COMPONENTS 4 DEVELOPERS

Creation of customized services

for kbmMW v. 1.00+

kbmMW supports RPC, or Remote Procedure Calls, using a different term – services.

A service is a grouping of business logic (or business objects which is another term) on the server which can be accessed by clients through remote procedure calls.

In a normal 2-tier application, the server side often does not contain any business logic at all, but only the raw data in the form of a database of some kind. In the n-tier architecture (usually 3-tier) there are two tiers (or servers) on the server side – the database server and the application server.

Instead of placing business related code (e.g. special calculations, lookups in databases, screenscraping on backend host systems etc.) in the client, this code would be moved to a service object on the server side application server.

The benefits of this are that it is much easier to maintain one server than to have to upgrade 100s of clients. Further, it makes it much easier to have several different types of clients use the business rules, since they would not have each of them to contain the business code themselves. They would instead simply call the application server that contains the business code.

This document will show two ways to create services, the easy way using a wizard and the manual way.

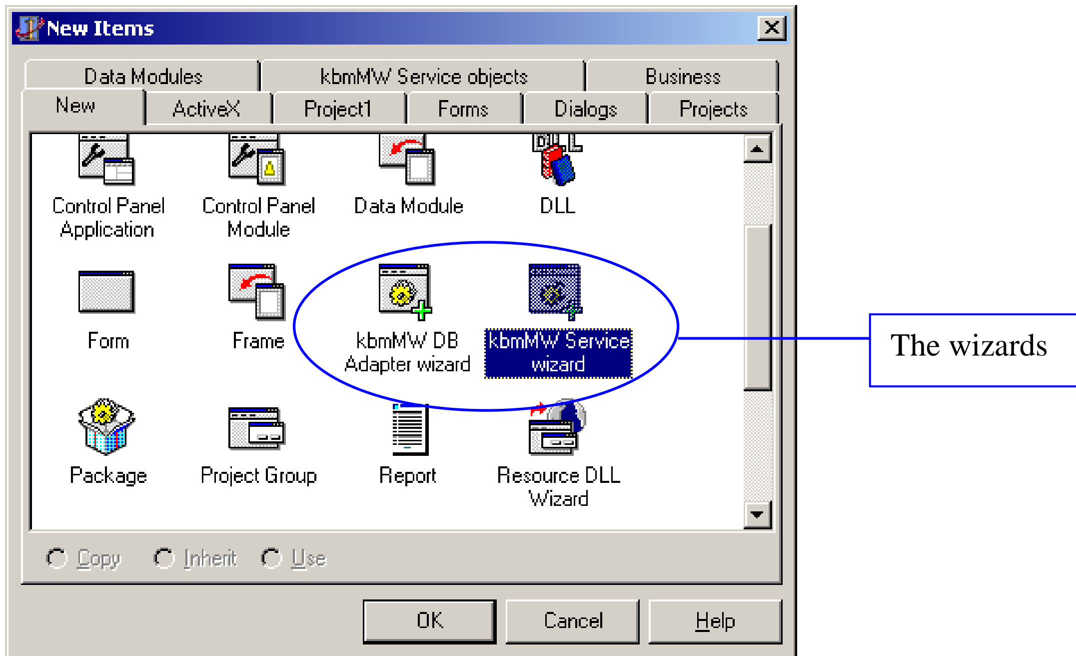
To better understand of the inner workings of services, it is strongly suggested that you read the manual method even if you choose the wizard for creating new services.
For our example, we will create a new service (KBMMW_TIME) version 1.0 which will return the server time to the client when called.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

The magic of wizards

The easiest way to create a new service is to use the kbmMW Service Wizard.

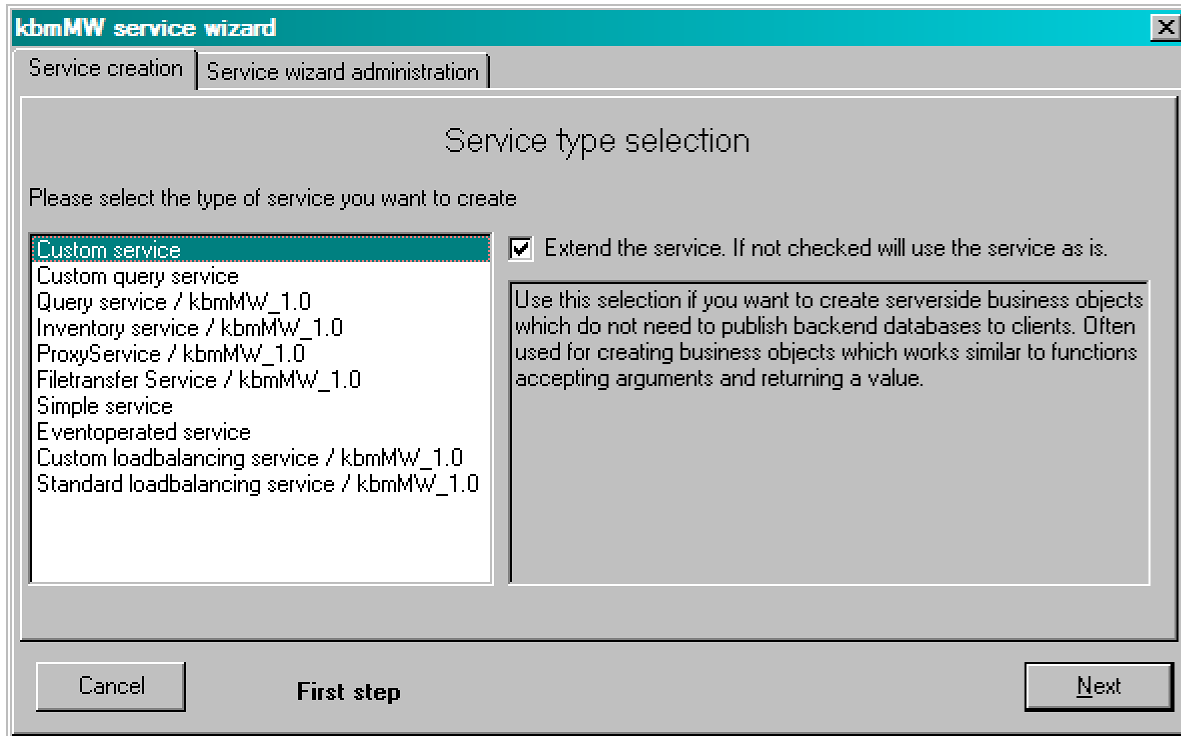
Find it in File->New (->Others for Delphi 6):



Select the kbmMW Service wizard and click OK.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

This will start the wizard with this initial dialog where you are asked to choose the type of service you want to create. You have several choices depending on the types of registered services on your installation. For creating a simple business object, you would normally either choose to base it on Custom service, Simple service or Eventoperated service. Read more about the different services in the description in the wizard.



