The native Java client

kbmMW supports a number of native clients. One of them is a Java archive which allows Java (J2EE, J2SE and possibly J2ME (not tested though)) code directly to reach a kbmMW based application server.

This document will show how to compile the extension, how to install it and how to use it.

Compiling the Java client library

The kbmMW Java client library is usually delivered as a JAR file. However with source distributions, you may have a requirement to recompile it for some reason. This section explains how to do that.

Prerequisites:

- For easiest compilation, use Borland JBuilder 7 or newer as project files are readily available.
  http://www.borland.com
  It is possible to compile the JAR file manually using standard Java tools, but its not described in this document.

- The kbmMW Java client source
  http://www.components4developers.com

Build procedure:

Please make sure to follow these steps very carefully!

1. Create a working directory where all files end up after extracting, e.g: `C:\work`.

2. Extract the kbmMW Java client source distribution into the working directory.

3. Start JBuilder, locate `JavaClient.jpx` in the JavaClient subdirectory of the working directory. Open it.

4. Select '*Rebuild project "JavaClient.jpx"*' in the Project menu item.

Now a JAR file with the name `kbmMWClient.jar` should be available in the same directory as JavaClient.jpx.

This JAR file is the ready to use with any Java development environment that supports `java.net.Socket`.

Installation and configuration:

1. Make the kbmMWClient.jar file available for your Java projects by including it in any of the following ways:
   a) Add the path to it (incl. its filename) to your machines CLASSPATH environment variable.
   b) When starting your Java application which use kbmMW, add the startup argument –cp <path_to_kbmMWClient.jar_file>
   c) Embed the class files into your application jar or ear file.

The Demo application explained

The application is contained in the same project as the kbmMW client, namely JavaClient.jpx, but is placed in another package, namely components4developers.demo. The demo files are not contained in the kbmMWClient.jar file (and nor should they).
The demo consists of two java source files, MainApp.java and MainFrame.java. MainApp.java is providing a main function and a few lines of code to make an instance of MainFrame. Thus MainFrame.java is the interesting file to look at.

```
package components4developers.demo;


import com.borland.jbcl.layout.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
```

Imports the standard GUI related packages.

```
import components4developers.client.*;
```

Imports the important kbmMW Java based client.

```
/**
 * <p>Title: </p>
 *
 * <p>Description: </p>
```

```
 *
 * <p>Copyright: Copyright (c) 2004</p>
 *
 * <p>Company: </p>
 *
 * @author not attributable
 * @version 1.0
 */
public class MainFrame extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
  private XYLayout xYLayout1 = new XYLayout();
  private JButton btnConnect = new JButton();
  private JButton btnDisconnect = new JButton();
  private JButton btnInventoryList = new JButton();
  private JButton btnGetAbstract = new JButton();
  private JButton btn100List = new JButton();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTextArea taDebug = new JTextArea();
```

Definition of the MainFrame class, and fields for GUI purposes.

```
  public TkbmMWSimpleClient SimpleClient = new TkbmMWSimpleClient();
```

One more field is defined for the purpose of holding a kbmMW client instance.

```
    public MainFrame() {
        try {
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            jbInit();
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
```

The constructor of MainFrame. In best JBuilder way, runs the method that sets all properties and events of the GUI elements.

```
    /**
     * Component initialization.
     *
     * @throws java.lang.Exception
     */
    private void jbInit() throws Exception {
        contentPane = (JPanel) getContentPane();
        contentPane.setLayout(borderLayout1);
        setTitle("kbmMW Java client");
    this.getContentPane().setLayout(xYLayout1);
    btnConnect.setText("Connect");
    btnConnect.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btnConnect_actionPerformed(e);
      }
    });
    btnDisconnect.setText("Disconnect");
    btnDisconnect.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btnDisconnect_actionPerformed(e);
      }
    });
    btnInventoryList.setText("List inventory");
    btnInventoryList.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btnInventoryList_actionPerformed(e);
      }
    });
    setSize(480,350);
    xYLayout1.setWidth(405);
    xYLayout1.setHeight(300);
    btnGetAbstract.setText("Get Abstract");
    btnGetAbstract.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btnGetAbstract_actionPerformed(e);
      }
    });
    btn100List.setText("List 100x");
    btn100List.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        btn100List_actionPerformed(e);
      }
    });
    taDebug.setText("jTextArea1");
    this.getContentPane().add(btnConnect, new XYConstraints(10, 9, -1, -1));
    this.getContentPane().add(btnDisconnect, new XYConstraints(10, 42, -1, -1));
    this.getContentPane().add(btnInventoryList, new XYConstraints(122, 9, -1, -1));
    this.getContentPane().add(btnGetAbstract,    new XYConstraints(237, 10, -1, -1));
    this.getContentPane().add(btn100List,   new XYConstraints(123, 44, -1, -1));
    this.getContentPane().add(jScrollPane1,  new XYConstraints(6, 87, 393, 203));
    jScrollPane1.getViewport().add(taDebug, null);
    }
```

