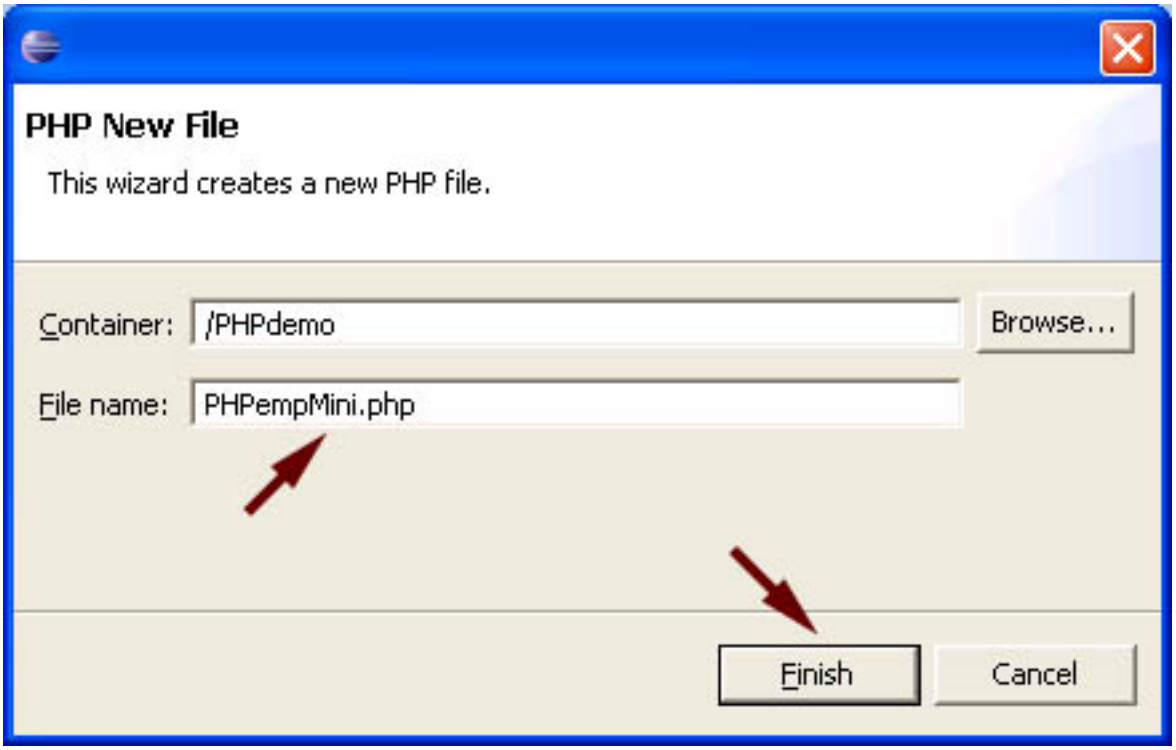


3. Create the PHP SOAP client

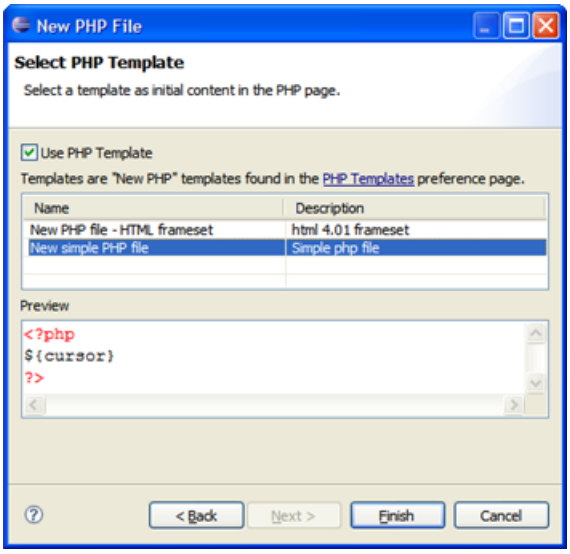
   Create a PHP script file by right-clicking into the Eclipse Navigator area, select **New** -> **PHP File**



Specify a File name, click **Finish**



Opt to create a "Simple PHP File"

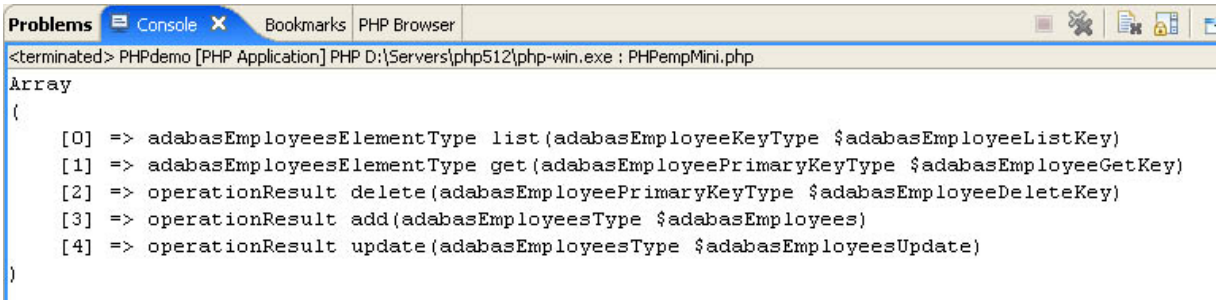


The PHP SOAP class to represent the Adabas Service is called SoapClient, the first step will be to instantiate SoapClient, passing the URL of an SOA Gateway WSDL as the parameter:

Now that we have instantiated our client we want to see what methods it provides and what parameters are required. Fortunately we can get PHP and the instantiated SoapClient class to do most of the work for us easily:

```
$functions = $soapclient->__getFunctions();
print_r($functions);
```

If you run this as a console application (via "Run") the output is much better formatted than running it in the PHP browser. The console window will show the following:



This shows that the service described by the WSDL provides five operations: list, get, delete, add and update; It also lists the required parameters and the responses given.

A description of the input and output parameters can be retrieve by calling the __getTypes class:

```
$types = $soapclient->__getTypes();
print_r($types);
```

The output will look like this:

This information is sufficient to construct the first simple call to the "list" operation.

First an array of the required input parameters needs to be constructed:

```
$AdabasEmployeeListKey = array (
'personnel_id'=> '300001*', 'firstName'=>'', 'name'=>'', 'city'=>'');
```

Ready to invoke the "list" operation as a method of the soapclient class:

```
$Adabasresponse = $soapclient->list($AdabasEmployeeListKey);
```

Now it is just a matter of taking the returned object and outputting the required results in a table:

```
echo "<br /><br />";
echo "<table border=1 cellpadding=5>";
echo "<tr><th>Personnel ID</th><th>Name</th><th>City</th><th>Address</th></tr>";

foreach ($Adabasresponse->adabasEmployees->adabasEmployee as $Employee) {

    echo "<tr>",
    "<td>$Employee->personnel_id</td>",
    "<td>$Employee->firstname $Employee->name</td>",
    "<td>$Employee->city</td>",
    "<td>";

    foreach ($Employee->address_line as $adline)
        echo "$adline<br />";

    echo "</td></tr>";

}

echo "</table>";
```

Run this in the built in PHP Browser (activated with "**Window**" -> "**Show view**" and select "**PHP Browser**") or an external browser: http://<your_localhost_url>/PHPempMini.php:

Personnel ID	Name	City	Address
30000100	Lloyd	Ilkeston	276 Cotmanhay Road Ilkeston Derbyshire
30000107	O'Brien	Nottingham	25 Main Street Radcliffe-On-Trent Nottingham
30000110	Stilwell	Nottingham	11 Westwood Road Sneinton Nottingham
30000112	Finch	Stamford	13 Kings Road Stamford Lincs
30000114	King	Peterborough	14 Main Street Peterborough Northants
30000124	Grebby	Derby	23 The Paddock Kegworth Derby
30000125	Maguire	Derby	113 Derby Road Spondon

PHP Examples

The following PHP examples can be copied from here, moved to your web server's DocumentRoot (for example) and executed:

- **First steps**
- **List Employees**
- **List Employees descending**
- **List Employees sorted**
- **List by Sub Descriptor**
- **List by Super Descriptor**
- **Add an Employee**
- **Get an Employee**
- **Delete an Employee**
- **A simple PHP form for accessing Adabas**
- **All-in-one PHP form accessing the Employees file**

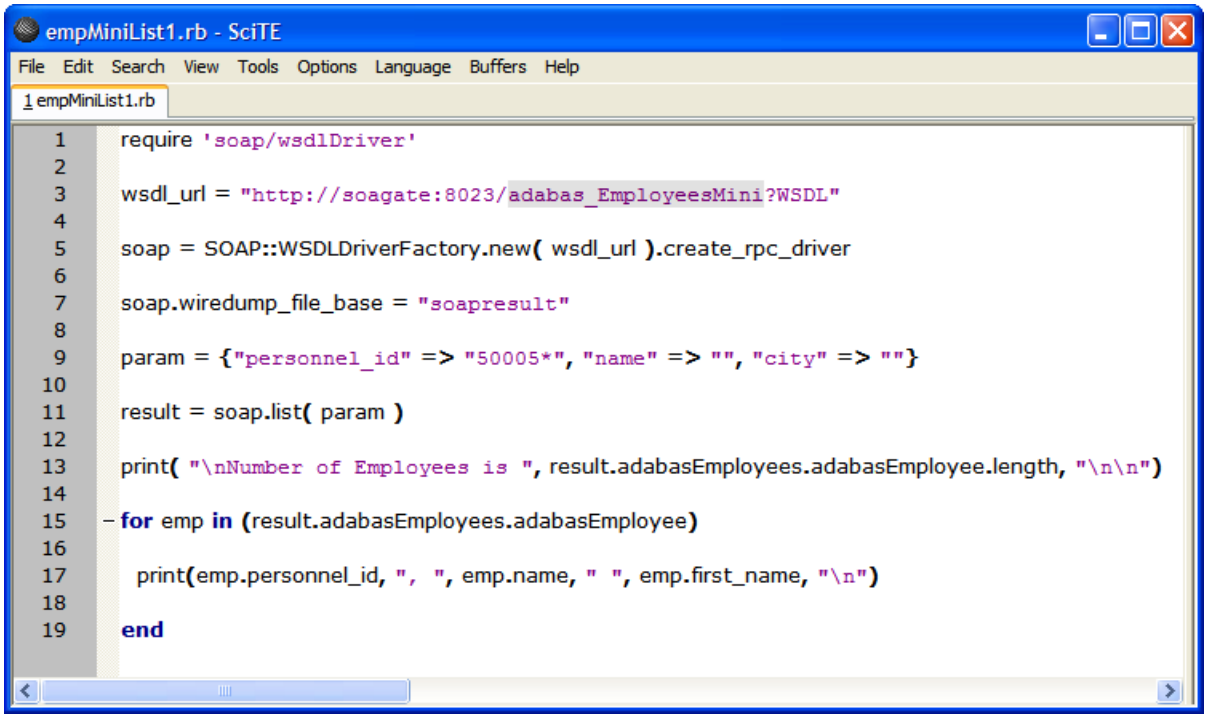
4 Accessing a SOA Gateway Resource from a Ruby program

Ruby (**downloadable here**) is an Open Source object oriented language with a very simple, yet powerful SOAP interface.

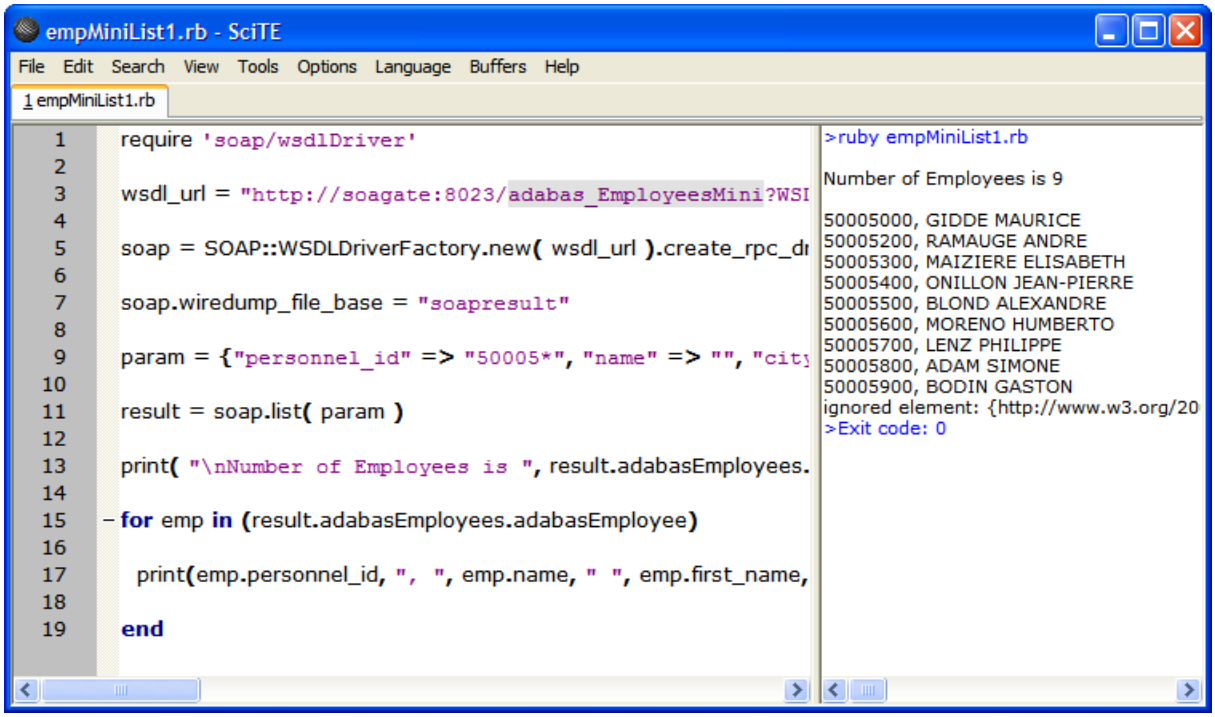
Running a Ruby program

Ruby comes with a very powerful editor, SciTE, which not only allows editing, but also compiling & execution of programs.

This is an example of a complete Ruby program listing all Employees with personnel-IDs starting with 50005*, using the 'adabas_EmployeesMini' Resource that comes with the sample SOA Gateway configuration.



When running this program, by either pressing the F5 key, or selecting Tools->Go from the SciTE menu, an output window will be attached to the editing window, and show the result of the Ruby query against SOA Gateway


empMiniList1.rb - SciTE
File Edit Search View Tools Options Language Buffers Help
1 empMiniList1.rb
1 require 'soap/wsdlDriver'
2
3 wsdl_url = "http://soagate:8023/adabas_EmployeesMini?WSI
4
5 soap = SOAP::WSDLDriverFactory.new(wsdl_url).create_rpc_dr
6
7 soap.wiredump_file_base = "soapresult"
8
9 param = {"personnel_id" => "50005*", "name" => "", "city
10
11 result = soap.list(param)
12
13 print("\nNumber of Employees is ", result.adabasEmployees.
14
15 for emp in (result.adabasEmployees.adabasEmployee)
16
17 print(emp.personnel_id, ", ", emp.name, " ", emp.first_name,
18
19 end
>ruby empMiniList1.rb
Number of Employees is 9
50005000, GIDDE MAURICE
50005200, RAMAUGE ANDRE
50005300, MAIZIERE ELISABETH
50005400, ONILLON JEAN-PIERRE
50005500, BLOND ALEXANDRE
50005600, MORENO HUMBERTO
50005700, LENZ PHILIPPE
50005800, ADAM SIMONE
50005900, BODIN GASTON
ignored element: {http://www.w3.org/20
>Exit code: 0

5 Creating a sample C# application

Tutorial: A sample C# application listing "Employees"

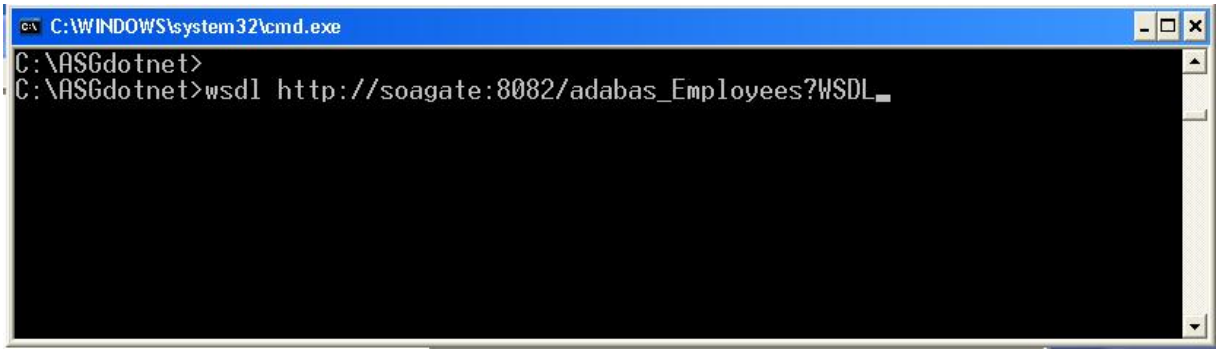
With a service description (WSDL), a proxy class can be created with the .NET Framework SDK Wsdl.exe tool. A XML Web service client can then invoke methods of the proxy class, which communicate with SOA Gateway over the network by processing the SOAP messages sent to and from the SOA Gateway server. The proxy class handles the work of mapping parameters to XML elements and then sending the SOAP message over the network.

Wsdl.exe is a Microsoft .NET tool which is used to create proxies for C#, Visual Basic .NET and JScript .NET. In this tutorial, we will be generating C#.

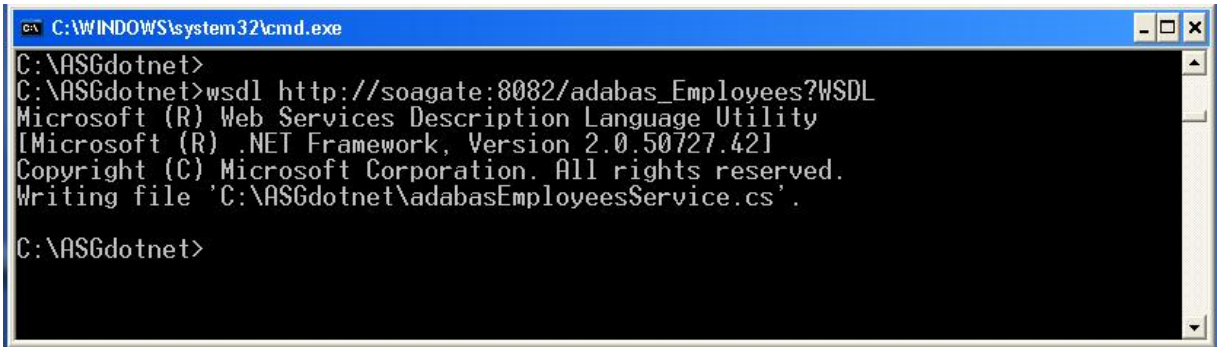
These are the steps required to generate the C# wrapper class using Wsdl.exe and create / run a program listing records from the Adabas demo file "Employees" using the generated proxy class:

1. From a command prompt, execute Wsdl.exe, specifying the URL / URI of the SOA Gateway DataSource to be exposed, append ?WSDL to instruct theSOA Gateway server to return the WSDL, not data:

   If the Wsdl.exe is not found, open the Visual Studio command prompt via the Start Menu. This location depends on what packages are installed, but often resides under "Microsoft Visual C# " or "Microsoft .NET Framework SDK ".



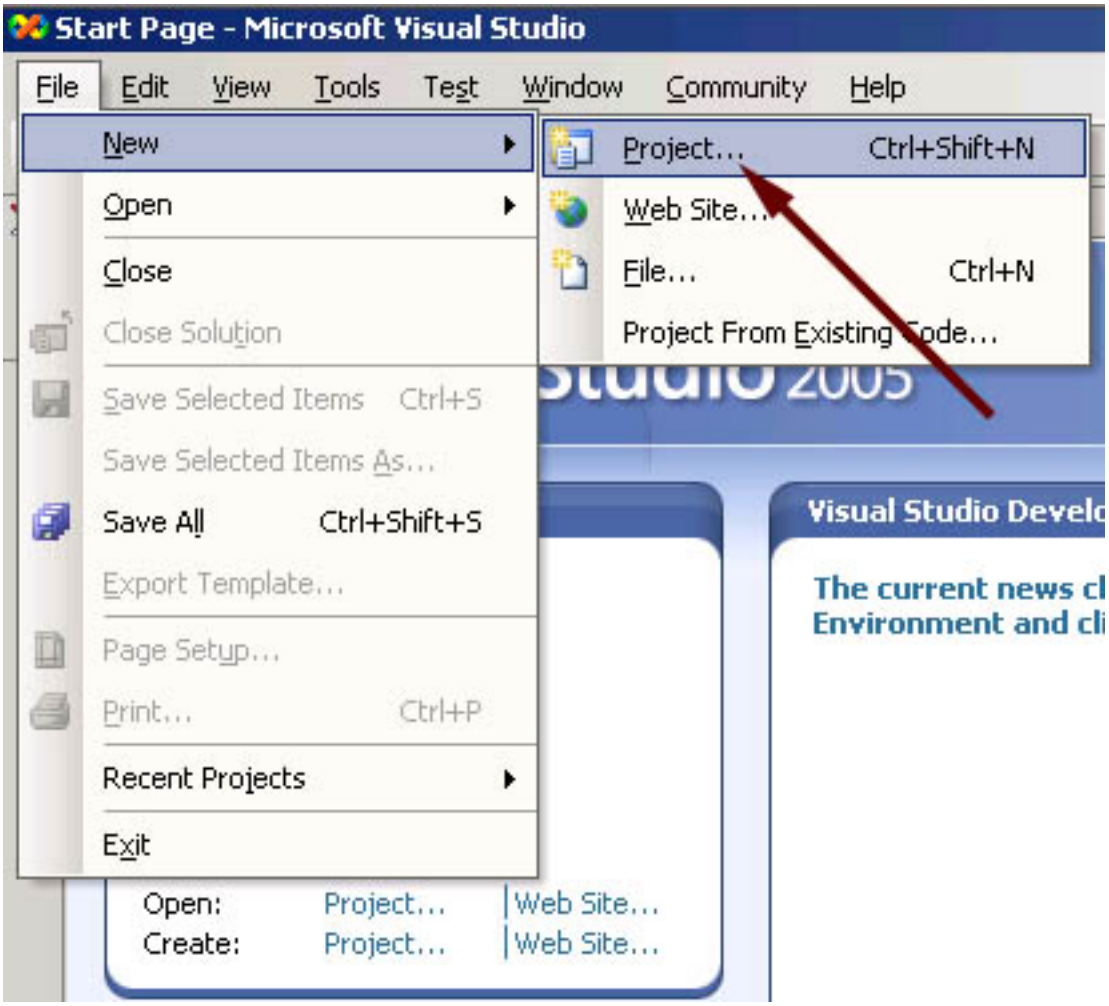
2. A single source file is generated, its name is <rootElementName>Service.cs, in this case the "root element" within the XSD is "adabasEmployees", thus the name of the proxy class source file adabasEmployeesService.cs



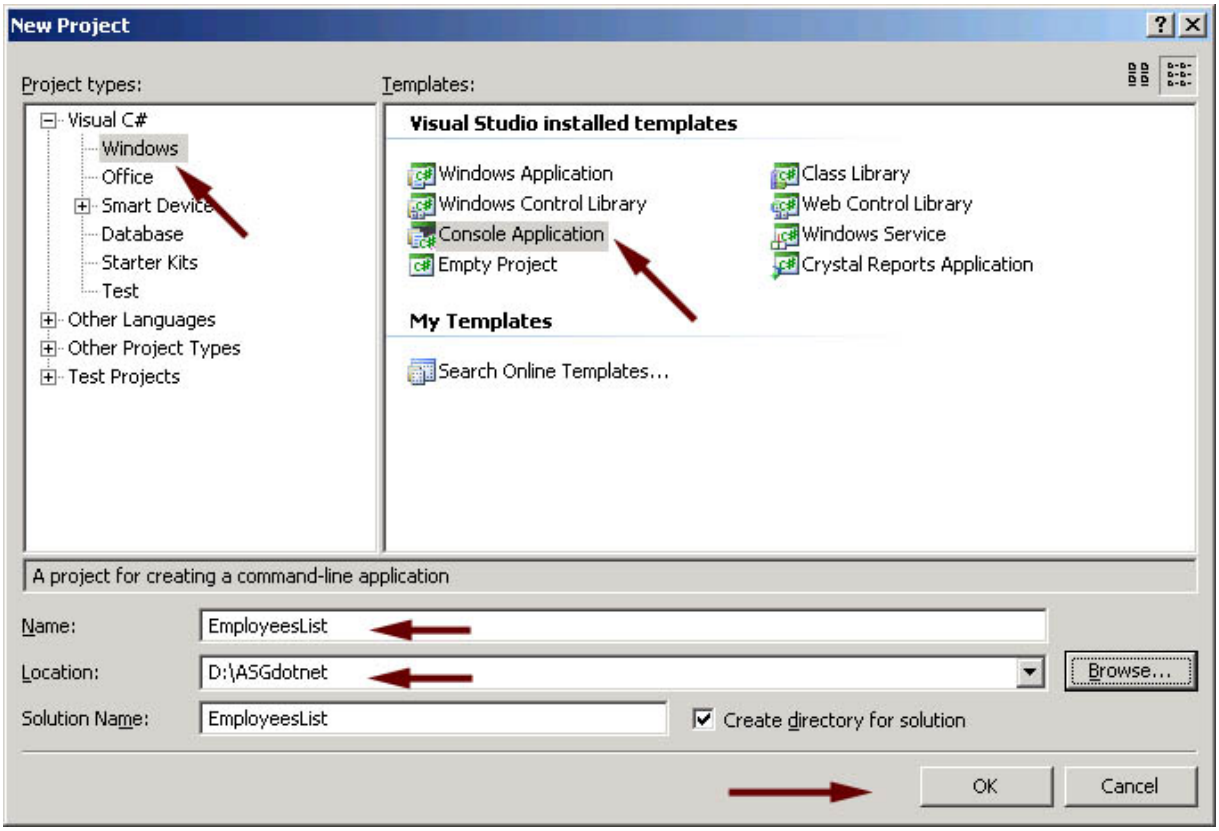
This file contains a proxy class exposing both synchronous and asynchronous methods for each SOAP operation provided by SOA Gateway for the DataSource. For instance, for the *list* operation, the proxy class has the following methods: *list*, *Beginlist*, and *Endlist*. The list method of the proxy class is used to communicate with SOA Gateway synchronously, but the Beginlist and Endlist methods are used to communicate with the SOA Gateway server asynchronously.

For more information about asynchronous communication with a Web Service please refer to the .NET documentation.

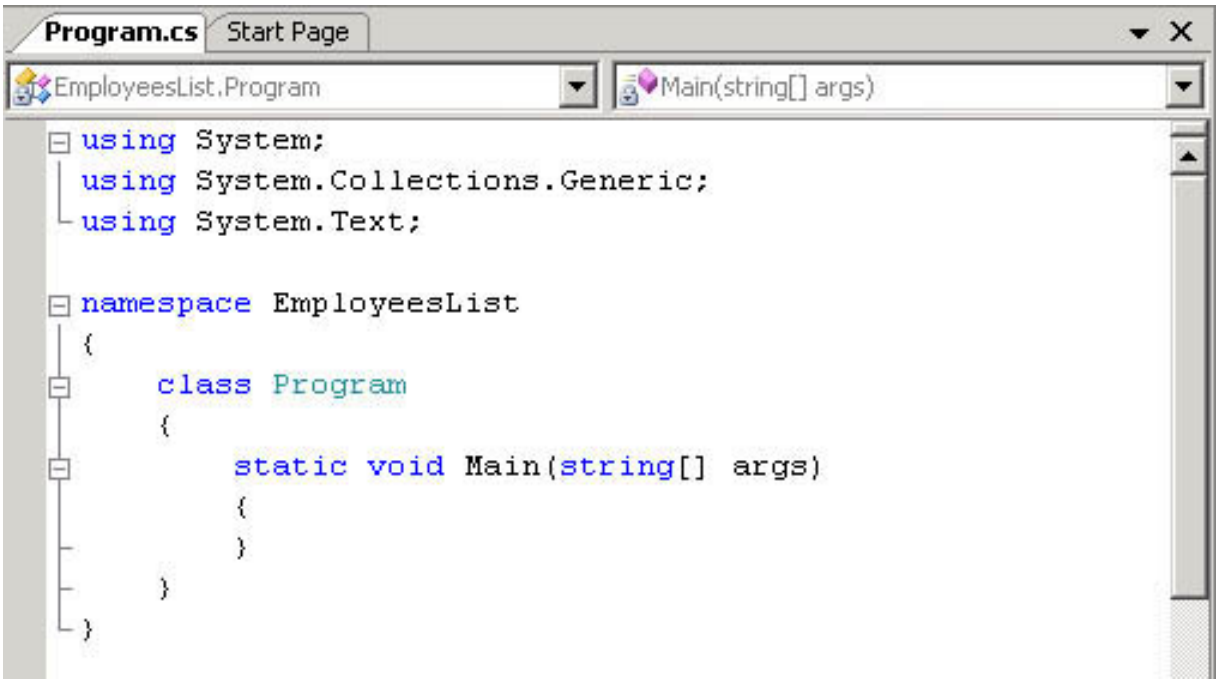
3. Start MS Visual Studio, create a new project with **File** -> **New - Project** (or the shortcut Ctrl+Shift+N):



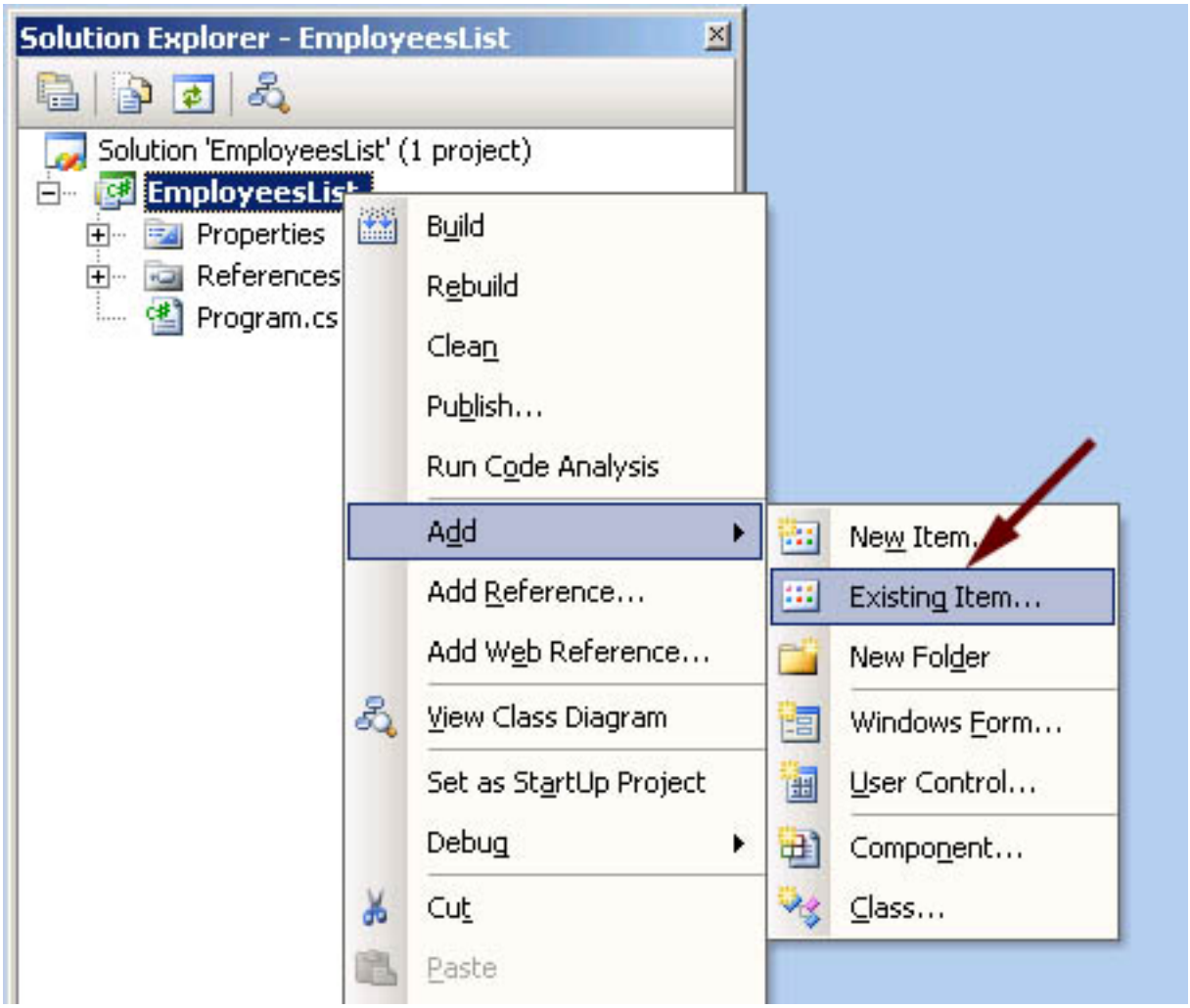
Create a C# Console Application, assign a name to it, specify the storage location, click **OK**