For the example in this document we want to create a custom service.

The 'Extend the service' checkbox means you are creating a descendent of the service (adding new functionality) instead of using it as is. If you would use as is, the wizard would tell you what to add to your project to use it.

All services are automatically registered in the registry for the benefit of the wizard. By selecting the 'Service administration' tab, that information can be viewed and altered.

After selecting the Custom service, click Next.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

kbmMW service wizard

Service creation | Service wizard administration

Basic service information

Required for non query service - Please enter the preferred name the service should be known as. No spaces or special characters in the name. Eg. MYSERVICE.

KBMMW_TIME

Optional - Please enter the version of the service. An empty string is allowed. The version is important if more versions exist of services with the same service name. Eg. 1.0

1.0

Cancel | **Step 2 of 9** | Previous | Next

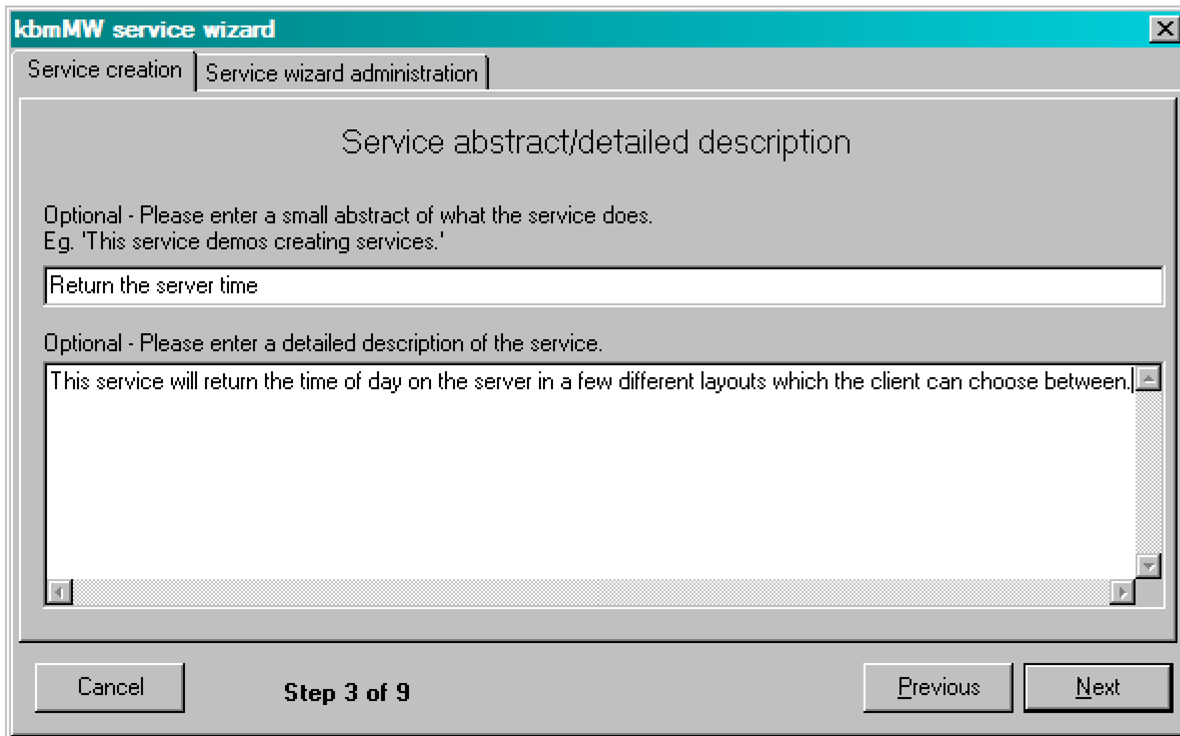
Here you are requested to give information about the preferred service name and optionally a version name/number. The preferred service name is the one that is used to identifying the service when clients want to access the service. The name is only a preferred name since the developer can choose to override the name during registration of the service in the server. More about that in the part about manually creating services.

The version information is available to allow for several versions or variants of the same service identified by the same service name. Clients will then have to request a specific version as well as the service name to access the service.

After providing the information click Next.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

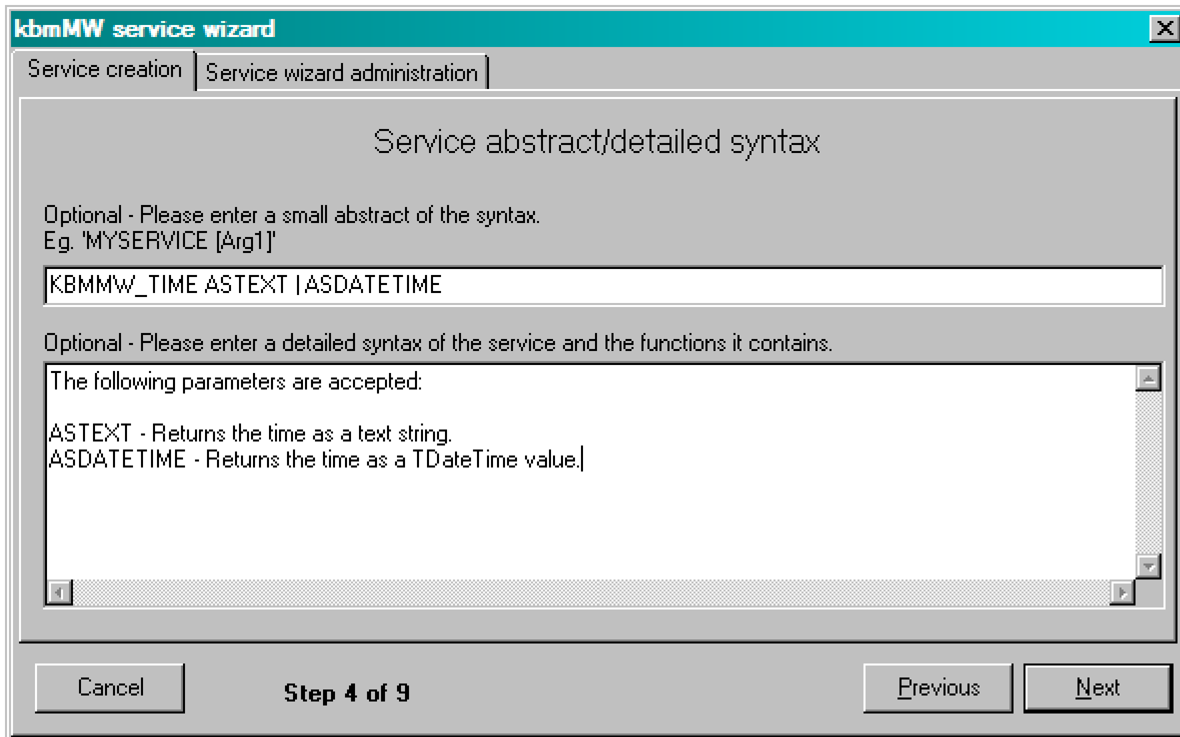
Here you will be asked to provide a small human readable description for what the service is about. This information is used by the service wizard, and can also be asked for by clients.



Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Then you are asked to give some optional information about how the service should be used.

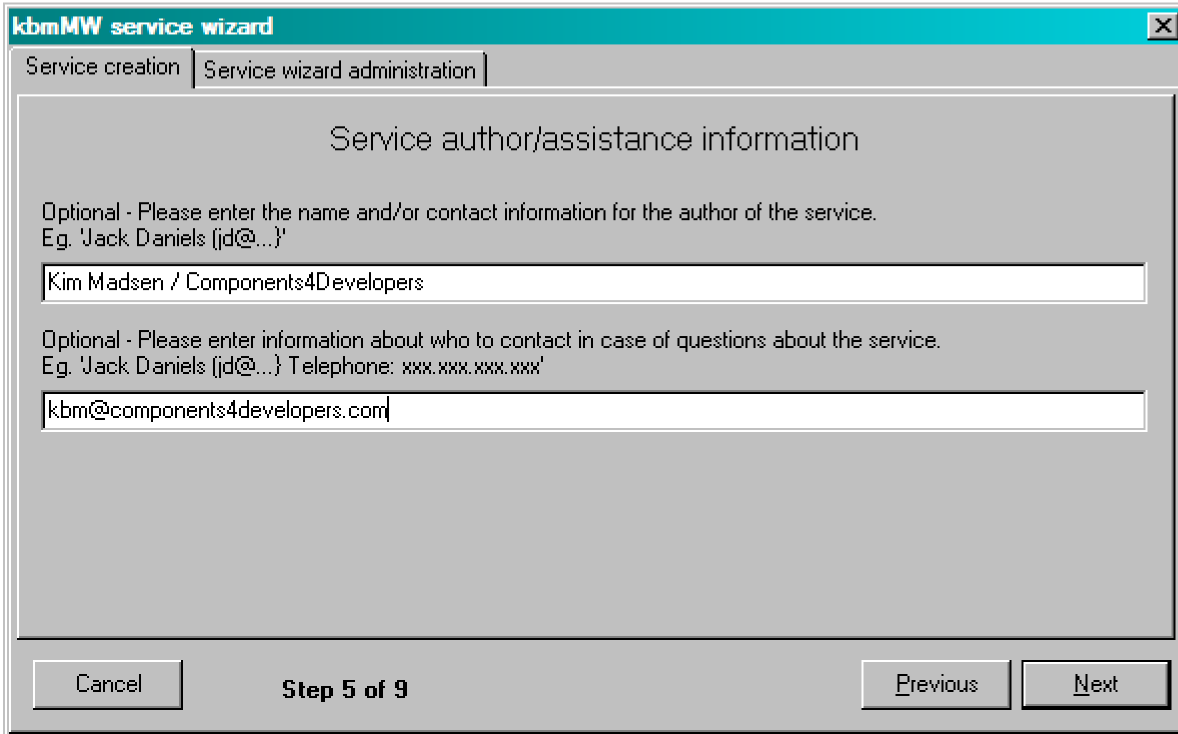
This information can be obtained by clients via the standard inventory service if the service has been registered on the server. This way, clients may be built that can query the server for what type of information the server can supply.



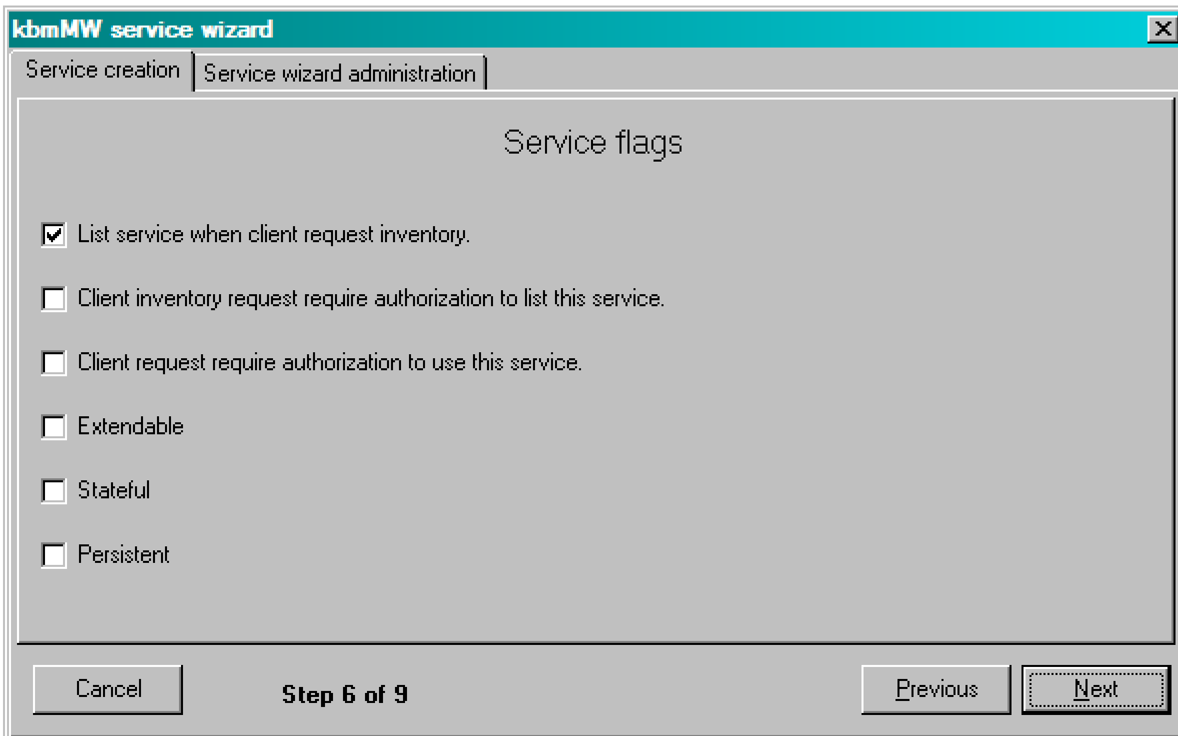
After optionally filling in the information, click Next.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Here some more optional informative information can be given about who is the author, and who to contact in case of questions/problems.

