SOAP with kbmMW

SOAP (Simple Object Access Protocol) is an XML based communication form which is designed to be portable between platforms. Because of this, SOAP have been getting quite allot of attention during the last few years. The actual implementation of SOAP usually goes under the name 'Web service'.

One of the things that make SOAP interesting is that any SOAP based message's layout, syntax and contents can be expressed in a separate XML based document, called a WSDL file. WSDL is an acronym for Web Service Definition Language.

By providing a well formed WSDL, any client will, in theory, be able to understand how to communicate with a SOAP server. I mention 'in theory' because in reality the actual implementation of SOAP varies quite allot from between platforms, programmer languages and applications. Thus its not uncommon to have a well formed WSDL file which is not completely or correctly understood by some SOAP implementation, even if the WSDL file have passed numerous public test suites for conformability, syntax etc.

Despite those and other problems, SOAP has become a defacto standard for loose inter platform communication.

'Loose inter platform' also indicates, that SOAP shouldn't be first choice of communication protocol in setups where the platforms are known, and are able to communicate using a binary, proprietary protocol.

Why is that you may ask? Well, just the fact that XML is what is used for communicating indicates that the amount of data transferred is much larger than if using a proprietary protocol. In addition the SOAP protocol itself has some rather lengthy so called envelope descriptions.
Another matter is the deciphering of a SOAP message. Since there are some degrees of freedom in how the message is generated based on a specific WSDL, the receiving end will have to parse the message, covering all the flexibilities. This of course uses extra CPU cycles and memory compared to a proprietary protocol.

Thus my advice is not to use SOAP as the primary communication protocol between entities who are capable of communicate via a binary proprietary protocol, but instead add SOAP as an additional (and optional) communication port, specifically for letting external, unknown systems interface with your server.

All this aside, kbmMW's XML and SOAP implementation is very fast, and is coded to use up as little memory as possible.

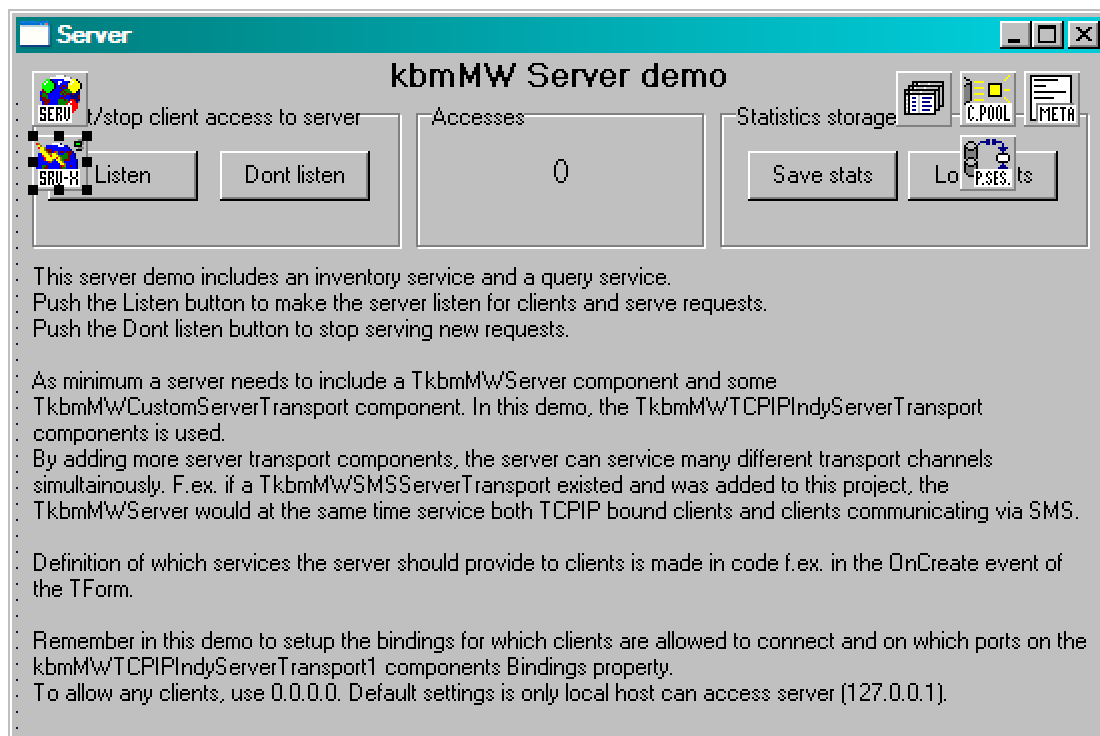
Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Adding SOAP capabilities to your server

It's really quite easy to add SOAP capabilities to your server. You will have to either choose an existing TCPIP based request/response type transport, like TkbmMWTCPIPIndyServerTransport, or add a new if you are adding SOAP as an additional interface to the world. For the samples in this document we use the Indy TCPIP transports.

Publish/subscribe based transports do not support using SOAP as a mean of transfer, simply because of the fundamental differences between the ways of communication.

For simplicity, let's modify the standard demo server to communicate SOAP.



