Under Construction: MIDAS, CORBA And TClientDataSet

by Bob Swart

The `TClientDataSet` component, (part of Delphi Client/Server) is one of the core components of Inprise's middleware strategy. It connects a client to a MIDAS, CORBA or MTS server.

TClientDataSet

Let's see how we can integrate some middleware technology (such as MIDAS or CORBA) with our client components and applications to turn them into multi-tier applications.

MIDAS stands for MIddle-tier Distributed Application Services. MIDAS 1 supports COM/DCOM, OLEnterprise/RPC and later TCP/IP sockets, whilst MIDAS 2 now also supports CORBA and DCOM/MTS.

CORBA stands for Common Object Request Broker Architecure, and can be seen as a multi-platform version of DCOM, supported by OMG. The Visigenic (now Inprise) VisiBroker ORB is one of the most common. Especially since it's used in Netscape and licensed by companies like Sun, Oracle and IBM.

Both MIDAS and CORBA can be used as a technique to share data and objects between computers on a network (clients and servers). The difference between MIDAS and CORBA is the fact that MIDAS Server only runs on NT, and the client can run on NT (DCOM) or anything else (using CORBA). A CORBA Server, on the other hand, can run anywhere.

The MIDAS Server

So much for the theory. Let's start with a very basic example of a distributed (N-Tier) application using MIDAS. First, we'll create the MIDAS Server Application. This can be done by starting with a regular project, renaming it to MidasServer, and adding a Remote Data Module to it (on the `Multitier` tab of the Object Repository). Note that we can actually create three Server Data Modules here: for CORBA, for MTS (Microsoft Transaction Server) and for MIDAS.

Once we double-click on the (MIDAS) Remote Data Module, we get the Remote Data Module Wizard, in which we can specify the `ClassName` (`DrBobMIDAS`), the instance type and threading model. For instancing, we can chose between Internal, Single Instance or Multiple Instance. Internal is the type we need to use when the Remote Data Module is added to an active Library (DLL). Since Instance makes sure only one instance of the Remote Data Module is created inside the executable, each client gets its own instance of the executable. With Multiple Instances we get one instance of the application (process) that will create all instances of the Remote Data Modules. Each client gets one Remote Data Module, but they all share the same process space.

If we select Internal instancing, we also need to specify the threading model to indicate how client calls are passed to the Remote Data Module. This can be Single, Apartment, Free or Both. Single means that the DLL will only get up to one request at a time, so no threading issues are involved. The other choices mean that each instance of the Remote Data Module will service up to one request at a time, while the DLL itself may handle multiple requests in separate threads.

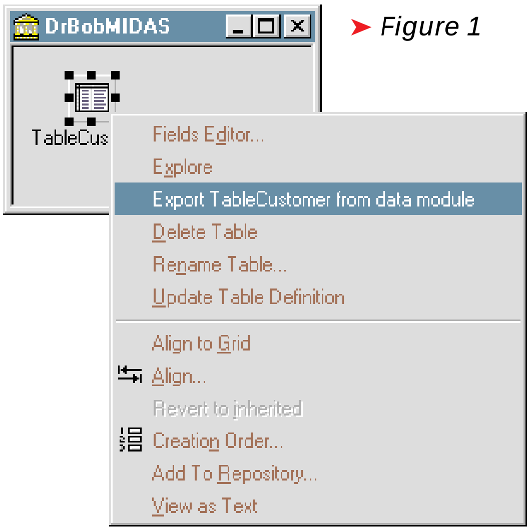
Since we only need a Single Instance, we don't need to worry about the Threading Model.

After we click `OK`, Delphi generates a new remote data module and adds it to the current project. We can now use this remote data module just like a normal data module and drop table and query components on it. In fact, let's drop a `TTable` component on it, set the `DatabaseName` to `DBDEMOS` and the `TableName` to `Customer.DB`. Set the `Active` property to `True`, so we know that the data can be seen.

This table, called `TableCustomer`, will be one that our MIDAS Server Application needs to 'provide' to our yet-to-build client application. We can specify this by right-clicking with the mouse on the `TTable` component and selecting the `Expert TableCustomer from data module` pop-up menu entry (Figure 1).

Note that the pop-up menu option is only visible the first time. Right after we actually exported the table, we don't see the option again when we right-click on the `TableCustomer` component (but if you close your project and re-open it again, the option is also available again, so it's not 100% perfect, yet).

