COMPONENTS 4 DEVELOPERS

Extending the Service Wizard

for kbmMW v2 and newer

One of the most overlooked features of kbmMW is the service wizard.

It is often used by people to create instances of one of the bundled services. But its little known that it can be extended to handle creation of your own special services.

This way its easily possible to share a specialized service with other people, who can use it as a base for their own service.

The service wizard can even be extended with a custom user interface for situations where the developer needs to make decisions about how the created service is to operate, similar to what the fileservice, java service and query service have.

This document will show how to create service wizard extensions.

Prerequisites

Open your kbmMW runtime package file, eg. kbmMWRunXXXXYYY.bpl, and make sure that 'Build Control' in the package project options is set to 'Explicit rebuild'.
Do the same with the designtime package file, eg kbmMWDesXXXXYYY.bpl.

This ensures that a bug in Delphi, that would surface when we compile our SWE project later on, is circumvented.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Creating a simple Service Wizard Extension (SWE)

As the service wizard is used only at designtime, our service wizard extension must also be created for designtime use only.

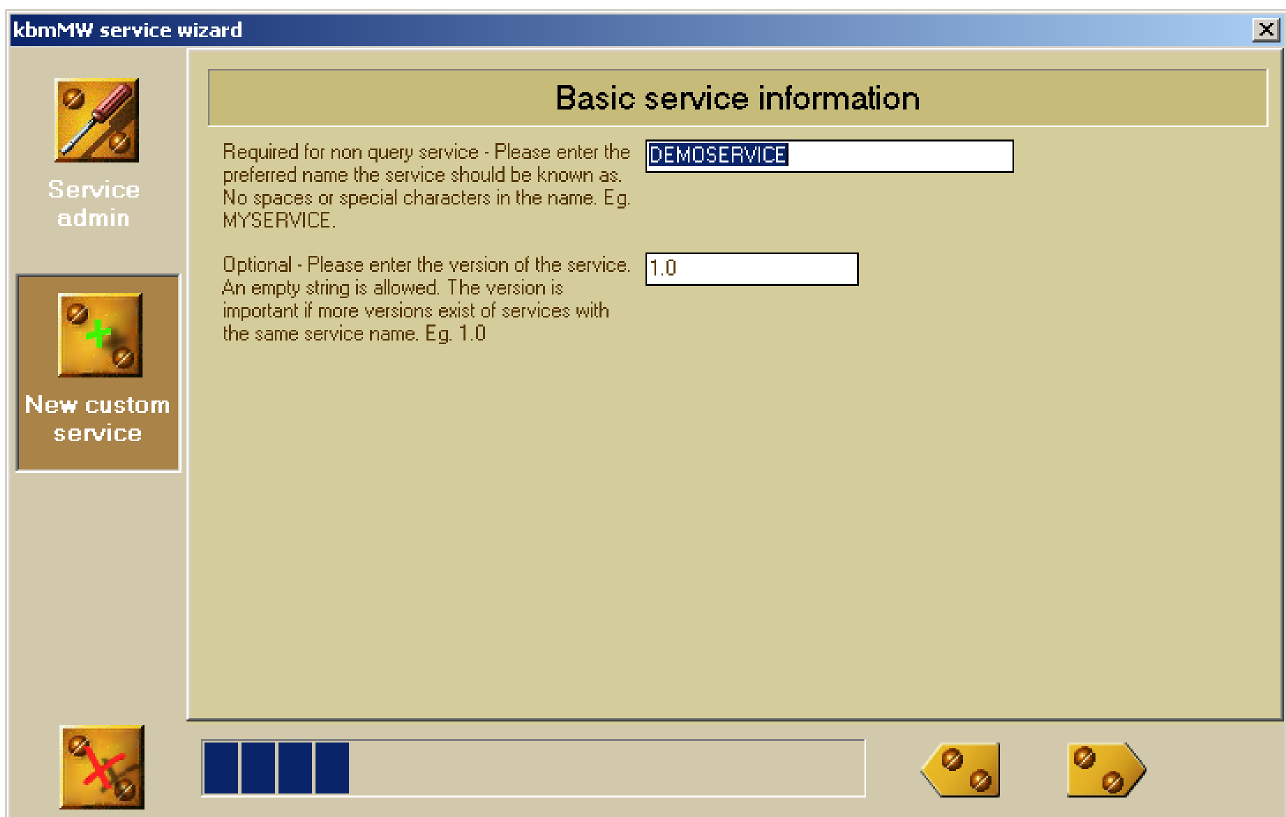
We start out with creating a new package in Delphi. (File/New/Delphi Package)

Select Project/Options to open the options panel for the current project (which is our package). Set the *Usage Options* radiobutton to be **Designtime only**. Make sure that *Build Control* is set to **Explicit rebuild**.

Save the package as for example: my_demo_swe

Next step is to add a custom service of some kind to the package. If you already have a service ready, you can use that. Otherwise for our sample, we add a Custom Service using the kbmMW Service Wizard.