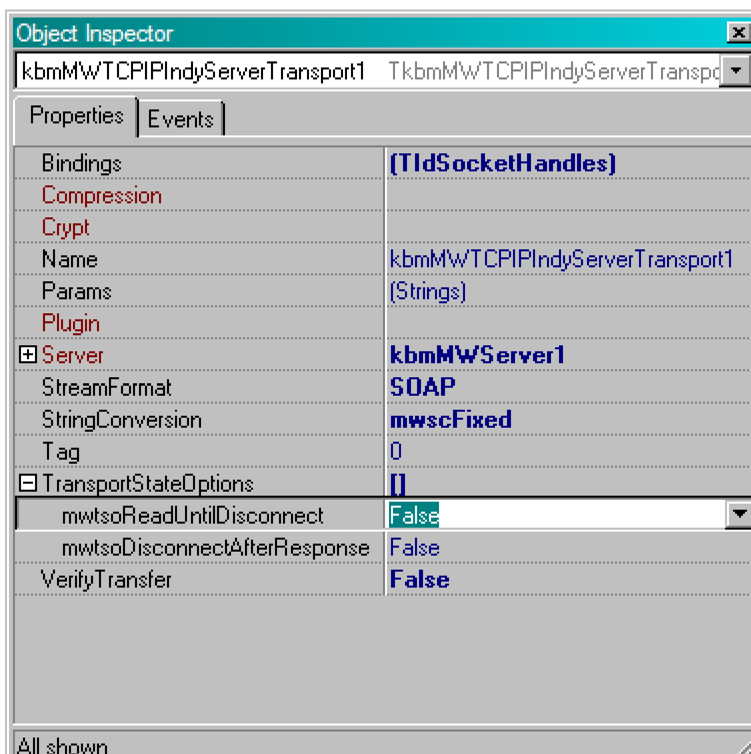
Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Select the **TkbmMWTCPIPIndyServerTransport**, and set its **StreamFormat** property to SOAP.

Optionally set the **TransportStateOptions** to **mwtsoDisconnectAfterResponse**. This is usually not required, unless the SOAP client is communicating through HTTP proxy servers. The setting instructs kbmMW to disconnect a client after the client have provided the request and received the response. It makes kbmMW operate stateless in a similar manner as a web server.

And set **VerifyTransfer** to **false**. We don't want to have kbmMW put its verification header into our SOAP traffic.

StringConversion should usually be left at **mwscFixed**, which ensures that date/time/datetime values are transferred according to XML standards. If set to mwscLocale, the format will be according to whatever format setting your server has defined.



Notice that these settings are transport oriented, rather than stream format oriented. Thus if you were using other stream formats, you could still choose to benefit from these settings in other situations.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next we have to look at some stream format settings. The stream formatter is configured through the **Params** property of the transport.

If your clients are entering via HTTP (which most external ones usually are) add the following to Params:

KBMMWSOAPVIAHTTP=1

This ensures that the SOAP stream format handles and generates HTTP headers, rather than having a simple 4 byte length counter at start of the message.

If you want to restrict the part of the incoming message that is parsed to check if its a valid message, you can set:

KBMMWMAXHTTPHEADERSIZE=somevalue

somevalue refers to the number of bytes that should be parsed of the header. Default is 512.
If you definitely know that a client is sending a correct request, but the server decides its an invalid request, it could be because the HTTP header from the client is larger than 512 bytes, in which case you should increase this value.

Why not just set it to 10.000.000? Because a malicious client could choose to send lots of crap requests with a large header which would make the server spend its energy on spam, rather than serving valid requests. Thus a good balanced value should be chosen.

Default the SOAP implementation is using a DOM parser. DOM (Document Object Model), is a parser which produce a tree containing information from the XML document. This makes it easy to extract information from an XML document. And in cases where XML referencing are used, it's essential to use the DOM model.

Another model is called SAX (Simple API for XML). Instead of building a tree in memory, events are being fired each time the parser have something it thinks the application can use. A SAX parser is very fast, and have a very low memory foot print, but since it's a single pass method, using XML referencing is not possible.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

It’s recommended to use the DOM parser because of its better capabilities. The DOM parser is based on the SAX parser, and is optimized for speed and low memory usage. However if you want to switch parser set the following parameter:

KBMMWSOAPUSESAX=1

Finally it’s possible to alter the default Mime type which is **text/xml** to something else by setting:

