C++Builder 4

Simple CORBA Servers in C++Builder 4.0

This article describes how to create the simplest possible CORBA client and server using Borland C++Builder 4.0. I also take a few brief moments to point out some very powerful new techniques for debugging multitier applications.

The CORBA tools described in this article only come with BCB 4.0 Enterprise edition, so you should have that high end version of the product if you want to follow along.

Source for Example Program

CORBA Setup

Before you get started, you want to make sure that CORBA is set up properly on your system.

The first thing you need to do is make sure that VisiBroker is installed on your system. If it is available, then you should be able to find it on your Start menu, under the Programs section. VisiBroker is usually installed in the C:\Inprise\VBroker directory. Other likely directories include the C:\VBroker directory and the C:\Program Files\VBroker directory. Of course the drive where VisiBroker is installed my vary.

You may have a VBroker directory on your system that contains nothing but a log directory. This only proves that VisiBroker was once installed on your system. To be sure you have the full install, look for a VBroker/Bin directory and a VBroker/Include directory.

If you are on Windows NT, there is another way to check if VisiBroker is on your system set up correctly. Go to the Control Panel, bring up the Services applet, and find the VisiBroker services listed near the bottom of the list of services. Make sure the Smart Agent is started. If it is not started, then use the menus to start it, and set things up so that it will be started automatically each time you reboot your system.

If you can't find VisiBroker on your system, then you should place the C++Builder Enterprise CD in your CDROM drive. Bring up the Windows Explorer, and search on the CD for the VBroker directory, where you will find the Setup.exe program that installs VisiBroker. The install is very straight forward, and requires little explanation. If you are install on Windows NT, make sure you set up the Smart Agent as a service.

On Windows 95/98, you can't run VisiBroker as a service, so you should go to the VBroker/Bin directory, and start OSAgent.exe manually. This process also works in Windows NT, but it really is more sensible to run the OSAgent as a service if that option is available.

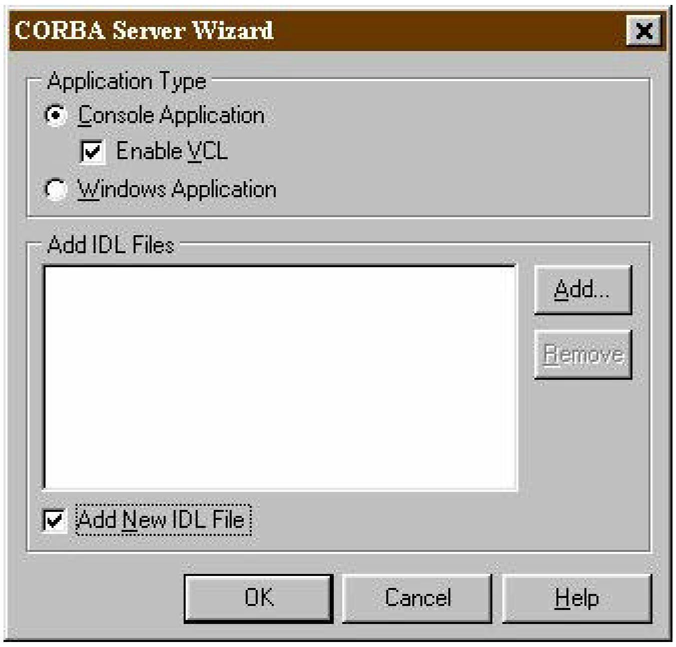
Inside the BCB4 IDE there is an option on the tools menu that allows you to start the OSAgent automatically. Selecting this option does the same thing as loading the OSagent from the VBroker/Bin directory.

To sum up, there are two things you need to do to make sure VisiBroker is properly installed on your system. First locate the VBroker directory on your hard drive. Second, make sure OSAgent is running. Remember that you need to install the version of VisiBroker that came with C++Builder. Other verions of the product may not work correctly with C++Builder.

Creating the Server

Now that you VisiBroker installed on your system, the next step is to create a client and server. To get started creating the server you should first launch the BCB4 IDE. Close any current projects you might be working on.

Choose File | New from the menu, turn to the Multitier page, and select the icon that lets you create a CORBA server. The CORBA Server Wizard dialog will then be automatically launched. Choose to create a Console Application, enable the VCL, and select the Add New IDL file button, as shown in Figure 1. Click on the OK button to finish the process.


