Using kbmMW with IntraWeb

Its really easy to get IntraWeb and kbmMW to play together in a nice way.
Some of the benefits of adding kbmMW to IntraWeb applications are:

- Highly controlled database access.
- Automatic connection pooling of database access.
- Automatic caching of data from the database.
- Reuse of existing application server.

In addition the ability to store business logic in an application server is another important asset. This allows for, easily, to use the same application server for full blown clients in combination with offering access via the web.

This document shows how to add kbmMW to an existing IntraWeb application, and require previous knowledge about how to use IntraWeb.

Ok, lets get to work...

IntraWeb applications can be build as standalone applications which contains a build in web server, or as the more traditional ISAPI or CGI types of applications.

kbmMW can be built into each of those application types, but to benefit from its caching and performance features, web applications should either be build as standalone apps, or as ISAPI's. The reason is that CGI's are loaded by the web server each time its used, and unloaded immediately after. This alone is a very slow process. Further it will require kbmMW to connect to and disconnect from the application server for each request. Finally kbmMW's caches are obviously cleared when the CGI application is unloaded.

The explanation in this whitepaper is based on having one of the precompiled kbmMW demo application servers running one a computer and listening. E.g. the kbmMW BDE application server.

How to modify an Intraweb project to use kbmMW?

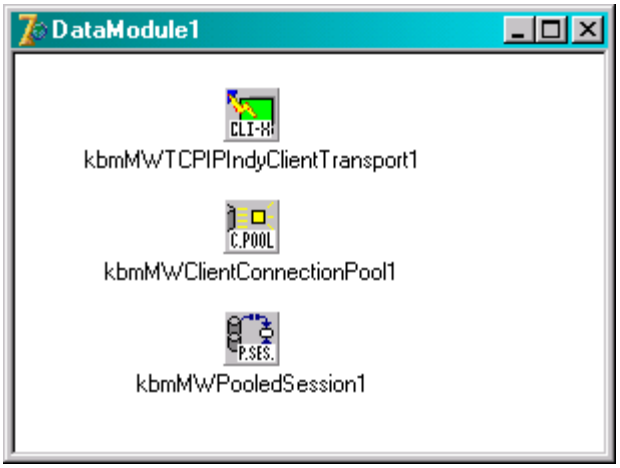
In this explanation we assume that you have an existing Intraweb standalone application similar to the FishFact demo. An Events IW demo with kbmMW support is available from the C4D site.

First an additional datamodule will have to be added. The purpose of it is to host the core kbmMW client components which should be shared between all instances of the Intraweb sessions.

Thus do a File/New/Datamodule.

Next add to the new datamodule one TkbmMWTCPIPIndyClientTransport component, one TkbmMWClientConnectionPool component and one TkbmMWPooledSession component.

Your datamodule should look something like this now:

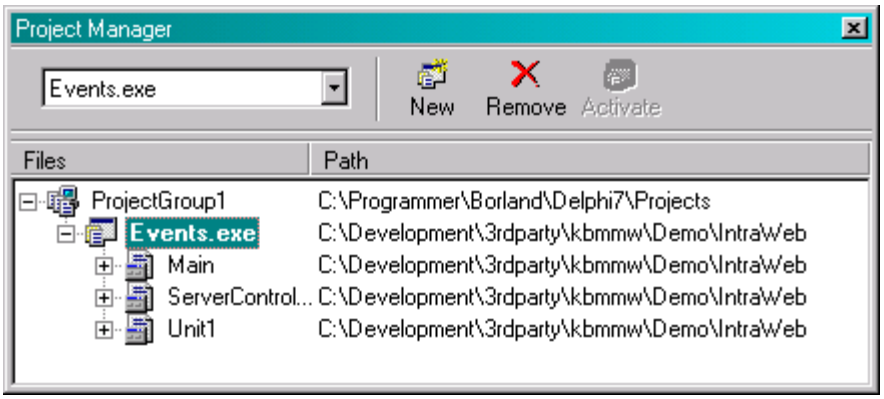


Setup the Host, Port, StreamFormat and VerifyTransfer of the kbmMWTCPIPIndyClientTransport1 component to match the application server. Default values for the demo application server running on the same machine you develop on are:
StreamFormat:=’STANDARD’, VerifyTransfer:=true, Post:=3000, Host:=127.0.0.1