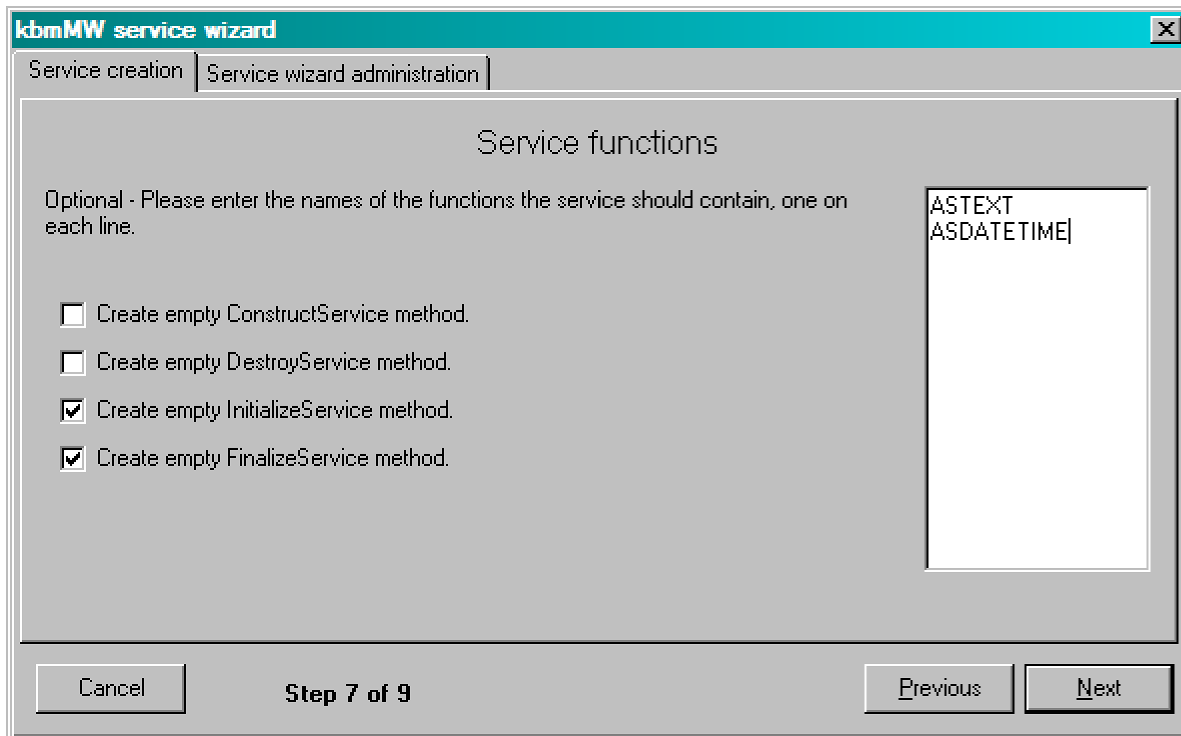


After optionally filling in the information, click Next.



Copyright 2001-2003 Components4Developers
http://www.components4developers.com

The service flags determine if the service is listed in the inventory. That is, they determine if the client will be able to get information about this service.
Further, it allows you to decide if the client must go through an authorization process before being allowed to retrieve inventory information or to use the service. For this example, we will leave the service open for anyone to use.
“Extendable” allows you to decide if other services may be built on top of the service. It is merely used by the service wizard to determine how to handle that specific service.
“Stateful” can be checked if the service is stateful. That means that a service instance belongs to a specific client until the client chooses to release it. Clients can use this to store information on the server which are supposed to be used from request to request. Generally, don’t make services stateful unless a very specific requirement exists, because it degrades server performance. Stateful service instances cannot be shared between clients.
“Persistent” is another type of stateful service, except it’s a service that’s globally stateful. The service is always running, but never owned by any specific client. Usually you only want one instance of a persistent service to run. Make sure to limit this either by overriding the GetMaxInstances class function or to set it during registration of the service via the returned service definition object’s MaxCount property. A persistent service do not start until someone requires it, or unless a minimum number of instances have been defined for it either by overriding the GetMinInstances class function or via the service definition object’s MinCount property. The persistent service will never be garbage collected.
After selecting the choices you want, click Next.



Finally you need to specify the functions the service should support. You can specify as many as you like. Along with the servicename and service version, the client also specifies which function to run. The function names can be anything using a combination of A-Z and 0-9 along with - and _.

You also have the option to let the wizard create empty ConstructService, DestroyService, InitializeService and FinalizeService methods. The ConstructService/DestroyService should be

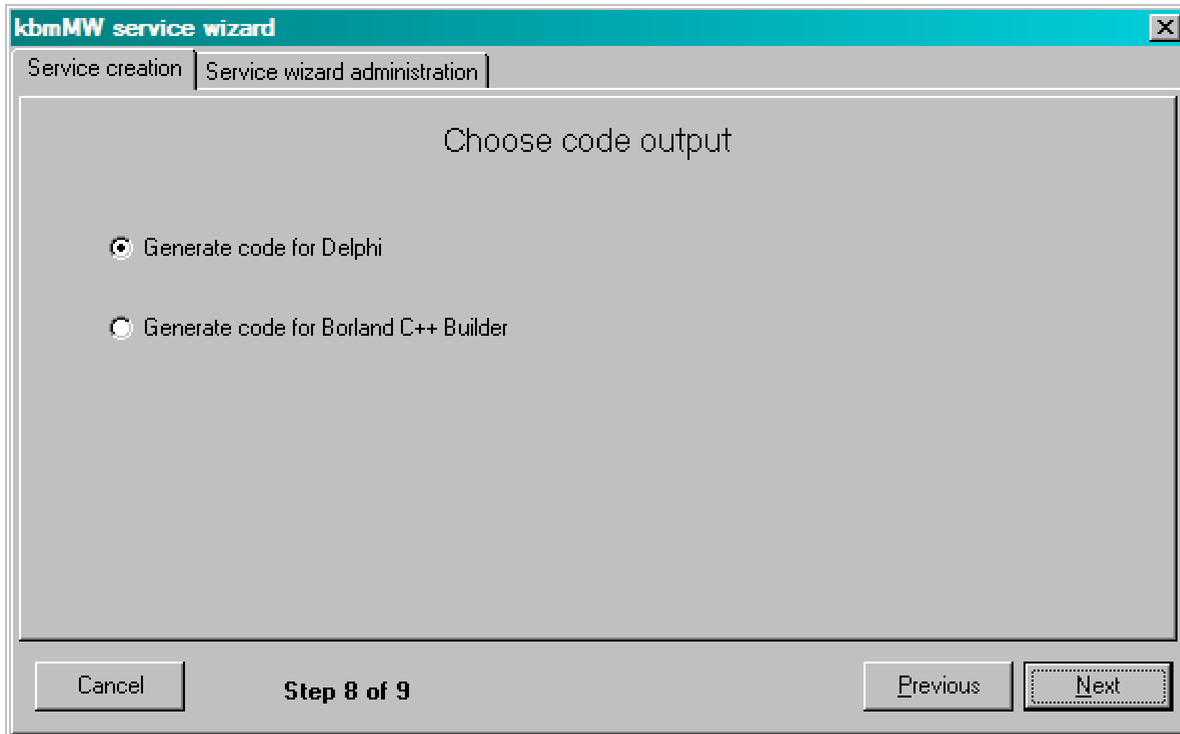
Copyright 2001-2003 Components4Developers
http://www.components4developers.com

used instead of the standard constructor/destructor scheme if you want your service to be directly portable between C++ and Pascal environments.