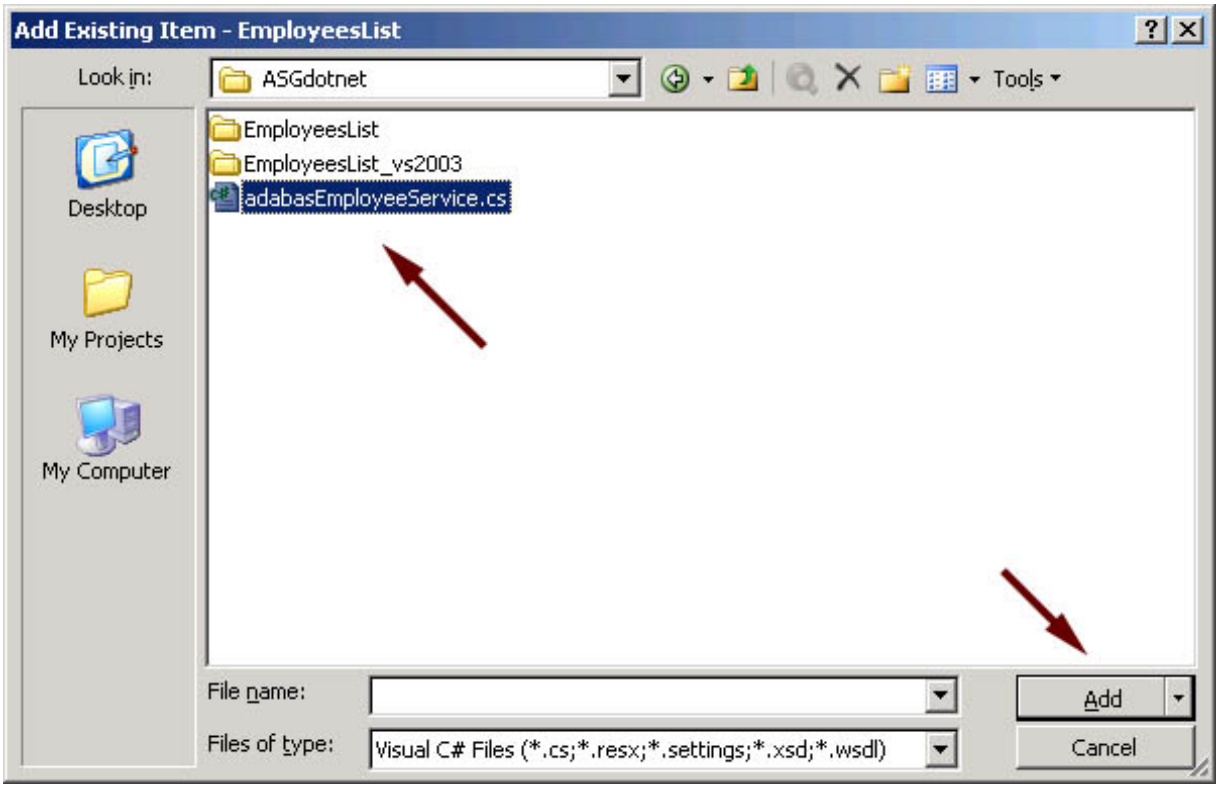
A skeleton class file has been generated into your project workspace, with the required class definition and an empty *Main* method



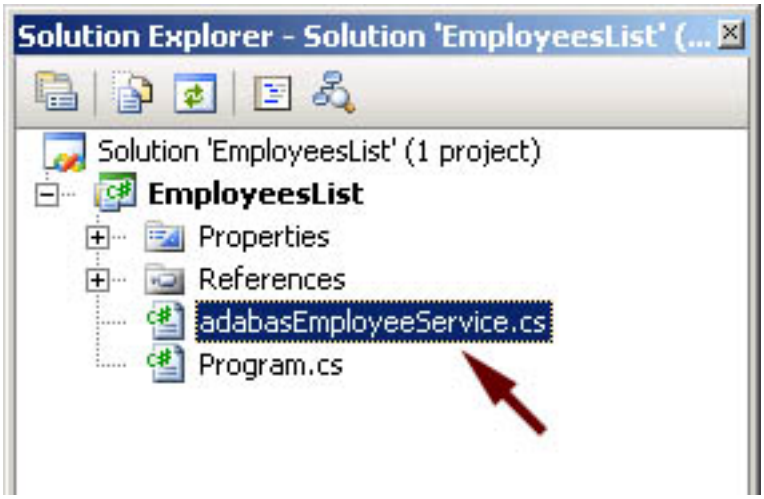
4. First of all, import the generated proxy into the project, right-click on the project name, select **Add** -> **Existing Item**



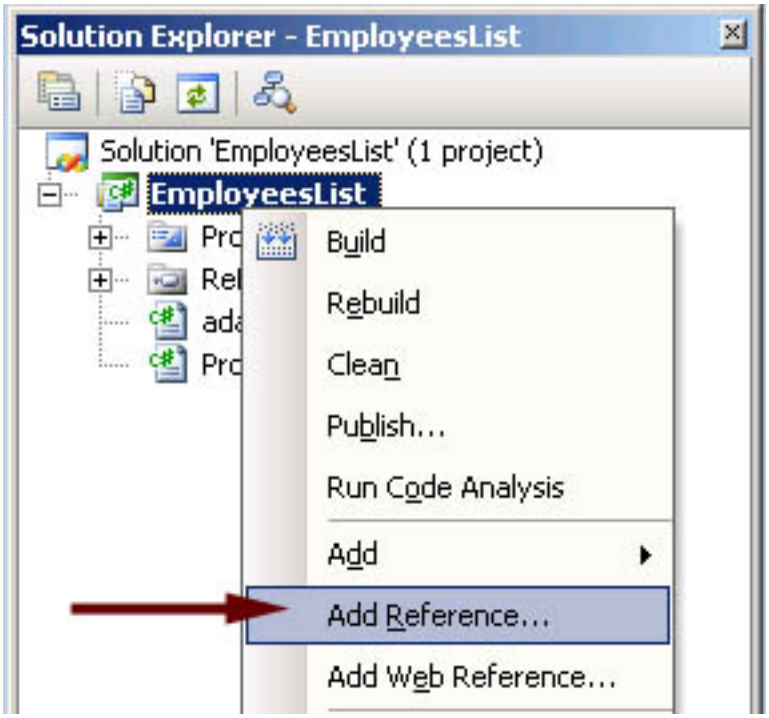
select the AdabasEmployeeService.cs proxy, click **Add**



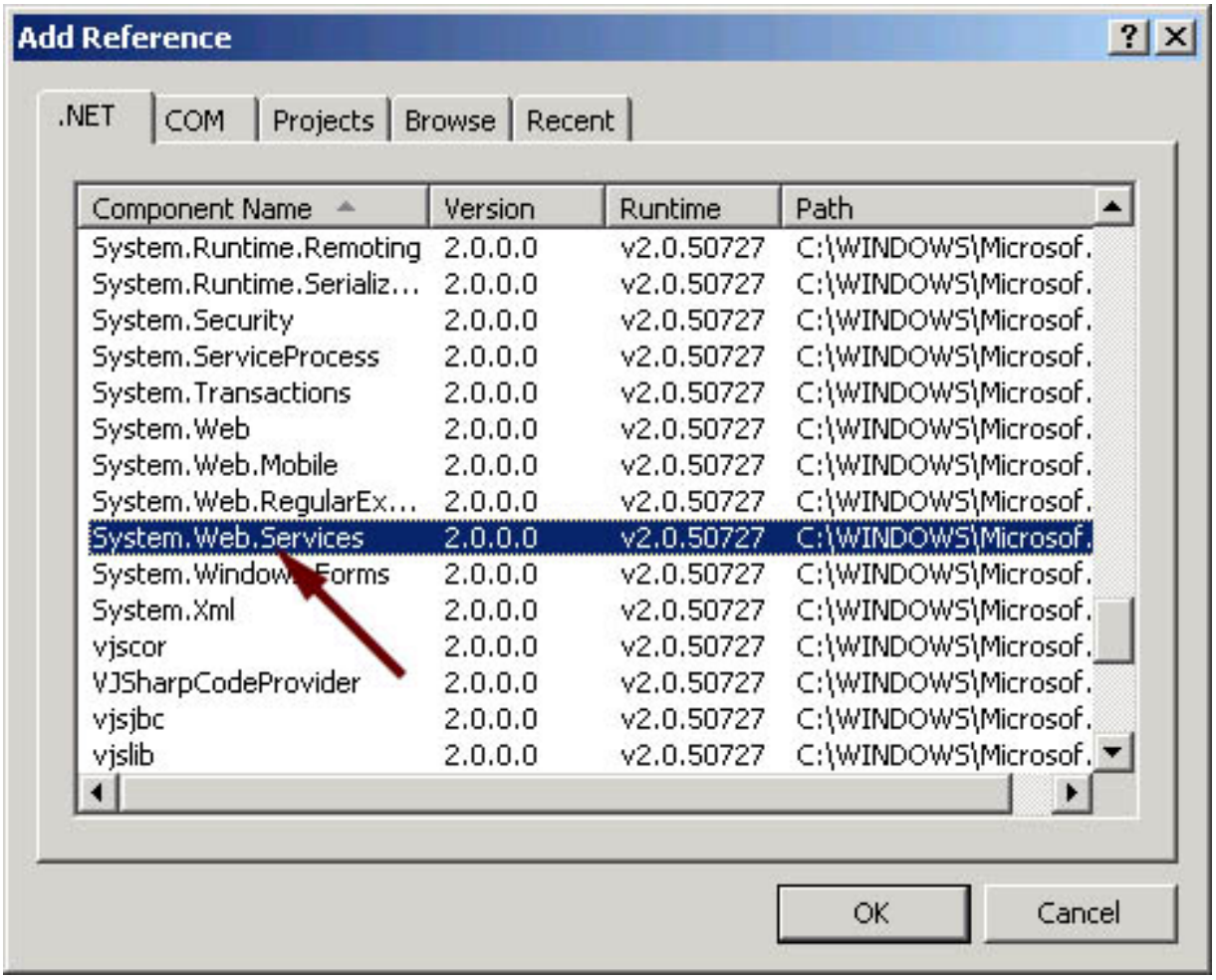
The proxy has been added to the project



You now need to add a reference to the .NET System.Web.Services component implementing the SOAP interface. In the project explorer, right click on the project name, select **Add Reference**



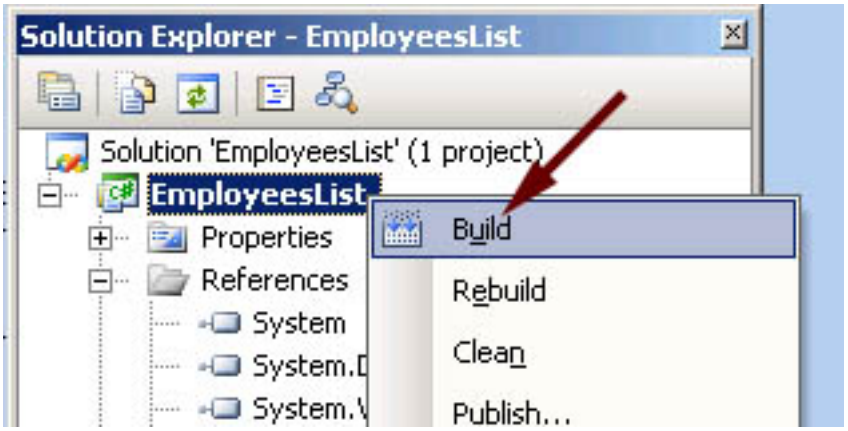
Scroll down to System.Web.Services, click to select it, click to select it, click **OK** to import the reference



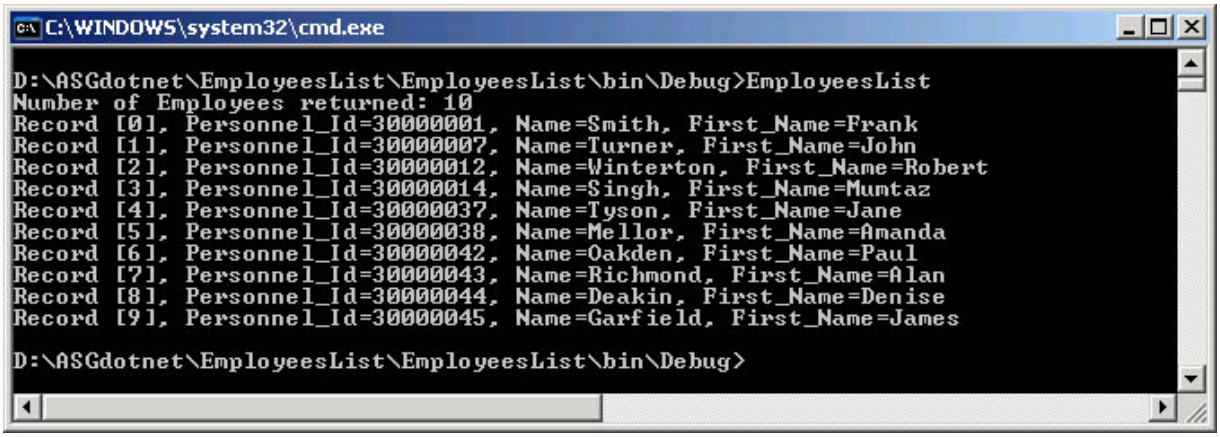
5. Remove the generated code from the newly added class entirely, use (paste) the code from **ASGDemo.cs** to create your first test program accessing Adabas data via SOA Gateway.

```
Program.cs
EmployeesList.Program                                   Main(string[] args)
using System;
using System.Collections.Generic;
using System.Text;

namespace EmployeesList
{
    class Program
    {
        static void Main(string[] args)
        {
            adabasEmployeesService emplService;
            adabasEmployeeKeyType keys;
            adabasEmployeesElementType result;
            adabasEmployeeType[] empl;

            keys = new adabasEmployeeKeyType();
            keys.personnel_id = "300000*";

            emplService = new adabasEmployeesService();

            try
            {
                result = emplService.list(keys);
            }
            catch (SystemException ex)
            {
                Console.WriteLine("exception: " + ex.Message);
                return;
            }

            empl = result.adabasEmployees;
```

6. Build the application. Right-click on the project name in the project explorer, click **Build**



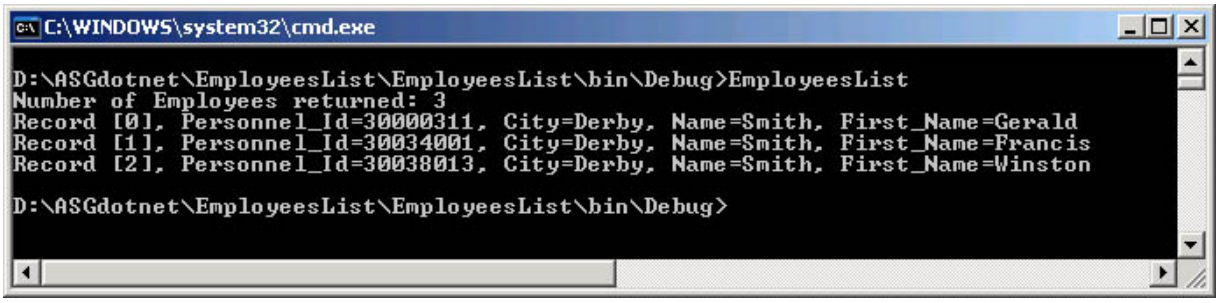
7. Open a command window, change to the project's build-directory Execute the compiled console application, EmployeesList, the output will look as follows:



8. This sample selects all "Employees" records with a personnel-id of 4000004n, you may want to experiment varying the key data, this is easily done by modifying the properties passed to the generated classes. E.g. try the following to list all records for "Employees" whose names start "SMI", living in cities with names starting "D".

```
keys.name = "SMI*";
keys.city = "D*;
```

The output will look like this:



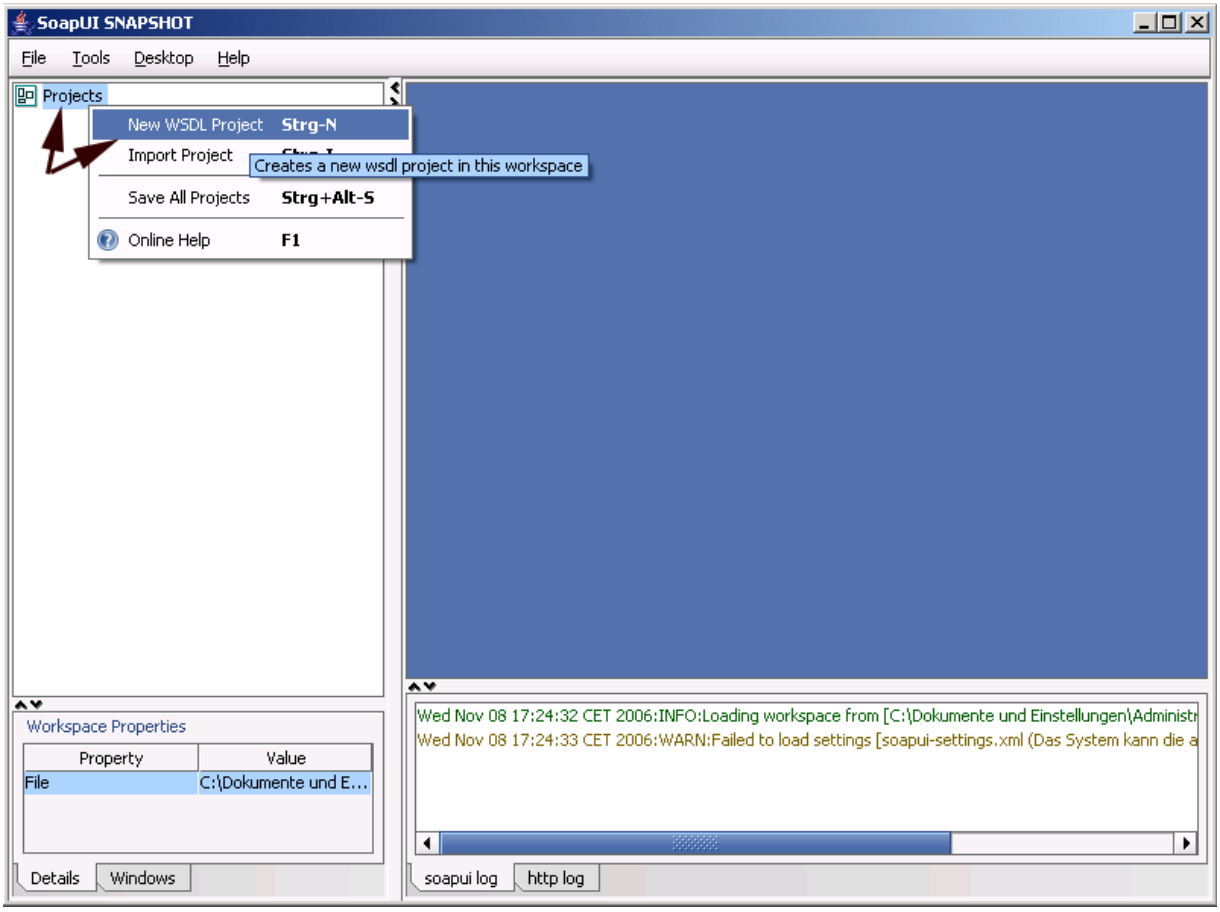
6 Accessing Adabas through SoapUI

This simple scenario demonstrates how to invoke operations on an Adabas DataSource exposed as a "Web service" through SOA Gateway from SoapUI.

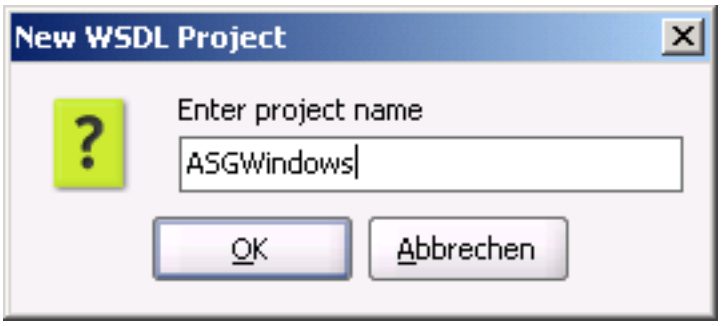
soapUI is a (freeware) desktop application for inspecting, invoking, developing and functional / load / compliance testing of web services over HTTP and can be downloaded **here**.

Additionally, soapUI can be integrated into the Eclipse framework, read **here** for more information.

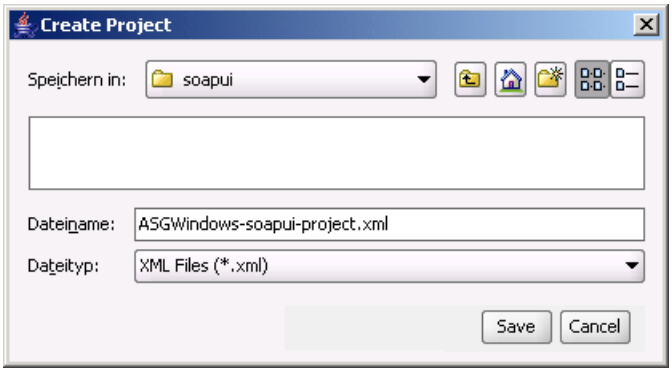
1. When starting soapUI for the very first time an "empty" workspace is generated, right-click on the top-level **Projects** item, select **New WSDL Project** to create your first soapUI project.



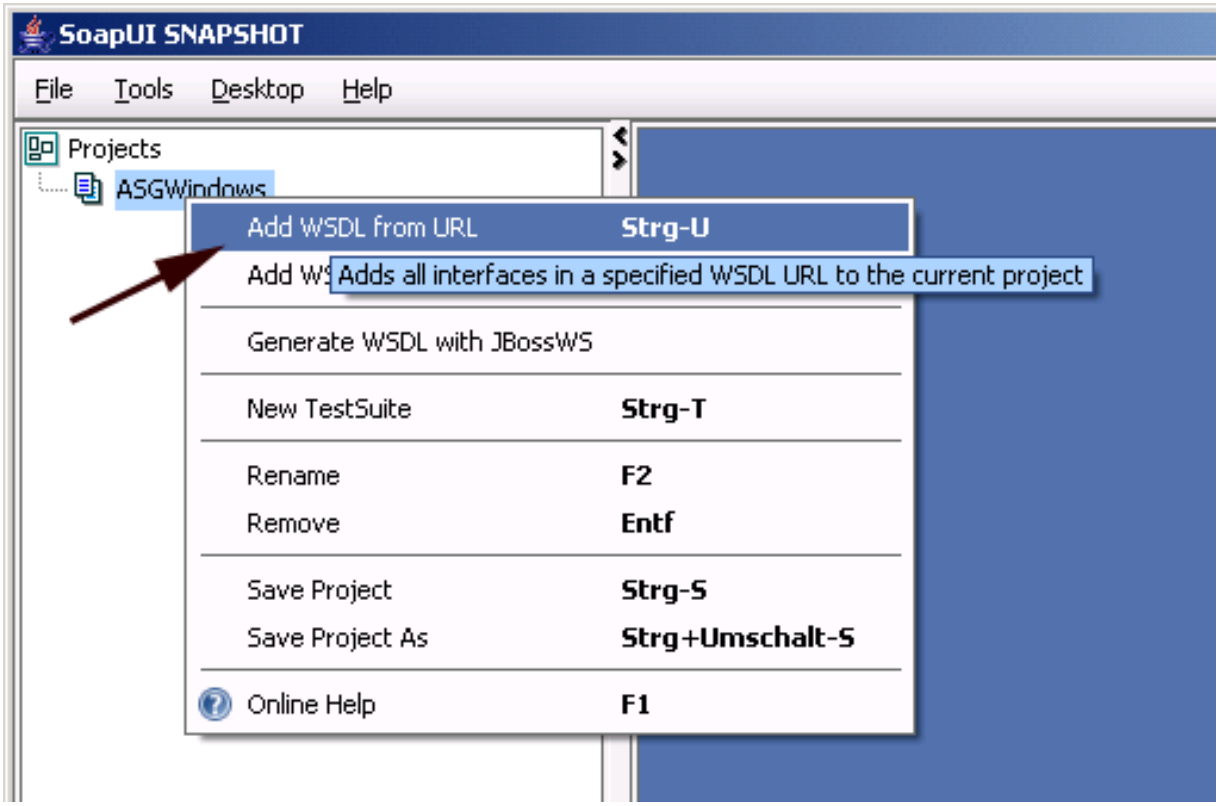
You will be prompted for a project name, enter one and click OK:



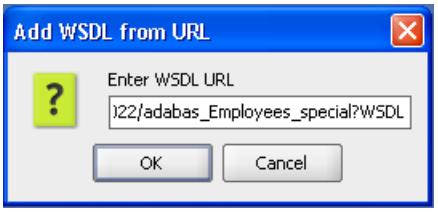
Specify a target location for the project files, click **Save**:



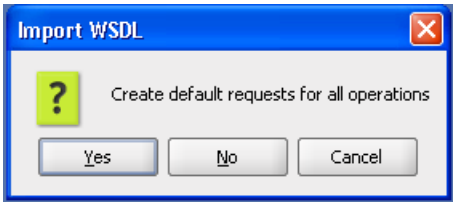
2. Now import an SOA Gateway "web service" into the newly created project, right-click the project name, then select **Add WSDL from URL**



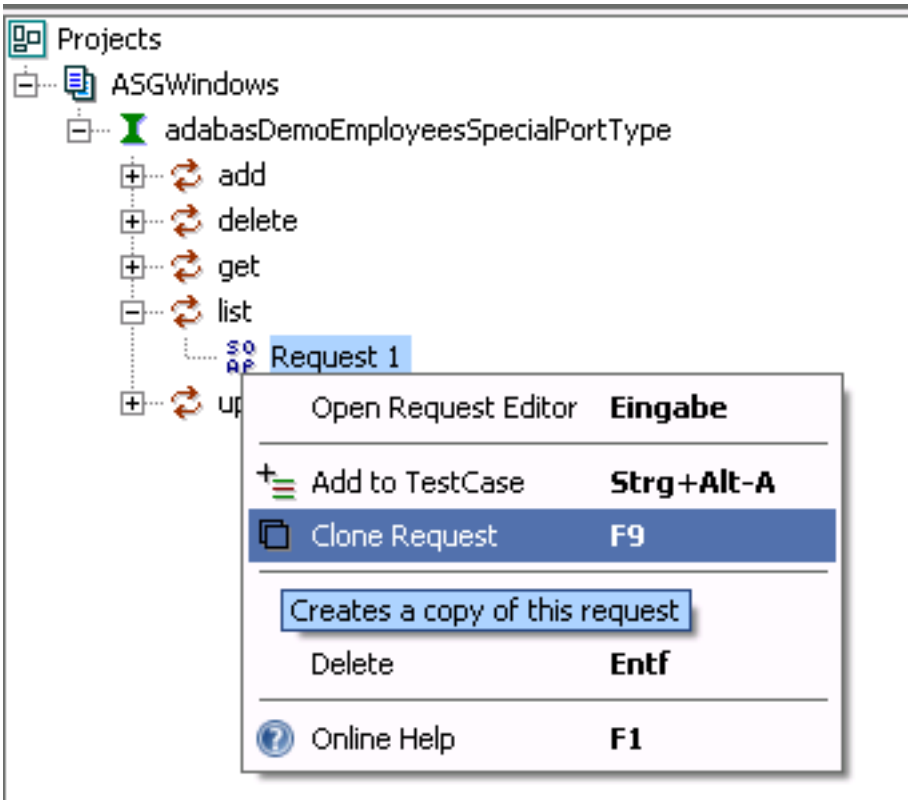
Enter the URI of the adabas_Employees_special resource, as in http://<yourASGserverhost>:<your-ASGserverport>/adabas_Employees_special?WSDL and click **OK** to import the webservice definitions from the resources / WSDL



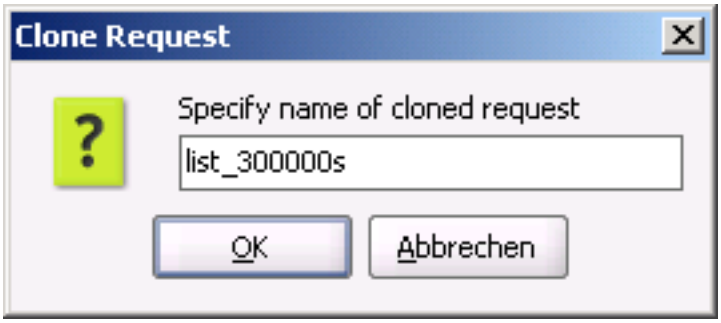
You will be asked if default requests for all operations are to be created, click **Yes**.



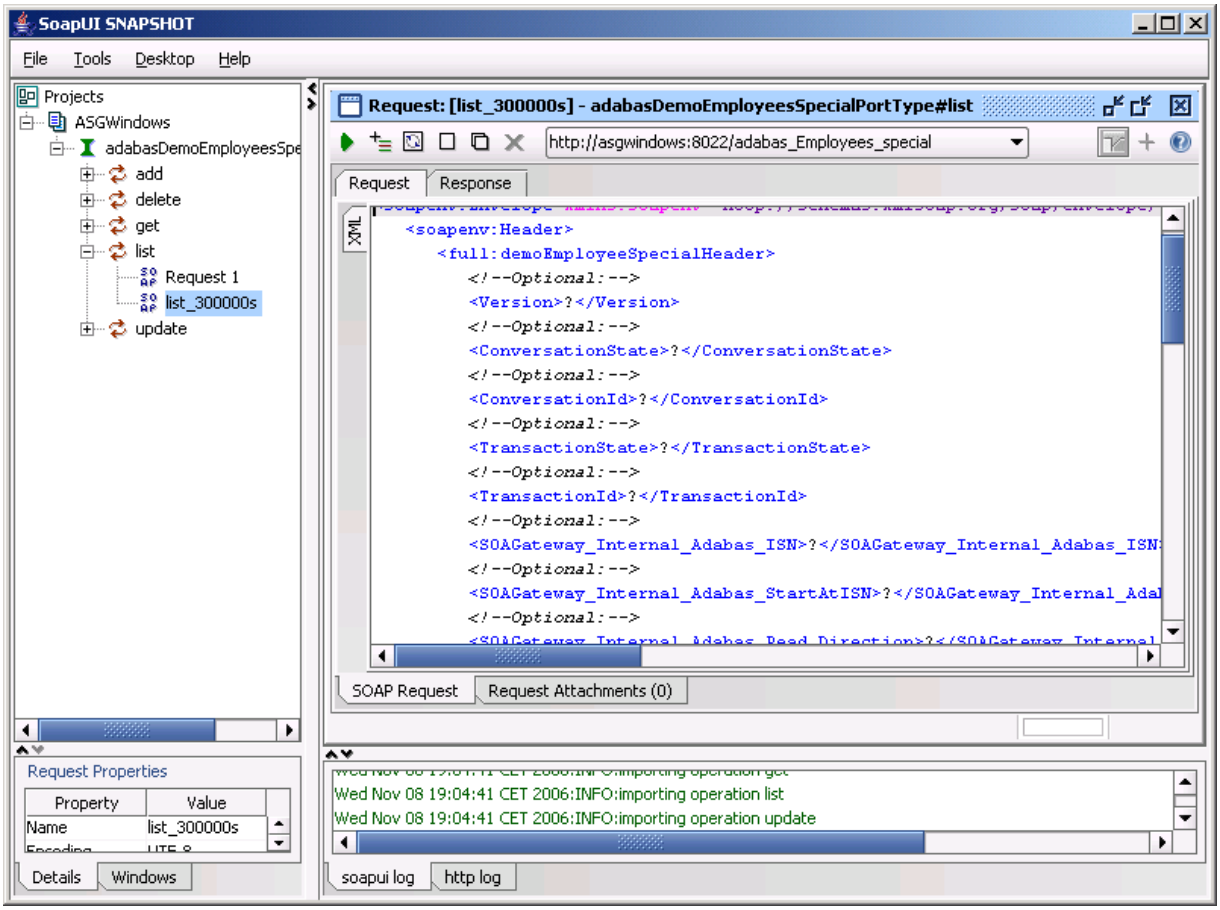
3. Default requests have been generated for all SOA Gateway operations for a webservice, undfold the "list" request by clicking the **plus** sign left of it, right-click the created **Request 1**, select **Clone Request**, this allows for unlimited duplication of the original request, which may be desirable when testing various options or "canning" requests.



Assign a name to the cloned request, click **OK**

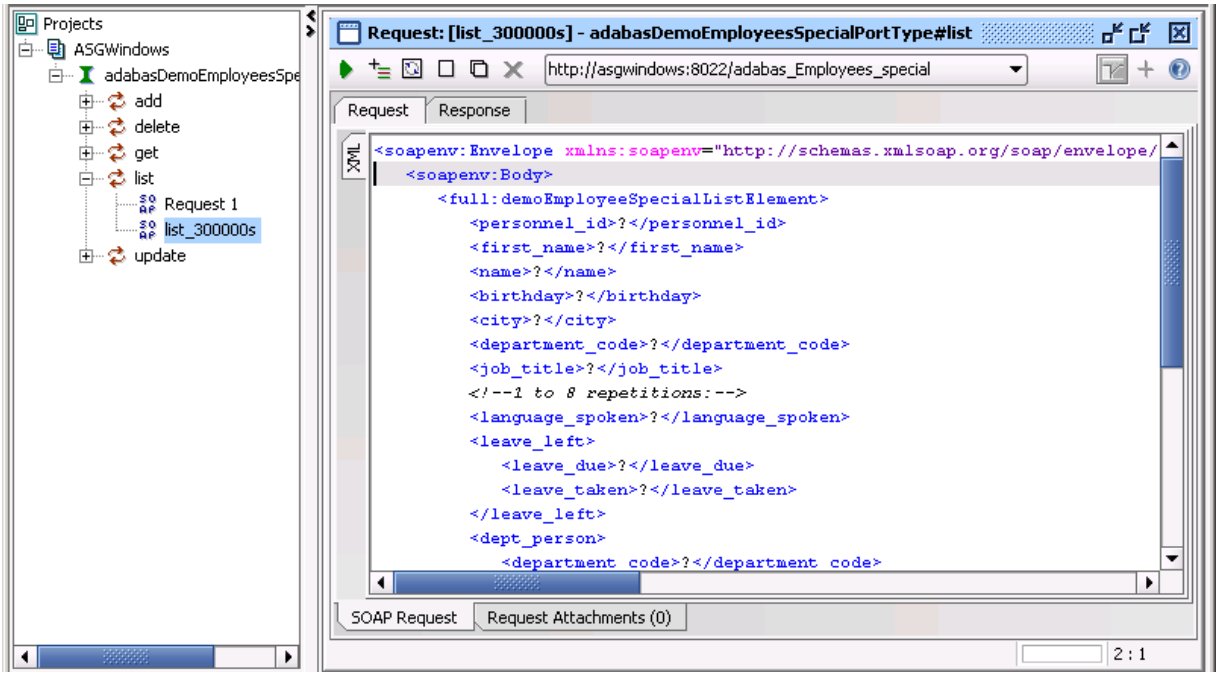


4. soapUI now opens the request document.