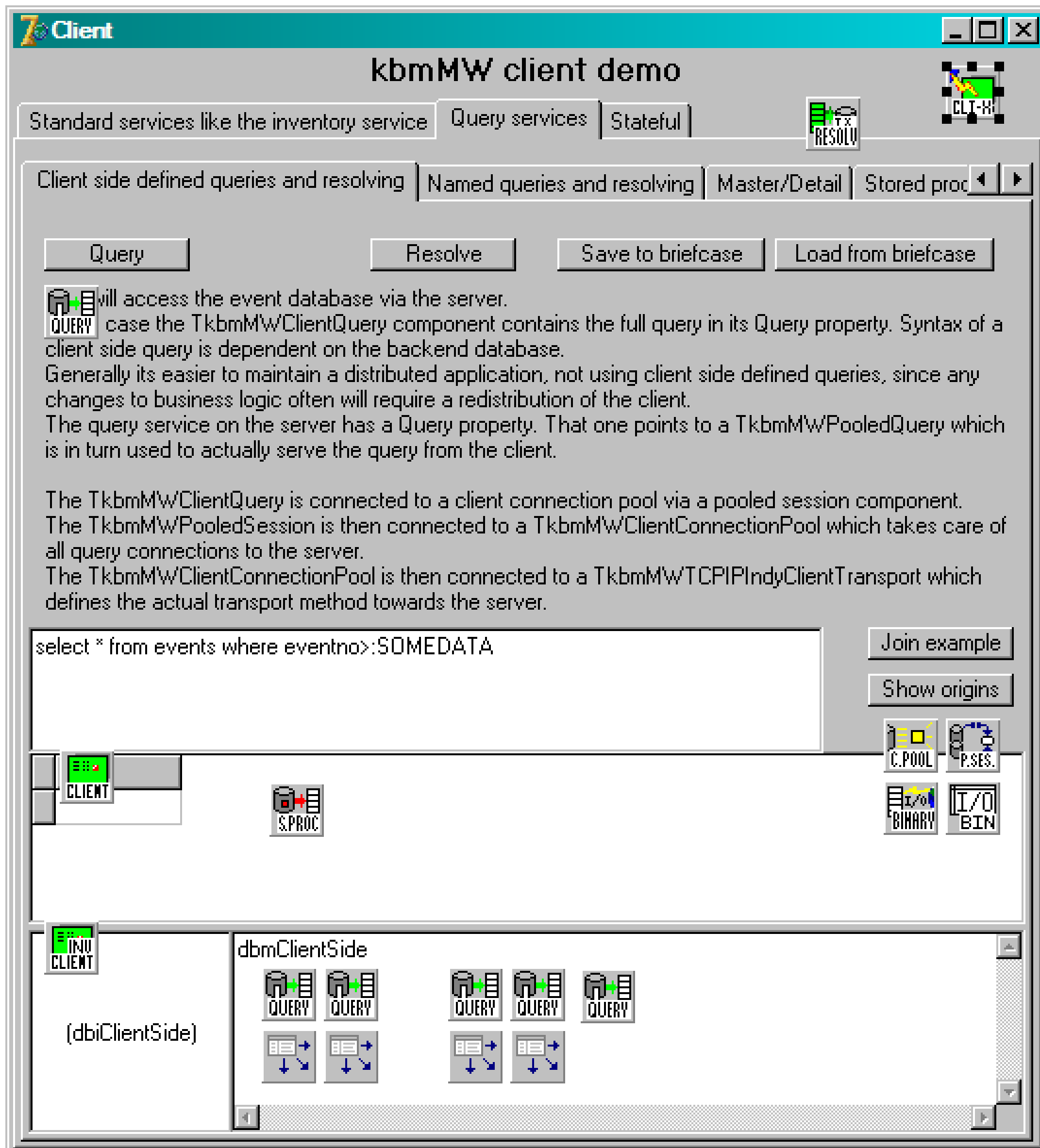
KBMMWMIMETYPE=somemimetype

somemimetype should be given without ‘ or “. It’s usually not recommended nor required to set this parameter.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

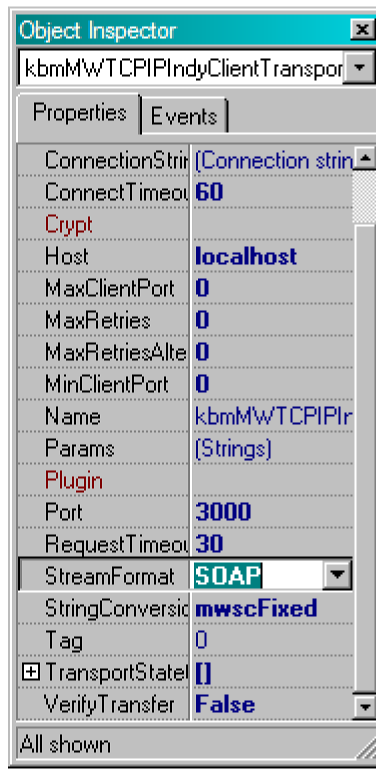
Making a kbmMW client communicate SOAP with a server

Next logical step would of course be to let an existing kbmMW demo client communicate with the demo application server using SOAP.



Copyright 2001-2004 Components4Developers
http://www.components4developers.com

It's important to set the **TkbmMWTCPIPIndyClientTransport** properties up the same as the server.



The **Params** property has some additional possible values:

KBMMWHTTPPOSTURL=someurl

This parameter controls what URL will be stated in the HTTP header (if the HTTP header generation is enabled... see KBMMWSOAPVIAHTTP parameter).
Ex: **KBMMWHTTPPOSTURL=http://servers.components4developers.com:4500/server1**

If you are not going through a HTTP proxy, its not important to set this parameter.
Otherwise it should be set to the true URL to your application server while the Host/Port properties of the transport should point to the HTTP proxy server.

Now your demo client and application server should be able to talk again, now using SOAP as their communication protocol.

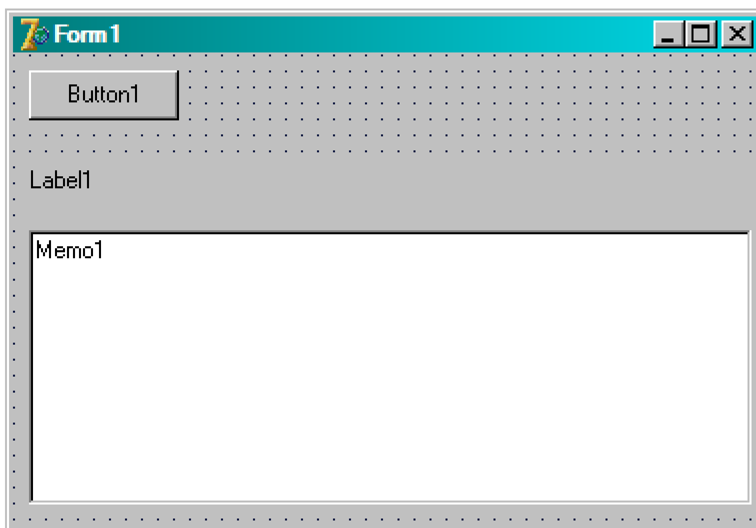
Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Creating a non kbmMW SOAP client using Delphi

If you have a Delphi application which is not kbmMW based, you can still utilize the SOAP transport as a mean of communication.