ConstructService is called immediately after a service instance has been created. DestroyService is called immediately before the service instance is destroyed.

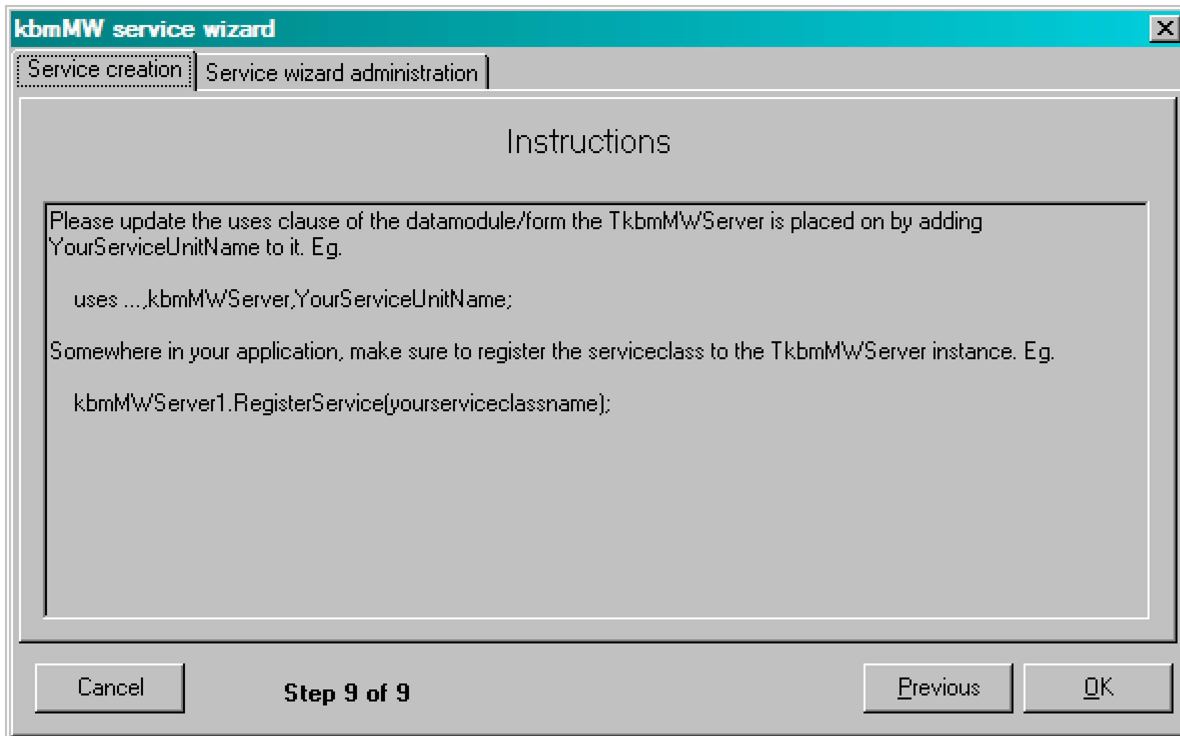
InitializeService is called when a client requests a service and a service instance has been made ready for use by the client (either by creation of new, or by using existing one from service pool).

FinalizeService is called after the client has finished using the service.



Finally choose what kind of code output should be generated - either code for Delphi or for C++ Builder. If you want your service to be portable between C++ Builder and Delphi environments, you should choose to generate Delphi code. If you strictly want to code in C++ Builder, choose C++ Builder code. Click Next.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com



Finally you get some basic instructions on how to continue.

Now all has been specified and the service can be generated. Click OK.

This will create a new service with support for the functions and information you have selected in the wizard.

Now we need to open the source to put some actual code in for the two functions the service supports (ASTEXT and ASDATETIME).

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Press F12 to change to view of source code.

```
  TkbmMWCustomService4 = class(TkbmMWCustomService)
  private
     { Private declarations }
  protected
     { Private declarations }
     function ProcessRequest(const Func:string; const ClientIdent:TkbmMWClientIdentity;
              const Args:array of Variant):Variant; override;
     function PerformASTEXT(ClientIdent:TkbmMWClientIdentity;
              const Args:array of Variant):Variant; virtual;
     function PerformASDATETIME(ClientIdent:TkbmMWClientIdentity;
              const Args:array of Variant):Variant; virtual;
  public
     { Public declarations }
{$IFNDEF CPP}class{$ENDIF} function GetPrefServiceName:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetVersion:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetAuthor:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetAssistance:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetSyntaxAbstract:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetSyntaxDetails:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetDescriptionAbstract:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetDescriptionDetails:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetExtendable:boolean; override;
{$IFNDEF CPP}class{$ENDIF} function GetFlags:TkbmMWServiceFlags; override;
     procedure InitializeService; override;
     procedure FinalizeService; override;
  end;
```

As you will see two special private functions have been created:
PerformASTEXT and PerformASDATETIME.
These two are the ones that we need to put some code in. You can look for the functions or search for the text ' Functions published by the service.' which is a remark just before the user-defined functions.

Further, you will notice that InitializeService and FinalizeService methods have been generated for you.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

COMPONENTS 4 DEVELOPERS

```
// Functions published by the service.
//-----------------------------------

function TkbmMWCustomService2.PerformASTEXT(ClientIdent:TkbmMWClientIdentity;
                                            const Args:array of Variant):Variant;
begin
     // Enter code here to perform function ASTEXT
     Result:='';
end;

function TkbmMWCustomService2.PerformASDATETIME(ClientIdent:TkbmMWClientIdentity;
                                                const Args:array of Variant):Variant;
begin
     // Enter code here to perform function ASDATETIME
     Result:='';
end;
```

Then enter the code for the functions so they will look like this:

```
// Functions published by the service.
//-----------------------------------

function TkbmMWCustomService2.PerformASTEXT(ClientIdent:TkbmMWClientIdentity;
                                            const Args:array of Variant):Variant;
begin
     // Enter code here to perform function ASTEXT
     Result:=DateTimeToStr(now);
end;

function TkbmMWCustomService2.PerformASDATETIME(ClientIdent:TkbmMWClientIdentity;
                                                const Args:array of Variant):Variant;
begin
     // Enter code here to perform function ASDATETIME
     Result:=now;
end;
```

Since we requested Initialize and Finalize methods to be created we should take a look at them too:

```
procedure TkbmMWCustomService2.InitializeService;
begin
     // Add code here which should be executed when a service instance is being initialized.
end;

procedure TkbmMWCustomService2.FinalizeService;
begin
     // Add code here which should be executed when a service instance is being finalized.
end;
```

Here some code could be added if needed.