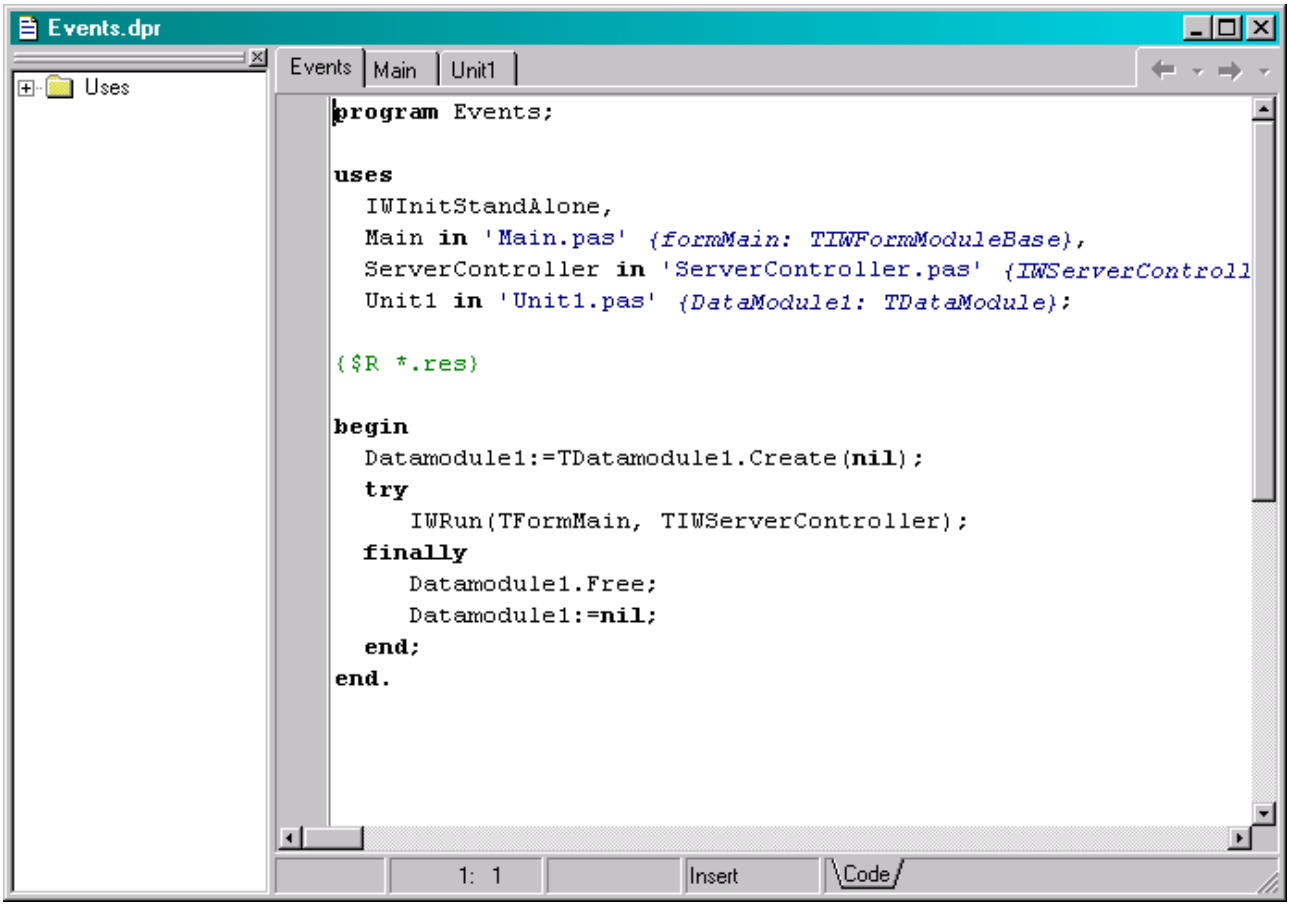
Next setup the kbmMWClientConnectionPool1.Transport to point on the kbmMWTCPIPIndyClientTransport1. Also set the connection pools MaxConnections to 5.

Finally setup the kbmMWPooledSession1.ConnectionPool to point on the kbmMWClientConnectionPool1 component. And set the SessionName of the kbmMWPooledSession1 component to ‘DEMO’ or whatever other unique name you can think of.