CORBA Server Wizard

Application Type
[● Console Application]
[✓ Enable VCL]
○ Windows Application

Add IDL Files
[✓ Add New IDL File]

Figure 1: The CORBA Server Wizard.

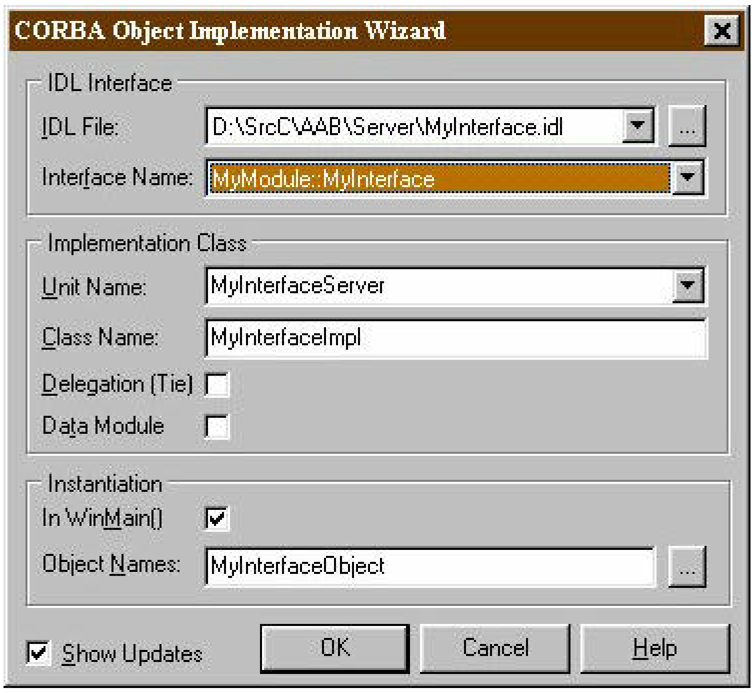
It is now time to save your work. Create a directory to house your project. Inside this directory create two directories, one called Client and the other called Server. Save your server in the Server directory under the name TempServer.dpr.

There should be a file in your project called File1.idl. Enter the following text in that file, and save the file under the name MyInterface.idl:

```
module MyModule
{
  interface MyInterface
  {
    wstring GetName();
  };
};
```

Select View | Toolbars form the menu and make sure the CORBA toolbar is visible. If it is visible, you will see three icons on the toolbar. The one on the far left shows a picture of a computer with a lightening bolt above it. The hint for this icon is New CORBA Object Implementation. When you select this icon MyInterface.idl will be compiled, and the CORBA Object Implementation Wizard will pop up.

Drop down the Interface Name combo box in the Implemenation Wizard and choose MyModule::MyInterface as the interface you want to implement. After you select this option, the wizard should look as it does in Figure 2. Be sure that the Show Updates button is selected.


CORBA Object Implementation Wizard

IDL Interface

IDL File: D:\SrcC\AAB\Server\MyInterface.idl

Interface Name: MyModule::MyInterface

Implementation Class

Unit Name: MyInterfaceServer

Class Name: MyInterfaceImpl

Delegation (Tie) ☐

Data Module ☐

Instantiation

In WinMain() ☑

Object Names: MyInterfaceObject

☑ Show Updates

Figure 2: Filling in the CORBA Object Implementation Wizard.

Click on the OK button on the Implementation Wizard. Because you selected Show Updates, the Project Updates dialog should appear. You can accept all the default options in the Project Updates dialog.

After you click OK in the Project Updates dialog, a file called MyInterface.cpp should be created, with the following source in it:

```
#include <vcl.h>
#pragma hdrstop

#include <corba.h>
#include "MyInterfaceServer.h"

#pragma package(smart_init)

MyInterfaceImpl::MyInterfaceImpl(const char *object_name):
  _sk_MyModule::_sk_MyInterface(object_name)
{
}

CORBA::WChar* MyInterfaceImpl::GetName()
{
  return WideString("MyInterface").Detach();
}
```