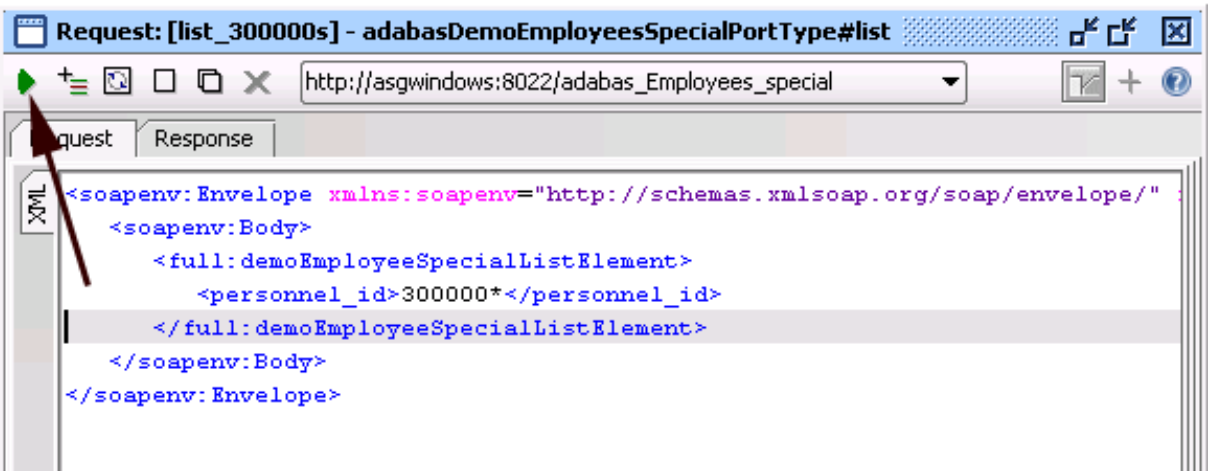


Remove the soapenv:Header section, everything from, and including, `<soapenv:Header>` to `</soapenv:Header>`

This leaves you with the soap Body section, which holds all key information:

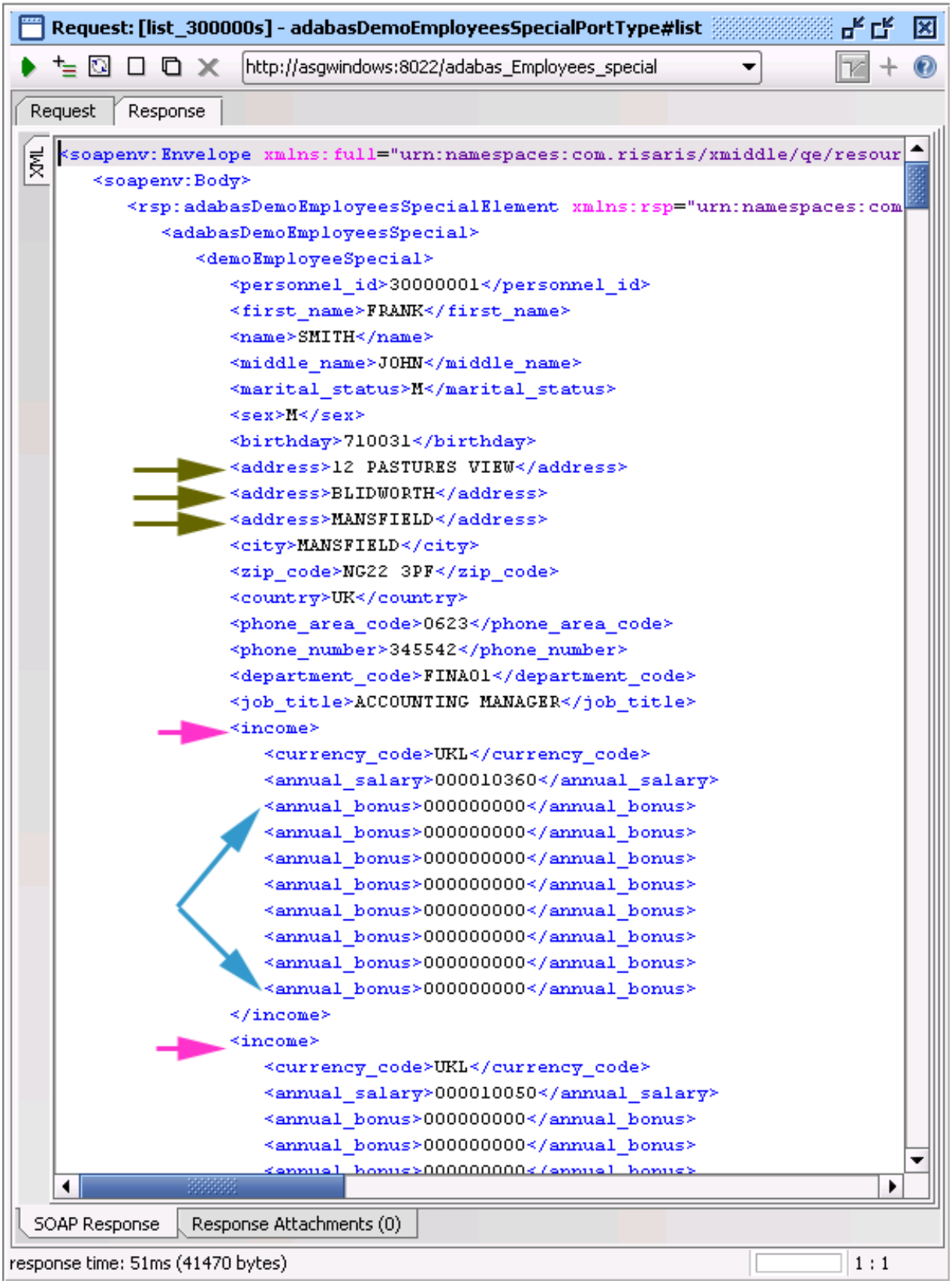


Remove all key elements but the personnel_id, enter 300000* as the key value, the request should look like this now, then click the **green arrow** to send the request to your SOA Gateway server:



5. The response will look like this - formatted XML

   This response shows "simple" as well as repeating fields like MUs (multiple value fields, here: olive arrows), PEs (periodic groups, pink arrows) and MUs within PEs (blue arrows):


Request: [list_300000s] - adabasDemoEmployeesSpecialPortType#list
http://asgwindows:8022/adabas_Employees_special
Request
Response
XML
<soapenv:Envelope xmlns:full="urn:namespaces:com.risaris/xmiddle/qe/resour
<soapenv:Body>
<rsp:adabasDemoEmployeesSpecialElement xmlns:rsp="urn:namespaces:com
<adabasDemoEmployeesSpecial>
<demoEmployeeSpecial>
<personnel_id>30000001</personnel_id>
<first_name>FRANK</first_name>
<name>SMITH</name>
<middle_name>JOHN</middle_name>
<marital_status>M</marital_status>
<sex>M</sex>
<birthday>710031</birthday>
<address>12 PASTURES VIEW</address>
<address>BLIDWORTH</address>
<address>MANSFIELD</address>
<city>MANSFIELD</city>
<zip_code>NG22 3PF</zip_code>
<country>UK</country>
<phone_area_code>0623</phone_area_code>
<phone_number>345542</phone_number>
<department_code>FINA01</department_code>
<job_title>ACCOUNTING MANAGER</job_title>
<income>
<currency_code>UKL</currency_code>
<annual_salary>000010360</annual_salary>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
</income>
<income>
<currency_code>UKL</currency_code>
<annual_salary>000010050</annual_salary>
<annual_bonus>000000000</annual_bonus>
<annual_bonus>000000000</annual_bonus>
SOAP Response
Response Attachments (0)
response time: 51ms (41470 bytes)
1 : 1

7 Preparing for the LOBs (Large OBjects) samples

While the basic samples use the well-known 'Emplyoees' file delivered with Adabas, the LOBs demo programs require loading of a small Adabas file containing LOB fields.

Note: LOB access requires a minimum Adabas version of v6 on OpenSystems platforms, Adabas v8 on mainframe platforms.

Loading the LOB file into an OpenSystems database

These are the steps to load the demo LOB file into an OpenSystems (Windows, *IX) Adabas database:

- Save **FILE90.FDT** on the target system.
- Save **FILE90.FDU** on the target system, adjust the dbid, file and lobfile parameters according to your needs.
- Set the environment variable FDUFDT, for example on a Linux system: `export FDUFDT=FILE90.FDT`
- Run the command: `adafdu <FILE90.FDU`

The demo 'base' and 'LOB' files are now loaded, which can be verified with the adarep utility: `adarep db=<yourdbid>,cont`

The LOB demo file is now ready to be used.

Loading the LOB file into a Mainframe database

The steps to create the LOB demo file in a mainframe Adabas database are as follows:

- Use **this FDT** as input to ADACMP, specifying dummy input (DDEBAND). Sample JCL can be found on the ADAvrm.JOBS library distributed with Adabas.
- Run the ADALOD utility, **these parameters** may serve as template input to the utility. Adjust the file number and size specifications based on your requirements.

Important: Note that Adabas space / buffer parameters may need to be increased for LOB access, please consult the Adabas documentation for a description of required changes to your Adabas nucleus parameters.

8 Using LOBs (Large OBjects) with soapUi

This tutorial shows how to add, delete and get BLOBs using SOA Gateway and Adabas.

SOA Gateway uses the **MTOM** specification to send/receive the XML and binary data to/from the required web service. This involves attaching the required binary file(s) to the SOAP message, and then transforming this into a MIME message to send across the wire.

For the purposes of this tutorial, we will use **soapUi** to send and receive messages to our web service.

Other web service clients may also be used, such as PHP with the **WSO2 Web Services Framework** extension

This tutorial assumes the following

- You have at least soapUi v1.7 installed.
- You are using a version of Adabas which support LOBs
- You have set up a LOB file at FNR 90.

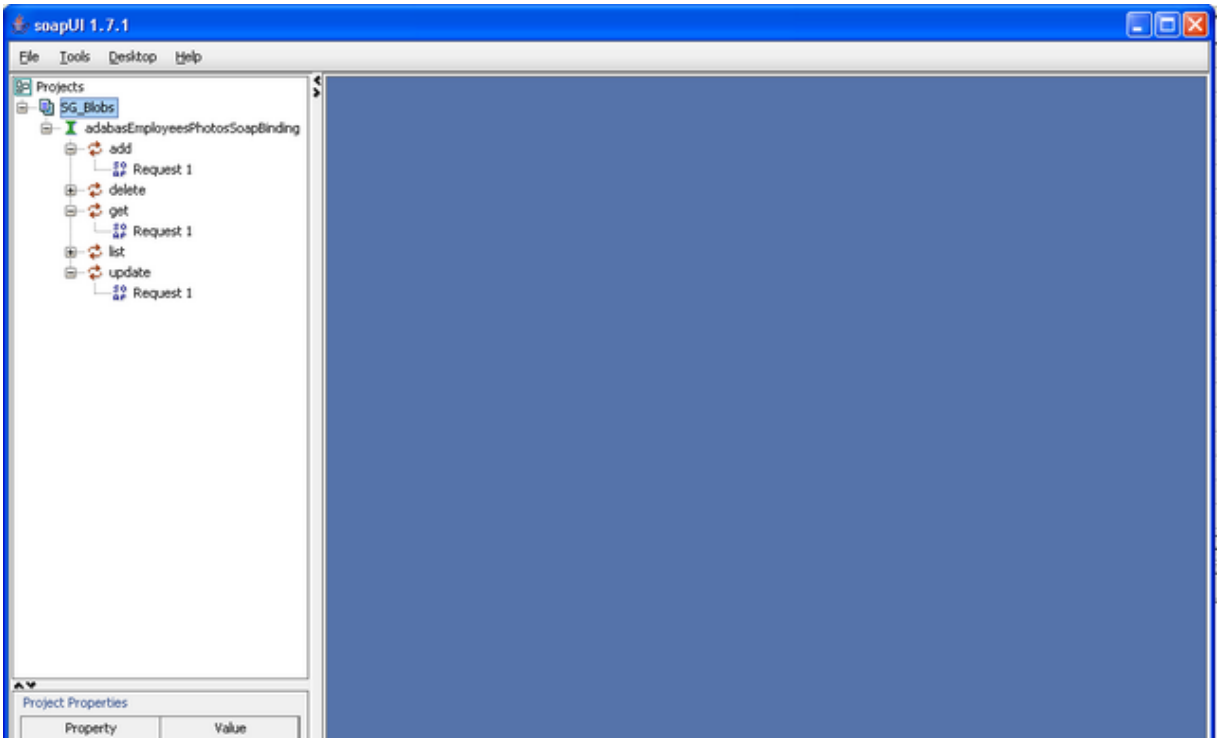
1. Start the Control Center and add a new resource.

   Enter the following information

   - Name: adabas_blobs
   - DataView : Select 'adabas_photoblobs', which is one of the sample definitions delivered with SOA Gateway
   - DatabaseId: 212 (or the DBID relevant on your system)
   - FileNumber: 90

   Publish your changes to the server.
2. Start soapUi, and add the WSDL for the adabas_blobs service. If you are unfamiliar with soapUi, please run through **this tutorial** first

3. You should now have a screen similar to this

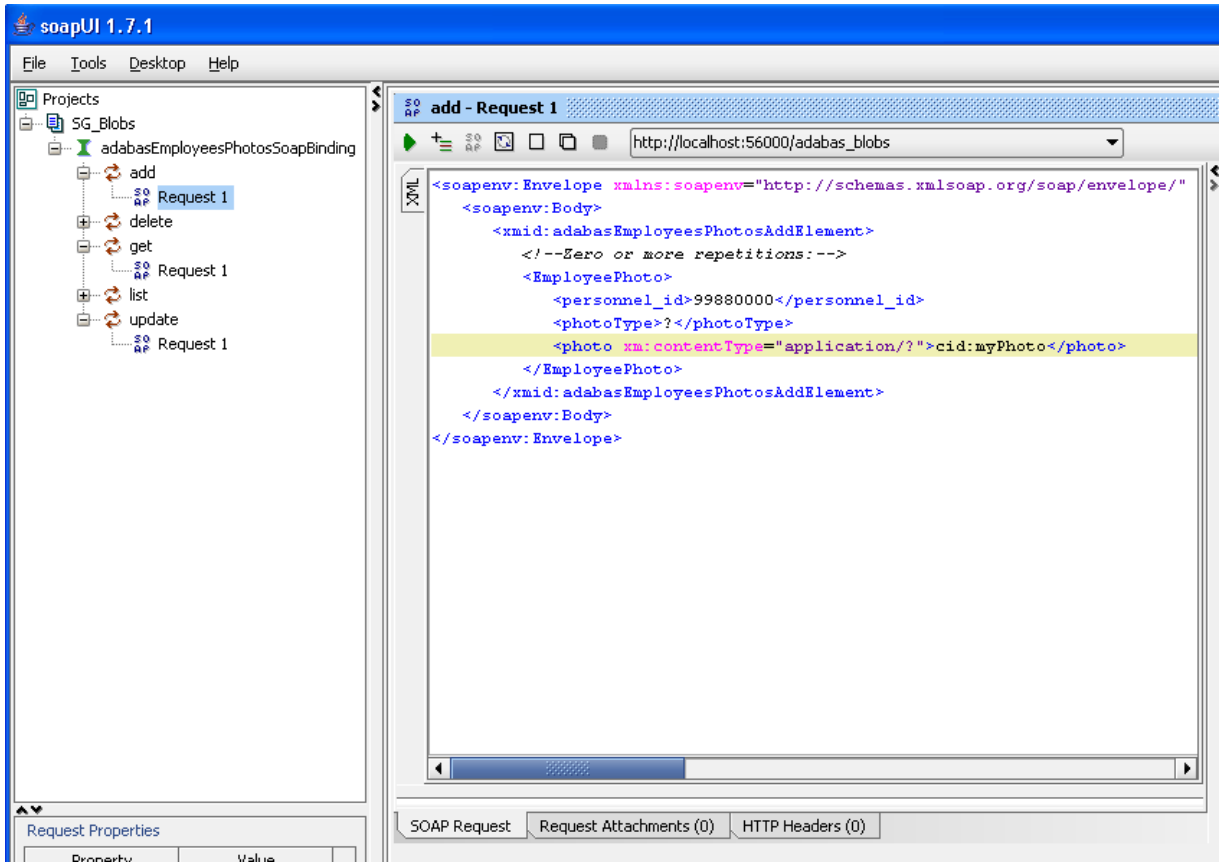


   Under **add** double-click **Request 1**

4. Remove the <soapenv:Header> element and all child elements.

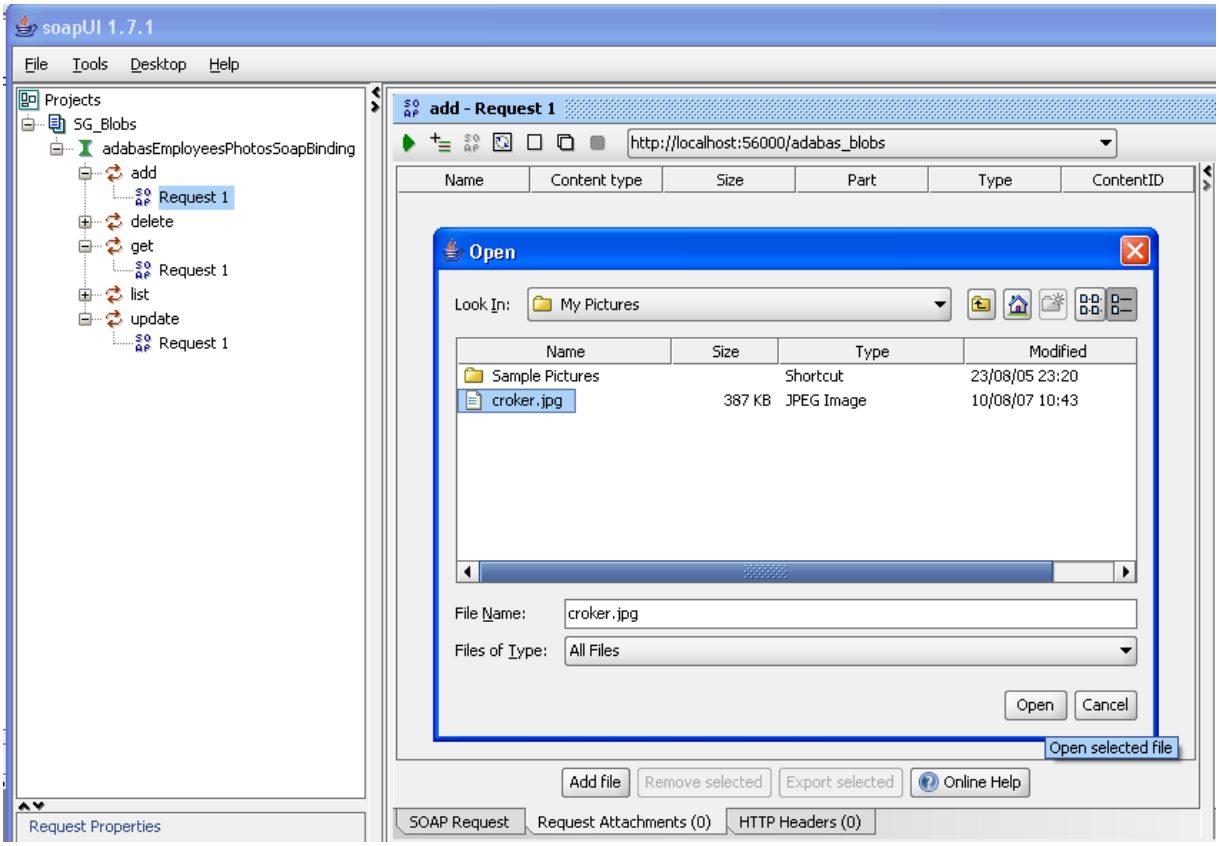
   Add the personnel_id of the record you would like the BLOB to be added

   Change the cid:XYZ reference to be something familiar, for example **cid:myPhoto**



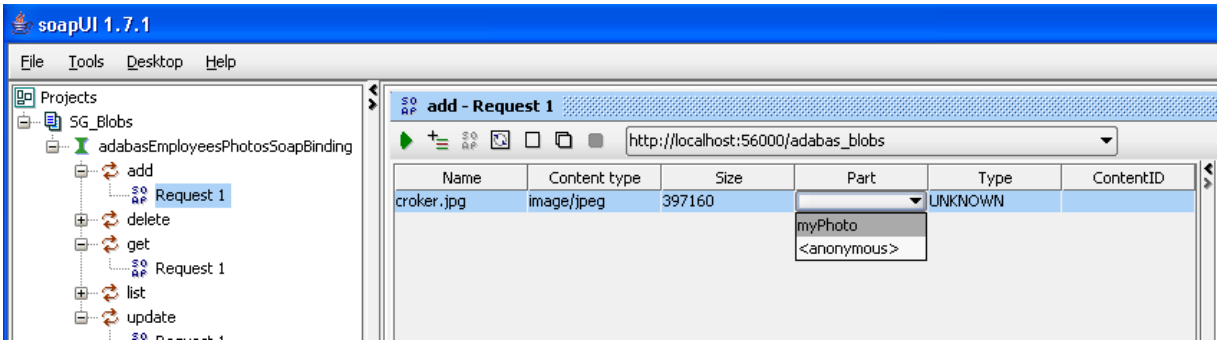
5. Click the **Request Attachments** tab, and click **Add File**

   Select the required LOB file and click **Open**

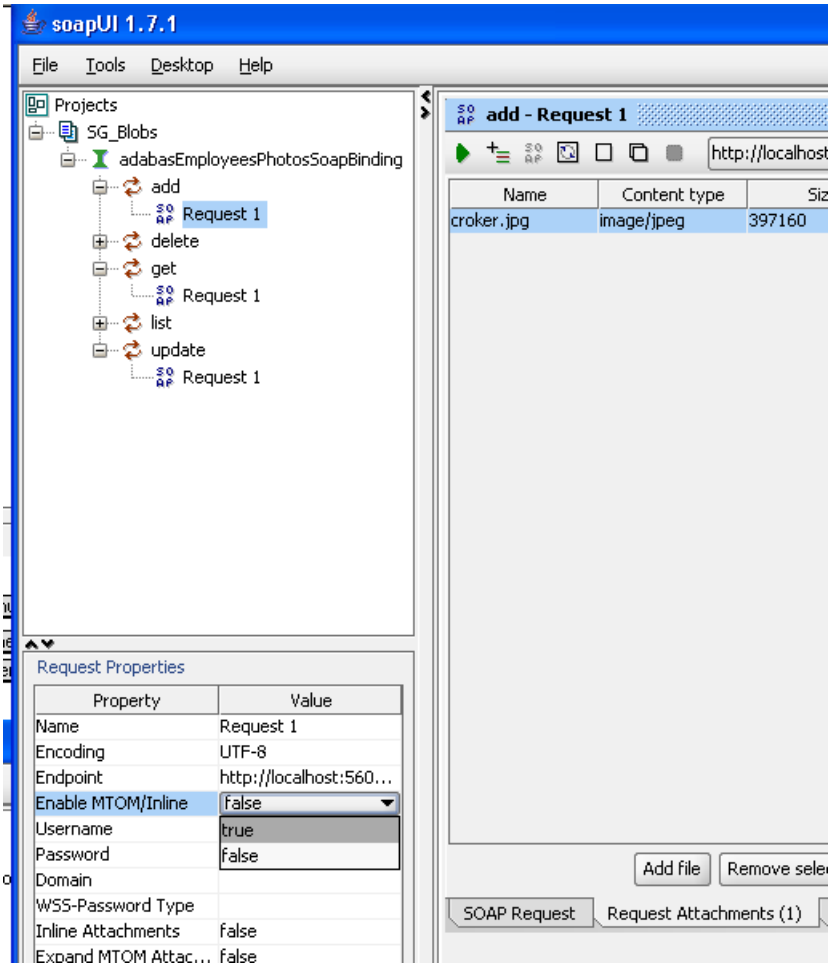


When asked to **Cache Attachment** select **No**

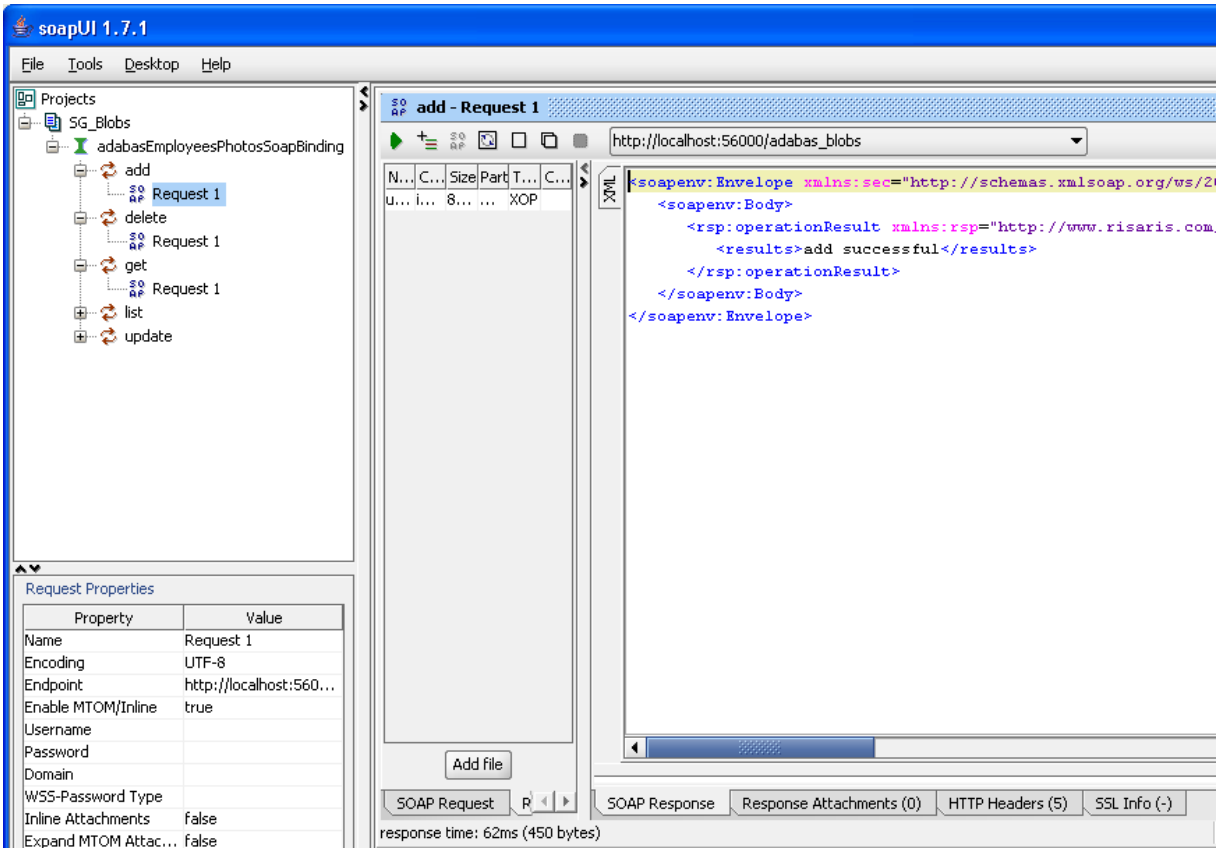
6. Click the **Part** column, and select the CID reference you changed earlier, for example **myPhoto**



7. In the **Request Properties** change **Enable MTOM/Inline** to true



8. Click the green arrow to send the request. The server's response will appear in the right pane

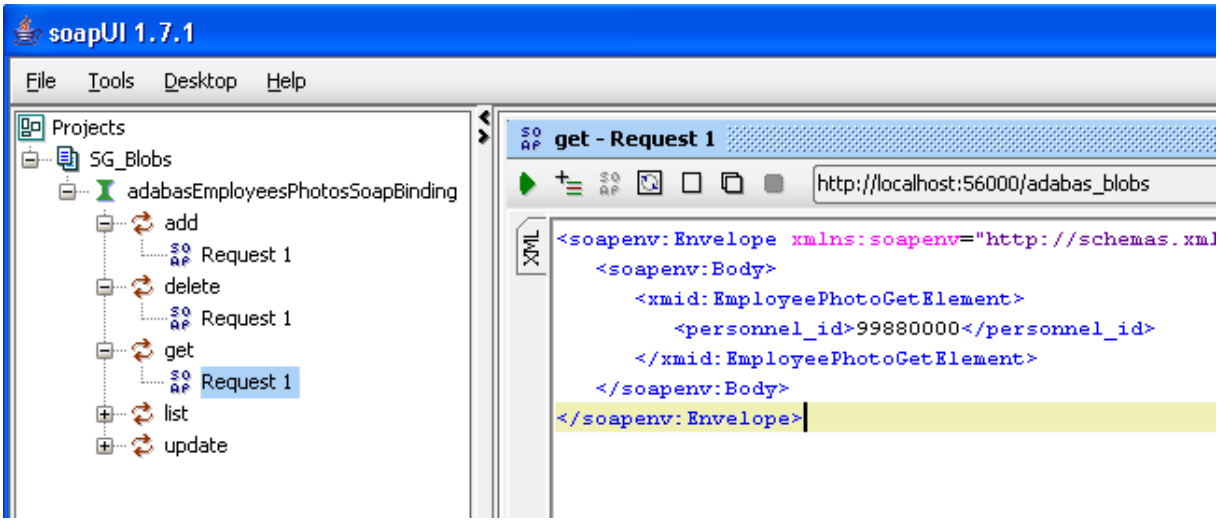


9. Now that the record, including the LOB, has been added to your Adabas file, you will want to retrieve it.

   In SOA Gateway LOBs can be retrieved using the **get** request

   Under **get**, double-click **Request 1**. Remove the <soapenv:Header> elements as before.

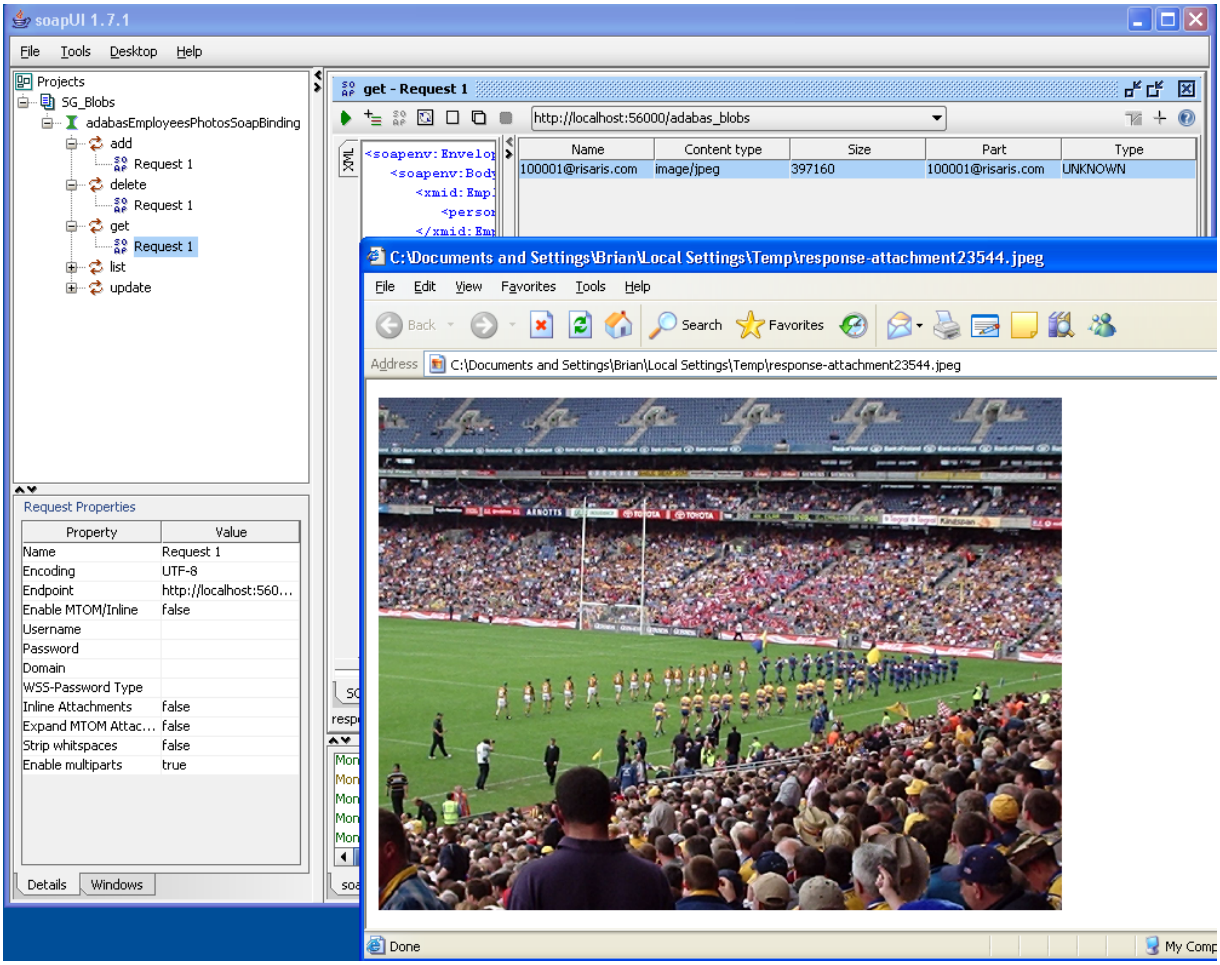
10. Add the personnel_id of the record that you wish to retrieve, for example 99880000.



11. Click the green arrow, and this record will be retrieved from Adabas.

    Select the **Response Attachments** tab, and double-click the attachment.

    It should open in a web-browser, or you can save the attachment to disk by selecting **Export selected**



12. You may delete the record, including the LOB, by selecting the delete operation and entering the corresponding personnel_id.

9 Accessing LOBs from PHP

LOBs sent as MTOM attachments on a SOAP response from SOA Gateway can be handled by a PHP program. This, however, requires the presence of the **WSO2 Web Services Framework**

Provided here are the following sample PHP programs dealing with LOBs access:

- **wsf_lobsGet: Get a record, save the LOB to a file**

10 Accessing LOBs using a browser

You can also use SOA Gateway to retrieve LOBs from Adabas into a web browser. This can be any browser of your choice, for example IE running on Windows, retrieving LOBs from Adabas running on z/OS.

In the tutorial below, we'll use the popular **Firefox** browser and the SOA Gateway running on z/OS.

This tutorial assumes you've already got some LOBs stored in your database. See the previous tutorial to find out how to do this.