Next part is to have the datamodule created automatically on start of the IW application.
This happens in the project source. In the project manager right click on the project name and select 'View source'.



Modify the source to look like this:

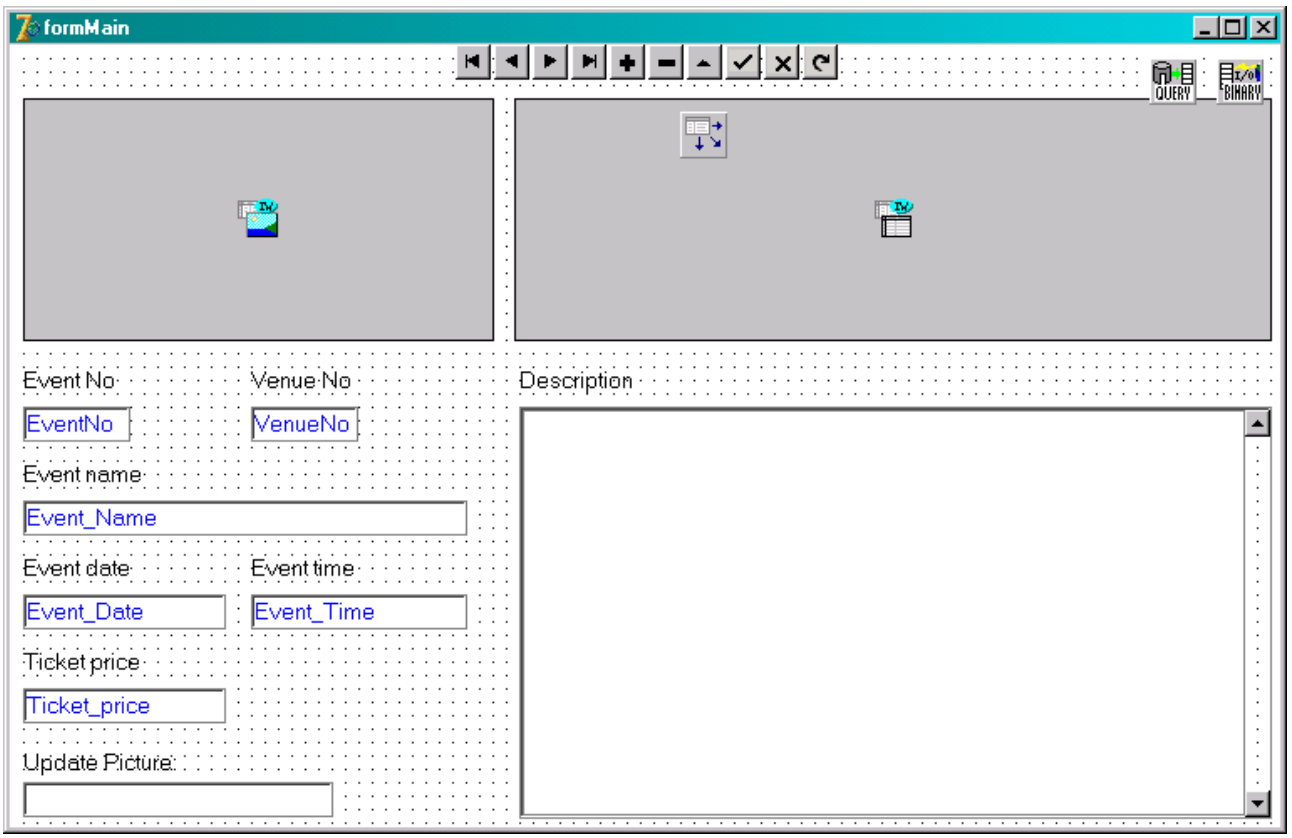


What it does is that it creates an instance of the datamodule hosting the kbmMW connection components just before actually starting the IW application. When the IW application has finished, it then nicely releases the memory allocated for the datamodule.

The screen dump is showing the code from the standalone IntraWeb server. The ISAPI based IntraWeb server looks very similar, except for the uses clause looking a little bit different.
But it's the same way a datamodule should be created and freed.

Now the basis is ready and the real job can begin.

Lets say the main form of the IW application looks like this:



Please notice the addition of two kbmMW client components, a TkbmMWClientQuery and a TkbmMWBinaryStreamFormat. The TkbmMWClientQuery is the equivalent of a TQuery, except its able to communicate with the application server instead of directly with a database. Its much like a TClientDataset component, although much easier to use.

