# OrbixCOMet Desktop Getting Started

**IONA Technologies PLC**
**April 1999**

**Orbix is a Registered Trademark of IONA Technologies PLC.**
**OrbixCOMet (TM) is a Trademark of IONA Technologies PLC.**

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Microsoft, Windows, Windows NT and Windows 95 are either trademarks or registered trademarks of Microsoft Corporation.

---

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 1998, 1999 by IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

**M 2 2 9 0**

# Contents

# Introduction

This book describes the basics for getting started on OrbixCOMet. OrbixCOMet combines the best of both the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) and Microsoft DCOM standards. It provides a high performance bi-directional dynamic bridge that enables two-way integration between DCOM/Automation and CORBA applications.

OrbixCOMet is designed to allow COM developers—who use tools like Visual C++, Visual Basic, PowerBuilder, Delphi or Active Server Pages on the Windows desktop—to access CORBA applications running on Windows, UNIX or OS/390 (formerly MVS) environments. It means COM developers can use the tools familiar to them to build heterogeneous systems that use both COM and CORBA components within a COM environment.

## Audience

This book is intended for use by application programmers who wish to get started in using OrbixCOMet to develop Orbix applications on the Windows desktop environment. After you have read this book, you can refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for more details.

## Requirements

OrbixCOMet requires Windows NT 4.0 with SP4. If you are installing OrbixCOMet on a machine with an existing Orbix installation that includes the OLE support module, OLE support will be disabled by the OrbixCOMet setup program. Alternatively, you can do this by typing the following command in the *%ORBIX%*\bin directory:

```
oleregit IOLEM23C.DLL /UNREGSERVER
```

**Note:** In this book, *%ORBIX%* represents your Orbix installation directory, and *%ORBIXCOMET%* represents your OrbixCOMet installation directory.

Refer to "DCOM Trouble-Shooting" on page 19 before you start any development. It contains important information about verifying that DCOM is correctly set up on your machine. You should also refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for information about topics such as IDL/Automation and IDL/COM mapping, implementing clients, error handling and callbacks. You should also read the *OrbixCOMet Desktop Release Notes* before you start any development.

# Installing OrbixCOMet

To install OrbixCOMet from CD-ROM:

1. From the `\NT` directory of the CD-ROM, run `autorunc.exe`.

   Depending on the CD-ROM you are using, you might first be presented with a screen containing a list of products that you can install. If so, select the **Install OrbixCOMet** button.
2. Follow the on-screen instructions.

During the installation you are asked for a valid OrbixCOMet licence code that should be included in your product package. If you supply an invalid code, the installation defaults to a 21-day evaluation licence. You can contact `shipping@iona.com` if you need to obtain a new licence code.

You are presented with the following list of components that can be installed:

| | |
|---|---|
| Run-time Files | These are the binaries that constitute the bridge. |
| Documents | This includes on-line technical documentation. |
| Development Utilities | This consists of command line tools and the type store GUI tool. |
| Demonstrations | These are sample programs that show how to use the bridge. |
| Orbix Files | This includes the Orbix 3.0c runtime. |
| DCOM test application | This validates that DCOM is correctly installed. |

You should only select the "Orbix Files" component if Orbix 3.0c is not already installed on the machine. If the patch level of an installed version of Orbix 3.0 differs from the patch level of the Orbix 3.0 runtime that is supplied with OrbixCOMet, refer to the *OrbixCOMet Desktop Release Notes* for further information regarding installation.

For details about OrbixCOMet configuration, refer to the *OrbixCOMet Desktop Programmer's Guide and Reference*.

# Using OrbixCOMet

Unlike IONA's previous offering of a static OLE Automation/CORBA bridge, OrbixCOMet is a fully dynamic bridge. This means there is no special code generation step with OrbixCOMet. Instead of generating a broker, you are only required to supply type information. In short, using OrbixCOMet simply involves configuring the bridge to pick up the correct type information for each interface/complex type that your applications use.