Remember to override the GetMaxInstances and GetMinInstances methods in case you are creating persistent services. Eg. add

```
{$IFNDEF CPP}class{$ENDIF} function GetMinInstances:integer; override;
{$IFNDEF CPP}class{$ENDIF} function GetMaxInstances:integer; override;
```

to the class definition and

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

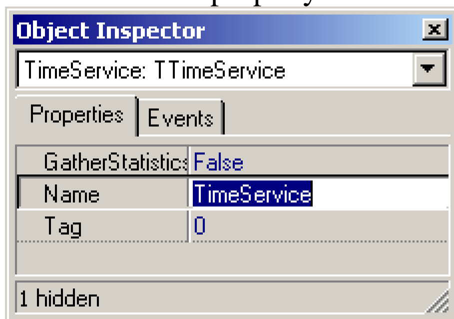
```
// Return the minimum number of instances that must be running at all times.
{$IFNDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetMinInstances:integer;
begin
     Result:=1;
end;

// Return the maximum number of instances that is allowed to run.
{$IFNDEF CPP}class{$ENDIF} function TkbmMWCustomService2.GetMaxInstances:integer;
begin
     Result:=1;
end;
```

to the implementation section. It’s optional to do so for all types of services since these values can be given at registration time of the service. But its highly recommended to add this for persistent services, not to forget it later on. This way it’s also possible to define default values for instance count for your service for other types of services.

Also notice the ifdef statements. They require CPP to be defined if the service is compiled under Borland C++ Builder. Thus the Delphi service code can be used as is in both Delphi and BCB. Of course if you chose to generate C++ code, you would already have a service written in C++, but that one would not be compatible with Delphi, only BCB.

That’s about it.

A small additional thing to do is to rename the datamodule your service is placed on by changing the name in the property editor:



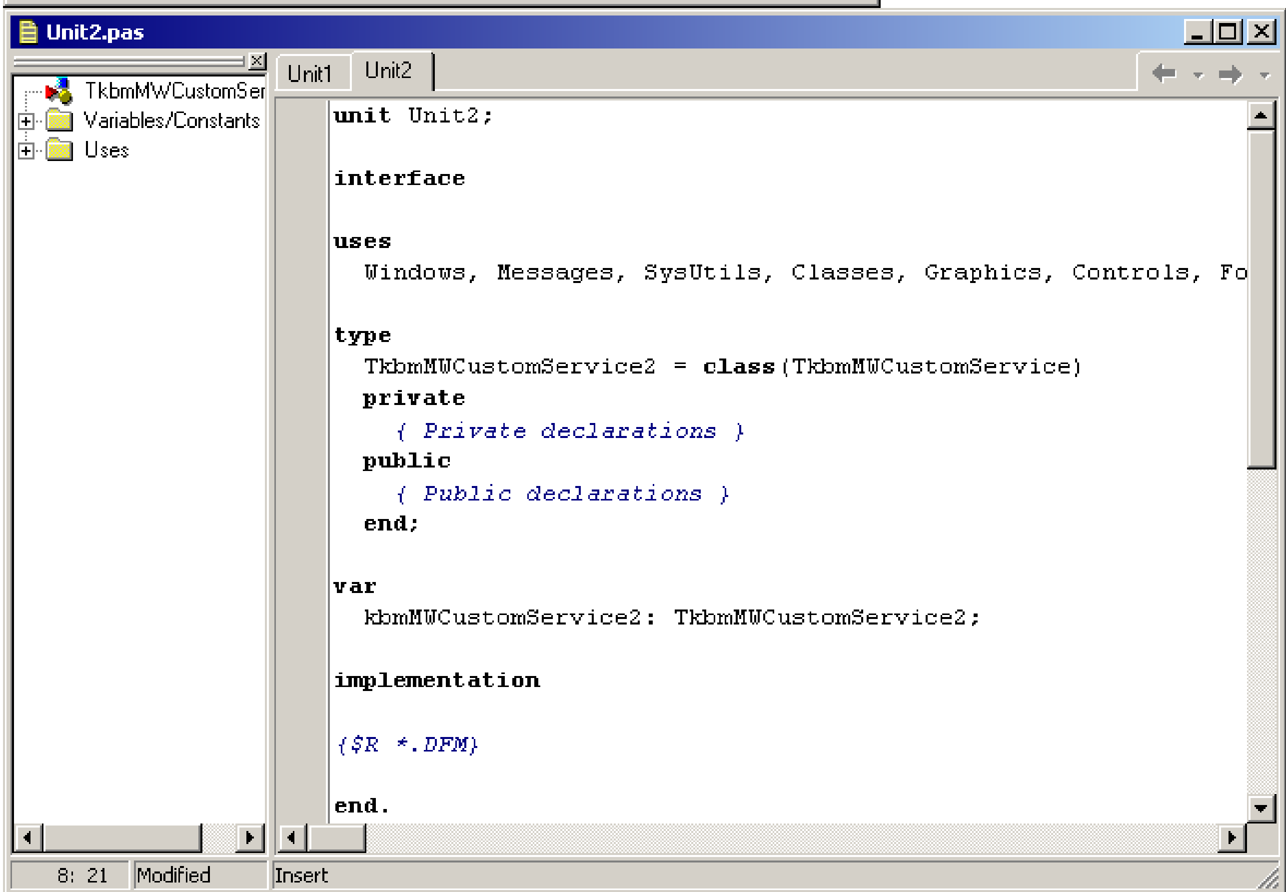
Change it for example to TimeService. Then save your service module. Finally you need to register the new service to the kbmMW server for it to be active. Check the '**Bind things together**' part of this document for how to do.

Inheritance is powerful

The other manual way is to create a new service **inheriting** from the TkbmMWCustomService datamodule.

This will leave you with an empty new service module which you can then put your business code into.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com



The service module also acts as a TDatamodule on which non visible components can be added to support your business objects.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Remember that more than one instance of the service module is most likely to run at the same time. Thus global variables are a no-no. If any access to global variables or any other shared resource is needed, the access must be wrapped into a critical section or other similar thread blocking methods. Also remember that kbmMW generally is stateless.
Thus your service object must not at any time rely on information stored in a previous call unless you know very well what you are doing. The order of which instance of the service object that is called is not possible to predict.
Each call from the clients should contain all variable information needed to perform the service operation.