Make sure to select 'File/Use unit' and select Datamodule1. This allows for this main form to know about what's on Datamodule1. Its simply added to the uses clause of this form.

Then set the kbmMWClientQuery1.SessionName to 'DEMO' or whatever you chose for the SessionName on the kbmMWPooledSession1 component on Datamodule1.

Set kbmMWClientQuery1.TransportStreamFormat to point on the kbmMWBinaryStreamFormat1 component.

Finally add some SQL statement to the kbmMWClientQuery1.Query property.
Eg. select * from events

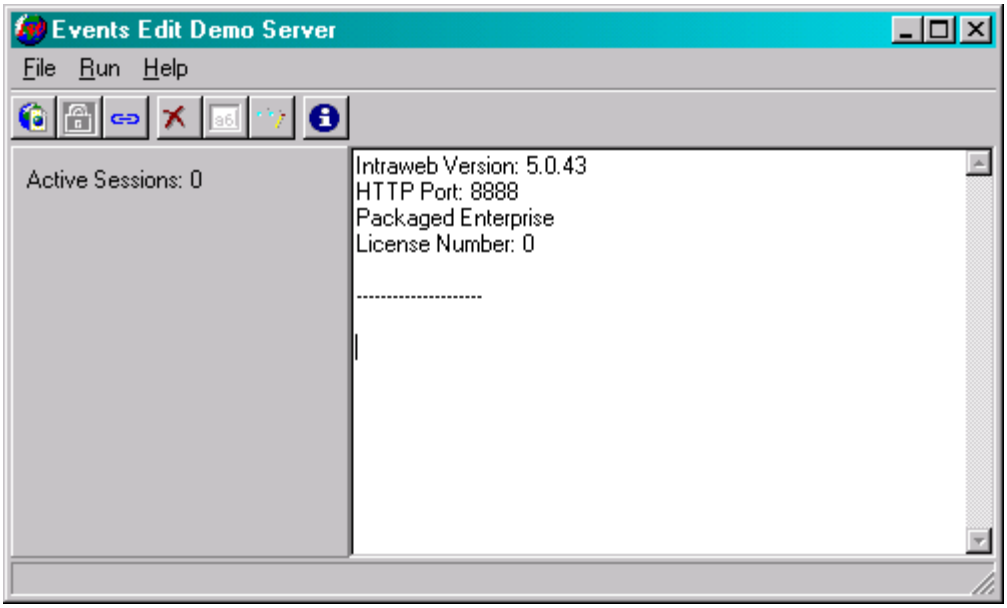
This is called a client side statement. Server side statements or so called named queries can be given too. Eg. @ALL_EVENTS which is a named query defined on the demo server.

Then update all IW dataaware components to point on the fields returned by the kbmMWClientQuery1 component.

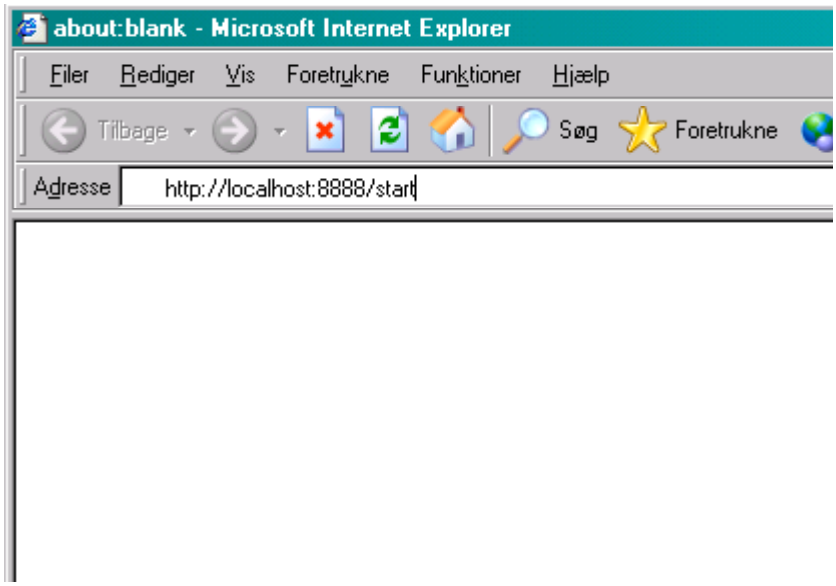
Finally remember to open the kbmMWClientQuery1 component by setting its Active property to true, or using its Open method.

Now data is automatically fetched from the application server.

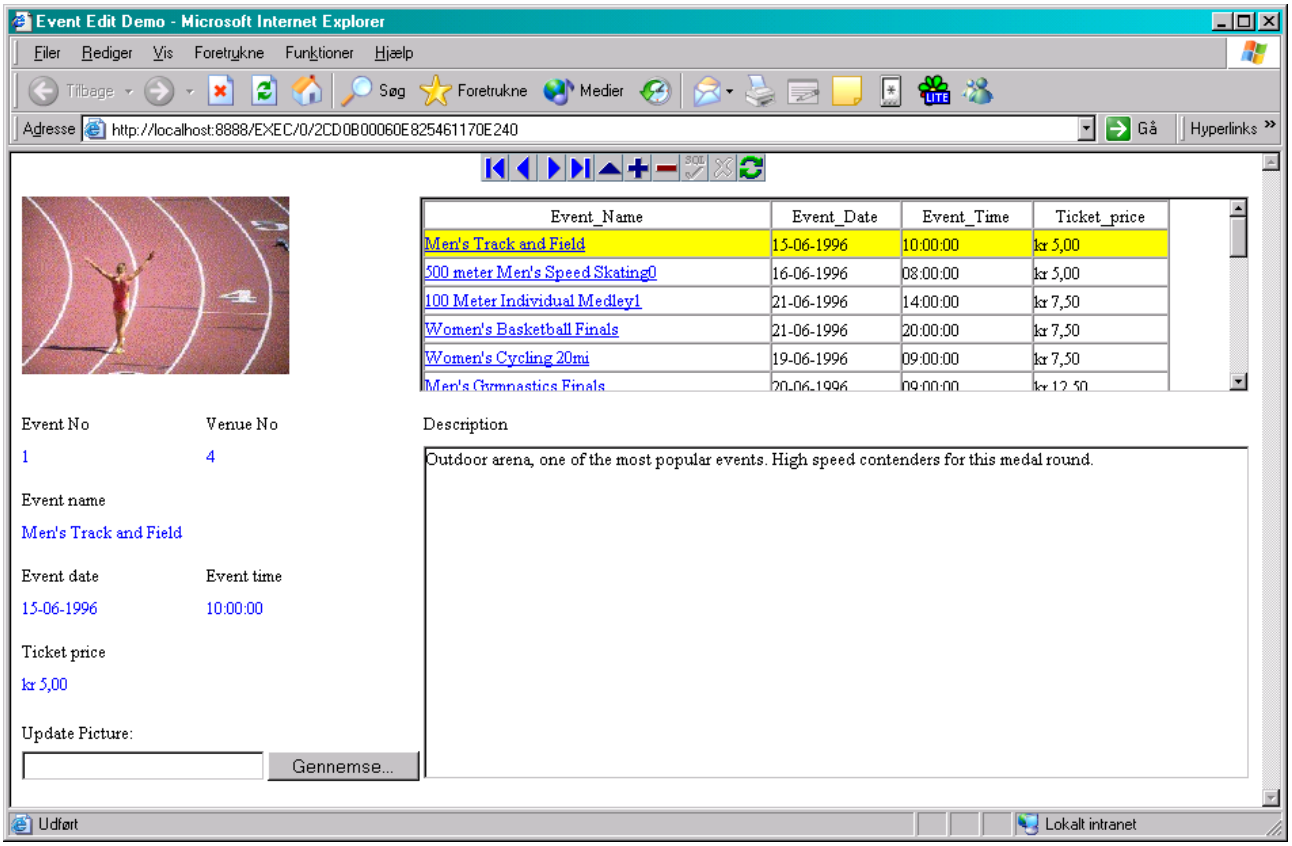
Lets run it:



Open a web browser and contact the web server:



And this is the result:



If you create a site which will update values, remember to call kbmMWClientQuery1.Resolve at relevant places to actually get the updates back to the application server and database. Otherwise all updates are only done in the local memory storage. More information about the resolving process is available in the 'Using kbmMW as a query server' whitepaper.

Kim Madsen
Components4Developers