The method that configures all the GUI elements.

```
void btnConnect_actionPerformed(ActionEvent e) {
  try {
     SimpleClient.Connect("localhost",3000);
     SimpleClient.RequestTimeout=1000;
     taDebug.append("Connected"+"\n");
  }
  catch (Exception ex) {
    taDebug.append(ex.toString()+"\n");
  }
}
```

Method that acts as an event handler for the button btnConnect. When clicked on, the kbmMW client instance will try to connect to the application server localhost on port 3000.
It also sets up a RequestTimeout value that indicates that if no response was returned in 1000 secs time, the request should time out.

```
void btnDisconnect_actionPerformed(ActionEvent e) {
  try {
     SimpleClient.Disconnect();
     taDebug.append("Disconnected"+"\n");
  }
  catch (Exception ex) {
    taDebug.append(ex.toString()+"\n");
  }
}
```

Method that acts as an event handler for the button btnDisconnect. When clicked on the kbmMW client instance will disconnect from the currently connected application server.

```
void btnInventoryList_actionPerformed(ActionEvent e) {
  try {
    Object o = SimpleClient.SendRequest("KBMMW_INVENTORY","KBMMW_1.0","LIST",null);
    taDebug.append(o.toString()+"\n");
  }
  catch (Exception ex) {
    taDebug.append(ex.toString()+"\n");
    ex.printStackTrace();
  }
}
```

Method that acts as an event handler for the button btnInventoryList. When clicked on, a request will be sent to the service ‘KBMMW_INVENTORY’ with version ‘KBMMW_1.0’ on the connected application server, asking for executing function ‘LIST’ with no parameters.

As a result we receive an object. All values provided in a SendRequest call must be objects and all result values returned are objects. Thus native types must be boxed.

```
  void btnGetAbstract_actionPerformed(ActionEvent e) {
    try {
      String[] sa=new String[1];
      sa[0]="KBMMW_INVENTORY";
      Object o = SimpleClient.SendRequest("KBMMW_INVENTORY","KBMMW_1.0","GET SYNTAX DETAILS",sa);
      taDebug.append(o.toString()+"\n");
    }
    catch (Exception ex) {
      taDebug.append(ex.toString()+"\n");
      ex.printStackTrace();
    }
  }
```

The event method for the btnGetAbstract button. When clicked it will call the application server for syntax details about the KBMMW_INVENTORY service.
This time an argument list is prepared. The argument list must always be an array of object (a string is also considered an object in Java).
In this sample, we allocate a string array with only one element, namely the name of the service to get syntax details about using the KBMMW_INVENTORY service itself.

```
  void btn100List_actionPerformed(ActionEvent e) {
    try {
      for (int i=0; i<100; i++) {
              Object o = SimpleClient.SendRequest("KBMMW_INVENTORY","KBMMW_1.0","LIST",null);
              taDebug.append(o.toString()+"\n");
      }
    }
    catch (Exception ex) {
      taDebug.append(ex.toString()+"\n");
      ex.printStackTrace();
    }
  }
```

This method is the implementation of the ‘click’ event of the button btn100List. It simply asks the application server for a list of services 100 times.

```
}
```