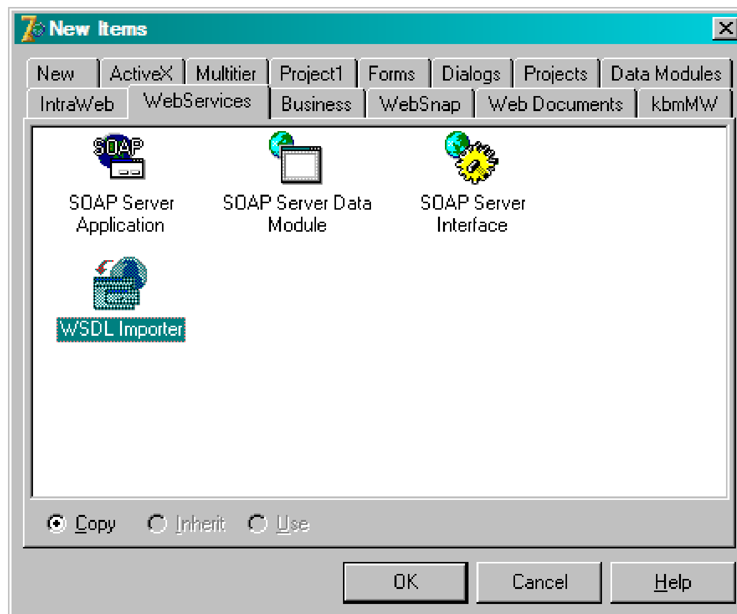
This time you need to import the kbmMW.wsdl file, which is delivered as part of the kbmMW installation, into the Delphi non kbmMW application.

For our sample first create a new Delphi application, and make it look similar to this:

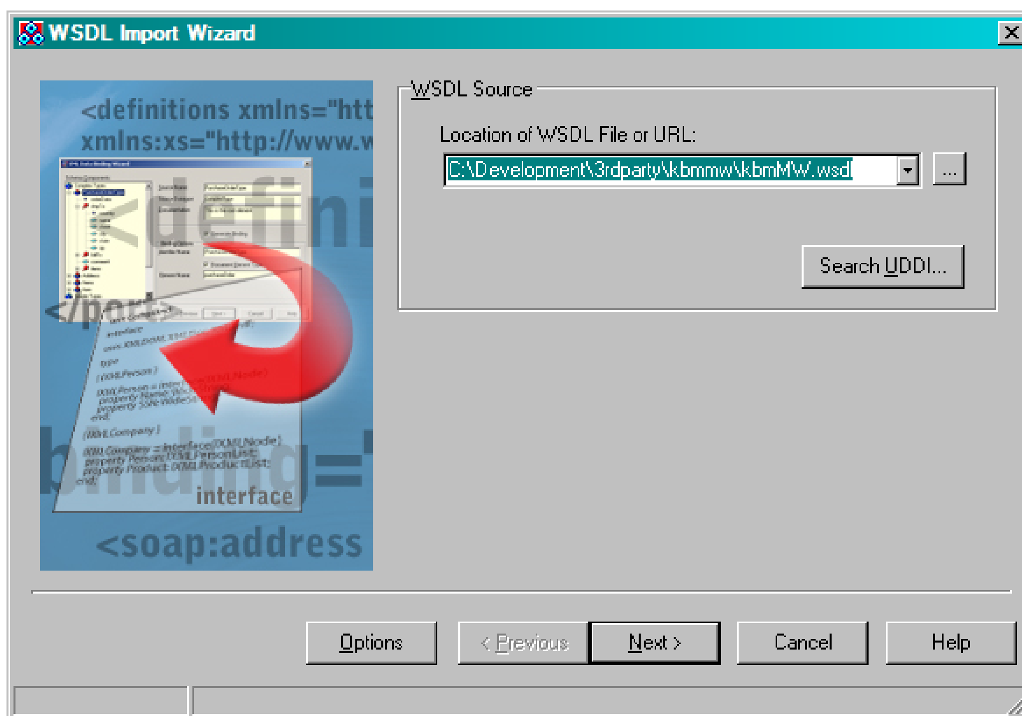


Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next step is to import the WSDL file using Delphi's WSDL importer.
Choose File->New->Other and select the WSDL importer.



Now select the kbmMW.wsdl file:



Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Click Next and Finish. You might want to set up some Options for example regarding proxy servers etc before clicking Finish.

Then a kbmMW.pas file has been generated. This contains all the stub code needed to communicate with a kbmMW based application server.

Make sure that you include kbmMW in unit1's uses clause.