Note that this only works for one table. If we have more than one table on the Remote Data Module,



➤ *Figure 1*

then, for every TTable or TQuery component we want to export, we need to put a TProvider component on the remote data module and connect it to that particular dataset. The reason it does work for one table is because a Remote Data Module also contains a default provider, which can be used in case of a single one dataset (as in our simple example).

The source code generated by Delphi for our Remote Data Module, including TableCustomer which is now exported, is shown in Listing 1. As we can see, exporting TableCustomer actually generated source code for the method Get_TableCustomer, which returns an IProvider (an interface to Provider), which means it can, *and will*, be called by the MIDAS client application connecting to this MIDAS server. Note the initialization section, where the TComponentFactory.Create method is called, with the arguments of our TDrBobMIDAS ComServer, specifying both the ciSingleInstance and tmSingle options. If we decide that we should specify Multiple Instances or Internal and the Apartment Threading model, then we can manually change these options here.

Now, there's one final thing we need to do to our MIDAS Server before we can safely compile and run it, and that's make sure we can recognise it when we see it later on in this article. We can do this by putting something familiar on the application's main form, such as a big TLabel component as in Figure 2.

Now, we can compile and run the application, which will also automatically register it as a MIDAS Server (so we'll be able to find it when we start to write the MIDAS client).