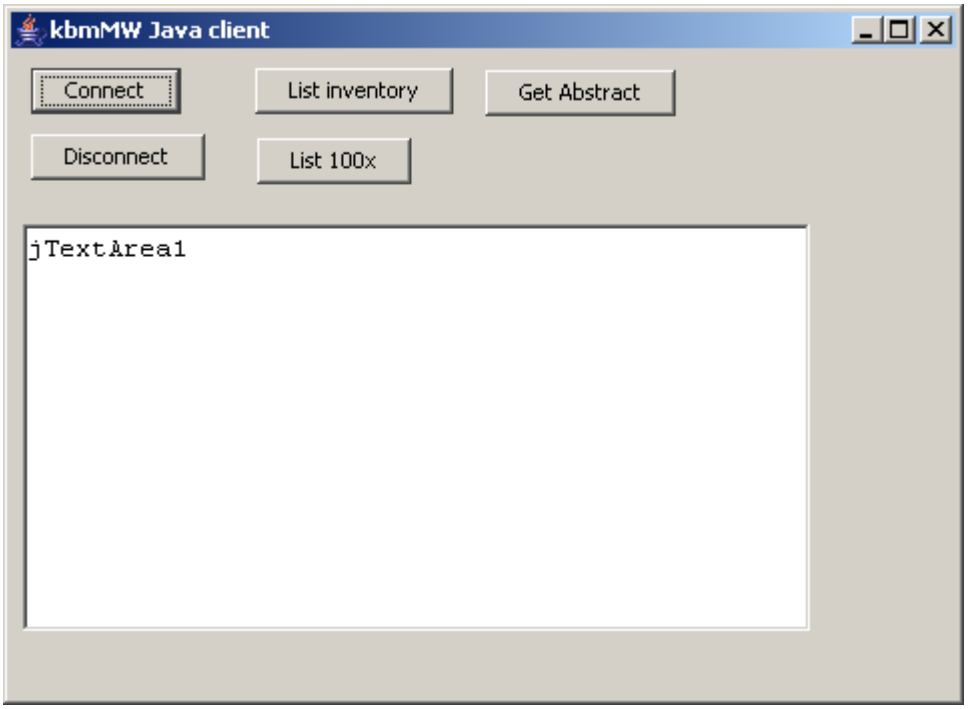
End of the MainFrame class.

To run the application, start a kbmMW Demo application server on your local machine, listening on port 3000 and with VerifyTransfer set to true.

Then from within JBuilder click the Run button, or from the command line issue a command similar to this:

**javaw -classpath "C:\work\JavaClient\classes;jbcl.jar;… other relevant jar files"
components4developers.demo.MainApp**

That should produce a screen like this:



Click connect, and try it out.

Reference

Exception handling

The kbmMW Java client is using standard Java type exception handling. In addition the TkbmMWSimpleClient object can be queried for status from the application server. Please refer to the documentation of the TkbmMWSimpleClient object.

Standard error codes (provided by the application server via the TkbmMWSimpleClient object):

KBMMW_ERR_EXCEPTION	100000

Provided if an undefined exception has occurred in a call to a kbmMW based application server. An error message explaining the problem is usually provided.

KBMMW_ERR_ABORT	100001

Provided if an abort exception has occurred in a call to a kbmMW based application server. An error message explaining the reason is usually provided.

KBMMW_ERR_SERVICENOTAVAIL	100100

Provided if a call has been made to an unavailable service.
An error message explaining the problem is usually provided.

KBMMW_ERR_FUNCNOTAVAIL	100101

Provided if an unknown function has been called on a kbmMW based application server.
An error message explaining the problem is usually provided.

KBMMW_ERR_AUTHFAILED	100200

Provided if the call to the application server was not authorized.
An error message explaining the reason is optionally provided.

KBMMW_ERR_MESSAGETYPE	108000

Provided if an unknown message has been received from the kbmMW based application server. Currently not in use.

KBMMW_SYS_REDIRECT	110000

Provided if the kbmMW based application server requests the client to disconnect and reconnect to another application server. The connection string to the new application server is in the error message.

Variant support

Java don't as such support variants directly. However Java is based on objects and even native types can be boxed in an object. I.e. int in an Integer object, long in a Long object, short in a Short object, float in a Float object, double in a Double object etc.
In addition, a string is already considered an object in Java, via the String class.

Hence a Java object, containing one of the kbmMW supported types, is equivalent to a kbmMW variant.

The following schema shows which supported Java object types match which kbmMW variant types:

byte[]	varByte+varArray: Variant byte array.
[]	varVariant+varArray: array of variants.
java.lang.Boolean	varBool: boolean value
java.lang.Integer	varInt: integer (32 bit) value
java.lang.Long	VarInt64: long (64 bit) value
java.lang.Short	varShortInt: short (16 bit) value.
java.lang.Double	varDouble: double sized floating point value.
java.lang.Float	varSingle: single sized floating point value.
java.lang.String	varOLEString: unicode enabled string value When receiving string values from the application server, kbmMW Java client also understands varString (ASCII strings).
null	varNull: the NULL value. When receiving varEmpty from the application server, it will also be understood as null.