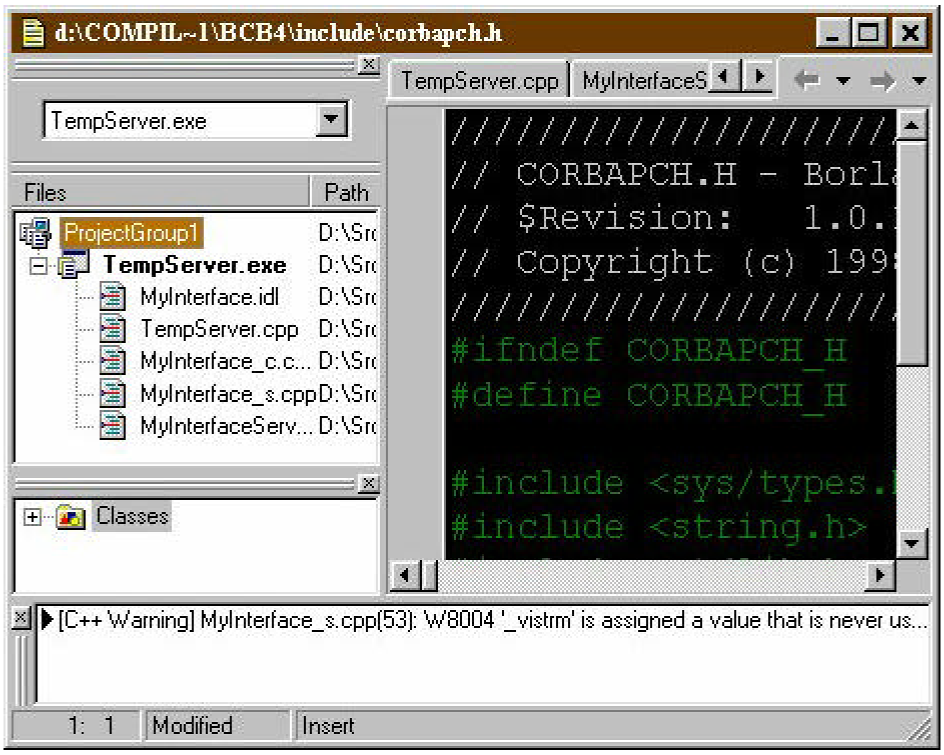
Note that I have filled in the GetName method so that it will return a simple string. I did not have to create the method itself. That was done for me automatically by the sytem. All I did was add the single line of code that returns a string.

The single line of code in the GetName method uses the VCL WideString class. I call the Detach method of the class to free up the memory for the WideString after the method is called. You should note that it was my desire to use the WideString class that led me to include the VCL in this project. If I did not use the WideString class, then I would not have had to include the VCL in this project. Choosing not to use or not use the VCL may be a significant issue if you are trying to generate code that will run on non-Windows systems.

At this stage you have finished creating your server, and you should be able to compile and link it by selecting Make from the BCB menus. The only time I ever had trouble with this process was when I was trying to compile a CORBA project without first installing VisiBroker. In that case, when I tried to compile a file called corbapch.h popped up and the IDE told me that it could not find the file vdef.h. Needless to say, vdef.h is kept in the VBroker/Include directory, and your compiler won't be able to find it if you have not installed the VisiBroker!

Creating the Client

Before creating the client, I find it wise to select View | Project Manager from the menu. Now dock the Project Manger in the upper left hand corner of the editor, as shown in Figure 3.


d:\COMPIL~1\BCB4\include\corbapch.h

TempServer.exe

Files	Path
ProjectGroup1	D:\Src
TempServer.exe	D:\Src
MyInterface.idl	D:\Src
TempServer.cpp	D:\Src
MyInterface_c.c...	D:\Src
MyInterface_s.cpp	D:\Src
MyInterfaceServ...	D:\Src