## Server-Side Requirements

OrbixCOMet requires no code changes to existing CORBA servers. You can simply register the server executable by using either the `putit` command or the server manager utility `srvmgr`. The following is an example of using `putit`:

```
[c:\]putit grid %ORBIXCOMET%\demos\corbasrv\grid\server.exe
```

At this point, you should ensure that the Interface Repository (and the Naming Service if you wish to use it from your application) is also registered.

## Registering Type Information

Because OrbixCOMet is a purely dynamic DCOM/CORBA bridge, it is driven by type information, either from COM type libraries or from a CORBA Interface Repository. The example provided in this book uses the Interface Repository. Register your OMG IDL file using the following command:

```
[c:\] putidl %ORBIXCOMET%\demos\corbasrv\grid\grid.idl
```

This launches the Interface Repository if it is not already running.

**Note:** This example assumes you are using Orbix as your server-side object request broker (ORB). Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for details about using other ORBs on your server side.

# Application Programming

You can develop your client-server applications in any language that supports COM or Automation. The examples shown in the following subsections use PowerBuilder and Visual Basic.

## Writing a Client Using PowerBuilder

This section describes the development of a simple client-server application using OrbixCOMet. The example can be found in:

*%ORBIXCOMET%*\demos\PB6\grid

Filenames mentioned in this section refer to files contained in this directory.

### Global Data

Start by declaring the following global data:

```
OleObject bridge
OleObject fact
OleObject grid_client
```

### Connecting to the Orbix Grid Server from PowerBuilder

The following code is executed when you click the **Connect** button on the GUI screen for the grid demonstration:

```
// Powerscript
// create the CORBA factory object
fact = CREATE OleObject
```

```
//DCOM on the wire - see OrbixCOMet Desktop
//                  Programmer's Guide and Reference
//bridge = CREATE OleObject
//bridge.ConnectToNewObject("IT_CCIExWrap.IT_CCIExWrap.1")
//fact = bridge.IT_CreateRemoteFactory(server_name.Text)

// IIOP on the wire (requires bridge on client machine)
// the CORBA.Factory object may be created in the normal
// fashion
fact.ConnectToNewObject("CORBA.Factory")

// Exception parameter in case a CORBA exception occurs
OleObject ex
ex = CREATE OleObject

grid_client = CREATE OleObject
grid_client = fact.GetObject("grid:grid:" + server_name.Text,
                             BYREF ex)

height_val.Text = string( grid_client.Height )
width_val.Text = string( grid_client.Width )

connect_button.Enabled = False
unplug_button.Enabled = True
set_button.Enabled = True
get_button.Enabled = True
```

The preceding code results in the creation of an instance of a `CORBA.Factory` object. After a `CORBA.Factory` object has been returned, a particular object is requested by calling the `GetObject()` method on the factory. (Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for the MIDL definition for `DICORBAFactory`, or examine `ItStdAuto.idl` in *%ORBIXCOMET%*`\idl`.)

## GetObject()

The OMG *COM/CORBA Interworking* document at `WWW.OMG.ORG` specifies that `GetObject()` should take a string as one parameter and return a pointer to the `IDispatch` interface on the created object. However, it does not specify the format for the string.

In OrbixCOMet, the formats for the parameter to `GetObject()` are as follows:

- Old format for backwards compatibility with the Orbix/ActiveX Integration product:

  `"broker.interface[[[:marker]:server]:host]"`

  (Broker is ignored.)

- COMet format:

  `"interface[[[:marker]:server]:host]"`

- Tagged format:

  `"interface:TAG:Tag data"`

  where `TAG` is one of the following:

  | | |
  |---|---|
  | `IOR` | The data is the hexadecimal string for the stringified `IOR`. For example:<br>`fact.GetObject("employee:IOR: 123456789......")` |
  | `NAME_SERVICE` | The data is the `NAME_SERVICE` compound name separated by "." For example:<br>`fact.GetObject("employee:NAME_SERVICE: IONA.employees.PD.Ronan")` |

- Simple Format:

  `"interface"`

  This assumes the server name is the same as the interface. It also assumes the Orbix locator is used to find the hostname. If there is no Orbix daemon running in the client machine, the configuration setting for `Comet.Config.COMET_DAEMON_HOST` in the OrbixCOMet configuration file should point at a machine where a daemon is running with its locator configured.