➤ Listing 1

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComServ, ComObj, VCLCom, StdVcl, BdeProv, DataBkr, DBClient, MidasServer_TLB,
  Db, DBTables;
type
  TDrBobMIDAS = class(TRemoteDataModule, IDrBobMIDAS)
    TableCustomer: TTable;
  private
  public
  protected
    function Get_TableCustomer: IProvider; safecall;
  end;
var DrBobMIDAS: TDrBobMIDAS;
implementation
{$R *.DFM}
function TDrBobMIDAS.Get_TableCustomer: IProvider;
begin
  Result := TableCustomer.Provider;
end;
initialization
  TComponentFactory.Create(ComServer, TDrBobMIDAS,
    Class_DrBobMIDAS, ciSingleInstance, tmSingle);
end.
```

➤ Figure 2

The MIDAS Client

As a separate MIDAS client, we can use just about any application, DLL, ActiveForm or whatever. All we need are a few components from the MIDAS tab and the DBCLIENT client support DLL.

Let's start the Delphi 4 Project Manager, save the first project as MidasServer and add a new application called MidasClient. The new Project Manager will show both applications as in Figure 3 *(a big thank you to Inprise for the Project Manager: I can finally add more than one project and target to a project group, without having to close and re-open projects!)*.

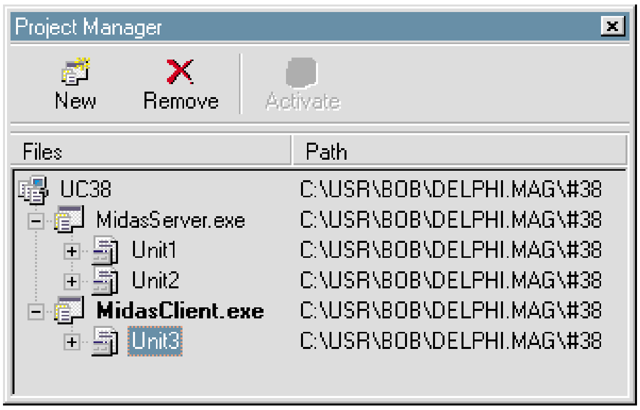
In order to make sure our MIDAS client application can be called a thin-client indeed, we should refrain from including the BDE. This means that the resulting client should be a standalone client, requiring no BDE, no BDE installation and no difficult BDE setup *[Hooray! Ed]*. In fact, apart from the 211,424 bytes DBCLIENT DLL, you could call the MIDAS thin client a zero configuration application, as some people indeed do.

To get an idea of the MIDAS components we can use, take a look at the MIDAS tab of the Component Palette (Figure 4). TRemoteServer and TMidasConnection are only included for backwards compatibility and should be avoided. Which leaves the two provider components, four connection components, the ObjectBroker and the ClientDataSet.

The provider components, used on the server, are used to export data from a dataset, and send it to a connected client. The connection components, used by the thin client, define the protocol used to connect the client to the server (using DCOM, CORBA, OLEnterprise or a simple TCP/IP Socket). The SimpleObjectBroker component can be used to locate a server for a connection component from a list of available application servers. Finally, TClientDataSet, also used on the client, is the most powerful of all, implementing a database-independent dataset that can be used in a thin client to receive data from a multi-tiered database server application.

We can drop a TDCOMConnection component on the main form of our thin client application. Now we need to set the ServerName property to the MIDAS Server that we created a minute ago. Remember that we also executed the MIDAS Server, to make sure it's registered on our machine, so we should see the name of the MIDAS Server application when we click on the dropdown listbox for the ServerName property of the DCOMConnection1 component. It should be MidasServer.DrBobMIDAS.

Now that we've defined the connection between the client and the server, it's time to drop on the TClientDataSet component and hook its RemoveServer property up to the DCOMConnection1 component. Next, we should set the value of the ProviderName, which can only take one possible value, namely



Files	Path
UC38	C:\USR\BOB\DELPHI.MAG\#38
MidasServer.exe	C:\USR\BOB\DELPHI.MAG\#38
Unit1	C:\USR\BOB\DELPHI.MAG\#38
Unit2	C:\USR\BOB\DELPHI.MAG\#38
MidasClient.exe	C:\USR\BOB\DELPHI.MAG\#38
Unit3	C:\USR\BOB\DELPHI.MAG\#38

➤ *Figure 3*

TableCustomer (Figure 5). Note that the list of possible ProviderNames are just the tables (providers) exported from the MIDAS Server application, which in this case is TableCustomer. Now all we need is a TDataSource connected to the ClientDataSet1 component, and a TDBGrid component (for example) connected to DataSource1.

Finally, we can set the Connected property of DCOMConnection1 and Active property of ClientDataSet1 to True to get a 'live' data feed from the MIDAS Server application. Note that once we set the Connected property of the DCOMConnection1 component to True, the MIDAS Server itself is started (it may take some time). As we made sure to put an identifying label on the MIDAS Server form, it's easy to recognise.

Briefcase Model

There's one more thing to note with respect to TClientDataSet. If we connect to the MIDAS Server and disconnect again, we still see data in the DBGrid component (provided we set ClientDataSet1.Active to True). This is due to caching, of course. However, *just re-think this statement once more*. The data inside the TClientDataSet is cached on the thin client, even when the server is not available!