Now write an event handler for Button1.OnClick

```
procedure TForm1.Button1Click(Sender: TObject);
var
   server:IProcessRequestSOAPPort;
   req:TkbmMWSOAPRequest;
   res:TkbmMWSOAPResponse;
begin
     server:=GetIProcessRequestSOAPPort(false,
       'http://servers.components4developers.com:4500/server1',nil);

     req:=TkbmMWSOAPRequest.Create;
     try
        // Setup request.
        req.ServiceName:='KBMMW_INVENTORY';
        req.ServiceVersion:='';
        req.StateID:=-1;
        req.Func:='LIST';

        // Call app server.
        res:=server.ProcessRequest(req);
        try

           // Check response.
           Label1.Caption:=res.StatusText;
           if res.StatusCode=0 then // OK
           begin
                Memo1.Text:=res.Result;
           end;
        finally
           res.Free;
        end;
     finally
        req.Free;
     end;
end;
```

First the `GetIProcessRequestSOAPPort` creates a connection to the application server using the specified URL.

Next a request object is built. In this sample we call the servers inventory service and asks for a list of services.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Then we call ProcessRequest and finally check the returned result.
Running this sample should end up in the memo box which contain a list of services/versions supported by the application server and a status message.

It's the responsibility of the developer to clean up the returned response object by freeing it at an appropriate time.

The request object can be reused for multiple calls if required, but must be freed when it's not required any longer.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Let’s modify the sample to ask for the syntax of a specific service. Alter the OnClick event code.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   server:IProcessRequestSOAPPort;
   req:TkbmMWSOAPRequest;
   res:TkbmMWSOAPResponse;
   args:variant;
begin
     server:=GetIProcessRequestSOAPPort(false,
       'http://servers.components4developers.com:4500/server1',nil);

     req:=TkbmMWSOAPRequest.Create;
     try
        // Setup request.
        req.ServiceName:='KBMMW_INVENTORY';
        req.ServiceVersion:='';
        req.StateID:=-1;
        req.Func:='GET SYNTAX ABSTRACT';

        // Prepare the arguments for the ‘get syntax abstract’ call.
        v:=VarArrayCreate([0,1],varVariant);
        v[0]:=’KBMMW_INVENTORY’;
        v[1]:=’KBMMW_1.0’;
        req.Args:=v;

        // Call app server.
        res:=server.ProcessRequest(req);
        try

           // Check response.
           Label1.Caption:=res.StatusText;
           if res.StatusCode=0 then // OK
           begin
                Memo1.Text:=res.Result;
           end;
        finally
           res.Free;
        end;
     finally
        req.Free;
     end;
end;
```

The sample now shows how to provide arguments for the call. It’s possible to nest arrays and use any simple type supported by a variant in the argument call. It’s similarly possible to have arrays, nested arrays and all simple data types returned as a result in the res.Result property.

It’s possible to provide information like username/password etc. via the req.Ident property.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

```
procedure TForm1.Button1Click(Sender: TObject);
var
   server:IProcessRequestSOAPPort;
   req:TkbmMWSOAPRequest;
   res:TkbmMWSOAPResponse;
   args:variant;
begin
     server:=GetIProcessRequestSOAPPort(false,
       'http://servers.components4developers.com:4500/server1',nil);

     req:=TkbmMWSOAPRequest.Create;
     try
        // Setup identity.
        req.Ident:=TkbmMWSOAPClientIdentity.Create;
        req.Ident.Username:='HANS';
        req.Ident.Password:='somepassword';

        // Setup request.
        req.ServiceName:='KBMMW_INVENTORY';
        req.ServiceVersion:='';
        req.StateID:=-1;
        req.Func:='GET SYNTAX ABSTRACT';

        // Prepare the arguments for the 'get syntax abstract' call.
        v:=VarArrayCreate([0,1],varVariant);
        v[0]:='KBMMW_INVENTORY';
        v[1]:='KBMMW_1.0';
        req.Args:=v;