Client transport

```
components4developers.client.TkbmMWCustomTransport
  |
  +--- components4developers.client.TkbmMWTCPTransport
```

The client transports responsibility is to manage a connection to a kbmMW based application server which is then used by the kbmMW Java simple client.

Its designed to use the standard java.net.Socket class as the communication layer. This should make it compatible with most Java implementations for most platforms.

Fields

```
public String Host;
```

Contains the host string (IP address or hostname) that the current/last connect tried to connet to.

```
public int Port;
```

Contains the port number that the current/last connect tried to connect to.

```
public int ConnectTimeout = 20;
```

Specify in seconds how long a connection attempt can take before its timed out. Default value is 20 secs.

```
public int RequestTimeout = 60;
```

Specify in seconds how long a client request can take before its timed out. Notice that a long running request will continue to run on the application server (unless it has been configured to time stalled requests out), even after the client has timed out. Default value is 60 secs.

Constructors

```
public TkbmMWTCPTransport()
```

Constructs an instance of the transport, without attempting to connect to an application server at this stage.

Methods

```
public void Connect(String AHost, int APort) throws IOException
```

Connects to the given host on the given port number if possible. Its important later on to disconnect the client transport from the application server by using the Disconnect methods of TkbmMWSimpleClient or TkbmMWTCPTransport.
If no connection could be made or the host name couldn't be resolved, an IOException is thrown.

```
public void Connect() throws IOException
```

Same as Connect(String AHost, int APort), except that the fields Host and Port is used to identify where to connect to.

```
public void Disconnect() throws IOException
```

Disconnects the client transport from the application server.

```
public boolean IsConnected()
```

Returns true if the clienttransport is connected to an application server, else false.

In addition there are several more methods, Read, ReadStream, Write, WriteStream, EndRead, EndWrite and IsDataAvailable. They are not supposed to be called directly, but are used when required by the TkbmMWSimpleClient.

Client

`components4developers.client.TkbmMWSimpleClient`

The clients responsibility is to provide an interface to the developer allowing requests to be made against a kbmMW based application server, and returned responses to be handled.

The client itself do not contain the actual transport layer. For that purpose the client transport is used.

Fields

```
public String UserName = "";
```

Provides access to the current username which the client should use when contacting the application server. UserName operates in close connection with Password and Token.
For more details, see Token description.

```
public String Password = "";
```

Provides access to the current password which the client should use when contacting the application server. Password operates in close connection with UserName and Token.
For more details, see Token description.

```
public String Token = "";
```

Provides access to the token returned from the application server, after an authentication process have happened where a username and password has been validated and accepted. As result of the validation, the application server usually returns a unique and temporary number, called a token, which identifies the current client/user session.

Until the client disconnects, the client should forward the provided Token instead of Username and Password, by remembering the Token value returned as the result of a call to the application server, and later setting the Token property with the remembered value on subsequent calls to the application server. As long as Token is set to a non empty string, the Token value will be forwarded instead of the UserName and Password values.

```
public String Location = "";
```

Provides access to a location string, which essentially can contain any string based value. It can be used by the client to tell the application server where the client is located, and thus the value can be used for extra security validation, logging, auditing etc. on the application server.

```
public int RequestTimeout = 0;
```

Controls how long a request may take (in secs) before it is timed out and an exception raised. Please notice that even though the client times out, the server most likely will continue to process the request until the server side request timeout has been reached (optional application server setting).

```
public String StatusText = "";
```

Returns the status text provided from the last call to the application server. If no error or warning was raised, the string “OK” is returned.

```
public int StatusCode = 0;
```

Returns the status code provided from the last call to the application server. If no error or warning was raised, 0 is returned. See the Exception handling section for a description of the predefined error codes a kbmMW application server can return (in addition to 0).
Warnings are positive values, errors negative.

```
public int StateID = -1;
```

Provides access to a state identifier. It will be –1 if the last call to the application server was a stateless call…ie. the service called is a stateless service.
In case a stateful service was called, StateID will contain a unique value which should be remembered by the caller. On later calls, the specific stateful service (currently ‘owned’ by the client) can be accessed by providing the state identifier returned by the application server in the StateID property. Its the responsibility of the developer to remember state id’s returned from the application server and later to release the given state id’s (except for state id –1).

Constructors

```
public TkbmMWSimpleClient()
```

Creates a TkbmMWSimpleClient instance which hosts its own transport.
No connection to an application server is attempted at construction time.