This is called the briefcase model, where the TClientDataSet caches its data to and from disk as long as we're disconnected from the (MIDAS) data Server. TClientDataSet is using the SaveToFile and LoadFromFile methods for this.

The consequence of this is that the TClientDataSet encapsulates the power of a mini-DBMS. Of course, a number of restrictions apply and we can never get the performance of the BDE or a real DBMS, but at least we can edit, append, insert, delete and modify records in the dataset without being connected to the real database. In real life, this means we can disconnect our client machine from the server, perform our necessary changes, additions or whatever, and re-connect whenever we need to synchronise our changes with the server database. Again, power almost beyond belief in this single TClientDataSet component.

TClientDataSet does not support multi-user access to data (it's stored on the local disk or .DFM file on this single client machine), nor does it support SQL, but it does support filters, indices, calculated fields, BLOBs, master-detail relationships and nested tables. A topic worthy of an article in itself!

CORBA

Now that we have seen the MIDAS Server and Client, we can do the same with CORBA. However, instead of creating a (Remote) CORBA Data Module and re-creating the same server again, we can enable our existing server to support CORBA as well, thereby making an application that can service both COM and CORBA clients simultaneously.

The trick is to go back to the source code editor for the Remote Data Module (in the MidasServer project), and right click with the mouse. This brings up a pop-up menu, where we need to select the Expose as CORBA Object menu entry.

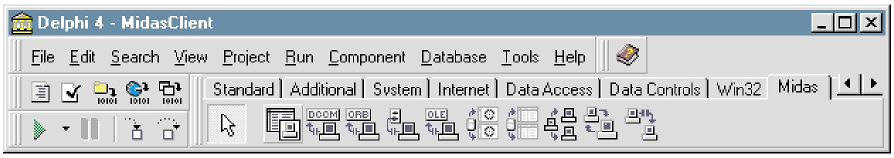
This simple, albeit non-obvious, action results in two changes in the MIDAS server source code for unit2. First, four extra units are added to the uses clause (CorbInit, CorbaObj, ComCorba and CorbaVcl). Second, a new line is inserted in the initialization section with a call to TCorbaVclComponentFactory.Create to create a CORBA interface for this server as well (Listing 2).

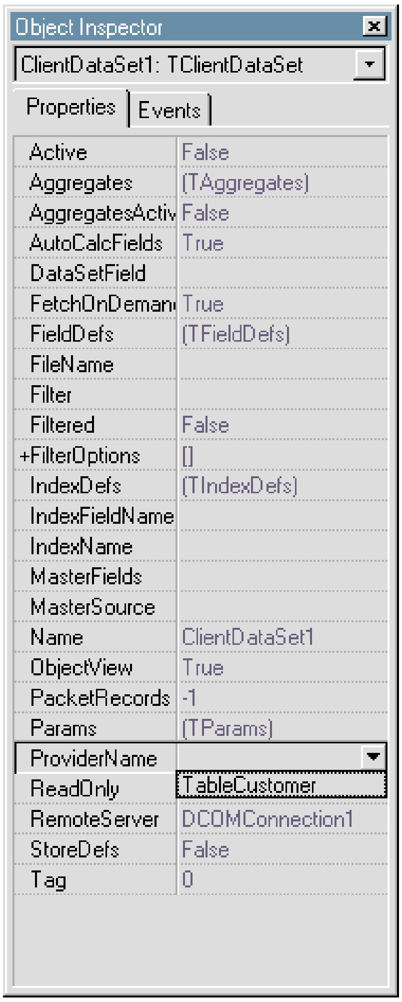
Having a MIDAS/CORBA Server and a MIDAS Client, all we need to do now is create a third project in our group, called CorbaClient. This time, we need a TCORBAConnection component (instead of DCOMConnection), and we need to set the RepositoryId of the CorbaConnection1 component to the value of the Server application and the Remote Data Module, being MidasServer/DrBobMidas in this case. Now we can add the TClientDataSet, TDataSource, TDBGrid and TDBNavigator components just like we did for the MidasClient application.

One word of caution: experience has taught me that when we try to connect the CORBA Client (ie set the Connected property of the CorbaConnection1 component to True), sometimes the MIDAS/CORBA Server cannot be found. This can be solved by making sure the VisiBroker Smart Agent is loaded first (some people even load the Smart Agent in their Startup group).

Now, let's change the label caption of this previously MIDAS-only server, and set it to Dr.Bob's MIDAS & CORBA Server. If we then both set the Connected property of the DCOMConnection1 and CORBAConnection1 components to True, then we can see the MidasServer now serving both the MidasClient and the CorbaClient (so the MidasServer is now actually using DCOM and CORBA at the same time): see Figure 6.

➤ *Figure 4*





Active	False
Aggregates	(TAggregates)
AggregatesActiv	False
AutoCalcFields	True
DataSetField	
FetchOnDemand	True
FieldDefs	(TFieldDefs)
FileName	
Filter	
Filtered	False
+FilterOptions	[]
IndexDefs	(TIndexDefs)
IndexFieldName	
IndexName	
MasterFields	
MasterSource	
Name	ClientDataSet1
ObjectView	True
PacketRecords	-1
Params	(TParams)
ProviderName	
ReadOnly	TableCustomer
RemoteServer	DCOMConnection1
StoreDefs	False
Tag	0

➤ *Figure 5*

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComServ, ComObj, VCLCom, StdVcl, BdeProv, DataBkr, DBClient, MidasServer_TLB,
  Db, DBTables, CorbInit, CorbaObj, ComCorba, CorbaVcl;
type
  TDrBobMidas = class(TRemoteDataModule, IDrBobMidas)
    TableCustomer: TTable;
  private
  public
  protected
    function Get_TableCustomer: IProvider; safecall;
  end;
var DrBobMidas: TDrBobMidas;
implementation
{$R *.DFM}
function TDrBobMidas.Get_TableCustomer: IProvider;
begin
  Result := TableCustomer.Provider;
end;
initialization
  TCorbaVclComponentFactory.Create('DrBobMidasFactory', 'DrBobMidas',
    'IDL:MidasServer/DrBobMidasFactory:1.0', IDrBobMidas, TDrBobMidas,
    iMultiInstance, tmSingleThread);
  TComponentFactory.Create(ComServer, TDrBobMidas,
    Class_DrBobMidas, ciSingleInstance, tmSingle);
end.
```