        // Call app server.
        res:=server.ProcessRequest(req);
        try
           // Check response.
           Label1.Caption:=res.StatusText;
           if res.StatusCode=0 then // OK
           begin
                Memo1.Text:=res.Result;
           end;
        finally
           res.Free;
        end;
     finally
        req.Free;
     end;
end;
```

Notice that the password is transferred in clear text within SOAP, and thus you should be careful about when to require passwords on a SOAP service.

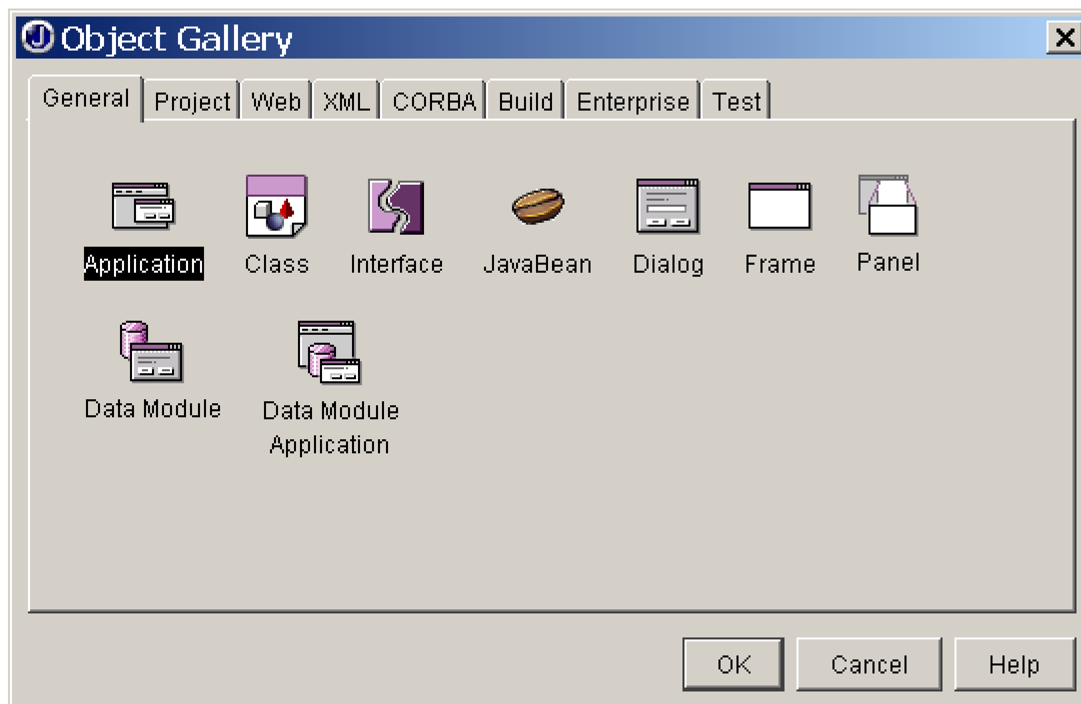
You don't have to explicitly free the created ident instance. Freeing the request instance will take care of that automatically.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Creating a non kbmMW SOAP client using Java

Now we make the same exercise as before, just implementing it in Java.
This sample has been coded using Apache's Axis SOAP support which can be found at http://ws.apache.org/axis/ and JBuilder 7.
Any other Java IDE and Java SOAP implementation can equally be used.

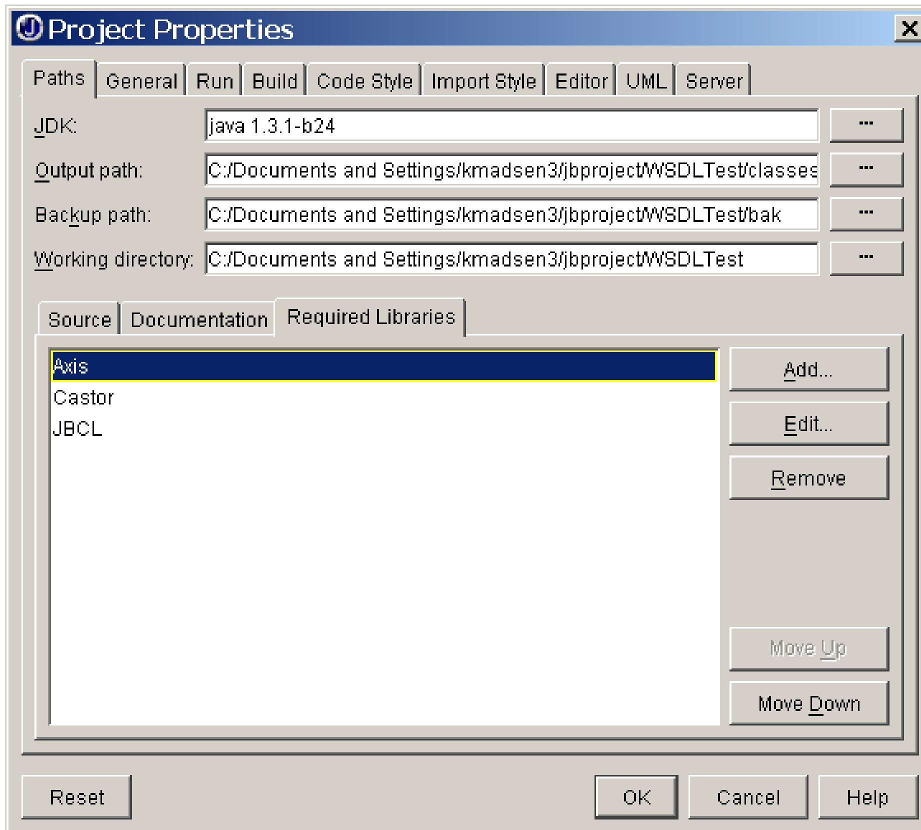
First create a new Java Application:



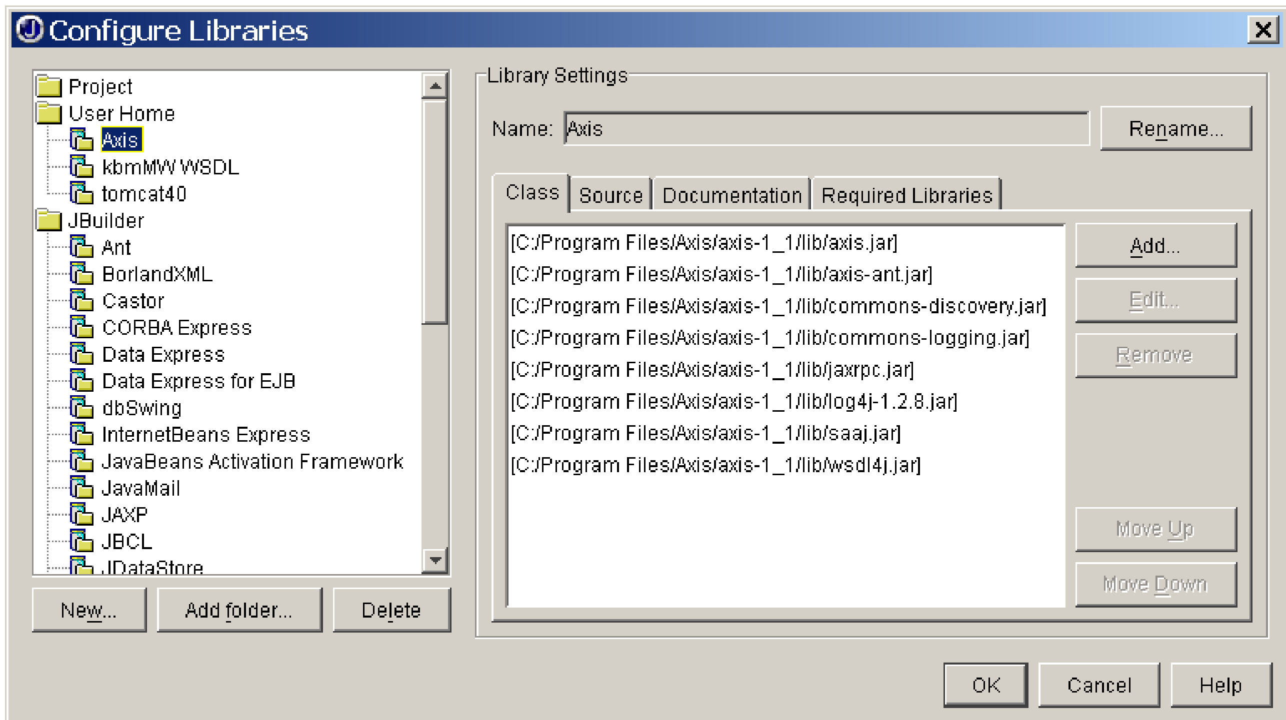
Setup the desired paths to where the application should be placed, class and package names etc.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Then ensure that Axis and Castor (a Java XML parser) is added as required libraries:



Copyright 2001-2004 Components4Developers
http://www.components4developers.com



Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next step is to arrange Frame1 to something similar to this (in this case we are using PaneLayout):

Next we need to import the kbmMW.wsdl file using Apache Axis's wsdl importer. To make the import as easy as possible I created a small WSDL2Java.bat file with the following contents (this is all on one line… no line breaks!):