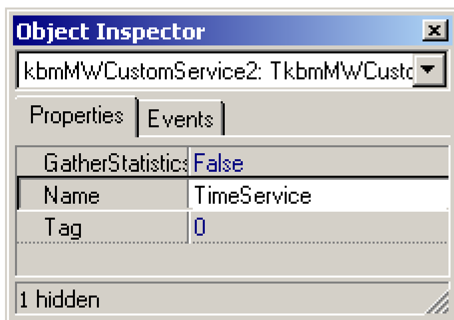
Down to business

A service object consists of an information part which the server can use to obtain information about for example the name of the service, the version and much more, and the actual implementation of business code.

The information part is actually a set of class functions which must be overridden to return some useful information and the implementation part is a single function named ProcessRequest.

As an example we will create a simple server time service which clients can ask to obtain what the time right now is on the server.

First we rename the service module to TimeService.



Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Then open the code window of our new Unit2 (our service object).

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  kbmMWServer, kbmMWGlobal, kbmMWSecurity;
```

The kbmMWGlobal and kbmMWSecurity needs to be added manually. The Forms, Dialogs, Graphics, Windows and Messages can be removed if so wished.

```
type
  // Inventory service.
  // Informs clients about which functionality is available.
  TTimeService = class(TkbmMWCustomService)
  private
    { Private declarations }

  protected
    { Protected declarations }
```

This is the function that the server will call when a client use this service. The implementation part that was mentioned before. You need to add this declaration to the service object.

```
    function ProcessRequest(const Func:string;
                      const ClientIdent:TkbmMWClientIdentity;
                      const Args:array of Variant):Variant; override;
```

This is a method which is called when a client needs to be authenticated to be allowed to use the service. It is an optional method to override.

```
    procedure Authenticate(const ClientIdent:TkbmMWClientIdentity;
           var Perm:TkbmMWAccessPermissions); override;

  public
    { Public declarations }
```

These are the class functions that the server use to know about the name of the service. GetPrefServiceName is required. The others are optional. You need to add these declarations to the service object.

```
{$IFNDEF CPP}class{$ENDIF} function GetPrefServiceName:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetVersion:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetAuthor:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetAssistance:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetSyntaxAbstract:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetSyntaxDetails:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetDescriptionAbstract:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetDescriptionDetails:string; override;
{$IFNDEF CPP}class{$ENDIF} function GetExtendable:boolean; override;
{$IFNDEF CPP}class{$ENDIF} function GetFlags:TkbmMWServiceFlags; override;
  end;
```


Copyright 2001-2003 Components4Developers
http://www.components4developers.com

COMPONENTS 4 DEVELOPERS

```
implementation

uses Unit1;

{$R *.DFM}

Here you add the implementation of the information functions.
```

GetPrefServiceName is used by the server to know the default name of the service, eg. the name the client should use to call an instance of this service. You would usually choose a name that somehow includes organisation or department. Further the name should only consist of uppercase characters (A-Z, 0-9 and underscore _).

```
class function TTimeService.GetPrefServiceName:string;
begin
   Result:='KBMMW_TIME';
end;
```

GetVersion is used by the server to know the version of the service. The combination of the service name and the version string is what identify a specific service. Thus you can have several versions of the same service to support both newer and older versions of clients if the interface to the service changes between versions. If GetVersion is not specified the default version string used is an empty string, in which case the fully qualified service name would not contain a version string.

```
class function TTimeService.GetVersion:string;
begin
   Result:='1.0';
end;
```

GetAuthor is used to identify the author of the service. If one of the standard services – the InventoryService - is registered on the server, clients can ask for information about specific services. Amongst those the name/email address of the author. If the GetAuthor function is not specified, the result would be an empty string.

```
class function TTimeService.GetAuthor:string;
begin
     Result:='Kim Bo Madsen (kbm@components4developers.com)';
end;
```

GetSyntaxAbstract is used to return a short resume of what the service is about. If the function is not specified, the result is an empty string.

```
class function TTimeService.GetSyntaxAbstract:string;
begin
     Result:='KBMMW_TIME ASTEXT|ASDATETIME';
end;
```


Copyright 2001-2003 Components4Developers
http://www.components4developers.com

GetSyntaxDetails is used to return a detailed description of what parameters should be used to access the service. Multiline answers should be build using the syntax: “Line1 information”,”Line2 information”....

```
class function TTimeService.GetSyntaxDetails:string;
begin
     Result:='"The following parameters are accepted:"'+
             '""'+
             '"ASTEXT - Returns the time as a text string."'+
             '"ASDATETIME - Returns the time as a TDateTime value."';
end;
```

GetDescriptionAbstract is used to return an abstract about what the service is all about.

```
class function TkbmMWCustomService2.GetDescriptionAbstract:string;
begin
     Result:='Return the server time';
end;
```

GetDescriptionAbstract is used to return a detailed explanation about what the service is all about.

```
class function TkbmMWCustomService2.GetDescriptionDetails:string;
begin
     Result:='"The service will return the time of the day on the server in a
few different layouts which the client ","can choose between."';
end;
```

GetExtendable simply returns a boolean value indicatiing if the service is intended to be extended.

```
class function TkbmMWCustomService2.GetExtendable:boolean;
begin
     Result:=false;
end;
```

GetFlags is used to determine how the service will be listed and be accessible.
The result returned should be a set of flags:

mwsfListed
: The service is listed in an inventory request from a client.

mwsfRunRequireAuth
: The service require authorisation to run. If this is set, the Authenticate method of the service is called. See later.

mwsfListRequireAuth
: The service require authorisation to be listed.

The default value is that services are listed.

```
class function TTimeService.GetFlags:TkbmMWServiceFlags;
begin
     Result:=[mwsfListed];
end;
```


Copyright 2001-2003 Components4Developers
http://www.components4developers.com

GetPermissions is used to get a specific clients permissions to a service object. It is called whenever a client tries to access the service.
A set of permissions should be returned:

mwapRead	A client can read info about the service.
mwapWrite	A client can write info to a service/configure it.
mwapDelete	A client can delete info from a service.
mwapExecute	A client is allowed to execute a service call.

Default is [mwapRead,mwapExecute]

```
class function TTimeService.GetPermissions(ClientIdent:TkbmMWClientIdentity):
           TkbmMWAccessPermissions;
begin
     Result:=[mwapRead,mwapExecute];
end;
```

The method Authenticate is called every time a client tries to access the service and the service flag requires the client to be authenticated. Its default behaviour is call the services OnAuthenticate event (needs to be published to be available during designtime) and then to call the server's OnAuthenticate event to return the permissions for the specific client.