➤ *Listing 2*

Disconnecting one of the thin clients from the server means that one is using cached data from the `TClientDataSet`, while the other is still connected to the Server itself.

Wrappers

Since almost all the source code is generated automatically by Delphi (we just set some property values), I've decided not to provide you with the source on disk. In fact, after having written this article, I'm more convinced than ever that the only way to truly learn how to use MIDAS and CORBA in your Delphi projects is to sit down and do it.

Next Time

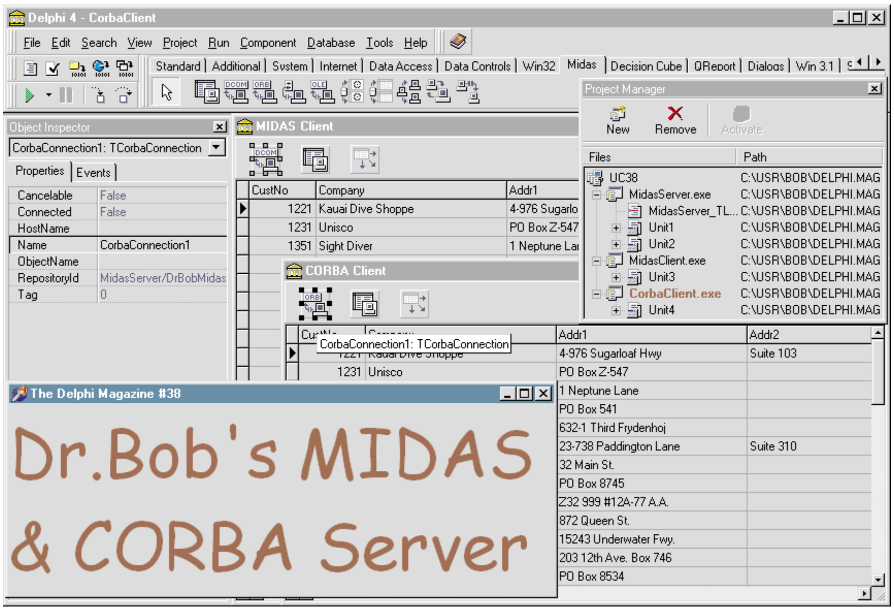
Next time, we'll actually use the described technique to write a distributed real-world application. The purpose of the application will be to monitor and track website visitors, going from page to page, and actually reporting frequently visited paths. Quite insightful for webmasters, and just plain fun to watch. The techniques will probably include a MIDAS/CORBA 'tracking' application (on the web server) and an ActiveForm thin client application (on the client machine). *Stay tuned...*

Acknowledgements

More technical information about MIDAS can be found on the Inprise website at www.inprise.com/midas. Thanks to Hubert A Klein Ikkink (aka Mr.Haki, visit www.drbob42.com/jbuilder) for his help and useful feedback while writing this article.

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is a professional knowledge engineer technical consultant using Delphi, JBuilder and C++Builder for Bolesian (www.bolesian.com) and freelance technical author.

➤ *Figure 6*



Cancelable	False
Connected	False
HostName	
Name	CorbaConnection1
ObjectName	
RepositoryId	MidasServer/DrBobMidas
Tag	0

CustNo	Company	Addr1
1221	Kauai Dive Shoppe	4-976 Sugarlo
1231	Unisco	PO Box Z-547
1351	Sight Diver	1 Neptune La

Files	Path
UC38	C:\USR\BOB\DELPHI.MAG
MidasServer.exe	C:\USR\BOB\DELPHI.MAG
MidasServer_TL...	C:\USR\BOB\DELPHI.MAG
Unit1	C:\USR\BOB\DELPHI.MAG
Unit2	C:\USR\BOB\DELPHI.MAG
MidasClient.exe	C:\USR\BOB\DELPHI.MAG
Unit3	C:\USR\BOB\DELPHI.MAG
CorbaClient.exe	C:\USR\BOB\DELPHI.MAG
Unit4	C:\USR\BOB\DELPHI.MAG

CustNo	Company	Addr1	Addr2
1221	Kauai Dive Shoppe	4-976 Sugarloaf Hwy	Suite 103
1231	Unisco	PO Box Z-547	
		1 Neptune Lane	
		PO Box 541	
		632-1 Third Frydenhoj	
		23-738 Paddington Lane	Suite 310
		32 Main St.	
		PO Box 8745	
		Z32 999 #12A-77 A.A.	
		872 Queen St.	
		15243 Underwater Fwy.	
		203 12th Ave. Box 746	
		PO Box 8534	