```
"C:\Program Files\Java\j2re1.4.1_02\bin\java.exe" -cp "C:\Program Files\Axis\axis-
1_1\lib\axis.jar";"C:\Program Files\Axis\axis-1_1\lib\commons-logging.jar";"C:\Program
Files\Axis\axis-1_1\lib\commons-discovery.jar";"C:\Program Files\Axis\axis-
1_1\lib\jaxrpc.jar";"C:\Program Files\Axis\axis-1_1\lib\saaj.jar";"C:\Program Files\Axis\axis-
1_1\lib\wsdl4j.jar";"C:\Program Files\Axis\axis-1_1\lib\axis-ant.jar";"C:\Program
Files\Java\j2re1.4.1_02\lib\rt.jar" org.apache.axis.wsdl.WSDL2Java %1
```

Replace the paths with what match your Axis and Java 1.4 installation (Axis's wsdl importer require Java 1.4, but after the import has been done, you can use Java v. 1.3, which is default for JB 7, to compile and run the demo application).

Place the kbmMW.wsdl file and the WSDL2Java.bat file in the same directory and run it like this:

WSDL2Java.bat kbmMW.wsdl

You will now have a file hierarchy under current directory, looking like this

Move this hierarchy (starting with com) to the source directory of your sample Java application.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Now back to the Java application. Add the following line to the imports section of the Frame1.java file:

```
import com.components4developers.www.namespaces.kbmMW.*;
```

Show the form/frame designer and double click the jButton1 component to create an event handler for its action event (click event).

```
  void jButton1_actionPerformed(ActionEvent e) {
    try {
            // Make a service
            KbmMWServiceLocator  service = new KbmMWServiceLocator();

            // Now use the service to get a stub which implements the SDI.
            IProcessRequestSOAPPort port =
service.getIProcessRequestSOAPPort(new java.net.URL("http://localhost:3000"));

            // Setup a request object.
            TkbmMWSOAPRequest req = new TkbmMWSOAPRequest();
            req.setServiceName("KBMMW_INVENTORY");
            req.setFunc("LIST");
            req.setStateID(new Integer(-1));

            // Process the request.
            TkbmMWSOAPResponse res = port.processRequest(req);

            // Show status and result value.
            jTextField1.setText(res.getStatusText());
            jTextArea1.setText(res.getResult().toString());
    }
    catch (Exception ex) {
      jTextArea1.setText(ex.getMessage());
    }
  }
```

Compile and run your Java application. Before proceeding, make sure you have the kbmMW SOAP demo server application running.

Now you will get a list of services supported by the application server.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

As with the Delphi SOAP client sample we will also show how to use arguments in Java by asking for the syntax abstracts for a specific service/version.

```
  void jButton1_actionPerformed(ActionEvent e) {
    try {
            // Make a service
            KbmMWServiceLocator  service = new KbmMWServiceLocator();

            // Now use the service to get a stub which implements the SDI.
            IProcessRequestSOAPPort port =
service.getIProcessRequestSOAPPort(new java.net.URL("http://localhost:3000"));

            // Setup a request object.
            TkbmMWSOAPRequest req = new TkbmMWSOAPRequest();
            req.setServiceName("KBMMW_INVENTORY");
            req.setFunc("GET SYNTAX ABSTRACT");
            req.setStateID(new Integer(-1));

            // Setup arguments.
            Object[] args = new Object[2];
            args[0]="KBMMW_INVENTORY";  // Service name
            args[1]="KBMMW_1.0";        // Service version
            req.setArgs(args);