1. Start the Control Center and add a new Web Service.

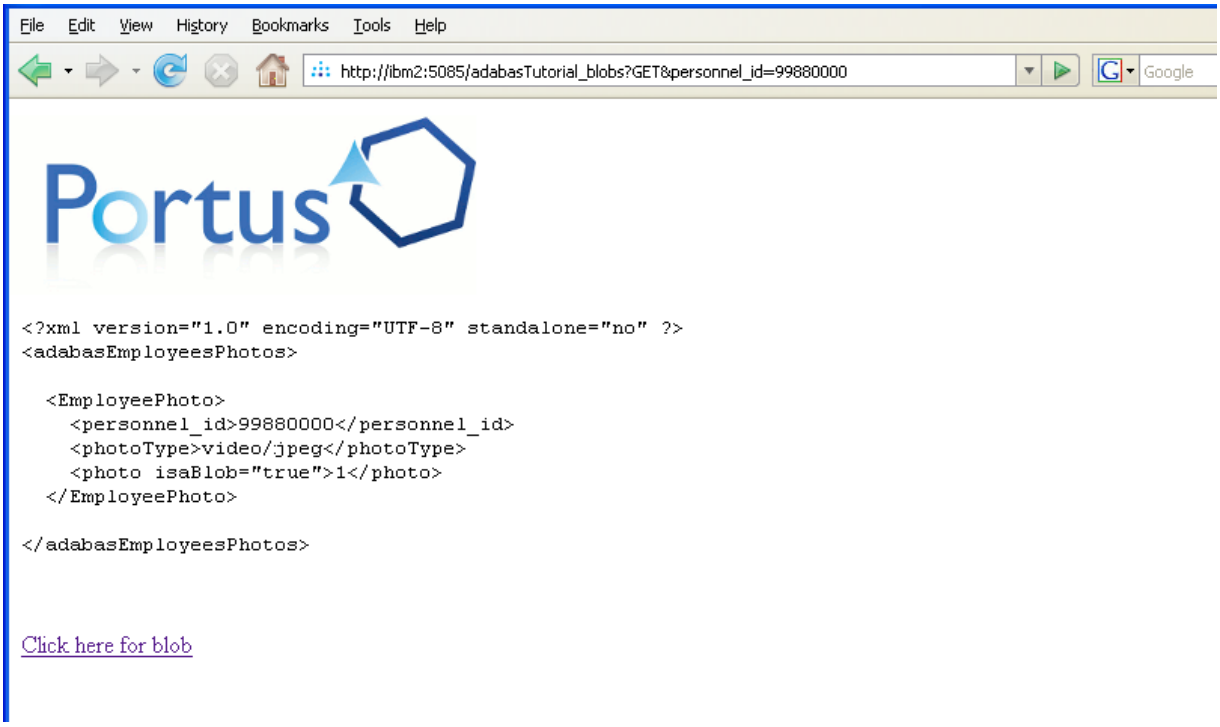
   Enter the following information

   - Name: adabasTutorial_blobs
   - DataView : Select 'adabas_photoblobs', which is one of the sample definitions delivered with SOA Gateway
   - DatabaseId: 212 (or the DBID relevant on your system)
   - FileNumber: 90

   Publish your changes to the server.
2. Start your browser, and enter the following URL

   http://soagate:56000/adabasTutorial_blobs?GET&personnel_id=99880000

   Replacing soagate:56000 with the hostname and port where SOA Gateway is running.
3. If there is a blob associated with this record, the following is returned



4. Click on the link "Click here for blob" to display the LOB file in the browser.

   Note: The LOB will open automatically in your browser, if it of a MIME type that your browser understands. Commonly used types are PDF, WMV, JPEG, etc. Refer to your browser documentation if you need more information.

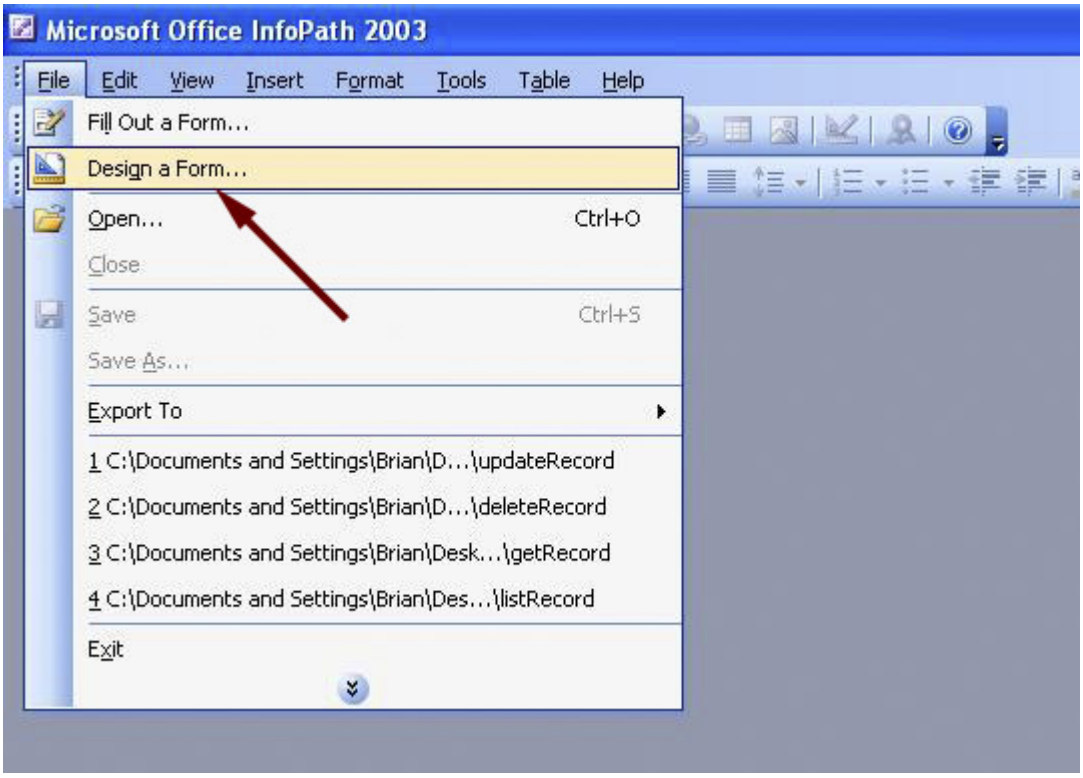
5. Here you can see a JPEG file opened up in a new tab in firefox after the LOB link was clicked.


adabasTutorial_blobs (JPEG Image, 1280x960 pixels) - Scaled (35%) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://ibm2:5085/adabasTutorial_blobs?GETBLOB&__blobRef=1&persc
SOA Gateway
adabasTutorial_blobs (JPEG Ima...
RUGBY WORLD CUP 2007
WWW.IRISHRUGBY.IE/RWC2007
Done

11 Accessing Adabas through Microsoft InfoPath

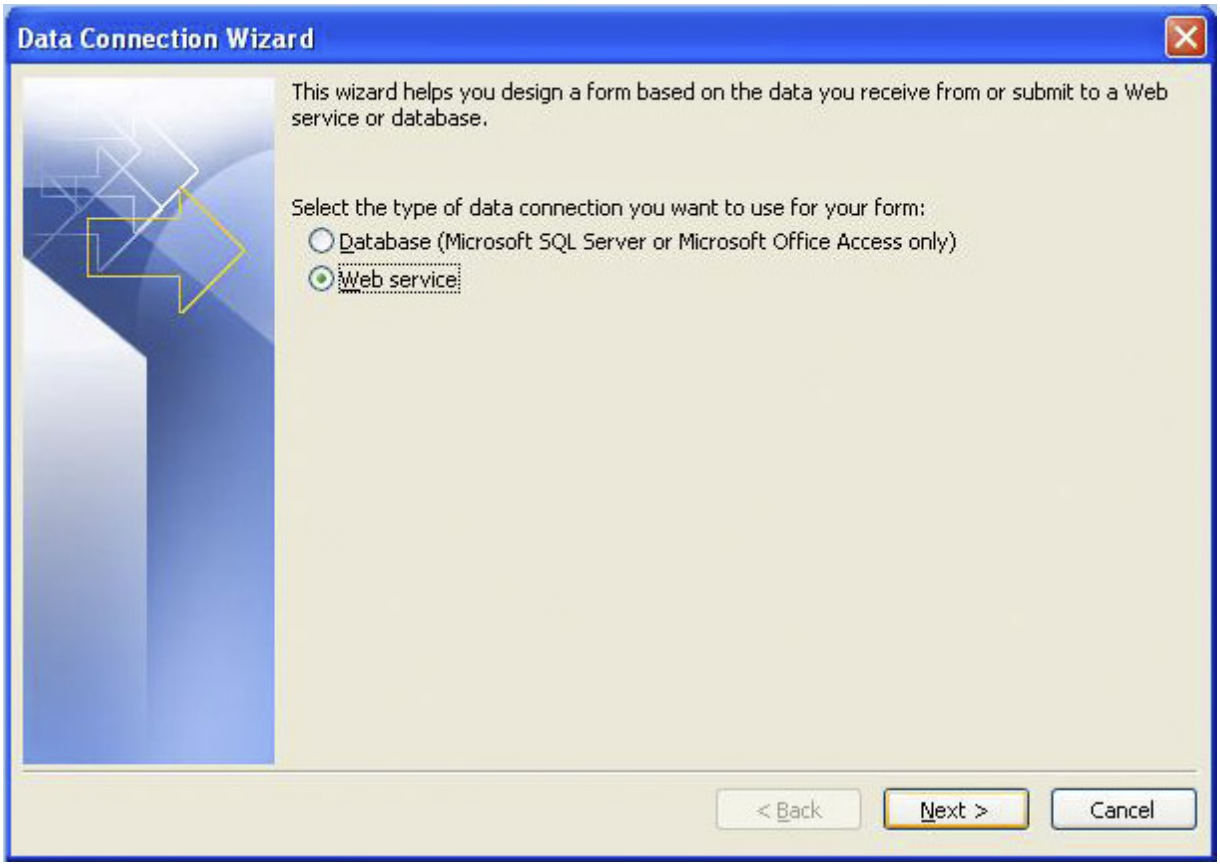
This tutorial demonstrates how to invoke operations on an Adabas DataSource exposed as a "Web service" through SOA Gateway from Microsoft InfoPath.

1. From the InfoPath main menu bar, select **File** -> **Design a Form**.

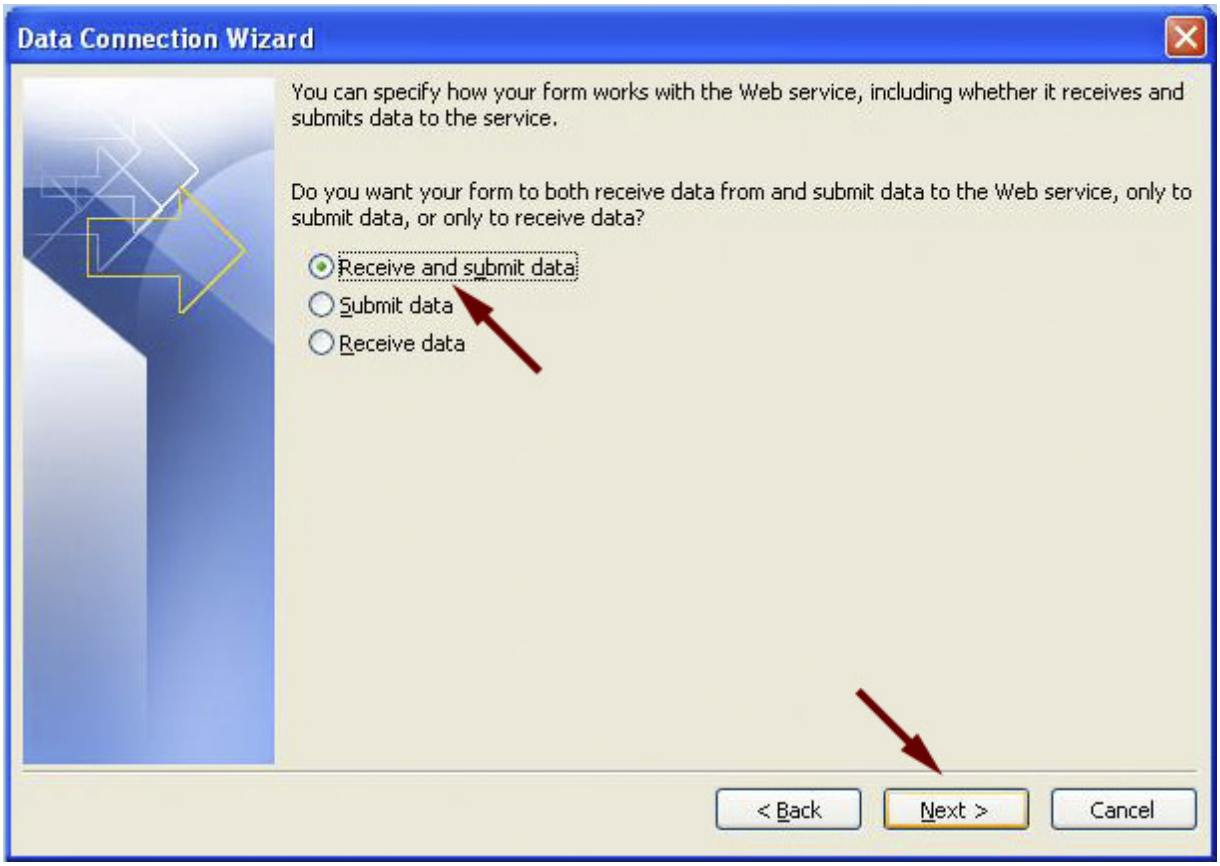


2. A panel will appear on your right hand side - choose **New** from **Data Connection**

   This will start a dialog, first select **Web Service**, click **Next**

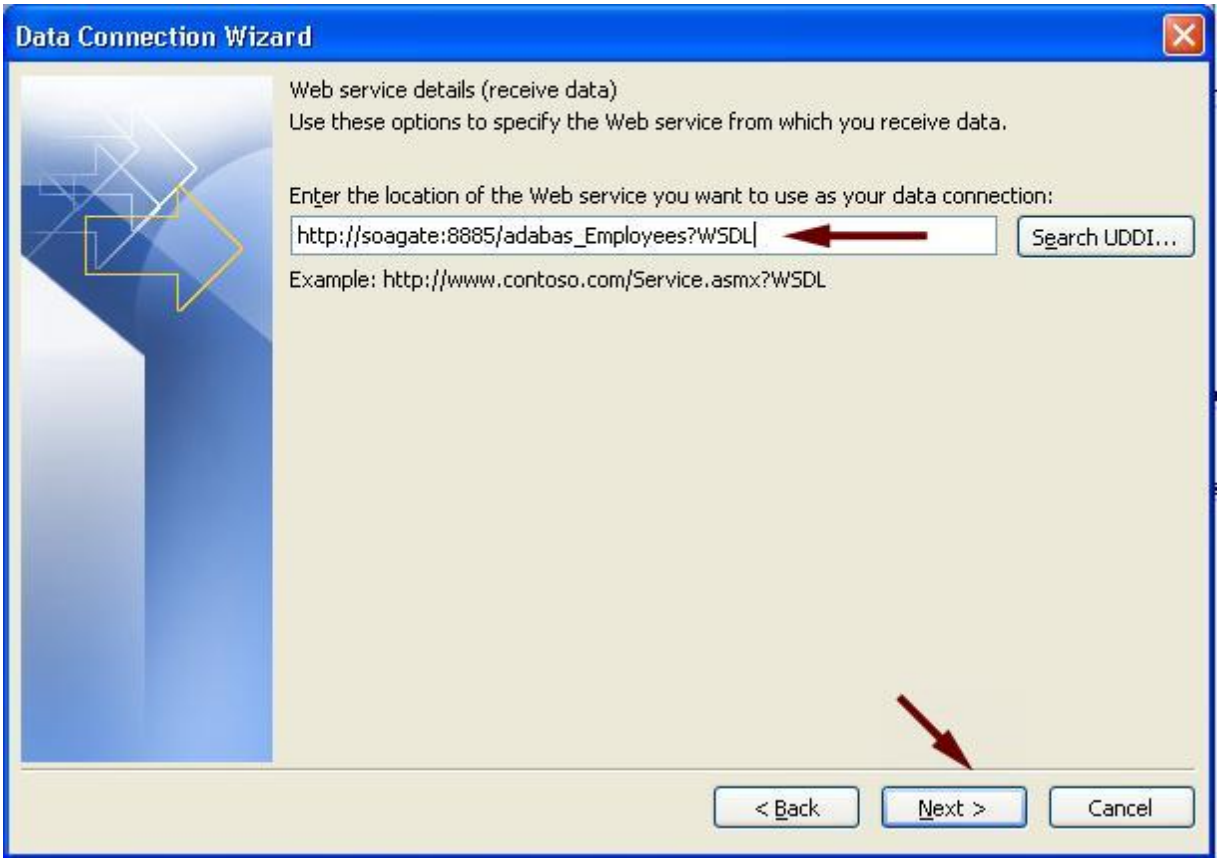


Select **Receive and submit data**, click **Next**

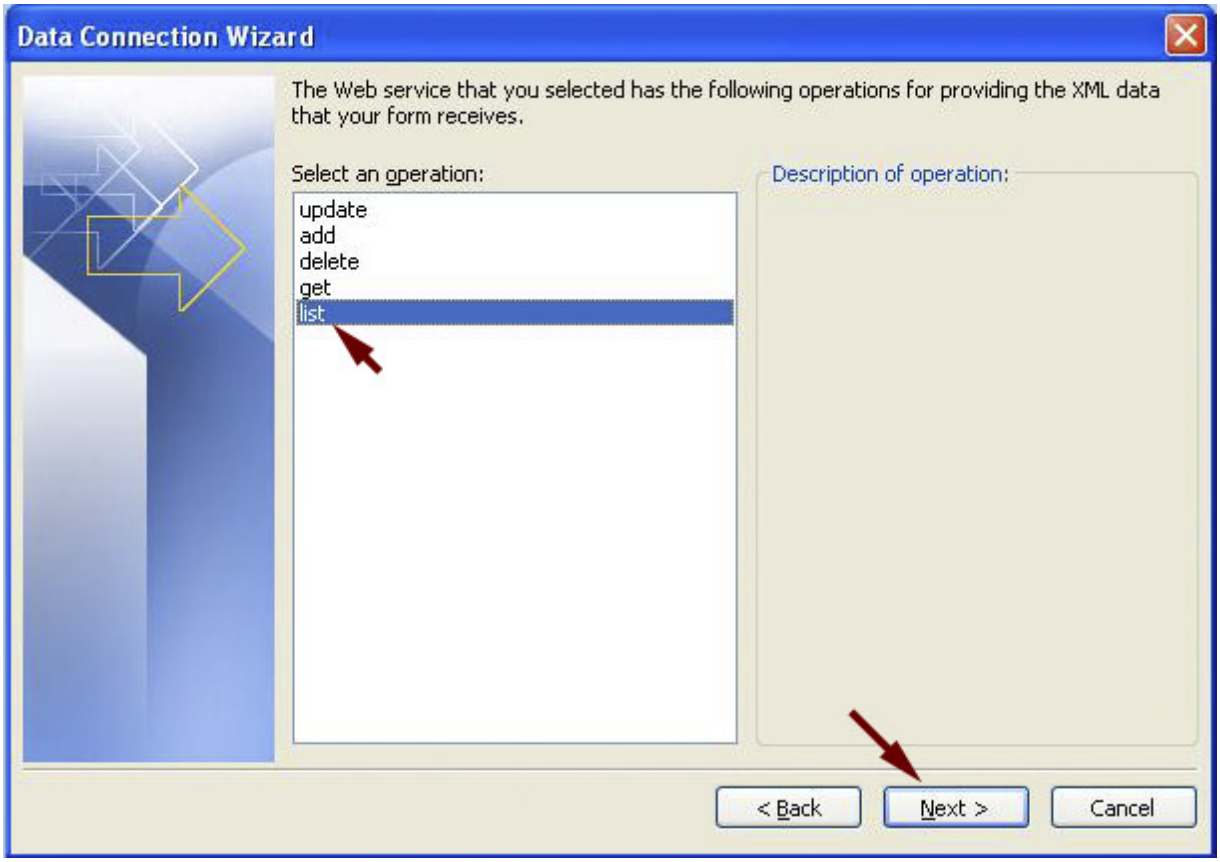


Enter the URL the WSDL for the "Employees" demo file is exposed as: *http://soagate:8885/adabas_Employees?WSDL*

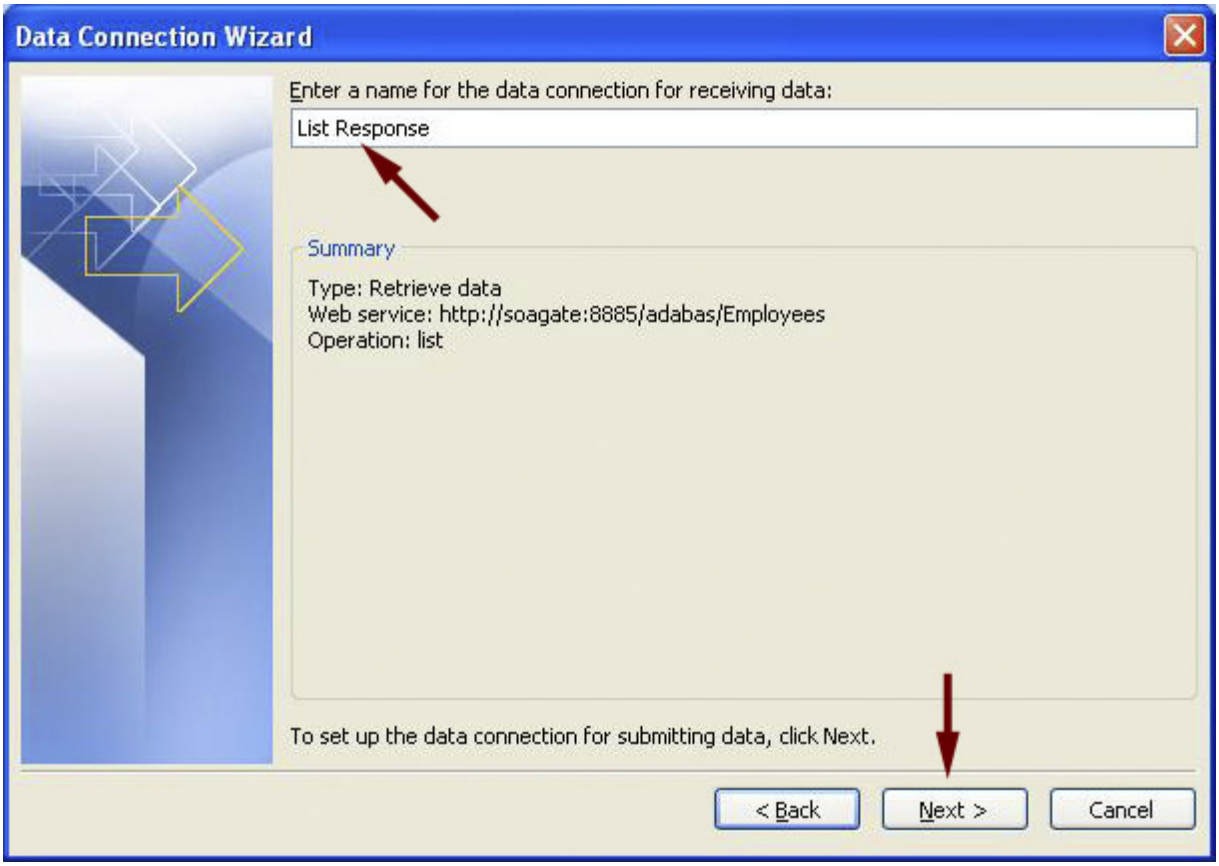
Click **Next**



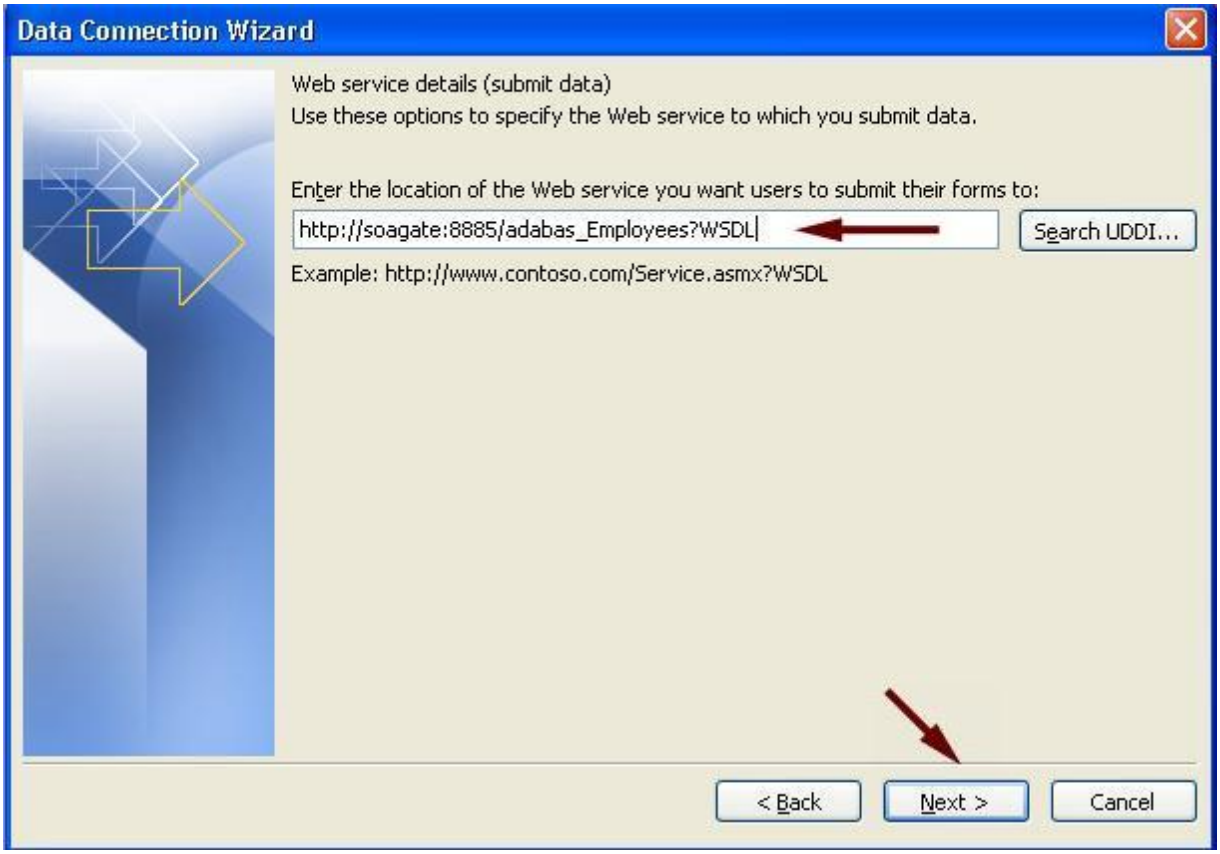
Select the **list** method, click **Next**



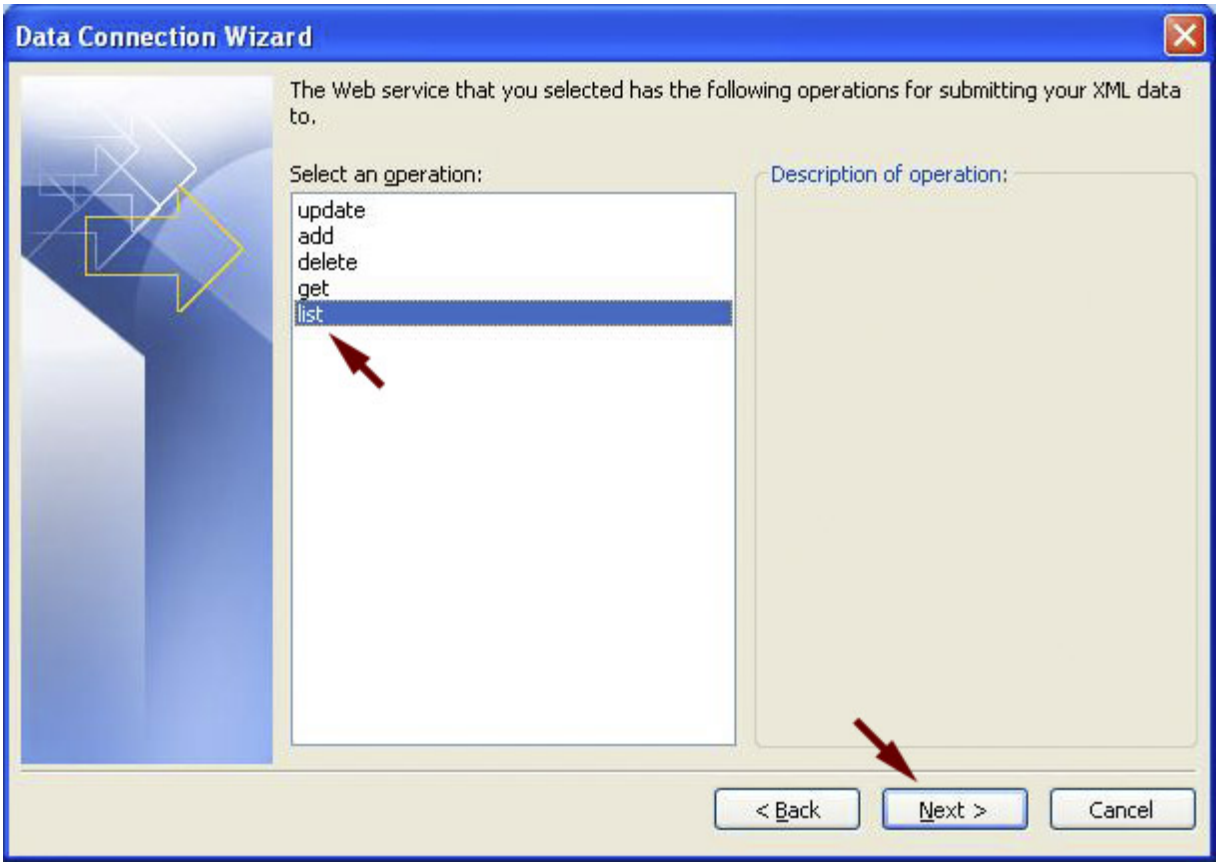
Assign a name to the response document, click **Next**



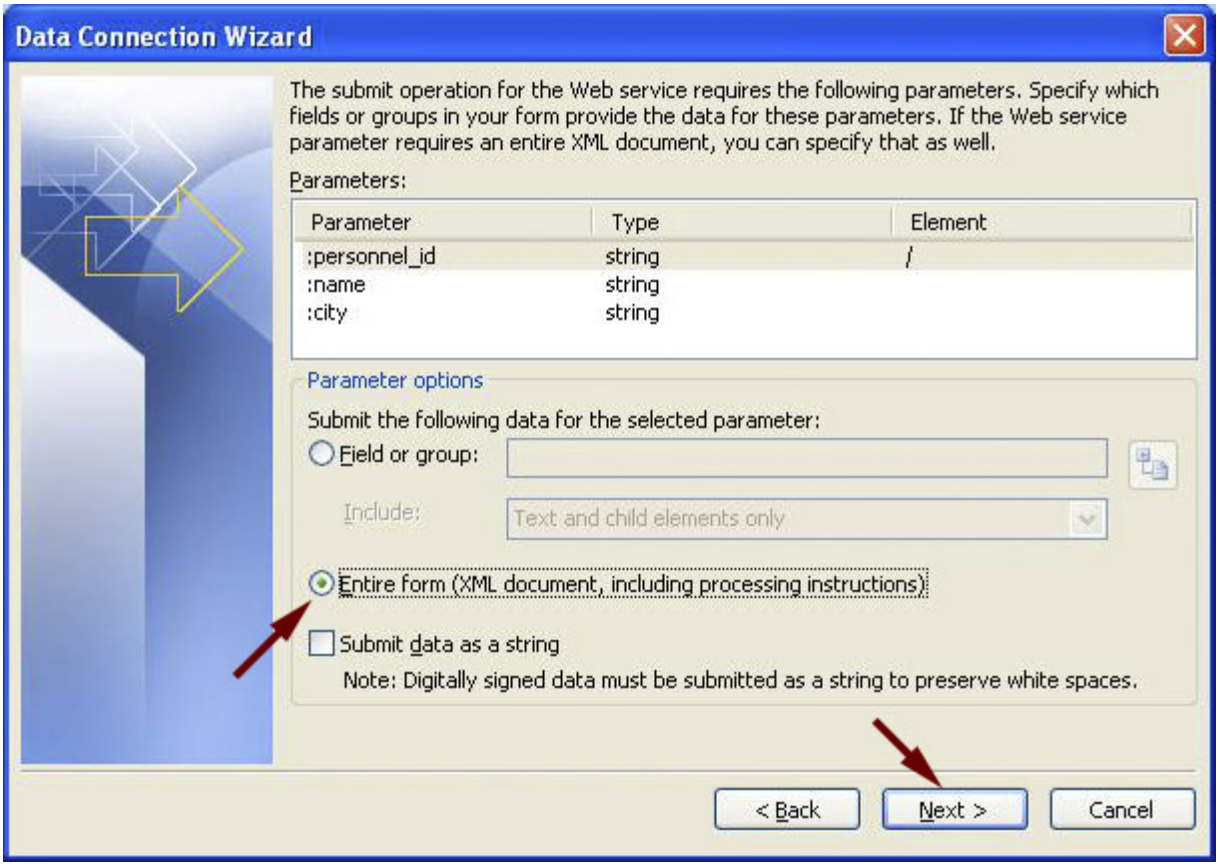
Enter the same URL again, click **Next**