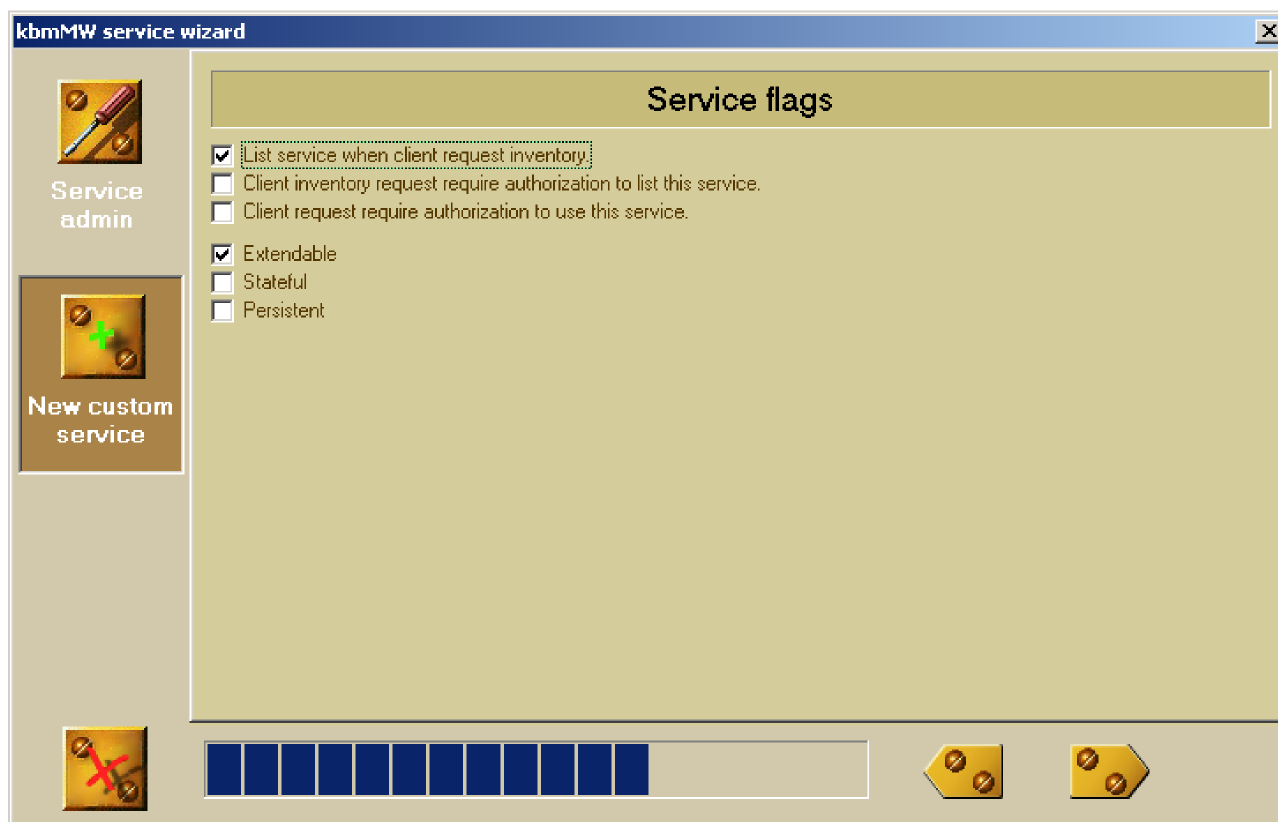
If we want the service to default to be known as for example DEMOSERVICE, we specify a service name in the wizard as usual.



Copyright 2001-2006 Components4Developers
http://www.components4developers.com

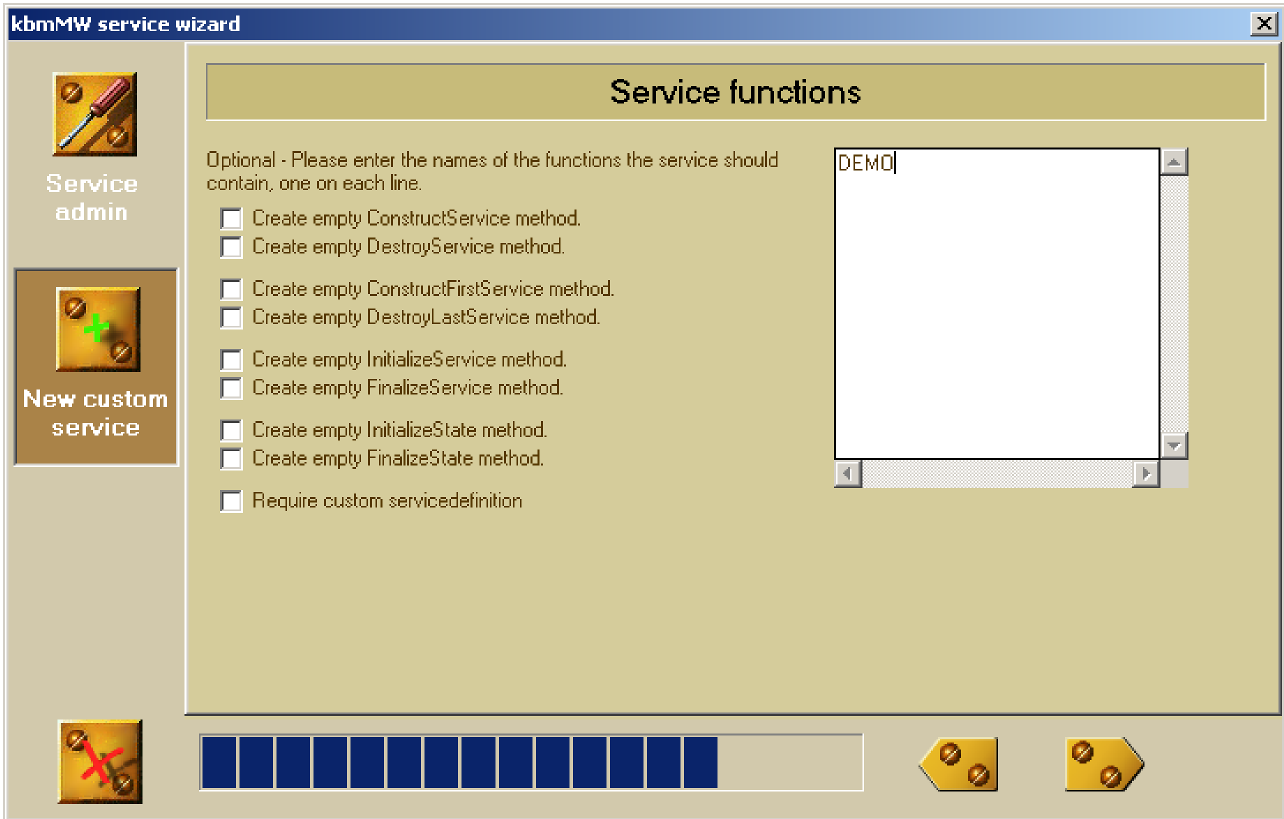
Optionally specify information for version, author, syntax description etc.