Methods

```
public void Connect(String AHost, int APort) throws IOException
```

Attempts a connection to AHost on port APort via the internal transport.
The client should be disconnected before attempting a new connect.
If a connection could'nt be established an IOException is raised.

```
public void Disconnect() throws IOException
```

Disconnects from the application server.

```
public Object SendRequestEx(String AServiceName,
                            String AServiceVersion,
                            int AStateID,
                            String AFunction,
                            Object[] AArgs)
       throws IOException, EkbmMWException,ParseException
```

Sends request to the application server.
If the client is not connected, an EkbmMWException is thrown.

AServiceName is a string identifying the name of the service to call.
AServiceVersion is a string identifying the version of the service to call.
AStateID is the state identifier that identifies a specific stateful service instance. Use –1 if calling a stateless service or a new stateful service.
AFunction is a string identifying the name of the function to call.
AArgs is an array of objects which consititutes the arguments sent to the application server. null can be given in case no arguments are required. Each argument element can contain a nested array or any other of the supported kbmMW Java client datatypes.

The function returns an object value which can also be an array.

IOException is thrown if a transport related exception is seen.
EkbmMWException is thrown if the server side raises a non specified exception.
EkbmMWAbortException is thrown if the server aborts the operation.
EkbmMWServiceNotAvailException is thrown if the service requested is not available.
EkbmMWFuncNotAvailException is thrown if the function requested is not available.
EkbmMWAuthFailedException is thrown if the call is not authorized.
EkbmMWRedirectException is thrown if the server wants to redirect the client to connect to another application service, for example due to loadbalancing.

The specific error/warning code can also be obtained via the StatusCode and StatusText field.

Eg.

```
TkbmMWSimpleClient client=new TkbmMWSimpleClient();
client.Connect("127.0.0.1",3000);

// Make the call.
try {
  Object result=client.SendRequestEx("KBMMW_INVENTORY",
                                     "kbmMW_1.0",
                                     -1,
                                     "LIST",
                                     null);
}
catch (Exception ex) {
  System.out.println("Error catched. "+
                     "ErrorCode="+Integer.toString(client.StatusCode));
  ex.printStackTrace();
}

// Check for server exception.
if (client.StatusCode==0) {

   // Is ok. Display result.
   System.out.println("Result is:"+result.toString());

   // Notice that the result also can be an array or byte array depending
   // on the servers functionality.
}

client.Disconnect();
```

```
public Object SendRequest(String AServiceName,
                          String AServiceVersion,
                          String AFunction,
                          Object[] AArgs)
     throws IOException, EkbmMWException, ParseException
```

Same as SendRequestEx, except that the current stateid defined via the StateID field is used.

```
public void SetRequestStream(byte[] AStream)
```

Sets the contents of the request stream. After the next call to the application server, the request stream will automatically have been cleared out. Thus its important to set it each time stream like data is to be transmitted to the application server.

```
public byte[] GetRequestStream()
```

Returns the currently defined request stream as a byte array. The request stream is a data stream that can be forwarded to the application server in addition to the normal arguments. Some services in the application server require use of request stream, like the file service for file upload functionality. As a result, the application server can choose to return a response stream in addition to the normal result.

```
public void SetResponseStream(byte[] AStream)
```

Sets the contents of the response stream. This is only provided for completeness. It usually do not server any purpose to set the contents of the response stream directly.

```
public byte[] GetResponseStream()
```

Returns the response stream of the last call if any has been provided by the application server. The response stream is for example needed to handle in case of using the standard kbmMW file service's download facility.

```
 public boolean ReleaseState(String AServiceName,
                             String AServiceVersion)
        throws IOException, EkbmMWException, ParseException
```

Release the state of the service instance identified by AServiceName, AServiceVersion and the currently set state ID of the client object.

After a stateful service has been called its very important to release the state as soon as the stateful service is not to be used by the client any longer. Not releasing it can result in serious server ressource drain which can lead to the server not being able to serve requests for the specific service, until the application server times the relevant services out (optional server side feature).

Eg.

```
System.out.println("Releasing state with state id=" +
                   Integer.toString(client.StateID));
client.ReleaseState("SOMESTATEFULSERVICE","ver1.0");
```

```
public boolean ReleaseStateEx(String AServiceName,
                              String AServiceVersion,
                              int AStateID)
       throws IOException, EkbmMWException, ParseException
```

Same as ReleaseState except a specific state ID can be directly provided.
A client can thus easily manage multiple stateful service instances if needed.

This concludes the whitepaper about the kbmMW native Java client.

Kim Madsen
Components4Developers