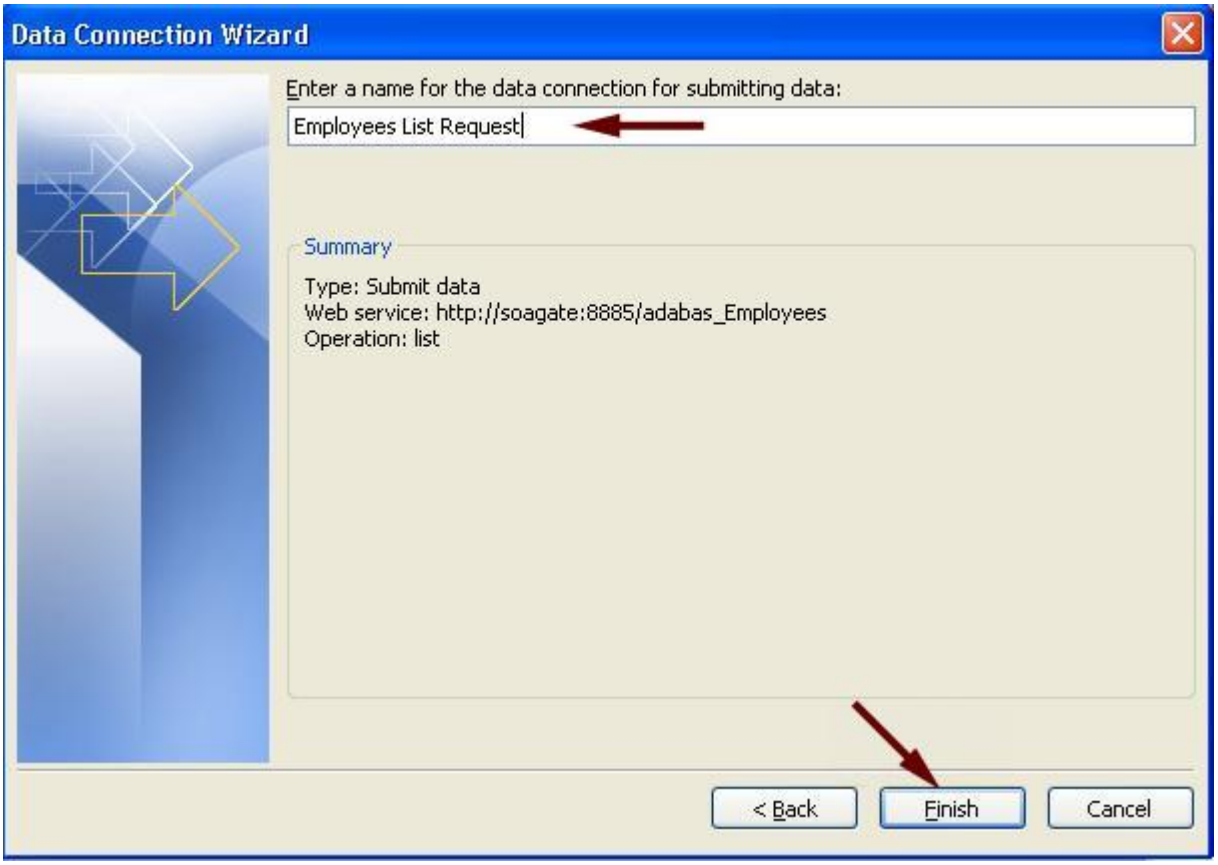
Choose list and click **Next**



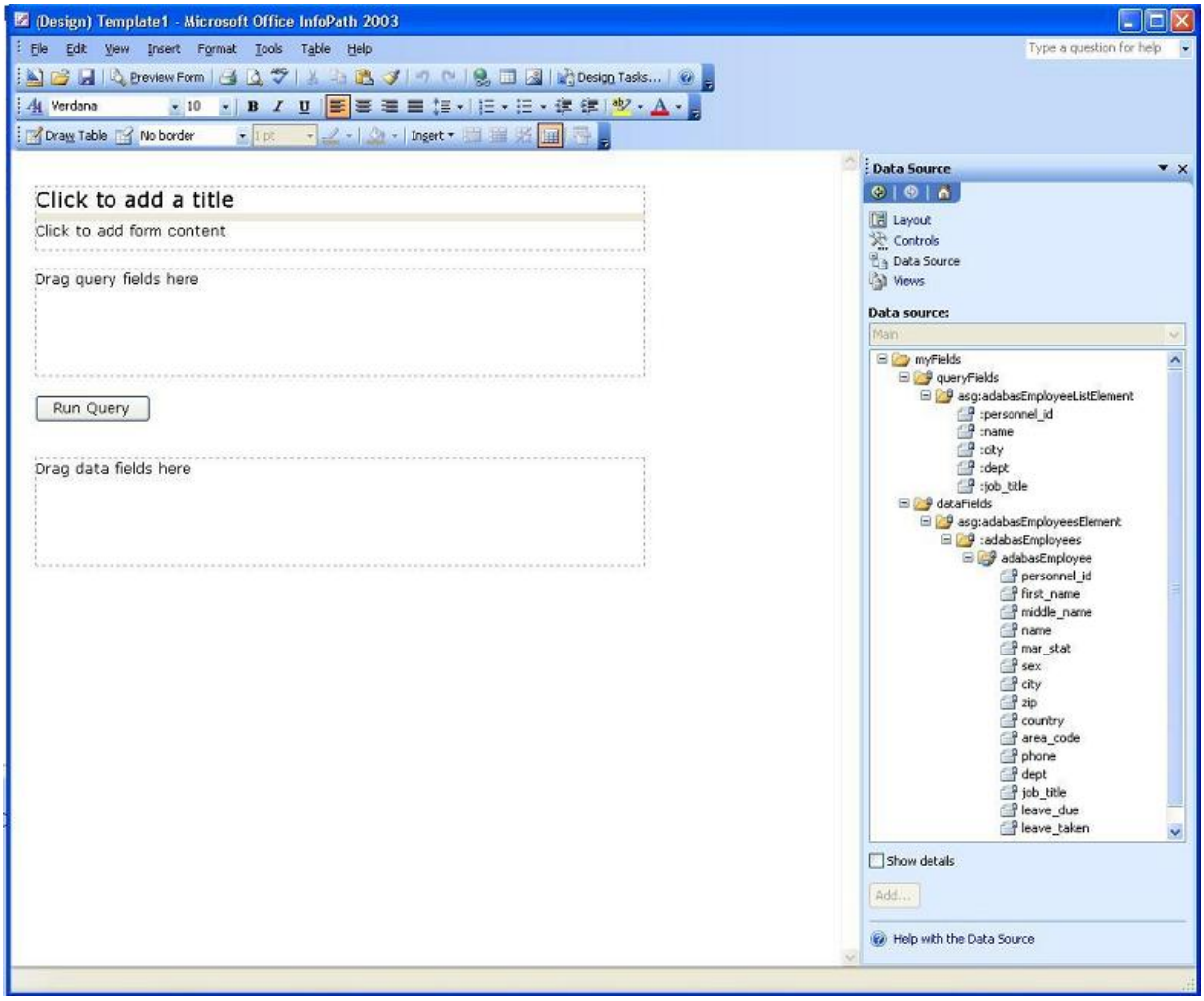
Select "**Entire form (XML document....**", click **Next**



Assign a name to the Send Connection, click **Finish**



3. You will now be presented with a form

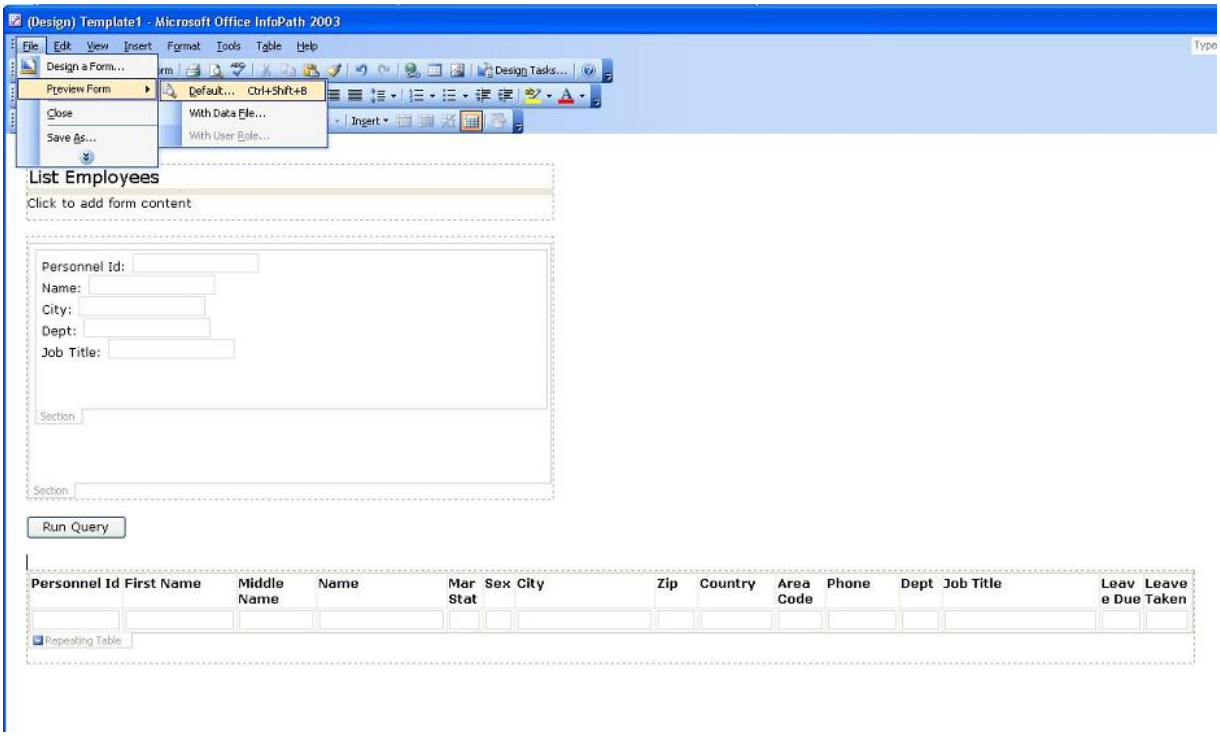


Do the following:

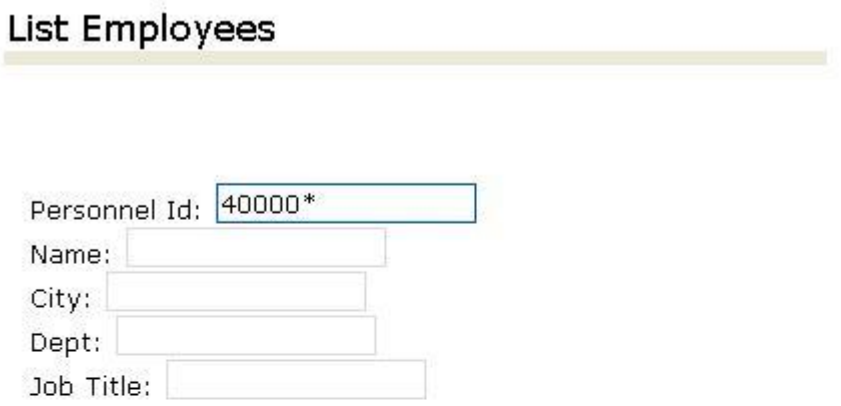
- Enter a title "List Employees" for example
- Expand the **queryFields**, drag them to the area "Drag query fields here"
- Expand the **dataFields**, right-click on the **adabasEmployee** element. Now drag the adabasEmployee to the area "Drag data fields here", insert it as a **Repeating table**

At this point, it might make sense to resize the table and the fields in the "repeating table"

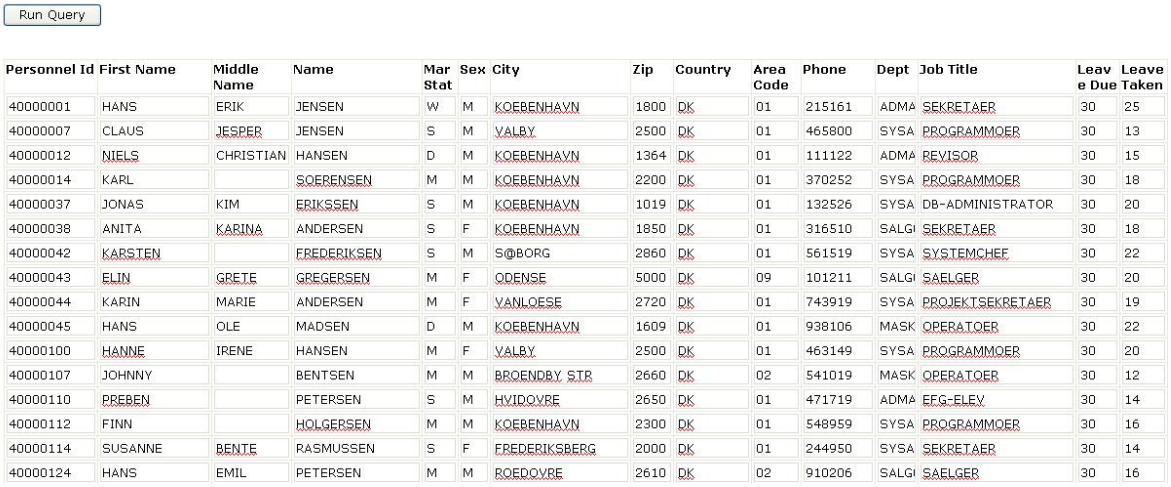
4. Once you are happy with how your form looks like, select **File** -> **Preview form** -> **Default**



5. You will be presented with a form, enter 4000004* in the Personnel Id field and send the request to the server



6. Your table will now be populated with the data based on your request:



Personnel Id	First Name	Middle Name	Name	Mar Stat	Sex	City	Zip	Country	Area Code	Phone	Dept	Job Title	Leave Due	Leave Taken
40000001	HANS	ERIK	JENSEN	W	M	KOEBENHAVN	1800	DK	01	215161	ADMA	SEKRETAER	30	25
40000007	CLAUS	JESPER	JENSEN	S	M	VALBY	2500	DK	01	465800	SYSA	PROGRAMMOER	30	13
40000012	NIELS	CHRISTIAN	HANSEN	D	M	KOEBENHAVN	1364	DK	01	111122	ADMA	REVISOR	30	15
40000014	KARL		SOERENSEN	M	M	KOEBENHAVN	2200	DK	01	370252	SYSA	PROGRAMMOER	30	18
40000037	JONAS	KIM	ERIKSSEN	S	M	KOEBENHAVN	1019	DK	01	132526	SYSA	DB-ADMINISTRATOR	30	20
40000038	ANITA	KARINA	ANDERSEN	S	F	KOEBENHAVN	1850	DK	01	316510	SALG	SEKRETAER	30	18
40000042	KARSTEN		FREDERIKSEN	S	M	S@BORG	2860	DK	01	561519	SYSA	SYSTEMCHEF	30	22
40000043	ELIN	GRETE	GREGERSEN	M	F	ODENSE	5000	DK	09	101211	SALG	SAELGER	30	20
40000044	KARIN	MARIE	ANDERSEN	M	F	VANLOESE	2720	DK	01	743919	SYSA	PROJEKTSEKRETAER	30	19
40000045	HANS	OLE	MADSEN	D	M	KOEBENHAVN	1609	DK	01	938106	MASK	OPERATOER	30	22
40000100	HANNE	IRENE	HANSEN	M	F	VALBY	2500	DK	01	463149	SYSA	PROGRAMMOER	30	20
40000107	JOHNNY		BENTSEN	M	M	BROENDBY STR	2660	DK	02	541019	MASK	OPERATOER	30	12
40000110	PREBEN		PETERSEN	S	M	HVIDOVRE	2650	DK	01	471719	ADMA	EFG-ELEV	30	14
40000112	FINN		HOLGERSEN	M	M	KOEBENHAVN	2300	DK	01	548959	SYSA	PROGRAMMOER	30	16
40000114	SUSANNE	BENTE	RASMUSSEN	S	F	FREDERIKSBERG	2000	DK	01	244950	SYSA	SEKRETAER	30	14
40000124	HANS	EMIL	PETERSEN	M	M	ROEDOVRE	2610	DK	02	910206	SALG	SAELGER	30	16

12 Using Transactions with soapUI

The following tutorial demonstrates how SOA Gateway can be used with Adabas transactions. It is assumed you are already familiar with the following

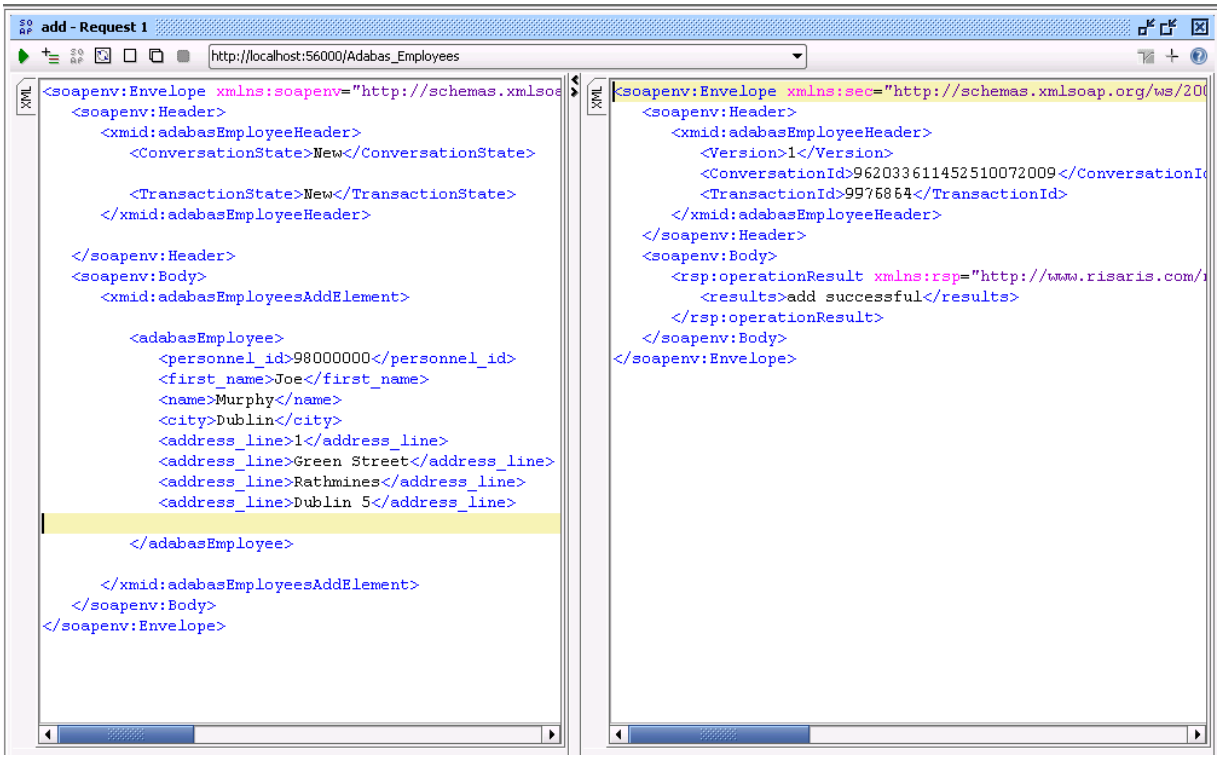
- soapUi (see **here**)
- Creating SOA Gateway web services from Adabas (see here)
- Familar with SOAP Header usage concepts (see here)

For this tutorial, we will use the Employees demo file (usually file number 11) that comes with Adabas, and the **adabas_employees_mini_view** data view. It is assumed you have already created a web service for your Adabas file.

Important: By default, SOA Gateway will time-out and kill existing Conversations after a period of time. This is can be configured using the Control Centre. with a maximum value of 3600 (10 minutes). See here for more information.

1. Import the WSDL into soapUi.
2. Choose the request for the `add` operation, and in the XML set the `ConversationState` to New and the `TransactionState` to New. Also remove the other Header values. Send the request to SOA Gateway

   E.g

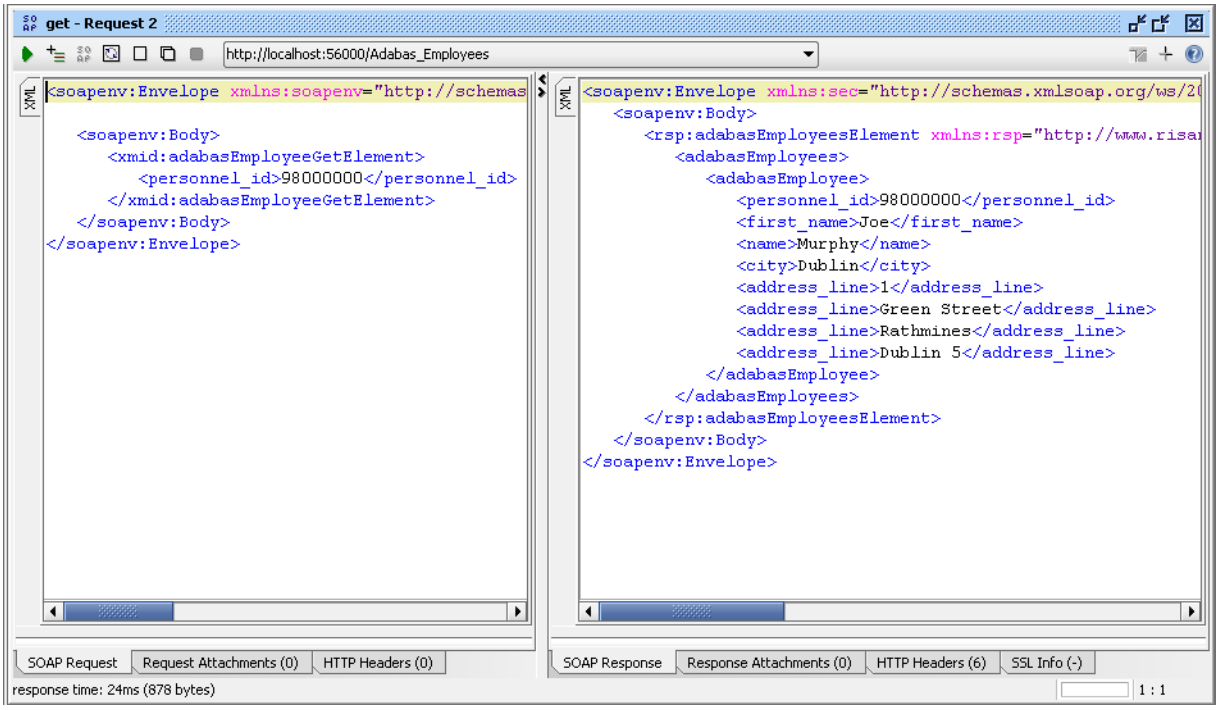


SOA Gateway has now created a new Conversation and a new Transaction for this request. The IDs for each of these are returned.

3. Verify that the record has been added successfully, (but not yet committed).

   Choose a `get` request, remote the soapenv:Header element, and enter get the record.

   E.g.



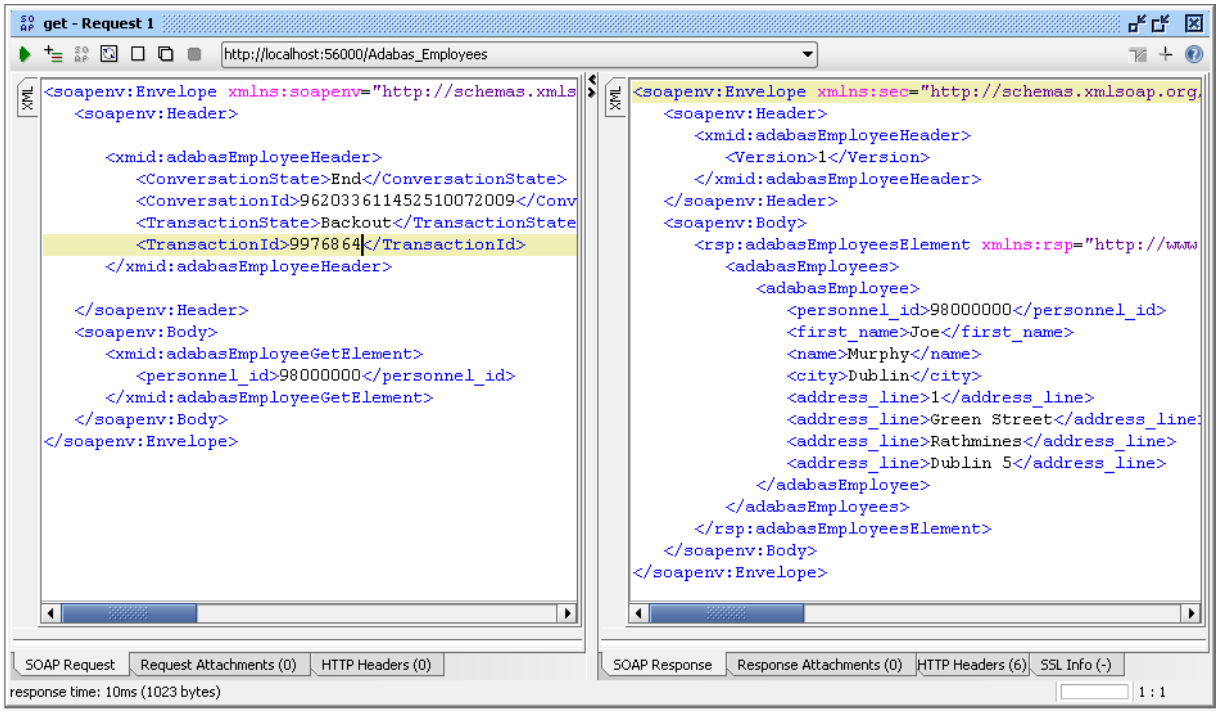
The record has been added, but not yet committed. Because Adabas' isolation level is "Read Uncommitted" (also known as "dirty read"), a request from a non-conversational request will still return the added, but-yet-uncommitted, record.

4. Now backout the transaction.

   Choose a `get` request, and enter the following:

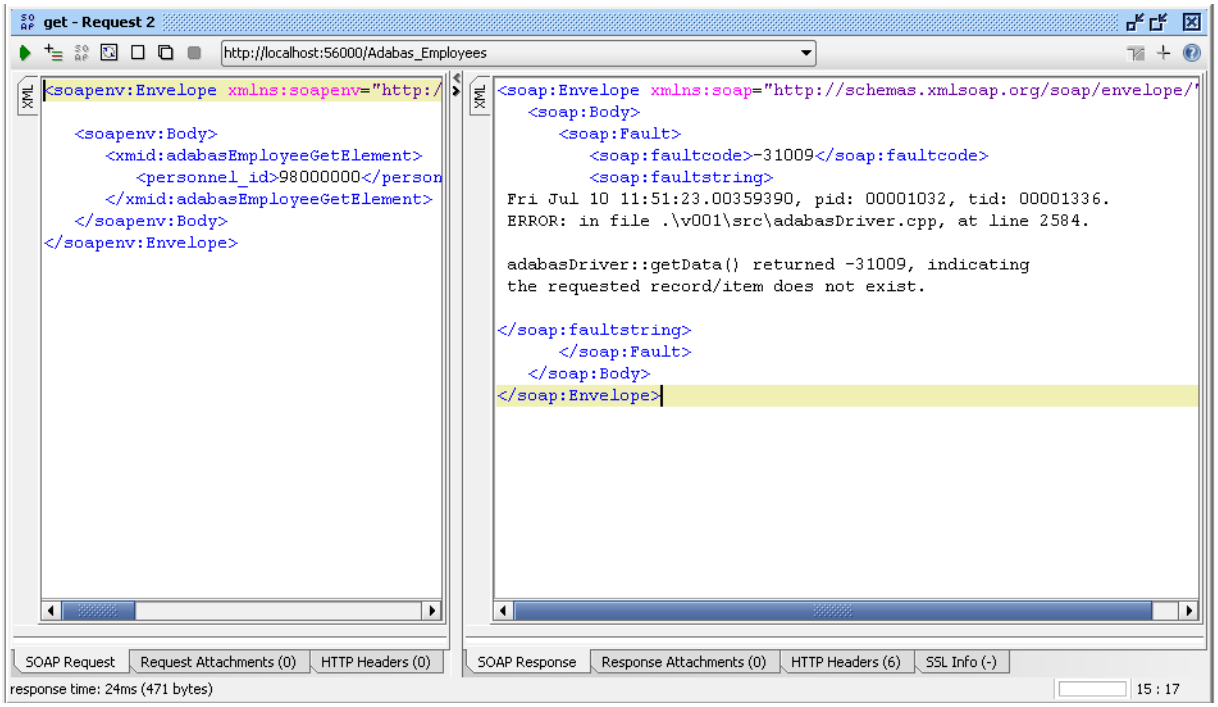
   - `ConversationState` to End
   - `ConversationID` to the value returned in the add response
   - `TransactionState` to Backout
   - `TransactionID` to the value returned in the add response

   E.g



5. Now, if you re-run the request from Step 3, the item does not exist as the previous add has been backed out.

   E.g.



13 SOA Gateway - Configuration Versioning with Eclipse and CVS

Introduction

If Eclipse is used as the management interface to SOA Gateway then versioning of the ASG configuration files can be achieved easily with CVS. Eclipse provides CVS interfaces, so much of the following will be familiar to many users of Eclipse already.

Using CVS can also bring advantages including security, auditing control, redeployment facilitation and metadata management.

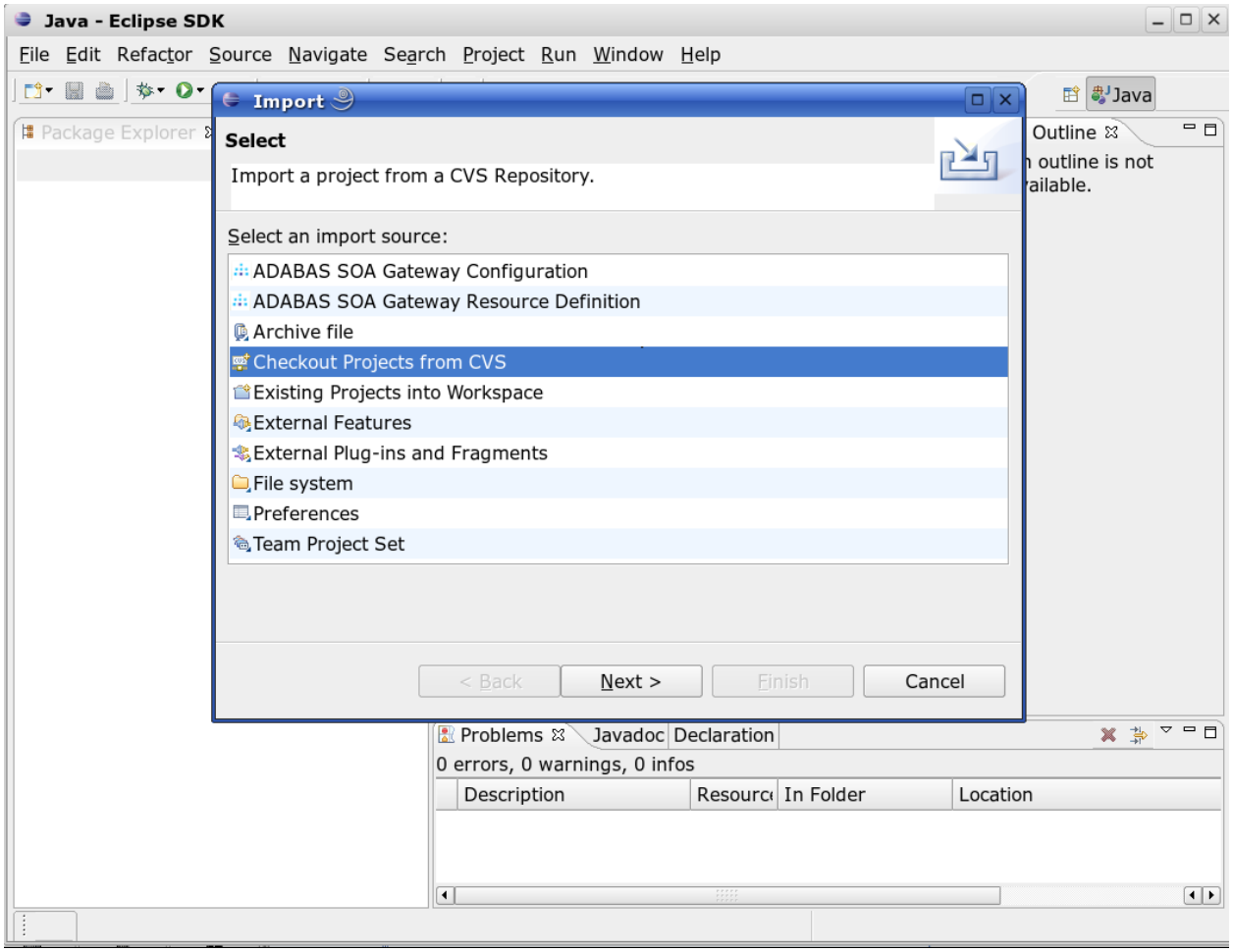
Requirements

In order to take advantage of this option, you must have available :

- a server running CVS and to which you have access
- a CVS 'module' where you can add the ASG related files you wish to maintain

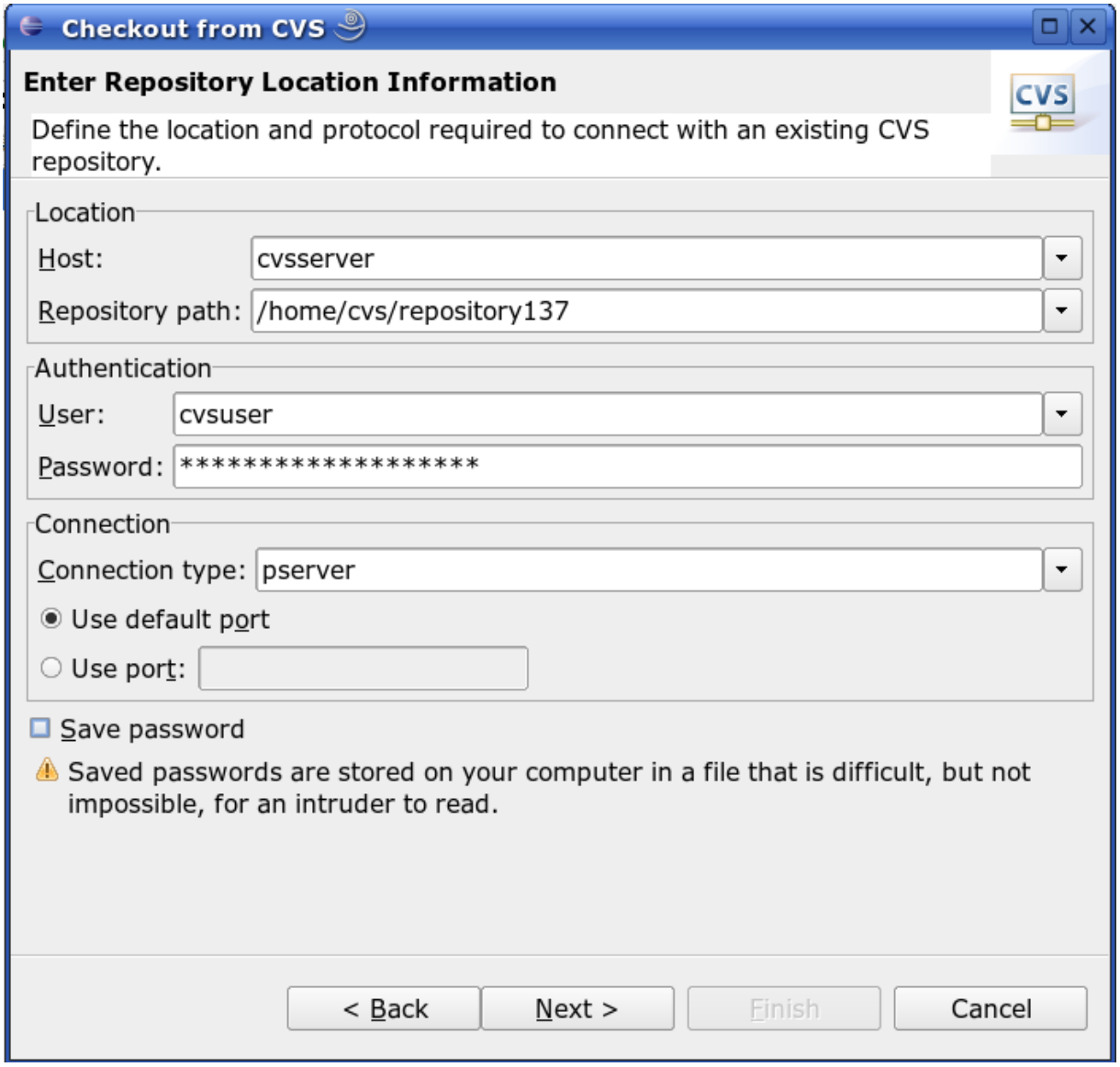
Example Setup

Select **File**->**Import** from the Eclipse Menus and then choose "Checkout Projects from CVS".