Set the service flags as required.



Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Next put some functionality into the service, like a function named DEMO:



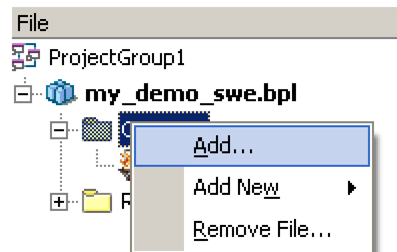
Proceed thru the remaining steps in the wizard and generate Delphi code.

Give the generated datamodule a new meaningful name instead of the default. Set its Name property at designtime. Eg. MyDemoService

Save the service giving the unit a unique name. In this sample we will save it as usvcDemoService.pas in the same directory as the package.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

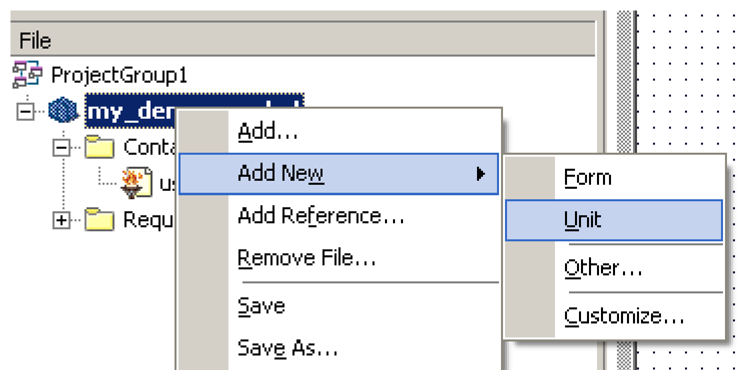
Next step is to add the generated service unit to your package.
On the package's contains element, rightclick and select Add:



Select the service unit (usvcDemoService.pas).

Then we need to add a new unit, containing the registration of all services within this package. Notice that its possible to have many services embedded in one package.

Rightclick the package project and select Add New – Unit.



Save this unit as for example uRegister.pas.

This unit should include at least one global procedure, the one called Register.

This special procedure is called by Delphi at designtime to register all required classes etc. contained in the package. The developer decides what to put into the procedure, but for our SWE, there are some required steps.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

The following explains the elements of the uRegister.pas unit.

```
unit uRegister;

interface
```

The following line includes some required definitions from kbmMW.inc. Simply copy kbmMW.inc and kbmMWConfig.inc from your kbmMW source directory to the directory of your service wizard extension, or add their directories to the search path for the SWE project.

```
{$I kbmMW.inc}
```

The following uses clause, should include all your own services that you want to register for use in the service wizard.

```
uses kbmMWReg,
     usvcDemoService;
```

The global Register procedure must be declared in the interface section to be accessible by Delphi.

```
procedure Register;

implementation

uses
```

The following ensures that the required Delphi designtime libraries are referenced.

```
{$ifdef LEVEL6}
 {$IFNDEF DOTNET}
  DesignEditors,
  DesignIntf
 {$ELSE}
  Borland.Studio.ToolsAPI,
  Borland.Vcl.Design.DesignIntf,
  Borland.Vcl.Design.DesignEditors
 {$ENDIF}
{$else}
  Dsgnintf
{$endif}
  ;
```


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

COMPONENTS 4 DEVELOPERS

The following is where the meat is. Its here all the registration is done. All the lines must be there.

```
procedure Register;
begin
```

The following line ensures that Delphi knows that our service module is indeed a container type module.

```
     RegisterCustomModule(TMyDemoService,TCustomModule);
```