```
procedure TTimeService.Authenticate(const ClientIdent:TkbmMWClientIdentity; var
Perm:TkbmMWAccessPermissions);
begin
     inherited;
end;
```

The function ProcessRequest is the workhorse. Its the one actually getting invoked when the client request the service. It is called with a Func (function name) which in our example is ASTEXT or ASDATETIME, a client identity object and an array of arguments if any was send from the client. The ClientIdent object contains information about who request this service. Most important members of this object is Username, Password, ClientLocation (the location the client has reported) and RemoteLocation (the location the server see the client as being from).

```
function TTimeService.ProcessRequest(const Func:string; const
ClientIdent:TkbmMWClientIdentity; const Args:array of Variant):Variant;
var
   f:string;
begin
     f:=UpperCase(Func);
     if f='ASTEXT' then result:=DateTimeToStr(now)
     else if f='ASDATETIME' then result:=now
     else kbmMWRaiseUnknownFunc(f);
end;

end.
```


Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Notice that all optional methods and functions do not need to be implemented. In this example they have been included with their default behaviour. But not including them would give the same result.

The arguments can be iterated like this:

```
for i:=low(Args) to high(Args) do
    ... Args[i] ...
```

Bind things together

Next step after creating your service object is to register it to the server.
You must have a main Tform or Tdatamodule which hosts a TkbmMWServer component along with some TkbmMWxxxxTransport component depending on your needs. The TkbmMWServer is the one responsible for distributing the client requests to the services registered to the server.

A good place to register the services is in the create event of the form. Registration of the service is done using either RegisterService or RegisterServiceByName. The difference between the two is that RegisterServiceByName allows you to overrule the service's preferred name.

```
function RegisterService(AServiceClass:TkbmMWCustomServiceClass;
         ADefault:boolean):TkbmMWServiceDefinition;
function RegisterServiceByName(AServiceName:string; AServiceClass:TkbmMWCustomServiceClass;
         ADefault:boolean):TkbmMWServiceDefinition;
```

AServiceClass is the class of your service.
ADefault must be set to true if the service is a default service (only one per TkbmMWServer). A default service is a service that is called when no other services are available or no services can be found matching the clients request.

Both these functions return a service definition object. The service definition object can be used to modify the behaviour of or get information about a service via the following properties:

property State:TkbmMWServiceState (readonly)
property Enabled:boolean
property GatherStatistics:boolean
property Name:string
property MinCount:integer
property MaxCount:integer
property MaxIdleTime:integer
property Stateful:boolean
property Persistent:boolean
property MaxIdleStatefulTime:integer
property Default:boolean
property DontQueue:boolean
property Instances:TThreadList (readonly)

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

State returns the current state of the service:
mwssOperational,mwssEnabled,mwssDisabled,mwssRemove

Enabled can be used to set if the service should be enabled or not for clients.

GatherStatistics determines if the service will gather service orientated statistics. Stats can be accessed via the TkbmMWServer.Statistics property.

MinCount determines how many instances must as a minimum be running.

MaxCount determines how many instances is allowed of this specific service. More instances will allow more clients simultaneous access to the service, but also require more resources since more threads could be running. –1 means that there are no restrictions to the number of concurrent running service instances (threads). The server starts a new instance only if all currently allocated instances are in use. If the MaxCount number of instances is reached, the client request is put on queue unless the DontQueue property is true.

MaxIdleTime determines how many seconds the non stateful instances of the service can be idle before its timed out and garbage collected (i.e. removed). Setting MaxIdleTime to 0 disables timing the service instances out.

MaxIdleStatefulTime determines how many seconds the in use stateful instances of the service can be idle before its timed out and garbage collected (i.e. removed). Setting MaxIdleStatefulTime to 0 disables timing any in use stateful service instances out.

Stateful determines if a service is stateful or not. A stateful service instance is owned by a client for as long time the client wants. Each time the clients requests the stateful service, the same service instance is handed the client. That means that the client can store information on the service which can be used between service calls. Generally do not specify a service to be stateful unless it's a specific requirement for your service. Not used properly, stateful services can degrade system performance severely.

Persistent determines if the service should ever be garbage collected. A persistent service will never be garbage collected. Generally only allow one instance of a persistent service unless specifically design considerations require otherwise.

Default determines if the service is a default service. A default service is one called when no other services are available matching the clients request. Only specify one service to be default. If more than one service is specified as default, only the first one registered will actually be used as default.

DontQueue determines if a client request should be queued waiting for a vacant service instance when the maximum number of service instances has been reached. If true, the request will fail immediately. If false, the client request will be queued.

Copyright 2001-2003 Components4Developers
http://www.components4developers.com

```
procedure TForm1.FormCreate(Sender: TObject);
var
   sd:TkbmMWServiceDefinition;
begin
 Server1.Active := true;
 sd:=Server1.RegisterService(TTimeService,false);
 // optionally modify the service definition settings here. Eg
 // sd.MaxCount:=10;
end;
```

To register the standard inventory service you need to add **kbmMWInventoryService** to the uses clause and an extra register statement:

```
  sd:=Server1.RegisterService(TkbmMWInventoryService,false);
```


Copyright 2001-2003 Components4Developers
http://www.components4developers.com

Putting it all into action

To test your new service you need to call it from a client application.
In the client application you have added for example a TkbmMWSimpleClient which is a simple way to be able to access most or all services. Further you must have a TkbmMWClientxxxxTransport which the TkbmMWSimpleClient.Transport points to.

Then the request towards the server could for example. be made in a TButton click event handler:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  v:string;
begin
  kbmMWSimpleClient1.Connect;
  v:=kbmMWSimpleClient1.Request('KBMMW_TIME','1.0','ASTEXT',[]);
  Edit1.Text:=v;
end;
```

If the service was defined without a service version, you could send an empty string instead of the version information (1.0).
In this case there are no arguments in the service call, thus the empty array [].

In addition to sending standard variable types in the argument array, its also possible to send and receive stream data via the kbmMWSimpleClient.RequestStream (for sending data in a stream to the service) and kbmMWSimpleClient.ResultStream (for receiving stream data from the service). The service has a similar named set of properties available for accessing and returning the stream data.

Notice that the TkbmMWSimpleClient directly uses a client transport.
If you already have a client connection pool in the client application, you can choose to use the TkbmMWPooledSimpleClient which instead connects via the client connection pool. This give the advantage of only using one connection for both simple client requests and dataset query requests.
Read more about the connection pool in the query service whitepaper.

That concludes the 'Creation of customized services' session.

Kim Madsen
Components4Developers

Copyright 2001-2003 Components4Developers
http://www.components4developers.com