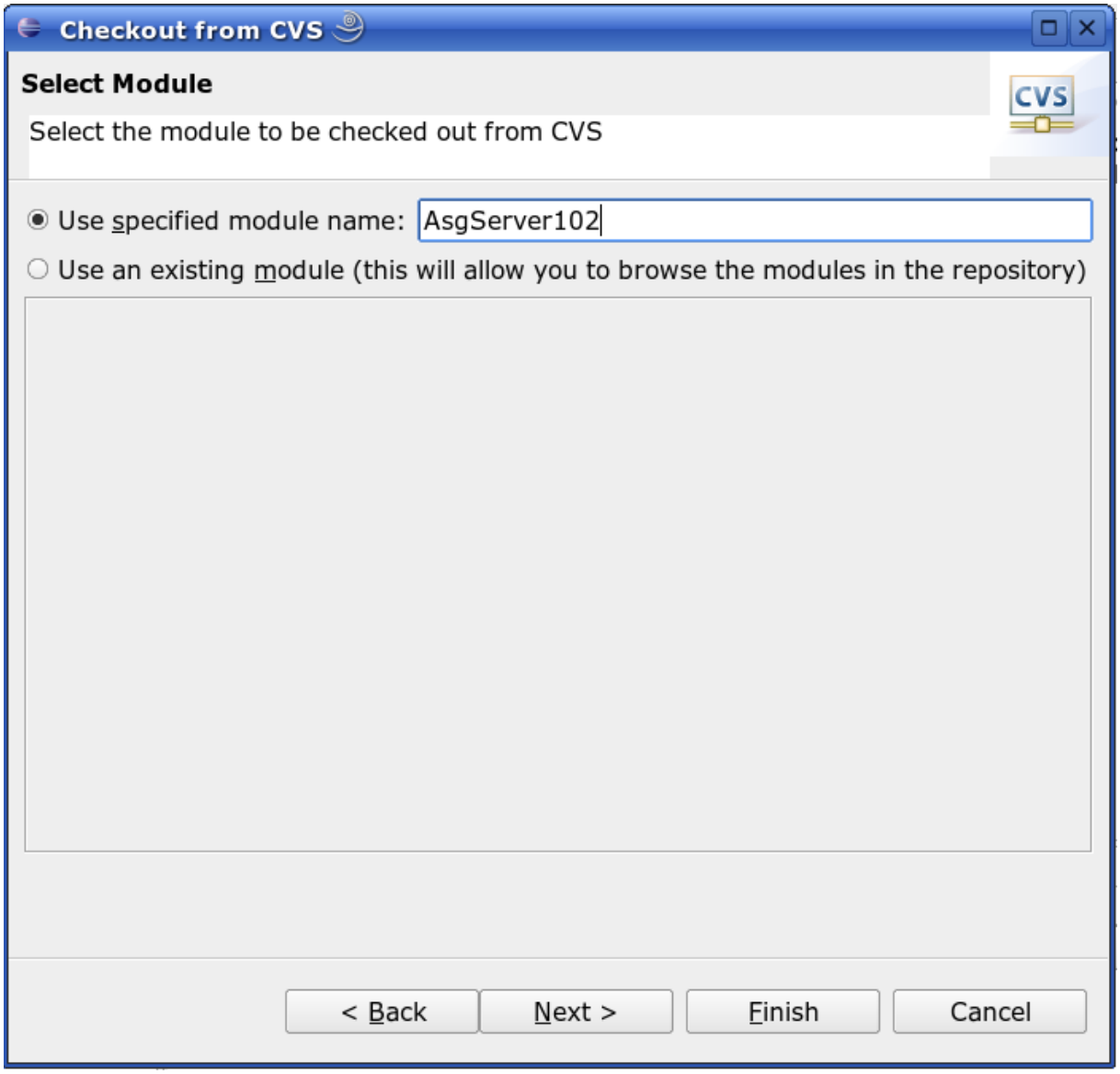
CVS Server Details

Enter the details required for your CVS server.



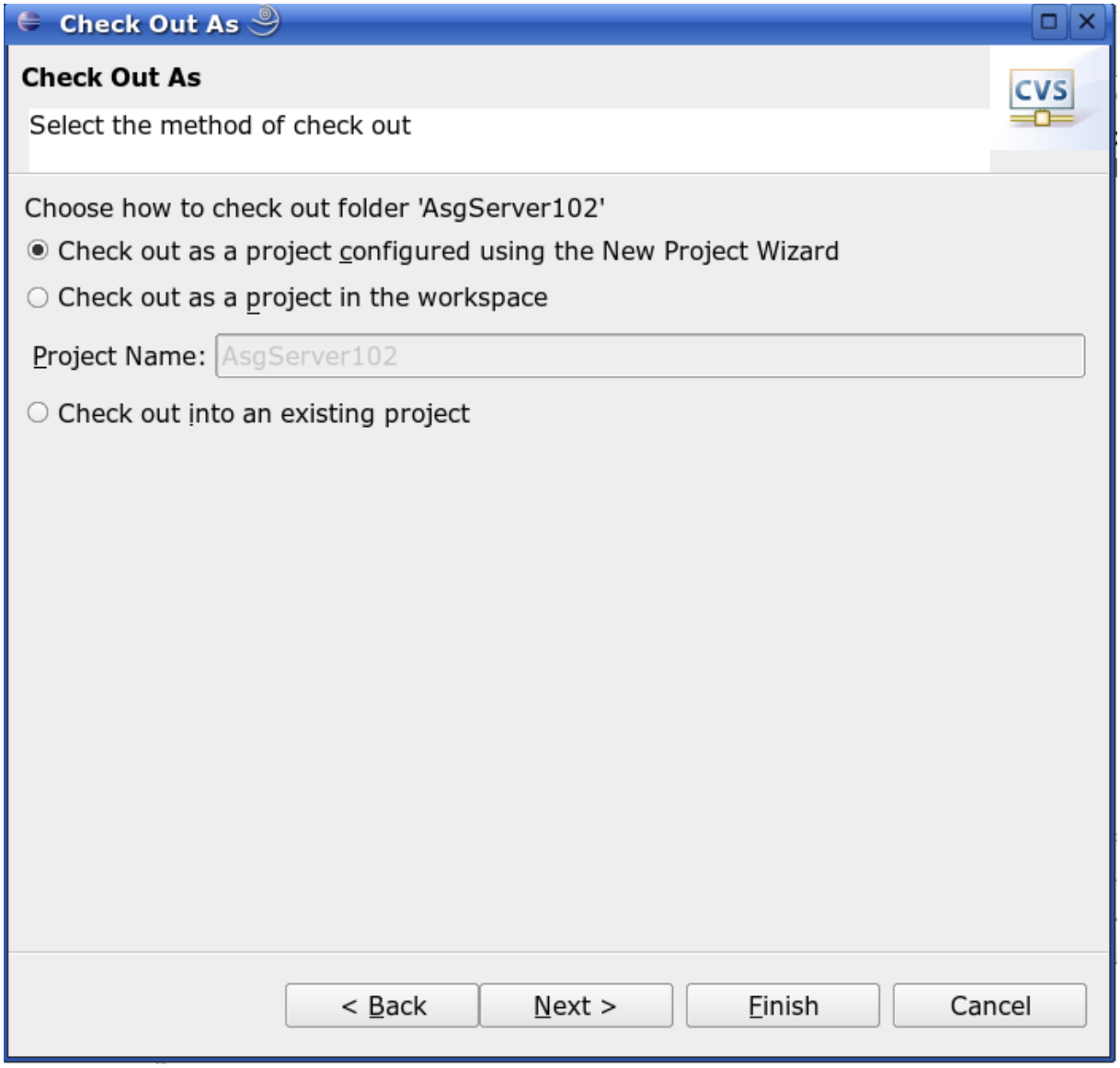
CVS Module Selection

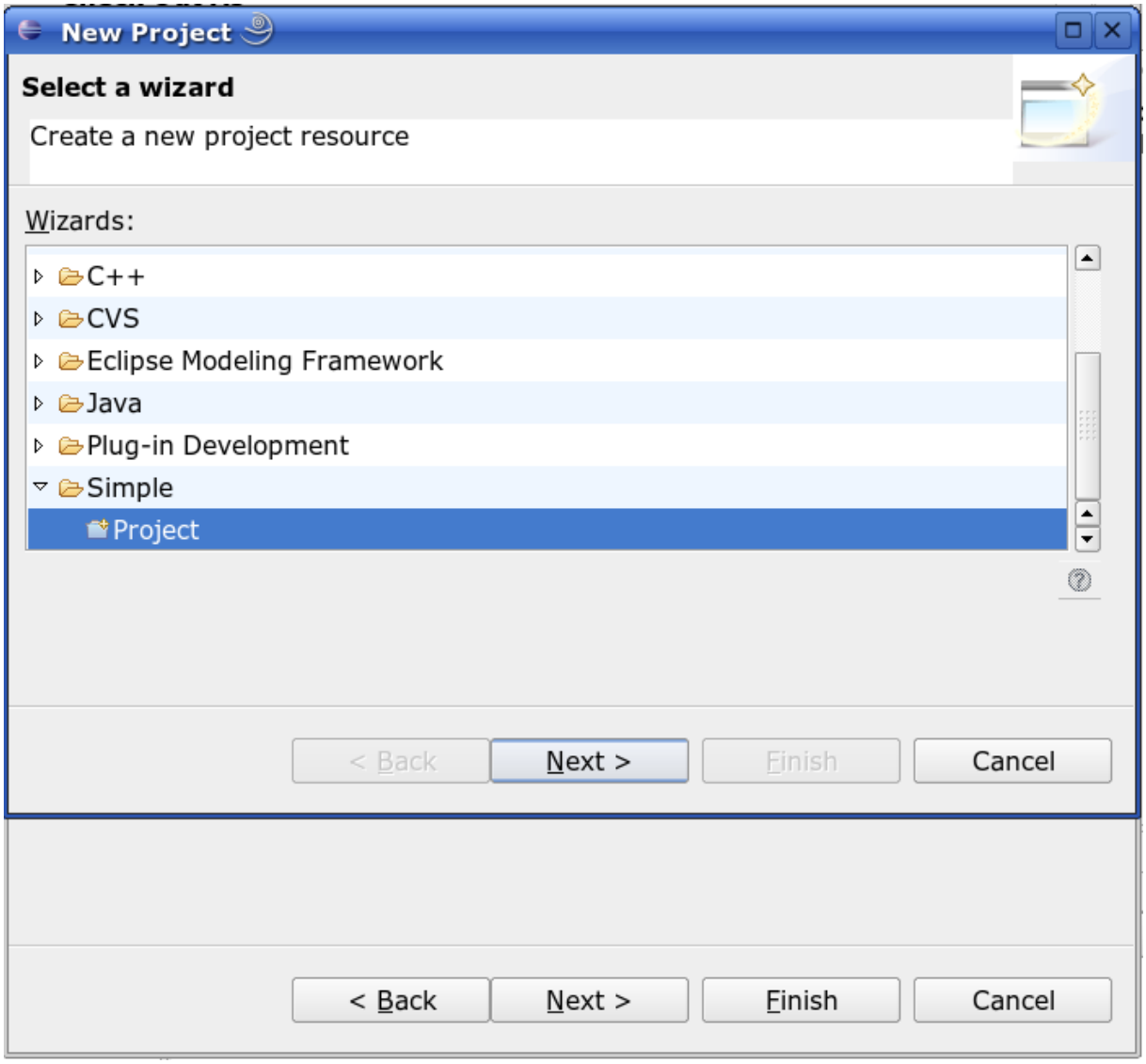
Enter the name of the CVS module where you are to keep your ASG files, or select the module from a list.

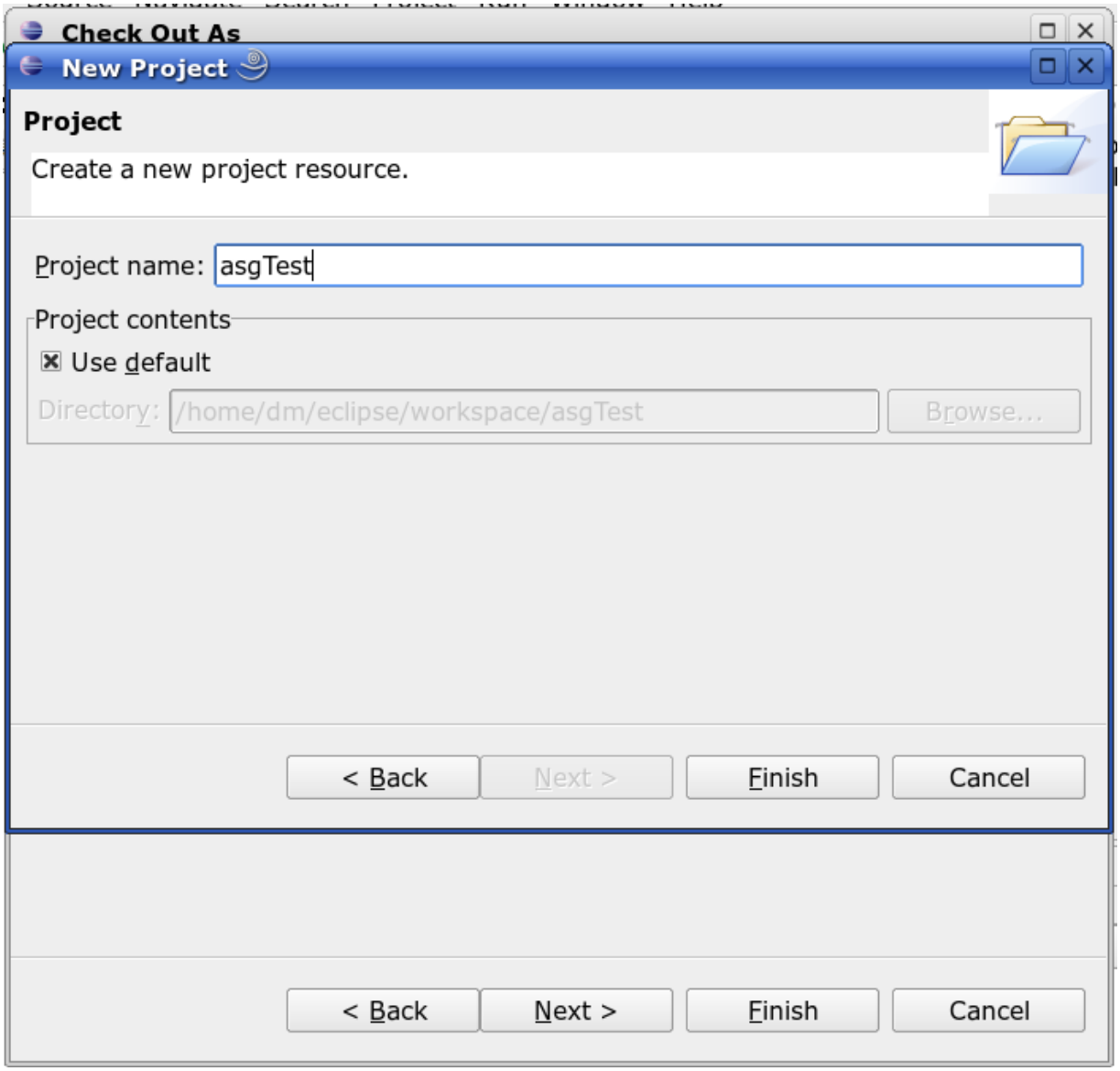


Selecting a Location in Eclipse

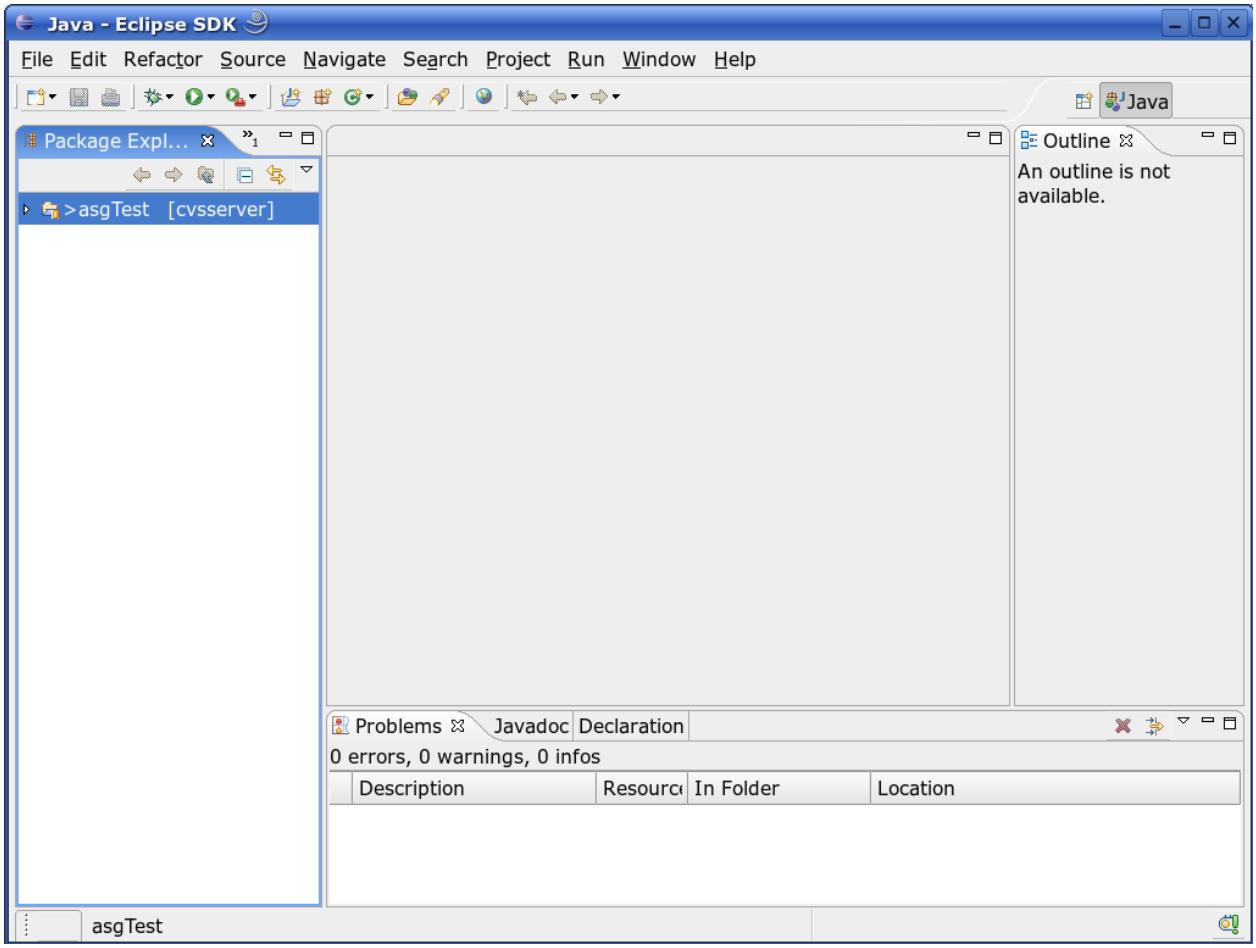
Select where you wish to check out the CVS module to. You may place it in various locations. If you use an existing Eclipse project, the module will appear as a folder within it. This example creates a new Simple Project.


Check Out As
Check Out As
Select the method of check out
CVS
Choose how to check out folder 'AsgServer102'
Check out as a project configured using the New Project Wizard
Check out as a project in the workspace
Project Name:
AsgServer102
Check out into an existing project
< Back
Next >
Finish
Cancel

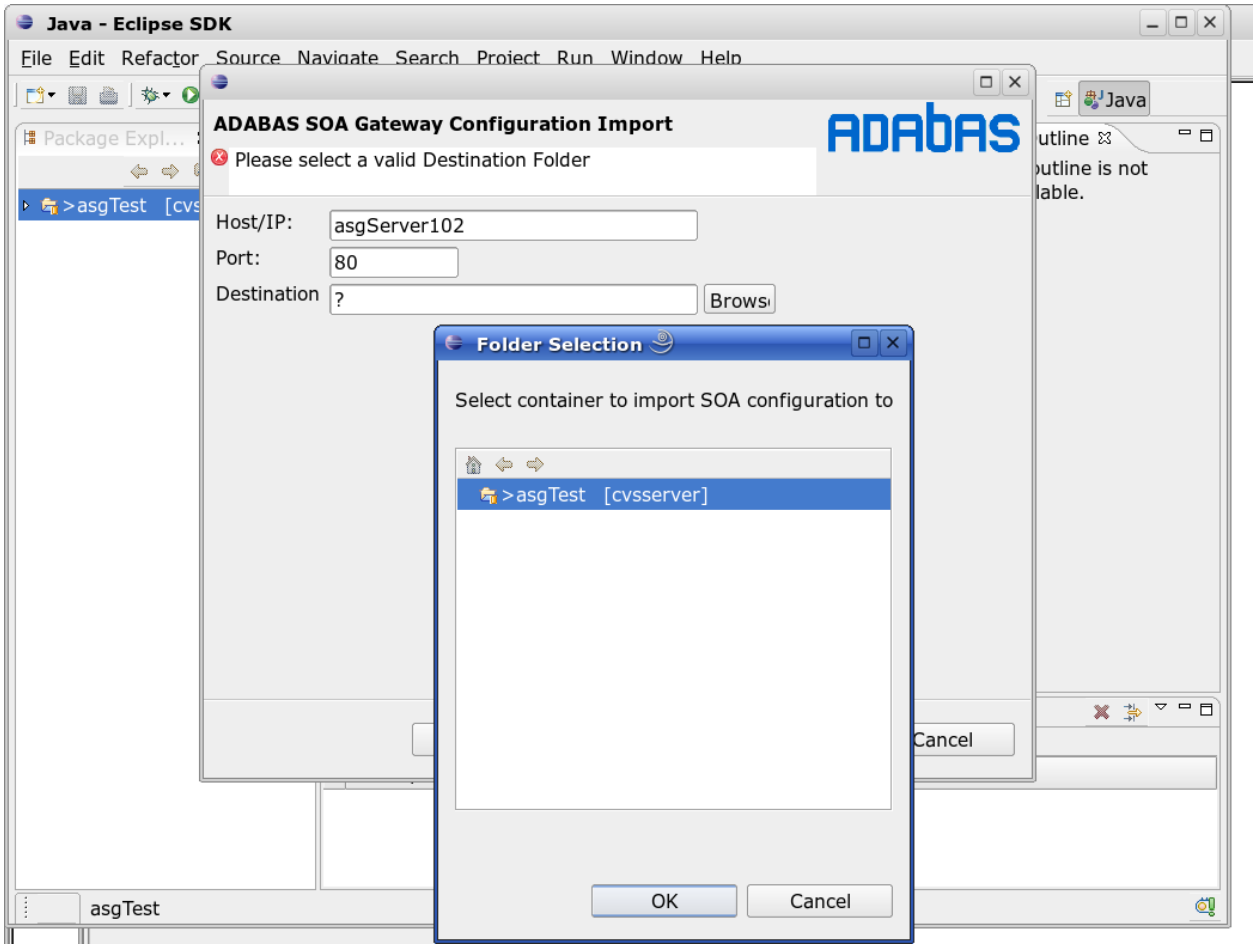

New Project
Select a wizard
Create a new project resource
Wizards:
C++
CVS
Eclipse Modeling Framework
Java
Plug-in Development
Simple
Project
< Back
Next >
Finish
Cancel
< Back
Next >
Finish
Cancel



The New Project is now shown in the Eclipse Package Explorer Window, the icon denoting it as a CVS item, and the server name shown at the end.

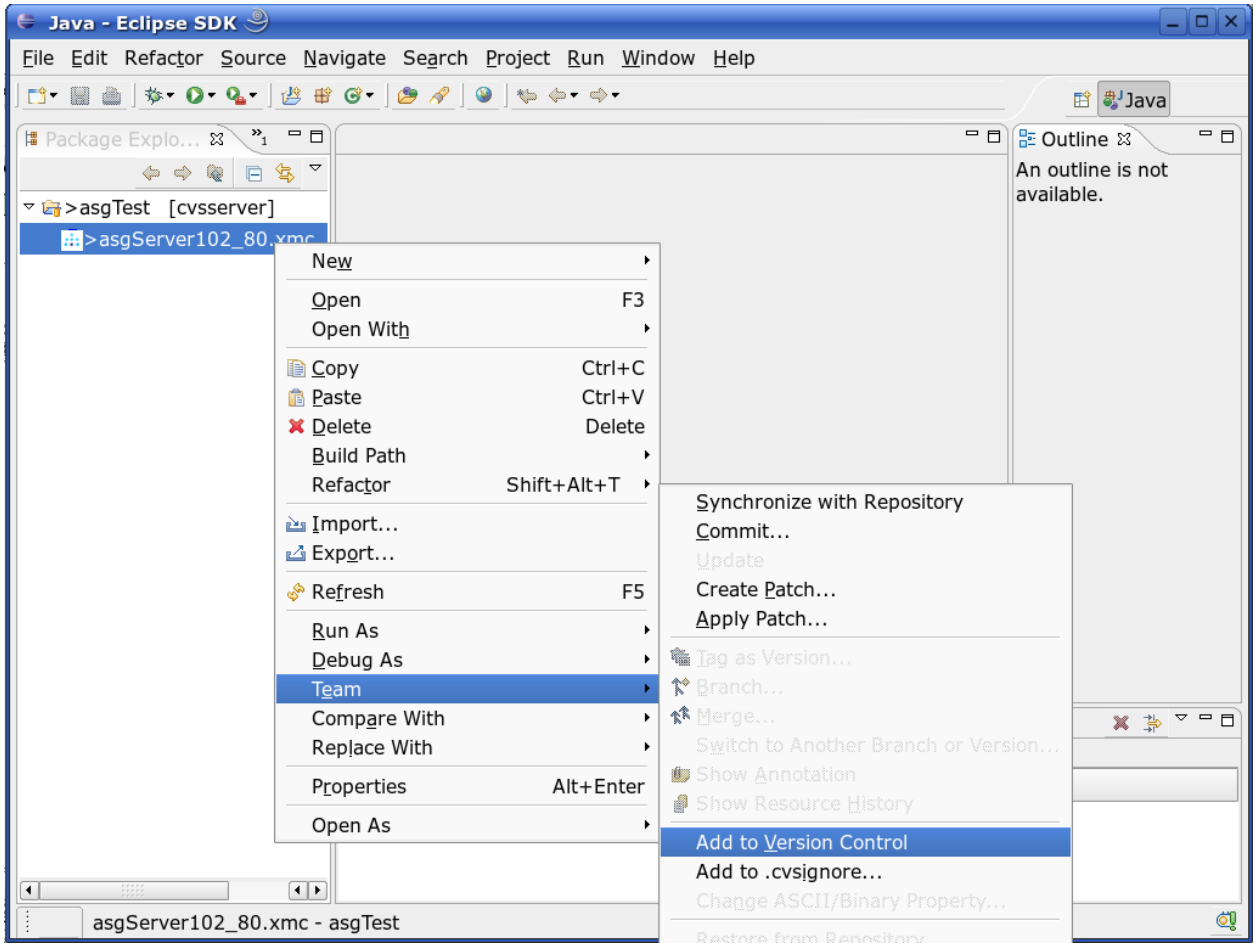


When you import items from ASG, you can now save them in this Project.



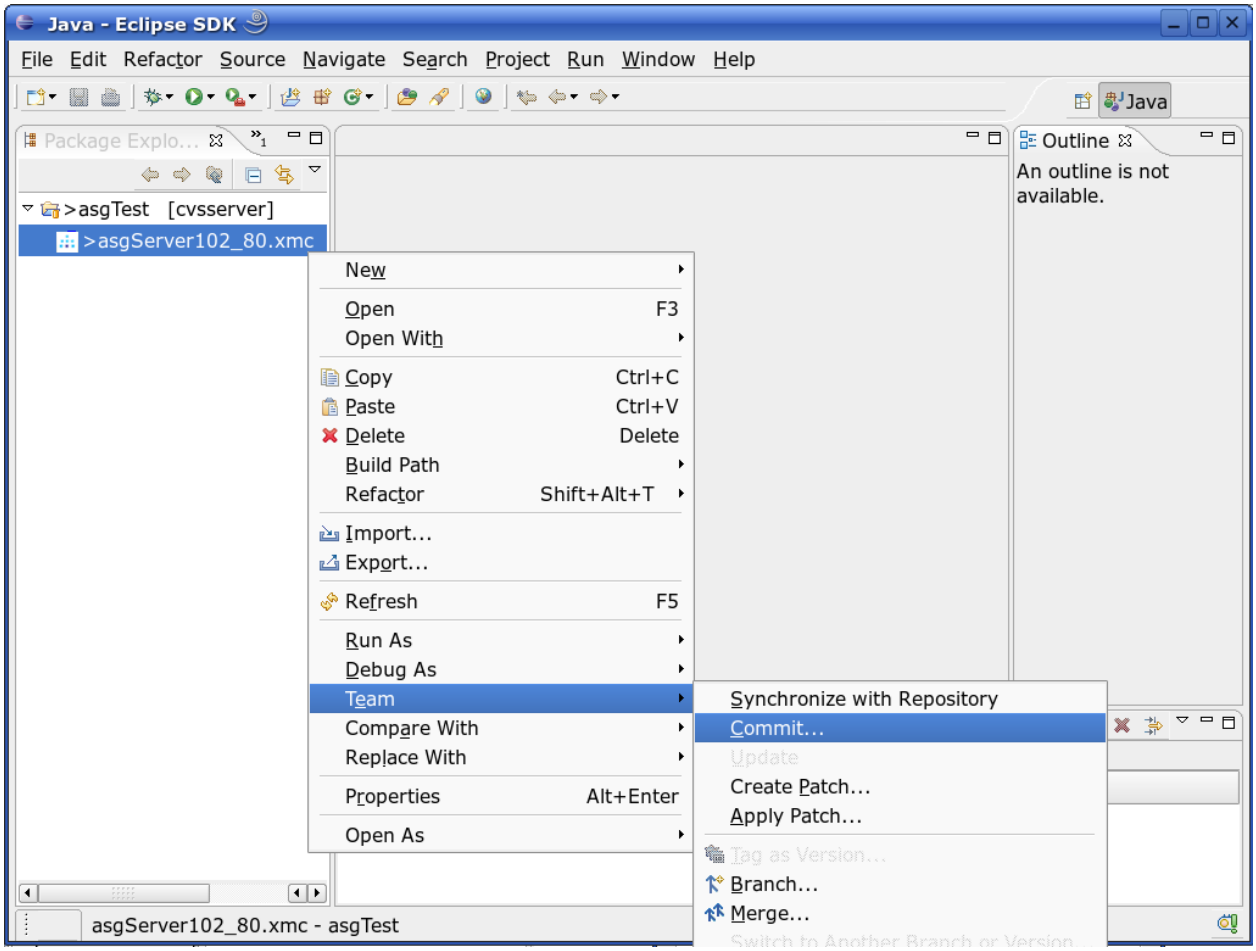
Adding Files to CVS

Note that your imported files will need to be explicitly added to the CVS repository. Also note that you will have to select the file type when adding it to CVS. This would normally be ASCII TEXT and not the default of Binary.