**Note:** If the interface has been scoped (for example, `"Module::Interface"`), the interface token would be `"Module/Interface"`.

### Disconnecting

When disconnecting, it is important to release all references to objects in the bridge, so that the process will terminate. In the `grid` demonstration, this action is performed by the following subroutine:

```
grid_client.DisconnectObject()
DESTROY grid_client
fact.DisconnectObject()
DESTROY fact
bridge.DisconnectObject()
DESTROY bridge
```

## Writing a Client Using Visual Basic

This section describes the development of a simple client-server application using OrbixCOMet. The example can be found in:

*%ORBIXCOMET%*\demos\VB6\grid

Filenames mentioned in this section refer to files contained in this directory.

### Global Data

Start by declaring the following global data:

```
Dim bridge As Object
Dim fact As Object
Dim gridDisp As Object
```

### Connecting to the Orbix Grid Server from Visual Basic

The following code is executed when you click the **Connect** button on the GUI screen for the `grid` demonstration:

```
‘ Visual Basic
Private Sub Connect_Click()

    ‘ DCOM on the wire - see later
    ‘ Set bridge =
    ‘ CreateObject("IT_CCIExWrap.IT_CCIExWrap.1")
    ‘ Set fact =
    ‘ bridge.IT_CreateRemoteFactory(bridgeHost.Text)
```

```
      ‘ IIOP on the wire
      Set fact = CreateObject(“CORBA.Factory”)
      Set gridDisp = fact.GetObject("grid:grid:" &
                                    server_name.Text)

      width_val.Caption = gridDisp.Width
      height_val.Caption = gridDisp.Height
      Command1.Enabled = False
      Command2.Enabled = True
      SetButton.Enabled = True
      GetButton.Enabled = True
End Sub
```

The preceding code results in the creation of an instance of a `CORBA.Factory` object. After a `CORBA.Factory` object has been returned, a particular object is requested by calling the `GetObject()` method on the factory. (Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for the MIDL definition for `DICORBAFactory`, or examine `ItStdAuto.idl` in the *%ORBIX_COMET%*`\idl` directory.) Finally, a particular object is requested by `GetObject()`.

## GetObject()

Refer to “GetObject()” on page 9 for more details.

## Disconnecting

When disconnecting, it is important to release all references to objects in the bridge, so that the process will terminate. In the `grid` demonstration, this action is performed by the following subroutine:

```
Private Sub Disconnect_Click()
   Set gridDisp = Nothing
   Set fact = Nothing
   Set bridge = Nothing
End Sub
```

### Running the Application

To run the application:

1. Run the client `gridClt.EXE`.
2. Specify the hostname in the appropriate single-line-edit and click **Connect**. This contacts the C++ grid server and obtains the width and height of the grid.
3. Enter `x` and `y` coordinates.
4. Click **Set** to modify values in the grid, or **Get** to obtain values from the grid.
5. Click **Disconnect** when you are finished.

## Using OrbixCOMet with Internet Explorer

The concepts of using OrbixCOMet with Internet Explorer are very similar to those for Visual Basic and PowerBuilder. For further details refer to the *OrbixCOMet Desktop Programmer's Guide and Reference*.

# Dual Interface Support

OrbixCOMet supports dual interfaces, allowing methods to be called directly though the `vtable` entries. (The `vtable` is a function table that contains entries that correspond to each operation defined in the interface.) The `vtable` allows a controller to perform early binding or late binding if required. To avail of this feature, a type library is necessary for the controller. OrbixCOMet provides a type library generation tool `ts2tlb.exe` ("`TypeStore2TLB`") that produces type libraries based on type information provided by the user. You can choose to run the `ts2tlb` tool either from the command line or from the GUI interface **OrbixCOMet tools** screen. (Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for more details about using development support tools.)