            // Process the request.
            TkbmMWSOAPResponse res = port.processRequest(req);
            jTextField1.setText(res.getStatusText());
            jTextArea1.setText(res.getResult().toString());
    }
    catch (Exception ex) {
      jTextArea1.setText(ex.getMessage());
    }
  }
```

As with the Delphi SOAP client, arguments and results can be simple types, arrays and nested arrays as you see fit.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Creating a non kbmMW SOAP client using C#

Finally we make the same exercise as before, just implementing it in C#.
This sample has been coded by hand, using MS dotNet SDK v. 1.1.

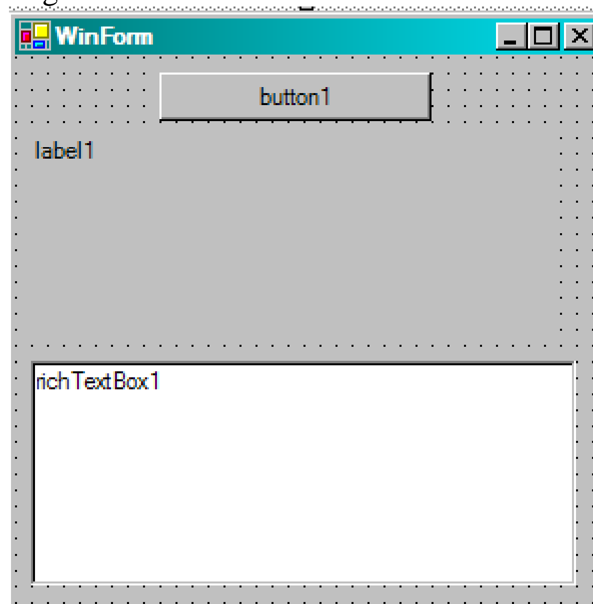
dotNet's WSDL importer unfortunately do have something to wish for. It's a rather poor implementation, and thus has problems correctly handling the XML data type *anyType* or similar types.

It is possible to not use the WSDL importer, and code the SOAP request in a more direct manner, thus avoiding those problems. There is more information about that on the internet by searching using www.deja.com.

Another way could be to use Delphi 8's WSDL importer, compile an assembly with all the things needed (incl. Borland's Variant class), and use that from your C# environment.

In any case, let's get this demo going, using native C# and MS WSDL importer.

Use the C# development tool of your choice (for example C#Builder or VS.Net) and create a new WinForms application looking similar to this:



Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next step is to generate some C# stub code from the kbmMW.wsdl file. If you are using an IDE like VS.Net or C#Builder, you probably can do the WSDL import directly from the IDE.
In our case we do it manually using Microsoft's WSDL importer.

Start a MS-DOS box and navigate to the place where you have your kbmMW.wsdl file placed.
You will have to figure out where your MS dotNet SDK is installed before continuing.
A typical place is: C:\Programmer\Microsoft.NET\SDK\v1.1

In the MS-DOS box, write the following:

```
C:\Programmer\Microsoft.NET\SDK\v1.1\bin\wsdl kbmMW.wsdl
```

The importer will now have generated a kbmMWService.cs file which you need to copy to the directory where you have your demo client source files.

Add kbmMWService.cs to the project.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next write some event handler for the OnClick event of the button1 component.

```
private void button1_Click(object sender, System.EventArgs e)
{
  kbmMWService service = new kbmMWService();
  TkbmMWSOAPRequest req = new TkbmMWSOAPRequest();

  // Setup the URL to the web service server.
  service.Url="http://localhost:3000";

  // Setup the request.
  req.ServiceName="KBMMW_INVENTORY";
  req.ServiceVersion="";
  req.Func="LIST";
  req.StateID=-1;

  // Execute the request.
  TkbmMWSOAPResponse res = service.ProcessRequest(req);
  if (res==null) return;

  // Display the response.
  label1.Text=res.StatusText;
  richTextBox1.Text=res.Result.ToString();
}
```

Then compile and run it. The result is that a list of services from the application server will be displayed.

Copyright 2001-2004 Components4Developers
http://www.components4developers.com

Next step is then to add some arguments to the call. Let's ask for the syntax abstract of the inventory service.

```
private void button1_Click(object sender, System.EventArgs e)
{
  kbmMWService service = new kbmMWService();
  TkbmMWSOAPRequest req = new TkbmMWSOAPRequest();

  // Setup the URL to the web service server.
  service.Url="http://localhost:3000";

  // Setup the request.
  req.ServiceName="KBMMW_INVENTORY";
  req.ServiceVersion="";
  req.Func="GET SYNTAX ABSTRACT";
  req.StateID=-1;
  req.Args=new string[] {"KBMMW_INVENTORY","KBMMW_1.0"};

  // Execute the request.
  TkbmMWSOAPResponse res = service.ProcessRequest(req);
  if (res==null) return;

  // Display the response.
  label1.Text=res.StatusText;
  richTextBox1.Text=res.Result.ToString();
}
```

As you may notice it's quite easy to provide arguments.

Alas the trees don't grow into the sky... here is also where we find the limitations of Microsoft's own WSDL importer. It always assumes that arguments and results are of string type.

Unfortunately there is no way around that problem until Microsoft fixes their WSDL importer, or unless you find another WSDL importer which works better (notice the D8 suggestion).

This concludes the whitepaper about SOAP with kbmMW.

Kim Madsen
Components4Developers

Copyright 2001-2004 Components4Developers
http://www.components4developers.com