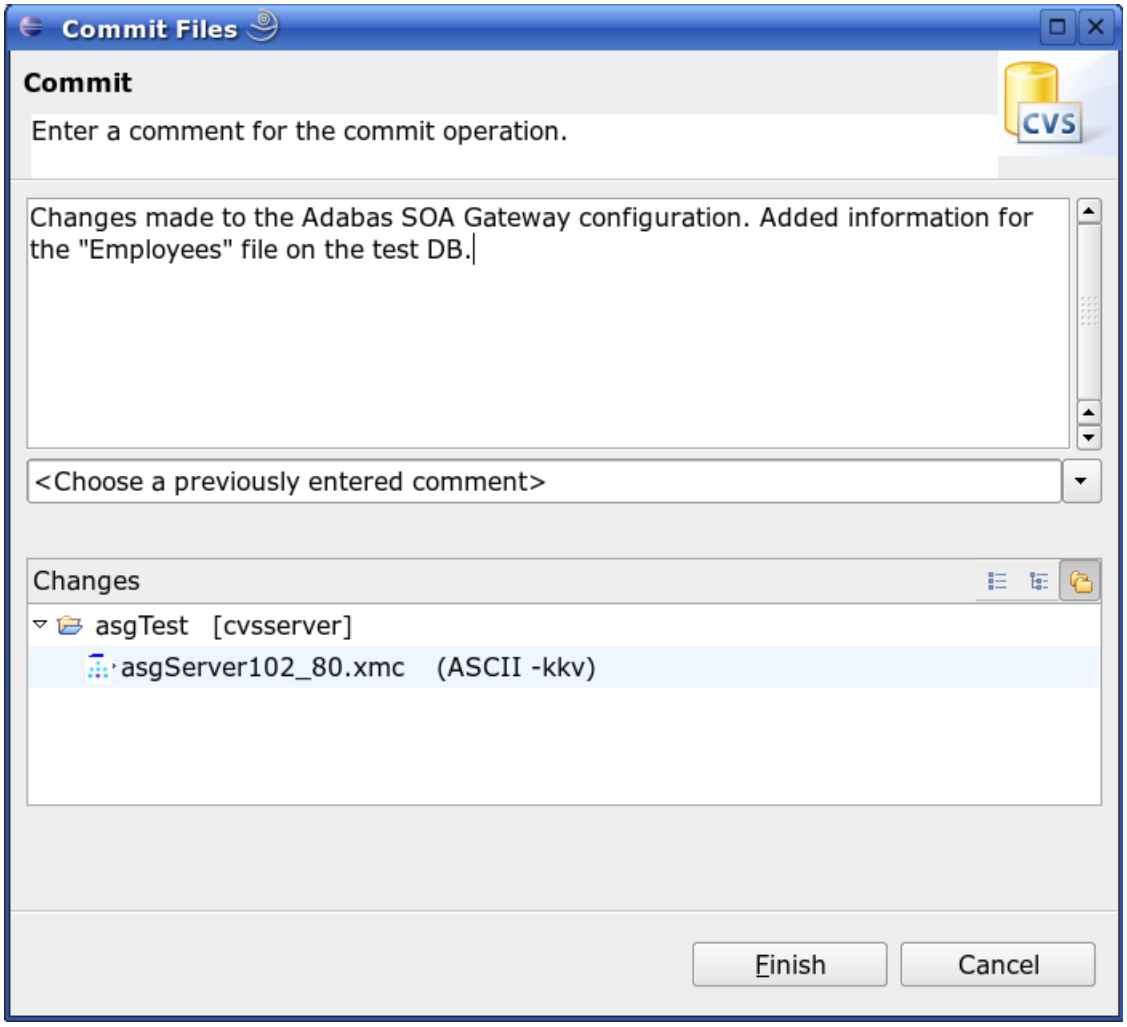


Making CVS aware of file changes

Every time you make a change to your ASG related files, you should commit those changes to the CVS repository.



A useful comment should be added during the commit describing the change(s) made. Ticket Id's and other related information can be added here also.



That should cover the basics required to use Eclipse in conjunction with CVS so that changes to your ASG configuration files are more controlled. Further information on the use of CVS with Eclipse, and how acquire and set-up a CVS server, can be found at the external sites listed below.

More Information

More information related to CVS can be found at **http://www.nongnu.org/cvs/**

More information related to CVS use with Eclipse can be found in the Eclipse documentation at **http://www.eclipse.org/**

14 Tut_02_List.java

```
import SoaG.*;
import SoaG.AdabasEmployeesMiniServiceStub.*;

/*
 *   The SOA Gateway LIST method will return all data matching the
 *   criteria at once, no matter how large the result set is.
 *
 *   So if you need (or want) to retrieve the result set in "chunks"
 *   of records, or your request requires complex search syntax,
 *   use the SELECT method.
 *
 *   Tut_03c_SelectConversational.java   provides an example for this.
 */

public class Tut_02_List {

  public static void main(String[] args) {

    try {
   AdabasEmployeesMiniServiceStub stub = new AdabasEmployeesMiniServiceStub();

   AdabasEmployeeKeyType keys = new AdabasEmployeeKeyType();
   keys.setPersonnel_id("50005*");

   AdabasEmployeeListElement listKey = new AdabasEmployeeListElement();
   listKey.setAdabasEmployeeListElement(keys);

   AdabasEmployeesMiniElement result = null;

   result = stub.list(listKey, null, null);

   AdabasEmployeesMiniElementType root = result.getAdabasEmployeesMiniElement();
   AdabasEmployeesMiniType group = root.getAdabasEmployeesMini();
   AdabasEmployeeType elements[] = group.getAdabasEmployee();
```

```
   System.out.println("Number of record read: " + elements.length);

   for (int i = 0; i < elements.length; i++) {
     AdabasEmployeeType r = elements[i];
     System.out.println("Record [" + i + "]"
                        + "Personnel Id=" + r.getPersonnel_id() + ", "
                        + "Name=" + r.getName() + ", "
                        + "First Name=" + r.getFirst_name() + ", "
                        + "City=" + r.getCity());
   }

    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

}
```

15 ex01_SoaGatewayFirst.php

```
<?php

/*
 *  This set of examples demonstrate the simplicity in accessing Adabas
 *  data as "WebServices" from PHP, based on the "EmployeesMini" view
 *  representing a subset ("View") of the Adabas demo file "Employees".
 *
 *  This first example outlines the usage of the PHP SoapClient class,
 *  which provides the infrastructure for issuing SOAP requests.
 *
 *  All examples are based on an Adabas SOA Gateway server running on
 *  host "soagate", port 8082, these need to be adjusted to the actual
 *  server host/port used at your site.
 *
 */


/*
 *  Instantiate the PHP "SoapClient" class
 */
try {
 /*
  * The only required parameter when instantiating "SoapClient" is the URL
  * pointing to the WSDL for the WebService to be accessed.
  */
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");
} catch (SoapFault $soapfault) {
 /*
  * In case of a SOAPFault being thrown, this will be caught and the fault
  * information printed. If a SOAPFault occurs outside of a "try/catch"
  * structure, PHP will abend the script with a generic message.
  *
  * Here we print the SOAPFault information in a structured way, the "<pre>"
  * tags are required to format the object nicely.
  */
```

```
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Without any knowledge of, or looking at Adabas definitions, a "WebService ↩
programmer"
 *  can easily retrieve the signature of any exposed Adabas SOA Gateway Service and
 *  code based on it.
 *
 * First we use a method of the SoapClient class to retrieve the functions exposed
 * by the specific "WebService", and print it:
 */

echo "<pre><B>Functions:</B>\n\n";
print_r($soapclient->__getFunctions());

/*
 *  The function prototypes shown by the __getFunction() method also depict the ↩
required
 *  parameters and the return values. Their definitions can be printed as well:
 */
echo ↩
"\n\n--------------------------------------------------------------------\n\n<B>Type ↩
Definitions:</B>\n\n";
print_r($soapclient->__getTypes());
echo "</pre>";

/*
 *  Proceed to ex02_SoaGatewayEmpList.php - List and format Employees data
 */

?>
```

16 ex02_SoaGatewayEmpList.php

```
<?php

/*
 *  This example demonstrates usage of the "list" function exposed by any
 *  Adabas SOA Gateway "WebService" from PHP, retrieving selected records
 * from the Adabas demo file "Employees", and formatting it, in 4 easy steps.
 */


/*
 *  Step 1: Instantiate the PHP "SoapClient" class
 */
try {
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 2: Build the key ("descriptor") array, due to parser requirements ALL
 *    key elements need to be specified, but elements may be left empty
 *    when unused.
 */
$listkey = array(
 'personnel_id' => '',
 'first_name' => '',
 'name' => '',
 'city' => 'CI*');

/*
 *  Step 3: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
```

```
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 4: Format the Employee records nicely into a HTML table.
 */

echo "<table border=1 cellpadding=5>";
echo "<tr><th>Personnel Id</th><th>Name</th><th>First Name</th><th>City</th><th ↩
width=200>Address Line</td>";

/*
 *  Loop through all "adabasEmployee" elements, creating a table row for every ↩
single one
 */

if ( isset($listresponse->adabasEmployees->adabasEmployee) )
{
 $Employees = $listresponse->adabasEmployees->adabasEmployee;
 if (!is_array($Employees))
  $Employees = $listresponse->adabasEmployees;

 foreach ($Employees as $Employee) {
  echo  "<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>",
   "$Employee->first_name</td><td>$Employee->city</td><td>";
  echo "<table>";

 /*
  *  Format the "address_line" element (a MU(ltiple value field)) as a table within ↩
the table
  */
  if (!is_array($Employee->address_line)) {
   echo "<tr><td width=200>$Employee->address_line</td></tr>";
  } else {
   foreach ($Employee->address_line as $addr) {
    echo "<tr><td width=200>$addr</td></tr>";
   }
  }
 echo "</table>";
 echo "</td></tr>";
}
}
/*
 *  Proceed to ex03_SoaGatewayEmpAdd.php - Add a new employee record
```

```
 */
?>
```

17 ex02a_SoaGatewayEmpListDescending.php

```
<?php

/*
 *  This example demonstrates usage of the "list" function exposed by any
 *  Adabas SOA Gateway "WebService" from PHP, retrieving selected records
 * from the Adabas demo file "Employees", and formatting it, in 4 easy steps.
 */


/*
 *  Step 1: Instantiate the PHP "SoapClient" class
 */
try {
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 2: Build the key ("descriptor") array, due to parser requirements ALL
 *    key elements need to be specified, but elements may be left empty
 *    when unused.
 */
$listkey = array(
 'personnel_id' => '50005*',
 'first_name' => '',
 'name' => '',
 'city' => '');

/*
 *  Step 3: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
```

```
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

echo "List of Employees by personnel id - Ascending<br><br>";
formatResponse($listresponse);

/*
 *  Step 4: Build the SOAP Header structure to trigger a DESCENDING
 *    read instead of an ascending one.
 */

$headers = array(
 'SOAGateway_Internal_Adabas_Read_Direction' => "Descending"
 );

$header =  new SoapHeader("http://www.risaris.com/namespaces/xmiddle",
       "adabasEmployeeHeader",
                           $headers, false);

$soapclient->__setSoapHeaders(array($header));

/*
 *  Step 5: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

echo "<br><hr><br>List of Employees by personnel id - Descending<br><br>";
formatResponse($listresponse);

/*
 *  Sub: Format the Employee records nicely into a HTML table.
 */
function formatResponse($listresponse){

 echo "<table border=1 cellpadding=5>";
```

```
 echo "<tr><th>Personnel Id</th><th>Name</th><th>First Name</th><th>City</th><th ↩
width=200>Address Line</td>";

 /*
  *  Loop through all "adabasEmployee" elements, creating a table row for every ↩
single one
  */
if ( isset($listresponse->adabasEmployees->adabasEmployee) )
{
 $Employees = $listresponse->adabasEmployees->adabasEmployee;
 if (!is_array($Employees))
  $Employees = $listresponse->adabasEmployees;

 foreach ($Employees as $Employee) {
  echo ↩
"<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>$Employee->first_name</td><td>$Employee->city</td><td>";

  echo "<table>";

  if (!is_array($Employee->address_line)) {
   echo "<tr><td width=200>$Employee->address_line</td></tr>";
  } else {
   foreach ($Employee->address_line as $addr) {
    echo "<tr><td width=200>$addr</td></tr>";
   }
  }
  echo "</table>";
  echo "</td></tr>";
 }

}

echo "</table>";
}
/*
 *  Proceed to ex03_SoaGatewayEmpAdd.php - Add a new employee record
 */
?>
```

18 ex02a_SoaGatewayEmpListSorted.php

```
<?php

/*
 *  This example demonstrates usage of the "list" function exposed by any
 *  Adabas SOA Gateway "WebService" from PHP, retrieving selected records
 * from the Adabas demo file "Employees", and formatting it, in 4 easy steps.
 */


/*
 *  Step 1: Instantiate the PHP "SoapClient" class
 */
try {
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 2: Build the key ("descriptor") array, due to parser requirements ALL
 *    key elements need to be specified, but elements may be left empty
 *    when unused.
 */
$listkey = array(
 'personnel_id' => '50005*',
 'first_name' => '',
 'name' => '',
 'city' => '');

/*
 *  Step 3: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
```

```
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

echo "List of Employees by personnel id = 50005* - Default sort order<br><br>";
formatResponse($listresponse);

/*
 *  Step 4: Build the SOAP Header structure to trigger a DESCENDING
 *    read instead of an ascending one.
 */

$headers = array(
 'SOAGateway_Internal_Adabas_Sort_Order' => "city"
 );

$header =  new SoapHeader("http://www.risaris.com/namespaces/xmiddle",
       "adabasEmployeeHeader",
                          $headers, false);

$soapclient->__setSoapHeaders(array($header));

/*
 *  Step 5: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

echo "<br><hr><br>List of Employees by personnel id = 50005* - now sorted by ↩
City<br><br>";
formatResponse($listresponse);

/*
 *  Sub: Format the Employee records nicely into a HTML table.
 */
function formatResponse($listresponse){
```

```
 echo "<table border=1 cellpadding=5>";
 echo "<tr><th>Personnel Id</th><th>Name</th><th>First Name</th><th>City</th><th ↩
width=200>Address Line</td>";

 /*
  *  Loop through all "adabasEmployee" elements, creating a table row for every ↩
single one
  */
if ( isset($listresponse->adabasEmployees->adabasEmployee) )
{
 $Employees = $listresponse->adabasEmployees->adabasEmployee;
 if (!is_array($Employees))
  $Employees = $listresponse->adabasEmployees;

 foreach ($Employees as $Employee) {
  echo ↩
"<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>$Employee->first_name</td><td>$Employee->city</td><td>";

  echo "<table>";

  if (!is_array($Employee->address_line)) {
   echo "<tr><td width=200>$Employee->address_line</td></tr>";
  } else {
   foreach ($Employee->address_line as $addr) {
    echo "<tr><td width=200>$addr</td></tr>";
   }
  }
  echo "</table>";
  echo "</td></tr>";
 }

}

echo "</table>";
}
/*
 *  Proceed to ex03_SoaGatewayEmpAdd.php - Add a new employee record
 */
?>
```

19 ex06_SoaGatewaySpecial_SubDescriptor.php

```
<?php

/*
 *  This example demonstrates usage of the "list" function exposed by any
 *  Adabas SOA Gateway "WebService" from PHP, retrieving selected records
 * from the Adabas demo file "Employees", and formatting it, in 4 easy steps.
 */


/*
 *  Step 1: Instantiate the PHP "SoapClient" class
 */
try {
 $soapclient = new SoapClient("http://localhost:8022/adabas_Employees_special?WSDL");
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 2: Build the key ("descriptor") array, due to parser requirements ALL
 *    key elements need to be specified, but elements may be left empty
 *    when unused. Here we are using the SUB-descriptor "department".
 */
$listkey = array(
        'personnel_id' => "",
        'first_name'    => "",
        'name'                  => "",
        'birthday'              => "",
        'city'                  => "",
        'dept'=> "",
        'job_title'             => "",
        'language_spoken'=>"",
```

```
        'leave_left'    => "",
        'dept_person'   =>  "",
        'currency_salary'=> "",
        'department'    => array('dept' => "MGMT"),
        'phonetic_name' => "");

/*
 *  Step 3: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 4: Format the Employee records nicely into a HTML table.
 */

echo "Find all Employees in department group \"MGMT\" - using an Adabas ↩
SUB-Descriptor<br><br>";
echo "<table border=1 cellpadding=5>";
echo "<tr><th>Personnel Id</th><th>Name</th><th>First ↩
Name</th><th>City</th><th>Department</td>";

/*
 *  Loop through all "adabasEmployee" elements, creating a table row for every ↩
single one
 */
foreach ($listresponse->adabasDemoEmployeesSpecial->demoEmployeeSpecial as $Employee) ↩
{
 echo  "<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>",

"$Employee->first_name</td><td>$Employee->city</td><td>$Employee->department_code</td>";

 echo "</td></tr>";
}

/*
 *  Proceed to ex03_SoaGatewayEmpAdd.php - Add a new employee record
 */
?>
```

20 ex06_SoaGatewaySpecial_SuperDescriptor.php

```
<?php

/*
 *  This example demonstrates usage of the "list" function exposed by any
 *  Adabas SOA Gateway "WebService" from PHP, retrieving selected records
 * from the Adabas demo file "Employees", and formatting it, in 4 easy steps.
 */


/*
 *  Step 1: Instantiate the PHP "SoapClient" class
 */
try {
 $soapclient = new SoapClient("http://localhost:8022/adabas_Employees_special?WSDL");
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 2: Build the key ("descriptor") array, due to parser requirements ALL
 *    key elements need to be specified, but elements may be left empty
 *    when unused. Here we are using the SUB-descriptor "department".
 */
$listkey = array(
 'personnel_id' => "",
 'first_name' => "",
 'name'   => "",
 'birthday'  => "",
 'city'   => "",
 'dept'=> "",
 'job_title'  => "",
 'language_spoken'=>"",
```

```
 'leave_left' => "",
 'dept_person' => array('dept' => "MGMT10", 'name' => "K*"),
 'currency_salary'=> "",
 'department' => "",
 'phonetic_name' => "");

/*
 *  Step 3: Execute the "list" request, passing the key array as the only parameter,
 *    the response object will consist of an "adabasEmployees" element containing
 *    an array of "adabasEmployee" elements.
 */
try {
 $listresponse = $soapclient->list($listkey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Step 4: Format the Employee records nicely into a HTML table.
 */

echo "Find all Employees in department group 'MGMT10' whose names start with 'K' - ↩
using an Adabas SUPER-Descriptor<br><br>";
echo "<table border=1 cellpadding=5>";
echo "<tr><th>Personnel Id</th><th>Name</th><th>First ↩
Name</th><th>City</th><th>Department</td>";

/*
 *  Loop through all "adabasEmployee" elements, creating a table row for every ↩
single one
 */
foreach ($listresponse->adabasDemoEmployeesSpecial->demoEmployeeSpecial as $Employee) ↩
{
 echo  "<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>",

"$Employee->first_name</td><td>$Employee->city</td><td>$Employee->department_code</td>";

 echo "</td></tr>";
}

/*
 *  Proceed to ex03_SoaGatewayEmpAdd.php - Add a new employee record
 */
?>
```

21 ex03_SoaGatewayEmpAdd.php

```
<?php

/*
 *  We now add a new Employee record, which is just as simple.
 */

try{
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");
} catch (SoapFault $soapfault) {
 printSoapFault($soapclient, $soapfault);
 return;
}

/*
 *  Constructing the "data record" is similar to building the key array
 *  for a "list" operation, MUs are represented by an array within the
 *  array.
 */
$adabasEmployee = array (
 'personnel_id' => '99999999',
 'first_name'  => 'Kirk',
 'name'   => 'Newlyadded',
 'city'    => 'City',
 'address_line' => array ('route 66', 'from here', 'to there', 'CA')
);

/*
 *  The expected structure is equivalent to the one returned by "list",
 *  thus we need to create an array of "adabasEmployee" elements, even
 *  though there is just one:
 */
$adabasEmployees = array($adabasEmployee);

/*
 *  Now add the Employee
```

```
 */
try {
 $Adabasresponse = $soapclient->add($adabasEmployees);
} catch (SoapFault $soapfault) {
 printSoapFault($soapclient, $soapfault);
 return;
}

/*
 *  An "add" results in a "short response", print the message:
 */
echo "<pre>result: $Adabasresponse->results</pre>";

/*
 *  The SOAPFault is handled in a function:
 */
function printSoapFault ($soapclient, $soapfault) {
 echo "<pre>";
 echo "\n\nSoap Fault occurred\n\nFaultCode..: ↩
".$soapfault->faultcode."\nFaultString: ".$soapfault->faultstring;
 echo "</pre>";
}

/*
 *  Proceed to ex04_SoaGatewayEmpGet.php - Get and display the newlyadded employee ↩
record
 */
?>
```

22 ex04_SoaGatewayEmpGet.php

```
<?php

/*
 *  The "get" operation is even easier.
 */
try{
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL", ↩
array('trace' => 1));
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
}

/*
 *  Construct the "key array", which consists of just one element,
 *  the primary key for the employees file - "personnel_id";
 */
$primKey = array('personnel_id' => '99999999');

/*
 *  Get the record
 */
try {
 $Adabasresponse = $soapclient->get($primKey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Check if we actually got the record we are looking for
 */
```

```
if (is_null($Adabasresponse->adabasEmployees->adabasEmployee)) {
 echo "No Employee with personnel_id=".$primKey['personnel_id'];
 return;
}

/*
 *  Print the formatted response
 */
echo "<pre>";
print_r($Adabasresponse);
echo "</pre>";

/*
 *  Proceed to ex05_SoaGatewayEmpDel.php - Delete the record added in ↩
ex03_SoaGatewayEmpAdd.php
 */
?>
```

23 ex05_SoaGatewayEmpDel.php

```
<?php

/*
 *  Finally we delete the Employee record with personnel_id=99999999 again
 */

try{
 $soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL", ↩
array('trace' => 1));
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
}

/*
 *  All we need is the primary key
 */
$primKey = array('personnel_id' => '99999999');

/*
 *  Delete takes the key as the input and returns a "short response" (just a message)
 */
try {
 $Adabasresponse = $soapclient->delete($primKey);
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

/*
 *  Print the response
 */
```

```
echo "<pre>";
print_r($Adabasresponse);
echo "</pre>";
?>
```

24 ex10_SoaGatewaySimpleForm.php

```
<?php
/*
 *  On entry to the form determine if the "List Employees" button has been pressed,
 *  if this is the case, retrieve the form field values.
 */
if (isset($_POST['submit'])) {
 $Fname  = $_POST["Fname"];
 $Lname  = $_POST["Lname"];
 $Persid = $_POST["Persid"];
 $City   = $_POST["City"];
}
?>
<html>
<head>
<title>Personnel Info</title>
</head>
<body>
<font face="courier">
<form method="post" action="<?php echo $PHP_SELF;?>">
Personnel Id: <input type="text" size="8" maxlength="8"   name="Persid" value="<? ↩
echo $Persid; ?>"><br />
First Name..: <input type="text" size="20" maxlength="20" name="Fname"  value="<? ↩
echo $Fname; ?>"><br />
Last Name...: <input type="text" size="20" maxlength="20" name="Lname"  value="<? ↩
echo $Lname; ?>"><br />
City........: <input type="text" size="20" maxlength="20" name="City"   value="<? ↩
echo $City; ?>"><br />

<br/>
<input type="submit" value="List Employees" name="submit">
</form>
<?
/*
 *  If the "List Employees" button has been pressed, retrieve the Employees record(s) ↩
and format
```

```
 *  them into a HTML table. The code used here is, with the exception of the ↩
variables used for
 *  building the keys array, equivalent to ex02_SoaGatewayEmpList.php
 */
if (isset($_POST['submit'])) {

 echo "Selected: Personnel Id=''".$Persid."'', ↩
first_name=''".$Fname."'',Name=''".$Lname;
 echo "'', City=''".$City."''<br/><br/>";

$soapclient = new SoapClient("http://localhost:8022/adabas_EmployeesMini?WSDL");

$key = array(
 'personnel_id' => $Persid,
 'first_name'  => $Fname,
 'name'   => $Lname,
 'city'   => $City
);

$result = $soapclient->list($key);

echo "<table border=1 cellpadding=5>";
echo "<tr><th>Personnel Id</th><th>Name</th><th>first Name</th><th>City</td><td ↩
width=200>Address</td></tr>";

if ( isset($result->adabasEmployees->adabasEmployee) )
{
 $Employees = $result->adabasEmployees->adabasEmployee;
 if (!is_array($Employees))
  $Employees = $result->adabasEmployees;

 foreach ($Employees as $Employee) {
  echo ↩
"<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>$Employee->first_name</td><td>$Employee->city</td><td>";

  echo "<table>";

  if (!is_array($Employee->address_line)) {
   echo "<tr><td width=200>$Employee->address_line</td></tr>";
  } else {
   foreach ($Employee->address_line as $addr) {
    echo "<tr><td width=200>$addr</td></tr>";
   }
  }
  echo "</table>";
  echo "</td></tr>";
 }

}

echo "</table>";
```

```
}
?>

</body></html>
```

25 ex15_SoaGatewayUpdateForm.php

```
<?php
/*
 *  This form incorporates ALL Adabas SOA Gateway access methods,
 *     list, get, add, update, delete
 *
 *  and demonstrates how easily web applications can be implemented
 *     based on the Adabas SOA Gateway
 */

global $PHP_SELF;

$Persid = "";
$Fname = "";
$Lname = "";
$City = "";
$Addr[0] = "";
$Addr[1] = "";
$Addr[2] = "";
$Addr[3] = "";
$msg = "";

if (isset($_POST['submit'])) {

 if ($_POST['submit'] != "Reset") {
  $Fname  = $_POST["Fname"];
  $Lname  = $_POST["Lname"];
  $Persid = $_POST["Persid"];
  $City   = $_POST["City"];
  $Addr = $_POST["Addr"];

  try {
   $soapclient = new SoapClient(
    "http://localhost:8022/adabas_EmployeesMini?WSDL");
  } catch (SoapFault $soapfault) {
   echo "<pre>";
```

```
  print_r($soapfault);
  echo "</pre>";
  return;
 }
}

$msg = "";

 if ($_POST['submit'] == "Delete") {

try {
 $Adabasresponse = $soapclient->delete(array('personnel_id' => $Persid));
} catch (SoapFault $soapfault) {
 echo "<pre>";

 print_r($soapfault);
 echo "</pre>";
 return;
}
$msg = $Adabasresponse->results;
 }

 if ($_POST['submit'] == "Get") {

try {
 $Adabasresponse = $soapclient->get(array('personnel_id' => $Persid));
} catch (SoapFault $soapfault) {
 echo "<pre>";
 print_r($soapfault);
 echo "</pre>";
 return;
}

if (!isset($Adabasresponse->adabasEmployees->adabasEmployee)) {
 $msg = "No Employee with personnel_id=".$Persid;
 $Lname = "";
 $Fname = "";
 $City = "";
 $Addr = array("", "", "", "");
} else {
 $Employee = $Adabasresponse->adabasEmployees->adabasEmployee;

 $Persid = $Employee->personnel_id;
 $Lname = $Employee->name;
 $Fname = $Employee->first_name;
 $City = $Employee->city;
 $Addr = $Employee->address_line;
}
 }

 if (($_POST['submit'] == "Add") || ($_POST['submit'] == "Update")) {
```

```
$adabasEmployee = array (
 'personnel_id' => $Persid,
 'first_name'  => $Fname,
 'name'   => $Lname,
 'city'    => $City,
 'address_line' => $Addr
);

$adabasEmployees = array($adabasEmployee);

if ($_POST['submit'] == "Add") {
 try {
  $Adabasresponse = $soapclient->add($adabasEmployees);
 } catch (SoapFault $soapfault) {
  echo "<pre>";
  print_r($soapfault);
  echo "</pre>";
 }
} else {
 try {
  $Adabasresponse = $soapclient->update($adabasEmployees);
 } catch (SoapFault $soapfault) {
  echo "<pre>";
  print_r($soapfault);
  echo "</pre>";
 }
}
 }
} else {
}
?>
<html>
<head>
<title>Personnel Info</title>

</head>
<body>
<font face="courier">
<form method="post" action="<?php echo $PHP_SELF;?>">
Personnel Id: <input type="text" size="8" maxlength="8"   name="Persid" value="<? ↩
echo $Persid; ?>"><br />
First Name..: <input type="text" size="20" maxlength="20" name="Fname"  value="<? ↩
echo $Fname; ?>"><br />
Last Name...: <input type="text" size="20" maxlength="20" name="Lname"  value="<? ↩
echo $Lname; ?>"><br />
City........: <input type="text" size="20" maxlength="20" name="City"   value="<? ↩
echo $City; ?>"><br />
Address.....: <input type="text" size="20" maxlength="20" name="Addr[]"   value="<? ↩
echo $Addr[0]; ?>"><br />
       .....: <input type="text" size="20" ↩
maxlength="20" name="Addr[]"   value="<? echo $Addr[1]; ?>"><br />
       .....: <input type="text" size="20" ↩
```

```
maxlength="20" name="Addr[]"   value="<? echo $Addr[2]; ?>"><br />

       .....: <input type="text" size="20" ↩
maxlength="20" name="Addr[]"   value="<? echo $Addr[3]; ?>"><br />
<br/>
<? echo $msg; ?><br><br>
<input type="submit" value="List" name="submit">
<input type="submit" value="Get"  name="submit">
<input type="submit" value="Add" name="submit">
<input type="submit" value="Update" name="submit">
<input type="submit" value="Delete" name="submit">
<input type="submit" value="Reset" name="submit">
</form>
<?
if (isset($_POST['submit'])) {

  if ($_POST['submit'] == 'List') {
 echo "Selected: Personnel Id=''".$Persid."'', ↩
first_name=''".$Fname."'',Name=''".$Lname;
 echo "'', City=''".$City."''<br/><br/>";

 $listkey = array('personnel_id' => $Persid, 'first_name' => $Fname, 'name' => ↩
$Lname, 'city' => $City);

 try {
  $listresponse = $soapclient->list($listkey);
 } catch (SoapFault $soapfault) {
  echo "<pre>";
  print_r($soapfault);
  echo "</pre>";
  return;
 }

 echo "<table border=1 cellpadding=5>";
 echo "<tr><th>Personnel Id</th><th>Name</th><th>First Name</th><th>City</th><th ↩
width=200>Address Line</td>";

 if ( isset($listresponse->adabasEmployees->adabasEmployee) )
 {
  $Employees = $listresponse->adabasEmployees->adabasEmployee;
  if (!is_array($Employees))
   $Employees = $listresponse->adabasEmployees;

  foreach ($Employees as $Employee) {
   echo ↩
"<tr><td>$Employee->personnel_id</td><td>$Employee->name</td><td>$Employee->first_name</td><td>$Employee->city</td><td>";

   echo "<table>";
   if (!is_array($Employee->address_line)) {
    echo "<tr><td width=200>$Employee->address_line</td></tr>";
   } else {
    foreach ($Employee->address_line as $addr) {
```

```
     echo "<tr><td width=200>$addr</td></tr>";
    }
   }
   echo "</table>";
   echo "</td></tr>";
  }

 }
  }

}
?>

</body></html>
```

26 empMiniList.rb (Ruby)

```
require 'soap/wsdlDriver'

wsdl_url = "http://soagate:8023/adabas_EmployeesMini?WSDL"

soap = SOAP::WSDLDriverFactory.new( wsdl_url ).create_rpc_driver

soap.wiredump_file_base = "soapresult"

param = {"personnel_id" => "50005*", "name" => "", "city" => ""}

result = soap.list( param )

print( "\nNumber of Employees is ", result.adabasEmployees.adabasEmployee.length, ↩
"\n\n")

for emp in (result.adabasEmployees.adabasEmployee)

  print(emp.personnel_id, ", ", emp.name, " ", emp.first_name, "\n")

end
```

27 ASGDemo.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EmployeesList
{
    class Program
    {
        static void Main(string[] args)
        {
            adabasEmployeesService emplService;
            adabasEmployeeKeyType keys;
            adabasEmployeesElementType result;
            adabasEmployeeType[] empl;

            keys = new adabasEmployeeKeyType();
            keys.personnel_id = "300000*";

            emplService = new adabasEmployeesService();

            try
            {
                result = emplService.list(keys);
            }
            catch (SystemException ex)
            {
                Console.WriteLine("exception: " + ex.Message);
                return;
            }

            empl = result.adabasEmployees;

            if (empl.Length > 0)
            {
                Console.WriteLine("Number of Employees returned: " + empl.Length);
```

```
            }

            for (int i = 0; i < empl.Length; i++)
            {
                Console.WriteLine("Record [" + i + "], Personnel_Id=" + ↩
empl[i].personnel_id +
                  ", Name=" + empl[i].name + ", First_Name=" + empl[i].first_name);
            }

        }
    }
}
```

28 MySQL_City.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CityList
{
    class Program
    {
        static void Main(string[] args)
        {
            RootElementNameService cityService;
            GroupElementNameKeyType key;
            RootElementNameElementType result;
            GroupElementNameType[] aCity;

            key = new GroupElementNameKeyType();
            key.ID = "305*";

            cityService = new RootElementNameService();

            try
            {
                result = cityService.list(key);
            }
            catch (SystemException ex)
            {
                Console.WriteLine("exception: " + ex.Message);
                return;
            }

            aCity = result.RootElementName;

            if (aCity.Length > 0)
            {
                Console.WriteLine("Number of Cities returned: " + aCity.Length);
```

```
            }

            for (int i = 0; i < aCity.Length; i++)
            {
                Console.WriteLine("City [" + i + "], ID=" + aCity[i].ID
                    + " Name = " + aCity[i].Name
                    + " CountryCode = " + aCity[i].CountryCode
                    + " District = " + aCity[i].District
                    + " Population = " + aCity[i].Population );
            }
        }
    }
}
```

29 FILE 90 (LOB demo file) FDT

ADACMP COMPRESS FILE=90

ADACMP FNDEF='1 , AA, 8 , A ,DE, UQ '

ADACMP FNDEF='1 , LM,32 , A ,NU '

ADACMP FNDEF='1 , LB, 0 , A ,LB, NV '

30 FILE 90 (LOB demo file) load parameters

Loading a LOB file involves two steps

1. loading the 'base' file
2. loading the LOB file

Sample load parameters for the 'base' file

```
//DDKARTE DD *

ADALOD LOAD FILE=90

ADALOD NAME=ASG-PHOTOS

ADALOD LOBFILE=91

ADALOD MAXISN=100

ADALOD DSSIZE=10B

ADALOD UISIZE=5B

ADALOD NISIZE=10B

ADALOD INDEXCOMPRESSION=YES

ADALOD ISNREUSE=YES

ADALOD LWP=1024K

ADALOD SORTSIZE=<sortsize>

ADALOD SORTDEV=<sortdev>

ADALOD TEMPSIZE=<tempsize>

ADALOD TEMPDEV=<tempdev>

/*
```

Sample load parameters for the 'LOB' file

```
//DDKARTE DD *
ADALOD LOAD FILE=91
ADALOD NAME=ASG-PHLOBS
ADALOD BASEFILE=90
ADALOD MAXISN=100
ADALOD DSSIZE=100B
ADALOD UISIZE=5B
ADALOD NISIZE=10B
ADALOD ISNREUSE=YES
ADALOD LWP=1024K
ADALOD SORTSIZE=<sortsize>
ADALOD SORTDEV=<sortdev>
ADALOD TEMPSIZE=<temspize>
ADALOD TEMPDEV=<tempdev>
/*
```

31 FILE90.FDT (LOB demo file FDT)

```
1 , AA, 8, A, DE, UQ ; personnel_id

1 , LM, 32, A, NU ; mime type for LOB

1 , LB, 0, A, NB,NV,LB ; LOB data
```

32 FILE90.FDU (LOB demo file load parameters)

```
dbid = 212

file = 90

name = photos

lobfile= 91

dssize = 50b

nisize = 10b

uisize = 5b

maxisn = 100

reuse = (isn,ds)
```

33 wsf_lobGet.php

Get a record from the Adabas demo file and save the LOB to a file, the extension is determined from the mime-type stored on the file,

```
<?php

$id = $_GET['id'];

$reqPayloadString = <<<XML

<emp:EmployeePhotoGetElement xmlns:emp="com.SOAGateway/EmployeePhoto">

<personnel_id>$id</personnel_id>

</emp:EmployeePhotoGetElement

> XML;

try {

$client = new WSClient(

array("to"=>"http://localhost:56000/adabas_blobs",

"useMTOM"=>TRUE,

"responseXOP"=>TRUE));

$reqMessage = new WSMessage($reqPayloadString);

$resMessage = new WSMessage('');

$resMessage = $client->request($reqMessage);

printf("Response = %s \n\n", $resMessage->str);
```

```
$cid2stringMap = $resMessage->attachments;

$cid2contentMap = $resMessage->cid2contentType;

$imageName;

if($cid2stringMap && $cid2contentMap){

foreach($cid2stringMap as $i=>$value){

$f = $cid2stringMap[$i];

$contentType = $cid2contentMap[$i];

if(strcmp($contentType,"image/pjpeg") ==0){

$imageName = "C:\\TEMP\\".$i."."jpg";

file_put_contents($imageName, $f);

echo "<pre>File saved as ".$imageName."</pre>";

}

if(strcmp($contentType,"image/gif") ==0){

$imageName = "C:\\TEMP\\".$i."."gif";

file_put_contents($imageName, $f);

echo "<pre>File saved as ".$imageName."</pre>";

}

if(strcmp($contentType,"audio/mpeg") ==0){

$imageName = "C:\\TEMP\\".$i."."mp3";

file_put_contents($imageName, $f);

echo "<pre>File saved as ".$imageName."</pre>";

}

}

}else{

printf("attachments were not found ");

}
```

```
} catch (Exception $e) {

if ($e instanceof WSFault) {

printf("Soap Fault: %s\n", $e->Reason);

} else {

printf("Message = %s\n",$e->getMessage());

}

echo "<pre>";

print_r($e);

print_r($reqMessage);

print_r($resMessage);

echo "</pre>";

}
```

34 Natural samples

35 Usage Governance Tutorials

Usage Governance can be reported in 3 ways and written to 3 output types.