The following line registers the service module to the kbmMW service wizard currently loaded in Delphi. The call takes two arguments, both arrays. The entries in the two arrays must match 1 to 1. The first argument is the service module class, the 2nd argument is an optional service wizard extionsion form which will be discussed later on. If no service wizard extension forms are available, pass nil.

```
     kbmMWRegisterServices([TMyDemoService],[nil]);
```

The following line ensures that the service module is registered in the registry (on Windows) or in a kbmMW service inifile (Linux). It is from here that a number of attributes are stored, that controls how the service wizard should handle this particular service. Eg. is the service extendable, descriptions etc.
The settings can all be altered at any time by the Administration section of the service wizard.

```
     kbmMWUpdateServiceTypes([TMyDemoService]);
end;

end.
```

Finally one should ensure that the package require kbmMW's runtime and designtime packages, and kbmMemTable runtime and whatever optional 3rdparty packages that are required by kbmMW's runtime package (DB support etc).

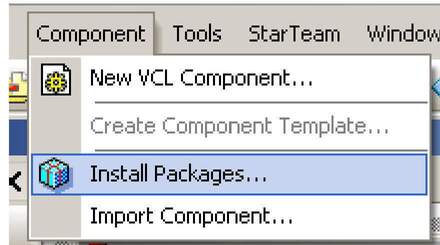
Now the package is ready to be compiled and sent to other people for installation in their Delphi environment (of same version and patch level!). After installation they will automatically have a new service type available in the wizard from which they can base their service instances on.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

COMPONENTS 4 DEVELOPERS

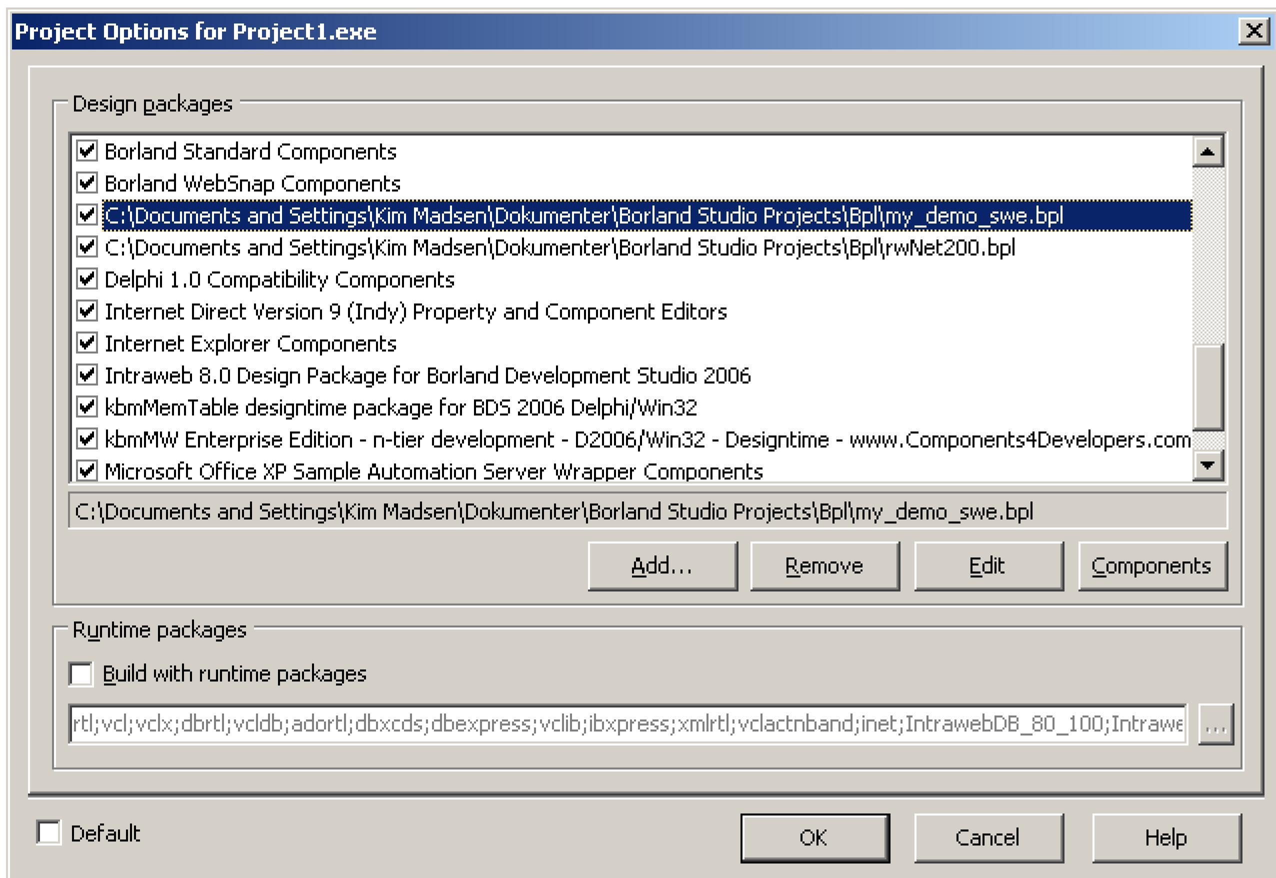
Installation of the SWE

In Delphi, select Component/Install Packages…



Then choose Add, and browse to your compiled package (in our sample its my_demo_swe.bpl) file and select it.

Then the package should automatically be listed in your currently known and loaded packages list in Delphi.

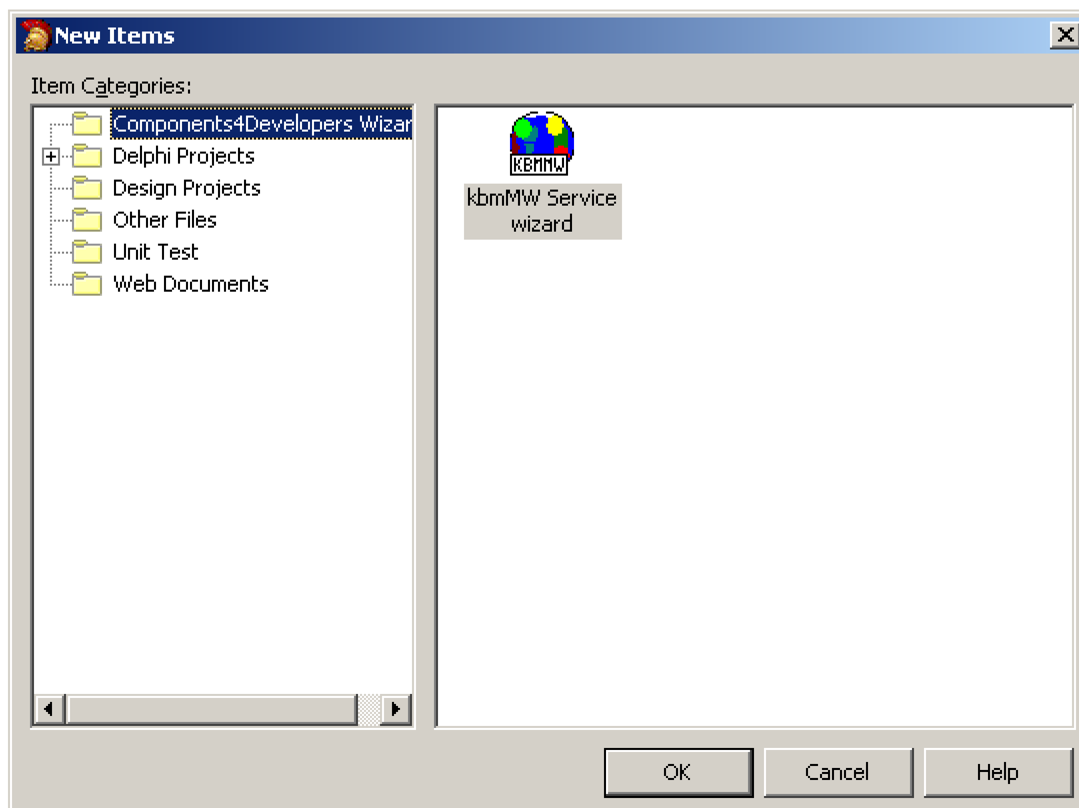


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Administrating the installed SWE's

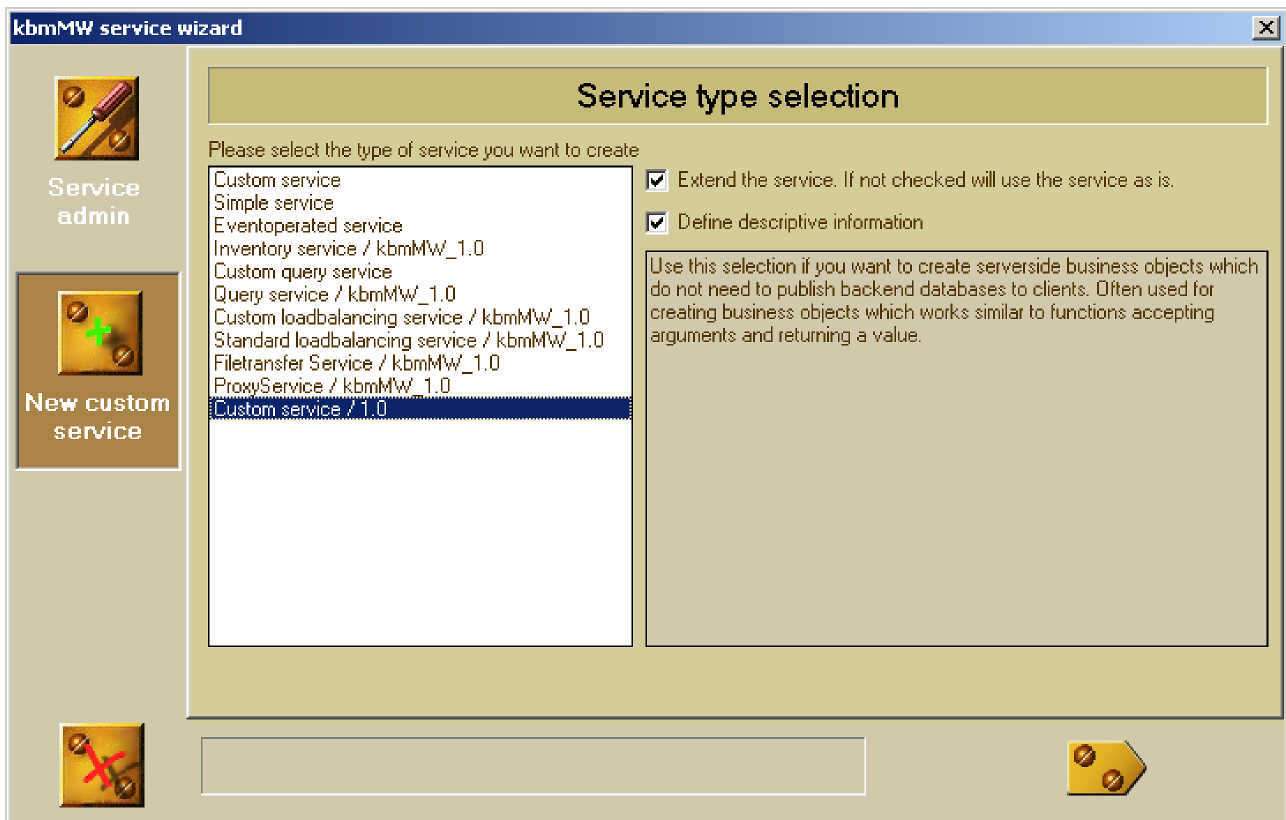
After the SWE's has been correctly installed, they can be administrated thru the service wizard.

To open the service wizard, make a new blank application project.
Then select New/Other/Components4Developers/kbmMW Service Wizard



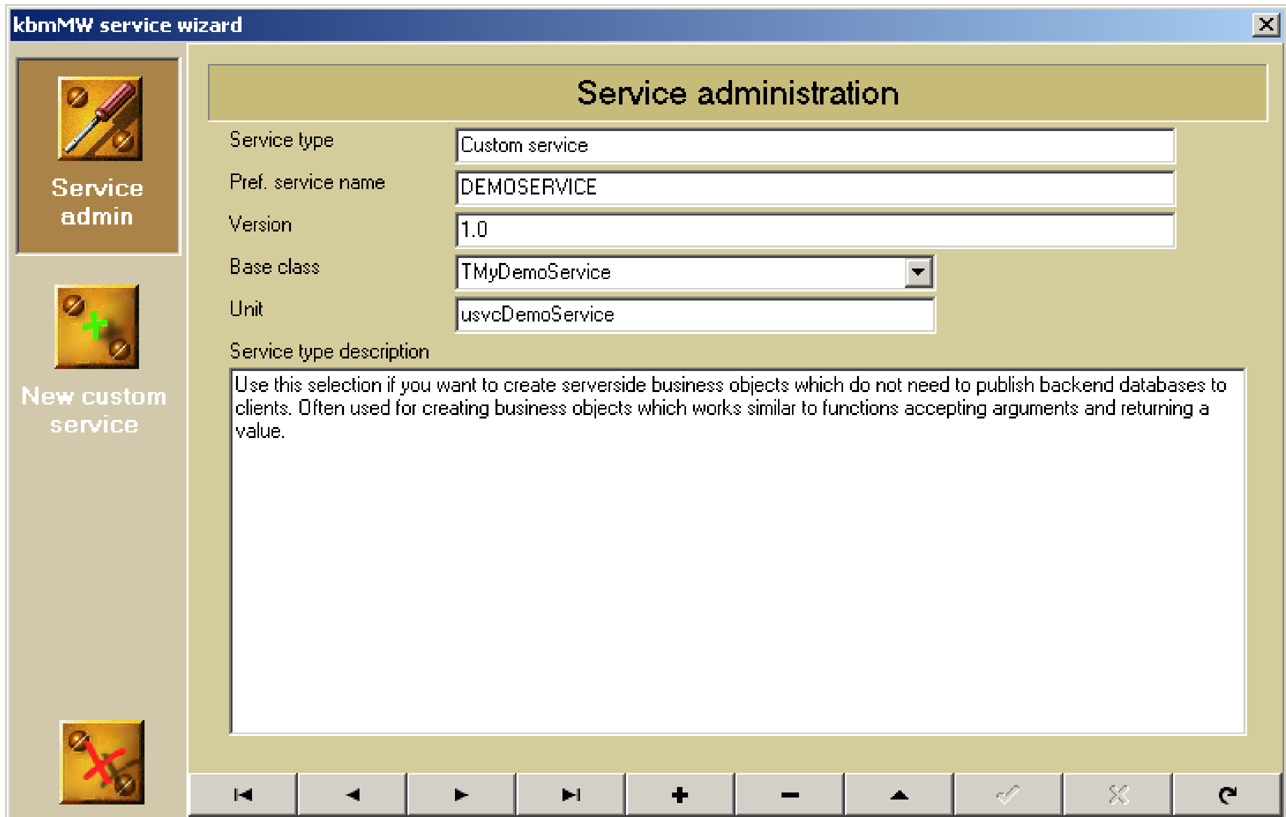
Copyright 2001-2006 Components4Developers
http://www.components4developers.com

You will find the newly added services in the bottom of the service list. They may not be named uniquely, but that can be done in the administration section of the wizard.

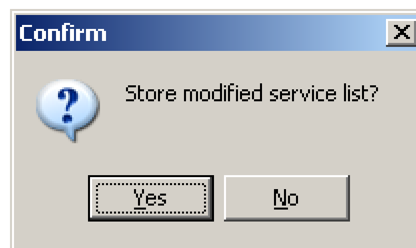


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

In the service administration, you can for example change the service type to something that better identifies what the service is. Eg. My Demo service. The description can also be updated if needed.



When you close the window, you will be asked if you want to save the modifications.



Select Yes and you are done.

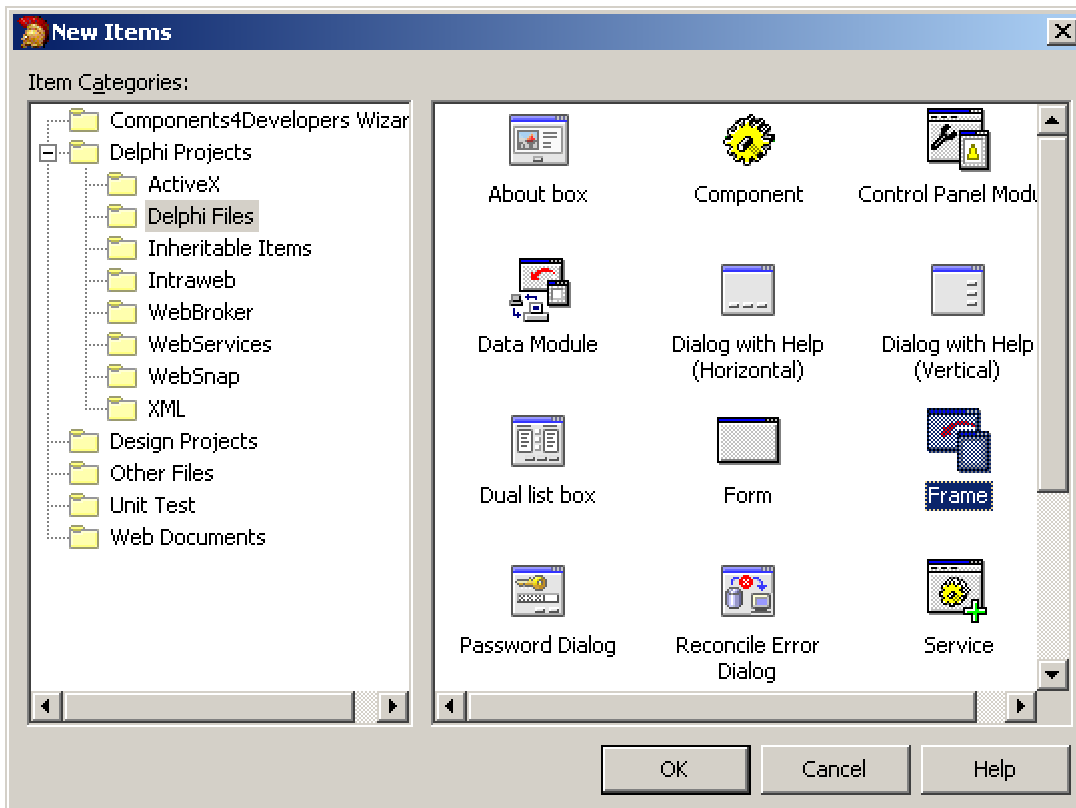
Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Advanced SWE's

Now that we know the basics of creating SWE's, we can start looking at more advanced things.

The query service, file service and Java service all have something in common… each of them have a special page in the service wizard in which the developer can choose how the service is going to be used, and which optionally results in components automatically being added to forms and datamodules.

This extra page is called a SWE form. In reality its not a TForm, but rather a TFrame.
A frame can be generated in Delphi by clicking File/New/Other/Delphi Projects/Delphi Files/Frame.



Lets say we want to let the developer put a checkmark in a checkbox to control if a non visual component, like a TkbmMemTable should automatically be added to the service upon creation by the wizard.

First we design the 'questionaire' for the developer.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

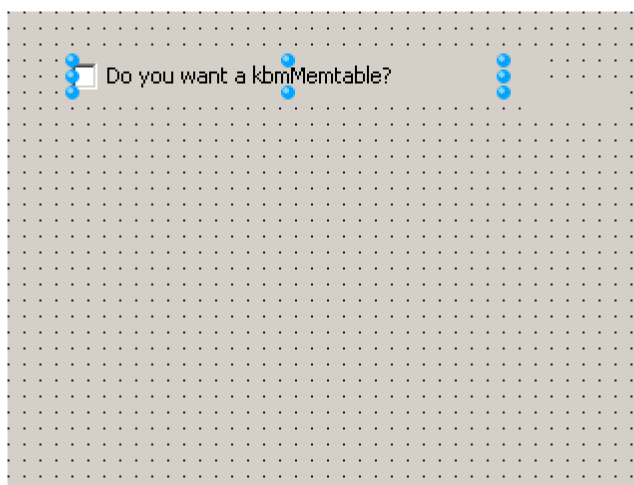
Open the SWE project from before, and add a new TFrame to it.
Set the frames name property at designtime to frDemoService.

Save the frame a DemoServiceSetup.pas.

(Please notice the naming conventions used as keeping the same style, makes it easier to understand for others later on)

Then add a TCheckBox and change its caption to ‘Do you want a kbmMemTable?’. Make the needed adjustments so its possible to see the caption in full etc.

Then you end up with something like this:



You can decorate the frame as you see fit.

Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Next step is to add some glue to the frame to make it accessible from the service wizard, and also to provide functionality for manipulating the generated service unit (and optionally other units).

This is done in code. Switch to seeing the source code for the frame.
The code needs a bit of altering. The following is a sample of code and includes explanations.

```
unit DemoServiceSetup;

interface
```

Add the include file again to get to some preprocessor declarations that we may need.

```
{$I kbmMW.inc}

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,
```

Remember to add kbmMWServiceSetup to the uses clause.

```
  kbmMWServiceSetup;
```

The standard TFrame declaration including our checkbox.

```
type
  TfrDemoService = class(TFrame)
    CheckBox1: TCheckBox;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Add the following service setup definition. It’s the workhorse in the SWE form management. GetFrame must be overriden, while its optional if to override the GenerateCode, GenerateInstructions and UpdateProject methods.

```
  TDemoServiceSetup = class(TkbmMWServiceSetup)
  public
     class function GetFrame:TkbmMWFrameClass; override;
     class function GenerateCode(const AFrame:TFrame; const FormIdent,AncestorIdent:string;
                                 const AType:TkbmMWComponentSourceType):string; override;
     class procedure GenerateInstructions(const AFrame:TFrame; const CPP:boolean;
                                          AStrings:TStrings); override;
     class procedure UpdateProject(const AFrame:TFrame); override;
  end;
```


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Optionally add the usual platform dependency code for reading the dfm/xfm/nfm form/frame definition file.

```
implementation

{$ifndef LINUX}
 {$IFDEF DOTNET}
  {$R *.nfm}
 {$ELSE}
  {$R *.dfm}
 {$ENDIF}
{$else}
 {$R *.xfm}
{$endif}
```

We need to refer to the kbmMWServiceWizard.

```
uses
   kbmMWServiceWizard;
```

The following is the implementation of the TDemoServiceSetup class.
The first method to implement is GetFrame. It should return the class of the frame to show in the service wizard when this service type has been selected.

```
// TDemoServiceSetup

class function TDemoServiceSetup.GetFrame:TkbmMWFrameClass;
begin
     Result:=TfrDemoService;
end;
```

The GenerateCode implementation is responsible for generating relevant code or text for different times in the service creation process. Depending on the IDE (Delphi vs C++) and the immediate situation, the method is called with different AType values.

```
class function TDemoServiceSetup.GenerateCode(const AFrame:TFrame; const
FormIdent,AncestorIdent:string; const AType:TkbmMWComponentSourceType):string;
begin
```

Here we can check for if the developed has checked our checkbox. If so, we want the wizard automatically to add a kbmMemTable to the generated service module.

```
     if TfrDemoService(AFrame).CheckBox1.Checked then
     begin
          case AType of
```

The following clause controls what should be added to a Pascal/Delphi uses clause if anything.

```
             mwcstPascalUses:
                begin
                     Result:='kbmMemTable';
                end;
```


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

This clause controls what to add to the include section of a C++ application.

```
              mwcstCPPInclude:
                 begin
                      Result:='#include "kbmMemTable.hpp"'+LF;
                 end;
```

This controls what should be added to the service datamodule file when its being generated. In the sample we add a kbmMemTable placed at X,Y (48,16)

```
              mwcstFormComponents:
                 begin
                      Result:=
                        '  object kbmMemTable1: TkbmMemTable'+LF+
                        '    Left = 48'+LF+
                        '    Top = 16'+LF+
                        '  end'+LF;
                 end;
```

And this section controls what to place in the datamodule implementation section.

```
              mwcstPascalImpl:
                 begin
                      Result:=
                        '    kbmMemTable1: TkbmMemTable;'+LF;
                 end;
           end;
     end
```

Finally if the developed unchecked the checkbox, no extra kbmMemTable instance should be added, so in all situations we simply return an empty string.

```
     else
         Result:='';
end;
```

The next method can optionally be overriden to add instructions to the developer. It could be telling him that he has to remember setting certain properties etc. Any text goes here.
The instructions are automatically added to the top of the generated service.

```
class procedure TDemoServiceSetup.GenerateInstructions(const AFrame:TFrame; const CPP:boolean;
AStrings:TStrings);
begin
     AStrings.Add('No special instructions for this DemoService');
end;
```

The next method can optionally be overriden to make other types modifications to the project, like adding components to other units than the generated service etc. For some examples of this, see the file service setup file.

```
class procedure TDemoServiceSetup.UpdateProject(const AFrame:TFrame);
begin
     // Here additional updates to the project can be made.
     // It for example includes adding components to other forms like
     // its done in the query service setup and the file service setup.
end;

end.
```


Copyright 2001-2006 Components4Developers
http://www.components4developers.com

Next compile the package again, and reinstall it.
You may need to shut down Delphi and reopen it, to make Delphi completely recognize the changes compared to our simpler SWE.

Now when we create a new service based on our demo service, we will be asked if we want to have a kbmMemTable placed on the new service instance or no.
Depending on the status of the checkbox, a kbmMemTable will be added automatically.

This concludes the whitepaper about Service Wizard Extensions.

Kim Madsen
Components4Developers

Copyright 2001-2006 Components4Developers
http://www.components4developers.com