Classes

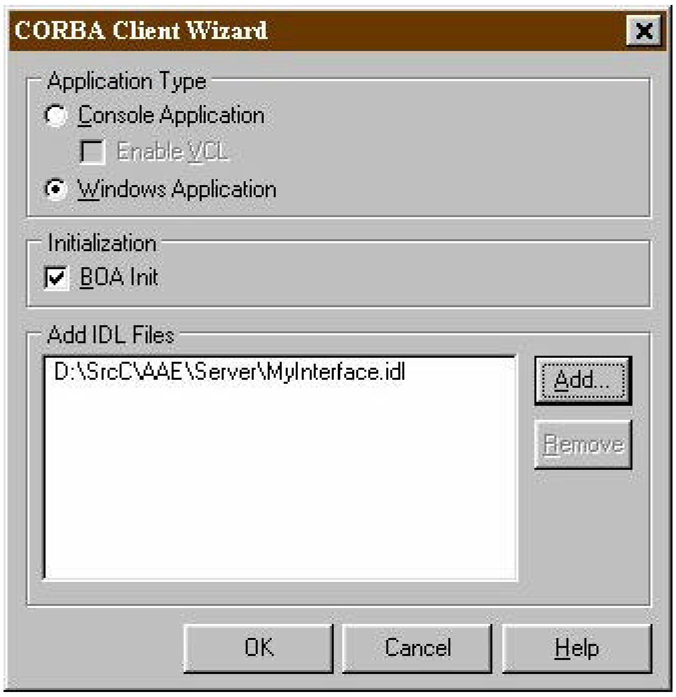
```
/////////////////////
// CORBAPCH.H - Borl
// $Revision:   1.0.
// Copyright (c) 199
/////////////////////
#ifndef CORBAPCH_H
#define CORBAPCH_H

#include <sys/types.
#include <string.h>
```

[C++ Warning] MyInterface_s.cpp(53): W8004 '_vistrm' is assigned a value that is never us...

Figure 3: Docking the Project Manager in the upper left hand corner of the Editor.

Click the New button in the Project Manager to create a new application. Turn to the Multitier page and select CORBA client. The CORBA Client Wizard will appear, and you should choose to create a Windows application and to initialize the BOA. Click the Add button and add MyInterface.idl to your application, as shown in Figure 4. Click the OK button and the source for your application will be generated automatically.


CORBA Client Wizard

Application Type

Console Application

Enable VCL

[✓ Windows Application]

Initialization

[✓ BOA Init]

Add IDL Files

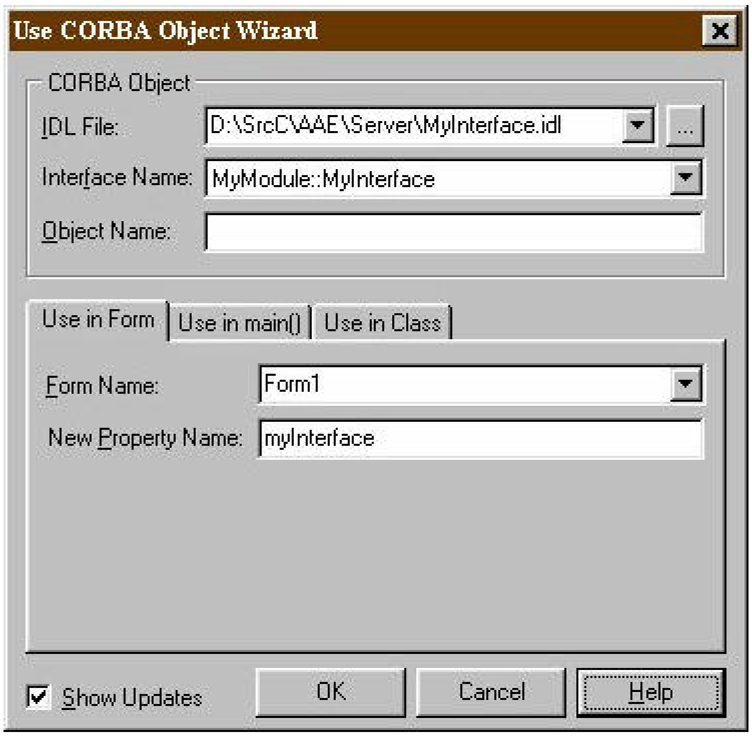
D:\SrcC\AAE\Server\MyInterface.idl

Figure 4: Add MyInterface.idl from the server directory to your CORBA Client Wizard.

Save your client application into the Client directory you created earlier. The main form should be saved as MainClient.cpp and the project itself should be saved as TempClient.dpr. Save the BPG file into the directory beneath the Client and Server directory under the name TempCORBAProj.bpg. (Obviously you can call these files whatever you want, I'm just giving you suggested names so that our work will match up exactly, thereby making it easier for you to check the steps while creating your first CORBA project.)

At the far right of the CORBA toolbar is an icon with the hint "Use CORBA Object" associated with it. Select this icon, or choose the "Use CORBA Object" menu item from the Edit menu.

The Use CORBA Object Wizard appears, as shown in Figure 5. Drop down the Inteface Name combo box and select MyModule::MyInterface. Leave the Object Name blank! Leave the Use in Form option so that it points at Form1. This will generate code in Form1 to create an instance of your server. Make sure Show Updates is selected, and then click the OK button.