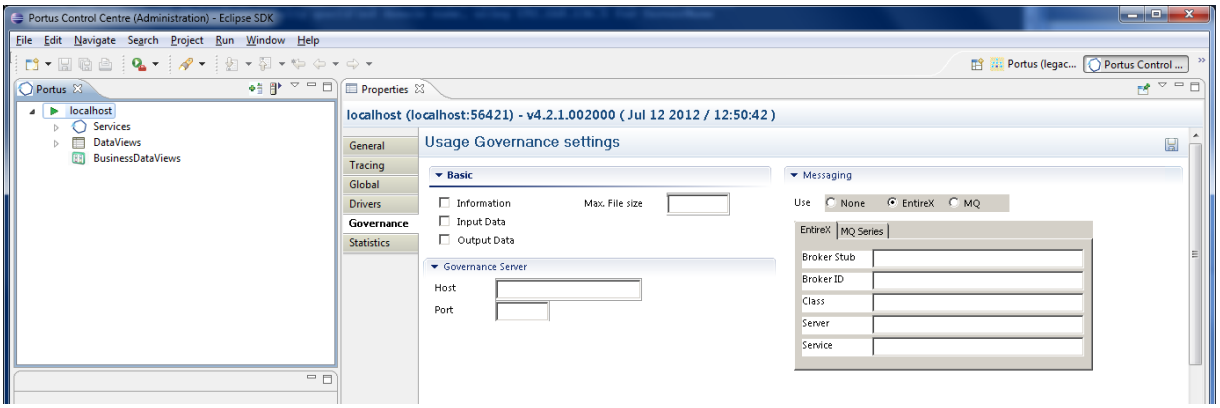
- **Using the local file system**
- **Using another SOA Gateway**
- **Using MOM**

36 Using the local file system

Tutorial: Write Usage Governance data to the local file system

1. In the Control Centre select the server for which usage governance data will be generated.
2. Select the Governance tab in the Properties view.

   To turn data collection on the Information box must be selected. If required Input Data and Output Data may also be selected.



3. Select the Save button.
4. Stop the server.See here on how to do this.
5. Start the server.
6. Issue a request to SOA Gateway e.g. a get request.
7. Go to SOA Gateway configuration directory.

   The default location of SOA Gateway configuration folder is [SERVER_INST]/Apache22/configuration replacing [SERVER_INST] with the location in which you have installed SOA Gateway.

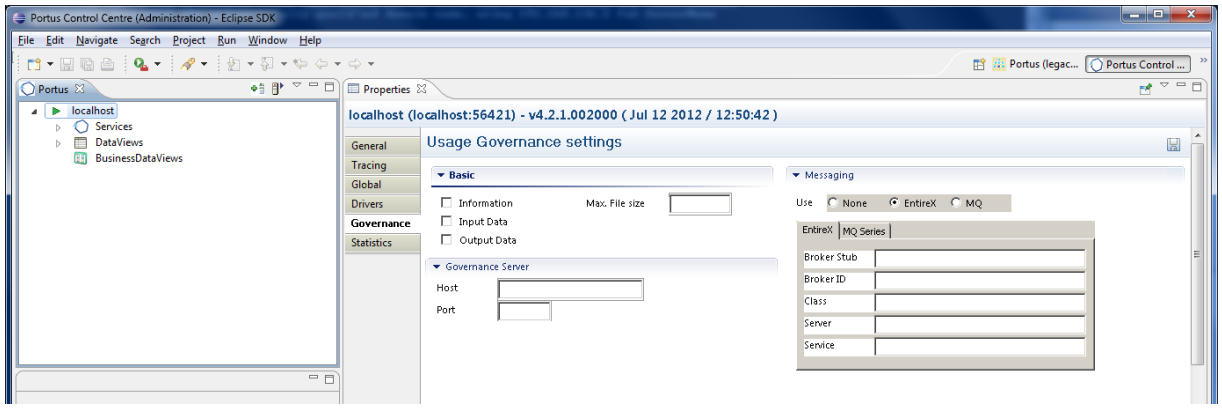
8. A file should be present in the format soag_usage_governance_yyyy_mm_dd_hh_mm_ss_ms.txt. Note that it is not possible to view this file while SOA Gateway is running. Stopping the server will create an XML file with the same name.
9. Open the file to check its contents.
10. There should be 1 entry for the get request plus all the governance data collected as per the options selected in 2.

37 Using another SOA Gateway

Tutorial: Writing Usage Governance data to another SOA Gateway

1. In the Control Centre select the server for which usage governance data will be generated.
2. Select the Governance tab in the Properties view.

   To turn data collection on the Information box must be selected. If required Input Data and Output Data may also be selected.
3. The Governance Server section is where we enter the details for another SOA Gateway.
   - Host : the IP address for the server.
   - Port : the port number of the server.
4. Enter these and save.



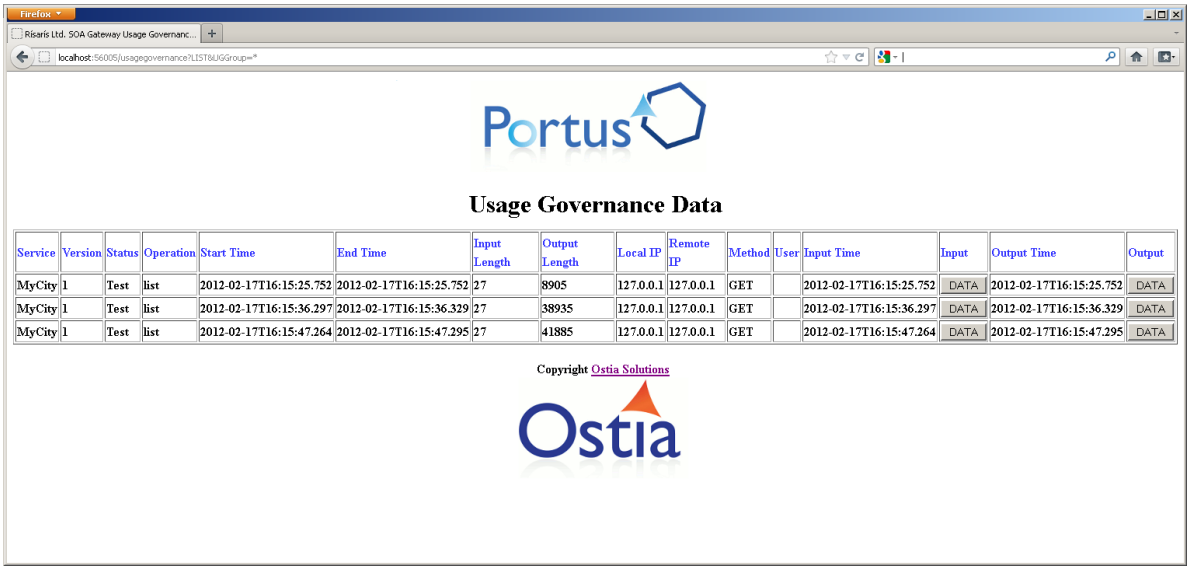
5. Stop the server. See here on how to do this.
6. Prior to restarting the server the following must be observed:
   - The Governance Server is running i.e. that SOA Gateway with the host and port number entered has been started and is awaiting requests.

- The governance web service, *usagegovernance*, has been created successfully and is loaded.

  If this has not yet been done, following the instructions **here** to do this.

7. Start the server. See here on how to do this.
8. On startup an initial connection is made to the governance server to verify its details. If this is unsuccessful an appropriate error will be written to the error log so this should be checked now.

   The default location of the Apache error_log is [SERVER_INST]/Apache22/logs/error_log replacing [SERVER_INST] with the location in which you have installed SOA Gateway.
9. Issue a request to SOA Gateway e.g. a list request.
10. There are a couple of ways to check that the data collection has reached its destination.
    - Query the database table directly..
    - Issue a list request on the usage governance web service e.g.



38 Using MOM

Tutorial: Write Usage Governance data to a MOM system

Usage Governance data can be written to a WebSphere MQ queue or an Software AG EntireX server.

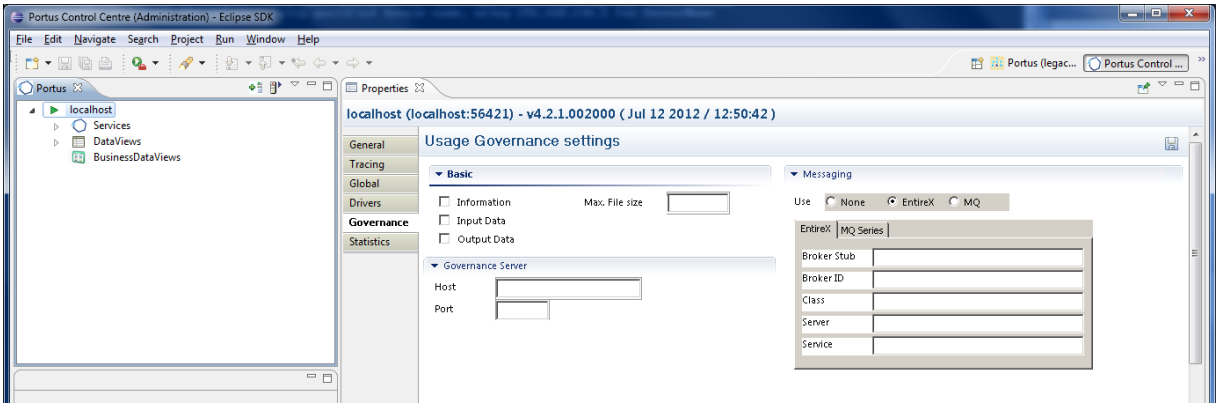
It is outside the scope of this tutorial to detail exactly how these are set up so it is assumed that the chosen Messaging system is configured correctly.

In this tutorial we will be using 2 SOA Gateway servers. One will direct usage governance data collected to a message queue and the other will read from this queue and process the input. The end result is that the target of the usage governance web service will receive an add request complete with the data.

1. In the Control Centre select the server for which usage governance data will be generated.
2. Select the Governance tab in the Properties view.

   To turn data collection on the Information box must be selected. If required Input Data and Output Data may also be selected.
3. The Messaging section allows one to enter the details required for None, MQ or EntireX :

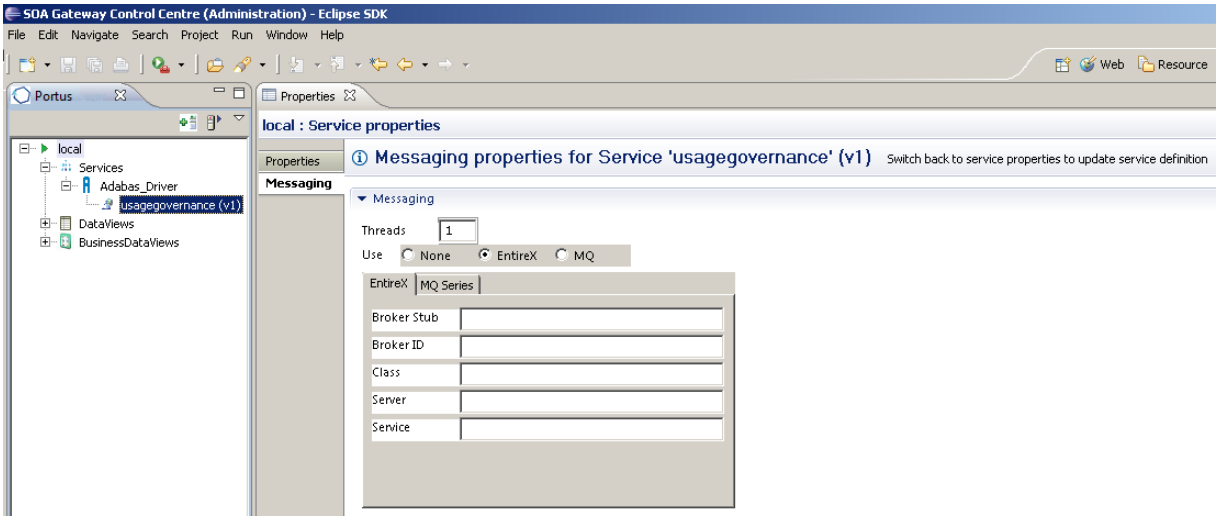
   Select the radio button of MQ or EntireX and fill in as appropriate for you MOM installation. Note that the queue will be opened for output and later on in this tutorial will be opened as an input queue.



4. Select the Save button.
5. Stop the server. See here on how to do this.
6. Start the server.
7. At this point SOA Gateway should have connected successfully to the manager and opened the output queue specified. Check the error log for any errors at this point.

   The default location of the Apache error_log is [SERVER_INST]/Apache22/logs/error_log replacing [SERVER_INST] with the location in which you have installed SOA Gateway.
8. Issue a request to SOA Gateway e.g. a get request
9. An add request, containing usage governance data pertaining to this request, should appear as an entry in the output queue.

   Use a search mechanism particular to your messaging system to check that this is the case. Note the syntax of this request is supported by the usage governance web service. If the latter has not yet been created, follow the steps outlined **here** to do so.
10. In the Control Centre, select the second SOA Gateway server. Open the Messaging tab in the Properties view for the usagegovernance web service:



11. Fill in the details for the messaging system you are using. Clearly, for MQ, the Input Q value has to be the value specified for Queue in step 3. For EntireX the details entered here should match those from 3.

Note: The Broker Stub value will depend on the system from which EntireX is being called i.e. broker32.dll for Windows, broker.so for Linux etc.

12. As per the hint, switch back to Service properties and select the Save button when complete.
13. SOA Gateway will now make contact with the appropriate messaging system. Check the error log for messages.

    The default location of the Apache error_log is [SERVER_INST]/Apache22/logs/error_log replacing [SERVER_INST] with the location in which you have installed SOA Gateway.
14. As soon as successful connections have been made, the input queue/service will be read and the message(s) processed.

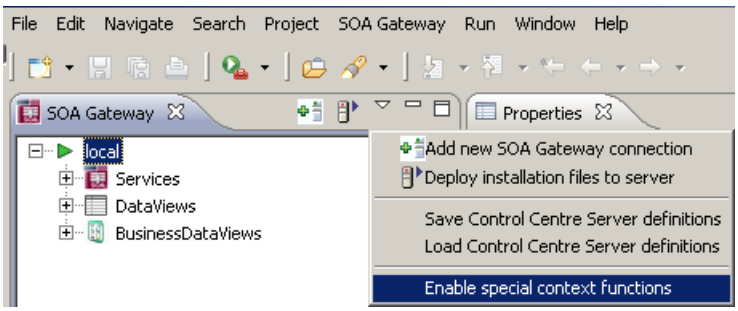
    There are a couple of ways to check the status of this.

    - Query the database table directly..
    - Issue a list request on the usage governance web service.
    - Check the Output Q (MQ) or the EntireX server for responses to the add request to the usage governance web service.

39 Create the usagegovernance web service

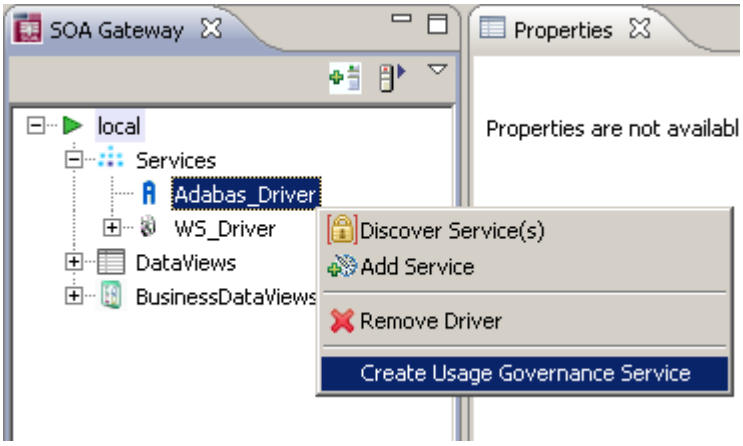
The usagegovernance web service is a bespoke service which is mandatory if usage governance data is being collected on a separate SOA Gateway. Here are the steps required to create it.

1. In the Control Centre select the server on which usage governance data will be stored.
2. Click on the View Menu icon and select the **Enable special context functions** entry.



3. Under Services select the driver which will support the new service. If one has not yet been created add one now. See here for details on this.

   Right click on the driver entry and select the **Create Usage Governance Service** item.



4. Fill in the details appropriate to your installation.

   Select the *Create database table now* check box.

   Select Create.

5. The web service is now created and its properties displayed .

40 SOAP over EntireX Tutorial

Tutorial: Send SOAP requests to EntireX

SOA Gateway and EntireX can be used together to enable sending SOAP requests to an EntireX service and to receive a response from a service. These request are asynchronous so it means that multiple requests can be sent from the client even though SOA Gateway may not be running. These requests are held until SOA Gateway reads and processes them. Likewise the responses are retained until such time that a client may request them.

Here is a top level view on the steps involved.

- For a particular web service specify the Broker details required.
- The resultant WSDL for the web service can be read by a client which will enable it to generate the appropriate SOAP request to send to the service.
- The resultant WSDL for the web service can be read by a client which will enable it to generate the appropriate SOAP request to read from the service.

Details

This tutorial will make use of Java wrapper/stub classes to access a SOA Gateway Service.

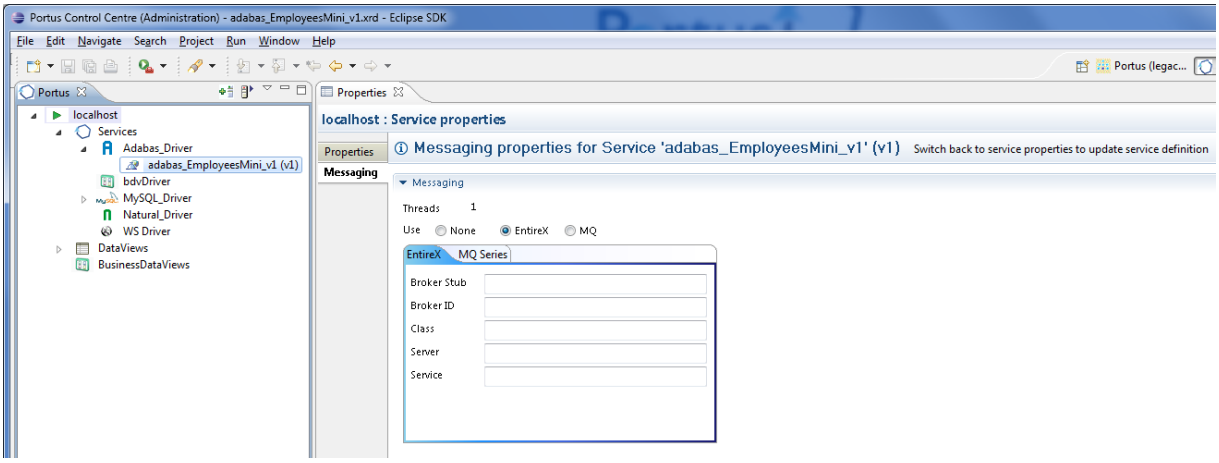
Java wrapper/stub classes are generated using the **Apache Axis2** feature **WSDL2Java**.

If you do not have it already, download and install the latest Axis2 kit.

The web service which we will be using is that described by the "adabas_EmployeesMini" DataView supplied with SOA Gateway.

1. Select the adabas_EmployeesMini web service and open the Messaging tab. Select the EntireX radio button and tab. Fill in the details appropriate to your installation.

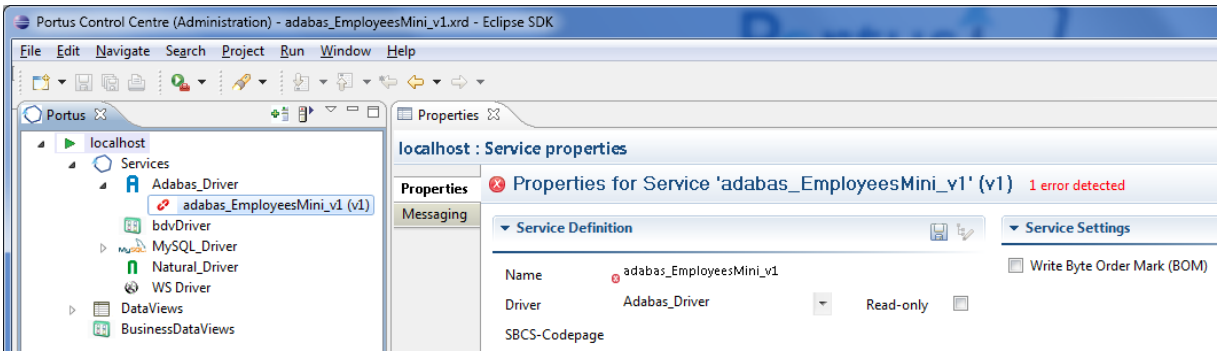
   Note: The Broker Stub value will depend on the system from which EntireX is being called i.e. broker32.dll for Windows, broker.so for Linux etc.



2. As per the hint, switch back to service properties and select the Save button when complete.
3. SOA Gateway will now make contact with the EntireX Broker. If an error occurs it will be highlighted as shown below.

   Check the error_log to find the cause. The default location of the Apache error_log is [SERVER_INST]/Apache22/logs/error_log replacing [SERVER_INST] with the location in which you have installed SOA Gateway. Please correct and try again.

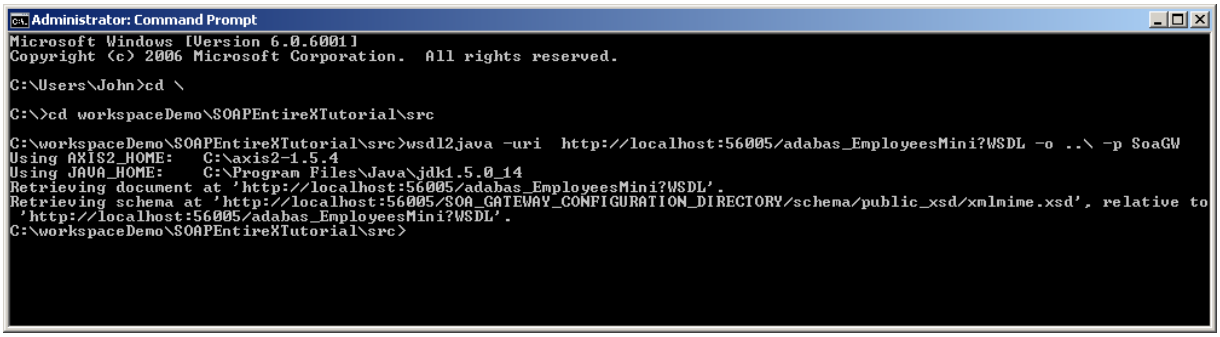
   If successful then carry on to next step.



4. Switch to the Java perspective (Window -> Open Perspective -> Other... -> Java (default). For more information on this aspect of Eclipse see Getting started with Eclipse.
5. Create a new Java-project naming it "SOAPEntireXTutorial".
6. Right-click the "SOAPEntireXTutorial" project folder, select "Build Path", then "Add External Archives.."
7. Add all .jar files from the axis2 "lib" directory to the project's Build-Path.
8. Right-click the "SOAPEntireXTutorial" project folder, select "Build Path", then "Add External Archives.."
9. Add the entirex.jar files from your EntireX installation e.g. ..\SoftwareAG\EntireX\classes

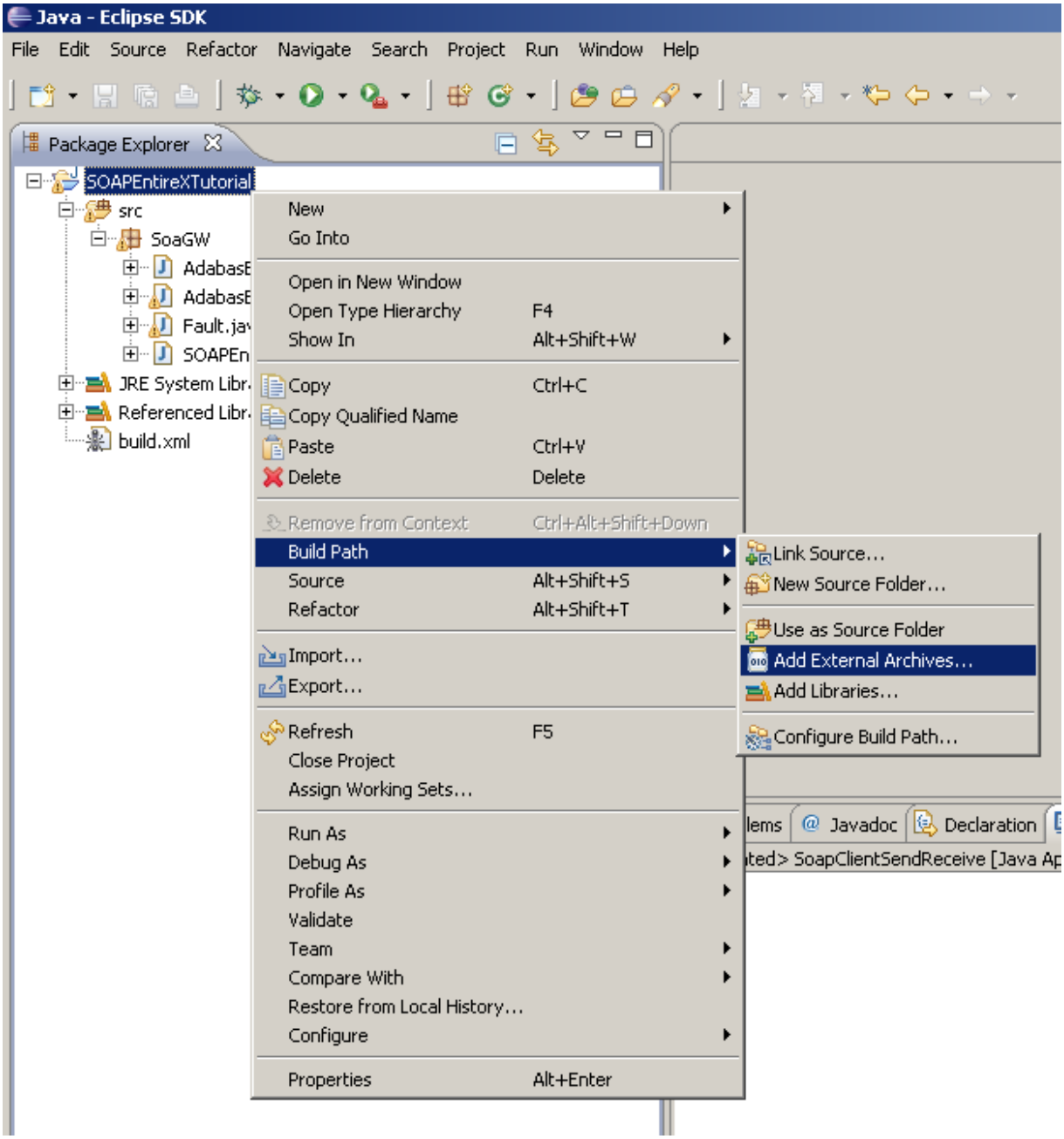
10. Open a command prompt (aka "DOS box"), change to the "SOAPEntireXTutorial\src" directory and run the following command:

    wsdl2java -uri http://<yourserver>:<yourport>/adabas_EmployeesMini?WSDL -o ..\ -p SoaGW

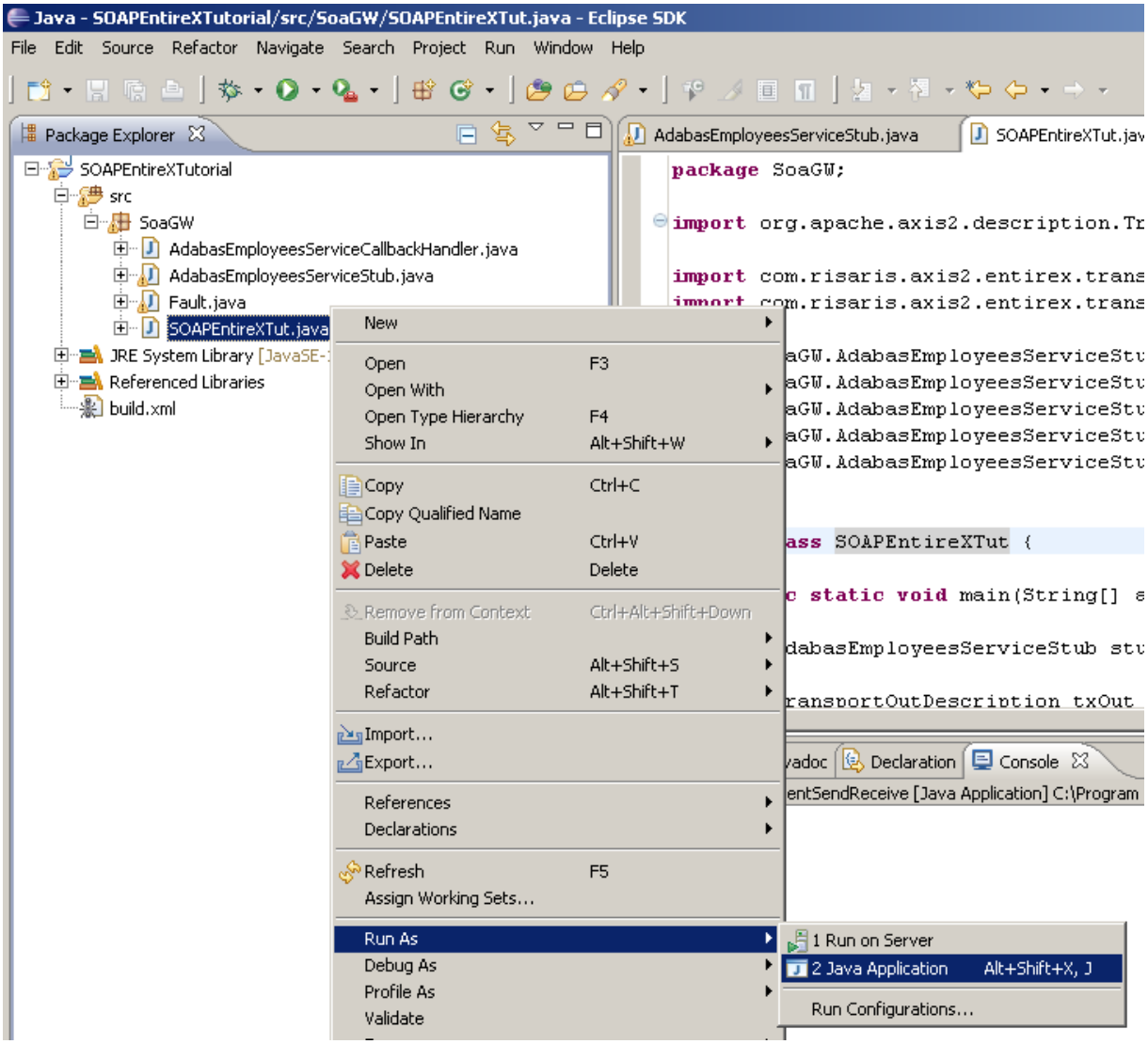


11. The following items are generated from the SOA Gateway WSDL:
    - A "Stub" class implementing all types and operations (ports / bindings).
    - A CallbackHandler - a stub class (not used in this tutorial) providing hooks for client-side extensions to the generated result- and error handlers.
    - A Fault class.
12. Get file SOAPEntireXTut.java and, when prompted, save in the ..\SOAPEntireXTutorial\src\SoaGW directory. Right-click on SOAPEntireXTutorial and select Refresh(F5). SOAPEntireXTut.java should appear in the explorer window.
13. Get file exxTransports-0.1.jar and, when prompted, save it in a local directory.
14. Right-click on SOAPEntireXTutorial and select Build Path -> Add External Archives...

    Navigate to where exxTransports-0.1.jar was saved and select Open to add it the project.



15. Right-click on the file and select Run As -> Java Application:



16. The output appears in the "Console" window: