kbmMW and the NexusDB transport

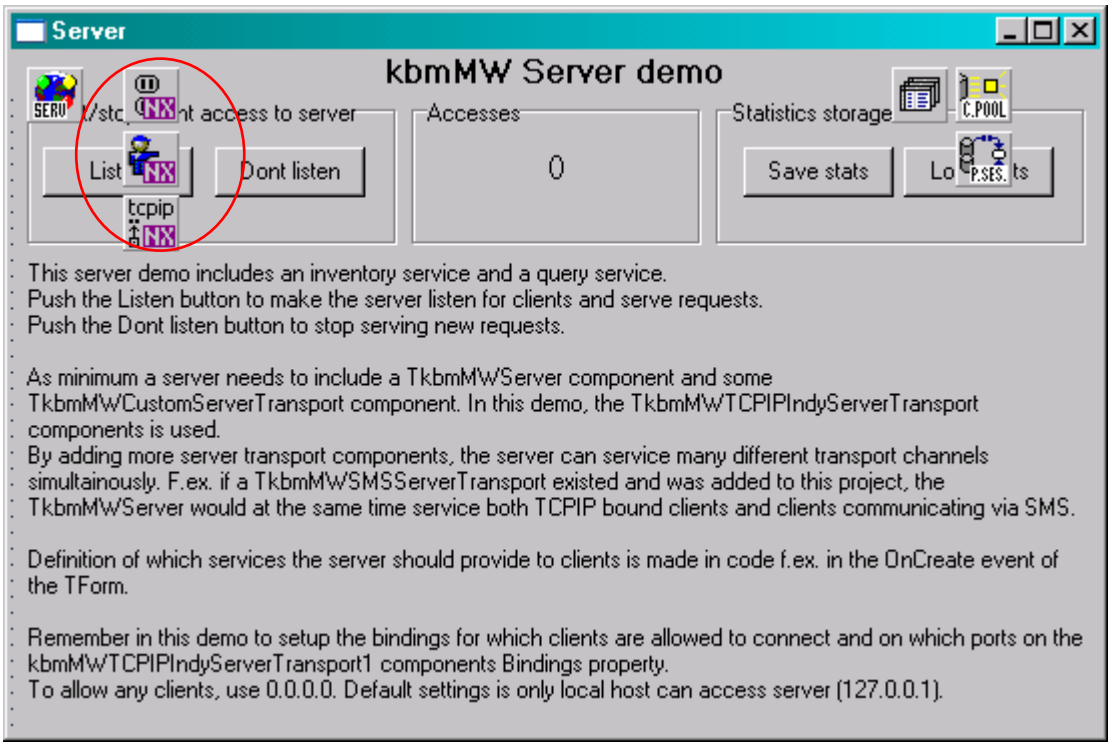
for kbmMW v. 1.07+

Preface

kbmMW directly supports several transport methods like TCPIP using Indy/DXSocks, ISAPI (transports via a webserver) etc.
With the introduction of the database NexusDB (www.nexusdb.com) the developers of it opened up for extending the exiting database core with new functionality which could communicate over NexusDB's own transport mechanisms.

The application server

As a sample we modify a standard server demo project which default use Indy for communication to instead use NexusDB's transport methods.



Start out with adding as minimum a **TnxSimpleCommandHandler** and a NexusDB transport of your choise (in this sample we use the **TnxWinSockTransport**).

Connect the **CommandHandler** property of the *TnxWinSockTransport* to the *TnxSimpleCommandHandler*.

Set the **Mode** property to *nstmListen* to make the NexusDB transport behave like a server.
Set the **ServerNameRuntime** and **ServerNameDesigntime** appropriately. Eg. '*kbmMWServer*'
Then add a **TkbmMWNexusDBServerPluginCommandHandler** (pheww… long word) component.
It's a special type of kbmMW server transport and as such is placed in the 'kbmMW Server Transports tab'.

Point its **Server** property to the *TkbmMWServer* component already part of the server app and finally point its **CommandHandler** property to the *TnxSimpleCommandHandler*.

Tune your transport's properties (eg. Port etc.) to match your needs according to NexusDB specifications.

Now all components are linked nicely together on the server side.

All we need to do is to modify the **OnClick** events for the Listen and Don't Listen buttons to activate and deactivate the server side NexusDB components. In other setups where kbmMW may be combined with a NexusDB in the same application and the NexusDB is directly accessible by NexusDB clients, you may want to have some other place to control the enabling and disabling of the NexusDB components.

For this sample the **OnClick** event for the Listen button should contain:

```
nxWinsockTransport1.Active:=true;
nxSimpleCommandHandler1.Active:=true;
kbmMWNexusDBServerPluginCommandHandler1.Active:=true;
kbmMWServer1.Active:=true;
```

and the **OnClick** of the Don't listen button should contain:

```
kbmMWServer1.Active:=false;
kbmMWNexusDBServerPluginCommandHandler1.Active:=false;
nxSimpleCommandHandler1.Active:=false;
nxWinsockTransport1.Active:=false;
```

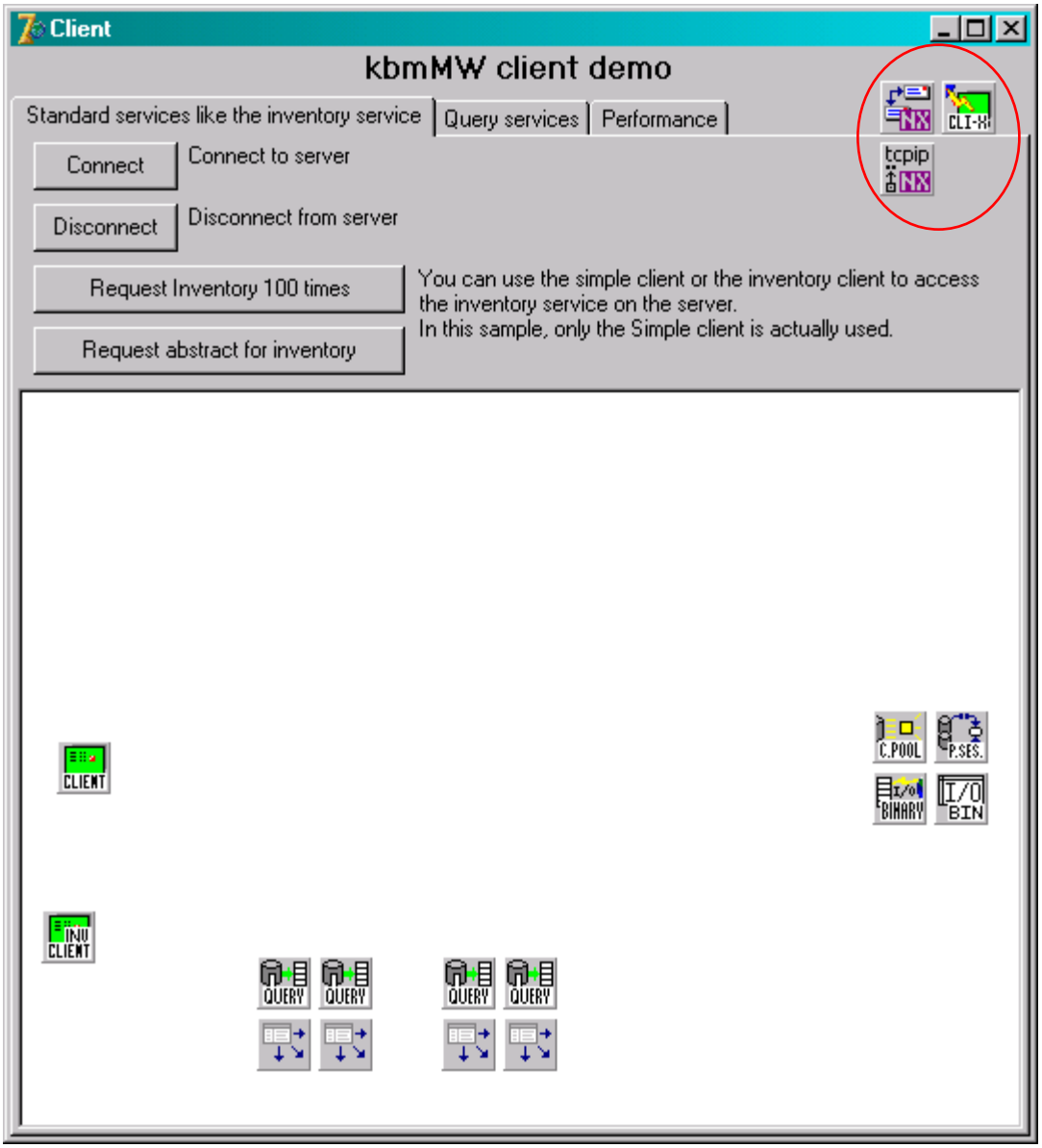
Notice that kbmMW supports multiple concurrent transports, and thus you can choose to combine this sample with other server side transports to keep support for non NexusDB oriented clients.

Compile your application server and its ready to rock and roll!

The client

Again we take the standard client demo as example for how to make a client work with the NexusDB transports.

On the client side you need to add a **TnxWinsockTransport** (or whatever NexusDB transport that match the one you chose to use on the application server), a **TnxSimpleSession** and a **TkbmMWNexusDBClientTransport** component.



Connect the *TnxSimpleSession's* **Transport** property to the selected NexusDB transport (i.e. the *TnxWinsockTransport*).

Set the *TnxWinsockTransport's* **ServerNameRuntime** and **ServerNameDesigntime** to 127.0.0.1 (in our case when the server and client runs on the same machine).
Finally set the *TkbmMWNexusDBClientTransport's* **Session** property to the *TnxSimpleSession* component.

Now point all *TkbmMWxxxClient's* and *TkbmMWClientConnectionPool's* **Transport** properties to point on the *TkbmMWNexusDBClientTransport*.

Make any adjustments like setting Port etc. on the NexusDB transport (TnxWinsockTransport) to match the settings on the application server.

Again the components needs to be activated for connection and deactivated for disconnection. This can be done either by setting NexusDB's **ActiveRuntime** properties to true/false at designtime, or by setting the **Active** properties to true/false at runtime e.g. in the Connect/Disconnect buttons OnClick events.

Compile and smile ☺

Why?

That's a good question… why?

Well if you have a special situation where the app server will have a NexusDB database embedded, and where clients must get direct access to this database, not going through kbmMW, you would have a situation where a connection is already established between the client and the server.
In this case the client is able to reuse the connection for transporting kbmMW related requests.

It could also be that you want to utilize one of NexusDB's transports which do not have an equivalent in the kbmMW world, like the named pipe or COM transports.

But there are also some disadvantages in doing this as you will not easily be able to use kbmMW's automated failover and loadbalancing features, and will have no control other control of the transport formats than what NexusDB gives. Thus you wont be able to use the transport over a http proxy server for example, and you are limited to the compression and security features supported by NexusDB.

Some preliminary performance tests shows that the NexusDB is around 15% faster than if using the standard Indy 9 client and server side transports.

This concludes the 'kbmMW and the NexusDB transport' whitepaper.

Kim Madsen
Components4Developers