Use CORBA Object Wizard

CORBA Object

IDL File: D:\SrcC\AAE\Server\MyInterface.idl

Interface Name: MyModule::MyInterface

Object Name:

Use in Form | Use in main() | Use in Class

Form Name: Form1

New Property Name: myInterface

☑ Show Updates

Figure 5: The Use CORBA Object Wizard as it appears after it has been correctly filled out.

Since you choose the Show Updates option, the Show Updates dialog will appear. You need make no changes in it, and can simply click the OK button.

If you turn to your main form, you will find that there are two methods in it designed to automatically create an instance of your server. The declaration for the form in MainClient.h looks like this:

```
class TForm1 : public TForm
{
__published:    // IDE-managed Components
private:        // User declarations
  MyModule::MyInterface_ptr __fastcall GetmyInterface();
  MyModule::MyInterface_var FmyInterface;
  void __fastcall SetmyInterface(MyModule::MyInterface_ptr _ptr);
public:         // User declarations
  __fastcall TForm1(TComponent* Owner);
  __property MyModule::MyInterface_ptr
    myInterface = {read=GetmyInterface, write=SetmyInterface};
};
```

Notice that there is a private variable called FmyInterface declared to point to your object on the server. There are also two methods called GetmyInterface and SetmyInterface which are implemented in your cpp file as follows:

```
MyModule::MyInterface_ptr
  __fastcall TForm1::GetmyInterface(void)
{
  if (FmyInterface == NULL)
  {
     FmyInterface = MyModule::MyInterface::_bind();
  }
  return FmyInterface;
}

void __fastcall TForm1::SetmyInterface(
  MyModule::MyInterface_ptr _ptr)
{
  FmyInterface = _ptr;
}
```

The key method here is GetmyInterface, which checks to see if FmyInterface is set to NULL. If it is, then an instance of the interface is created by calling the CORBA bind method.

Finally, you should look back at the declaration for TForm1 in the header, and notice that there is a property called myInterface. If you access this property, then GetmyInterface will be called automatically, thereby guaranteeing that you will create an instance of your object on the server. In short, all you have to do is reverence the myInterface property in order to access your server.

Given the declarations and implementations BCB has automatically inserted into your client, it becomes a trivial matter to call a method on your server. To get started, drop down a button on the main form, and create the following method:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ShowMessage(myInterface->GetName());
}
```

Note that this code references myInterface. As you have learned, this reference is all you need do to access your server. In other words, this single line of code both connects you to the server and calls the GetName method.

At this stage you can save your work, compile your application, and rest on your laurals. You have finished creating a simple CORBA client and server!

To test your work, select Project | Make all Projects from the BCB menu. This will ensure that both your client and your server are built. Go to the tools menu and make sure the VisiBroker Smart Agent is running.

On Windows NT, you can now run first your server, and then your client, both from within BCB. When your client is running, click on the button you placed on the main form, and confirm that you can call your server.

On Windows 95/98, you can't run two applications at the same time from the IDE. As a result, you should go to the Windows Explorer or the command prompt and start your server. Your server may not run properly if both the VBroker/Bin directory and the BCB4/Bin directory are not on your path. If this is the case, then you will need to set up your system properly before proceeding. Once you have the server running, come back into the IDE, and start the client. Click on the button on your client's form to test the call to your server.

Debugging Multitier Applications

On Windows NT, where you can run both the server and the client at the same time, you should be able to use the debugger to step from the client into the server project. To do this, set a breakpoint on the call to myInterface->GetName(). Run your application, click on the button, and you should be taken to the line where you set your breakpoint. Now press F7 to step into the code on your server. If you cannot step right through, try placing a breakpoint in your server on the implementation of your GetName method. Now trying stepping through again with debugger. If you get the debugger working correctly, and I have always been able to get it working correctly on my system, then you seeing something that is way cool. This kind of debugger technology is very, very nice!

Summary

In this relatively short and simple paper you have seen how to get up and running with

CORBA in BCB4. Hopefully you have found the process relatively simple and straightforward. In my opinion the team has done a beautiful job with this technology, and they efforts should help you get up to speed with this technology very quickly. In particular, the debugger engineers have done a beautiful job adding support for debugging multitier applications.

Back To Top
Home Page