**Note:** You must register your OMG IDL with the Interface Repository before `ts2tlb` can produce a type library.

The usage string for ts2tlb is as follows:

```
Usage:
ts2tlb [ options ] <type name> [[<type name>] ...]
         -f : file name (defaults to <type name #1>.tlb)
         -l : library name (defaults to IT_Library_<type name #1>
         -p : prefix parameter names with "it_"
         -i : Pass a pointer to interface Foo as IDispatch*
              rather than DIFoo* - necessary for some controllers
         -v : Print this message

Tip : use tlibreg.exe to register your type library!!
```

For example, the following command would create a file "grid.tlb", library name "IT_grid", for the grid interface:

```
[c:\] ts2tlb -f grid.tlb -l IT_grid  grid
```

When viewed with oleview, it appears as follows:

```
[odl,...]
interface DIgrid : IDispatch {
        [id(0x00000001)]
        HRESULT _stdcall get(
                        [in] short n,
                        [in] short m,
                        [out, optional] VARIANT* excep_OBJ,
                        [out, retval] long* val);
        [id(0x00000002)]
        HRESULT _stdcall set(
                        [in] short n,
                        [in] short m,
                        [in] long value,
                        [out, optional] VARIANT* excep_OBJ);
        [id(0x00000003), propget]
        HRESULT _stdcall height([out, retval] short* val);
        [id(0x00000004), propget]
        HRESULT _stdcall width([out, retval] short* val);
    };
```

**Note:** All UUIDs are generated using the MD5 algorithm specified in the OMG *COM/CORBA Interworking* document at WWW.OMG.ORG.

In Visual Basic, having created a reference to the type library, it would be used as follows:

```
Dim custGrid As IT_grid.DIgrid
```

For more complicated interfaces (for example, those that pass user-defined types as parameters), `ts2tlb` will attempt to resolve all those types from the disk cache and/or the Interface Repository. It will fail to produce a type library if any of the types it looks for are not found.

Finally, if you wish to register your type library in the system registry, the `tlibreg.exe` utility is provided for this purpose. It can also be used to unregister a type library.

# COM Interface Support

In addition to providing Automation/CORBA support, OrbixCOMet also provides support for COM custom interfaces. It adheres to the standards for mapping CORBA data types to COM that are laid down in the OMG *COM/CORBA Interworking* document at `WWW.OMG.ORG`. This support is aimed primarily at C/C++ programmers writing COM clients who wish to make use of the full set of COM types, rather than being restricted to Automation-compatible types. COM interfaces are described in MIDL (a derivative of DCE IDL), which is then compiled to produce marshalling code for the interface. OrbixCOMet provides a tool called `ts2idl.exe` ("`TypeStore2IDL`") that produces MIDL based on type information provided by the user. You can choose to run the `ts2idl` tool either from the command line or from the GUI interface **OrbixCOMet tools** screen. (Refer to the *OrbixCOMet Desktop Programmer's Guide and Reference* for more details about using development support tools.)

**Note:** You must register your OMG IDL with the Interface Repository before `ts2idl` can produce valid MIDL.

The usage string for `ts2idl` is as follows:

```
Usage:
ts2idl [options] <type name> [[<type name>] ...]
Options:
      -b : Pass object references as type Object in OMG IDL
      -c : Don't connect to the IFR (e.g. if cache is fully primed)
      -r : Resolve referenced types
      -m : Generate MIDL <default>
      -p : Generate makefile for proxy/stub DLL
      -f : <filename>
      -v : Print this message

      Tip : use -p to generate a makefile for the marshalling DLL!!
```

For example, the following command would create a MIDL file "`grid.idl`" for the `grid` interface:

```
[c:\] ts2idl -f grid.idl grid
```

**Note:** For more complicated interfaces that employ user-defined types, you can use the `-r` switch to completely resolve these types and to produce MIDL for them also.

The generated MIDL is:

```
[object,...]
    interface Igrid : IUnknown
    {
        HRESULT get([in] short n,
                            [in] short m,
                            [out] long *val);
        HRESULT set([in] short n,
                            [in] short m,
                            [in] long value);
        HRESULT _get_height([out] short *val);
        HRESULT _get_width([out] short *val);
    };

#endif
```

The `-p` switch is a useful labour-saving device that generates a makefile for compiling the IDL file and producing a DCOM proxy/stub DLL.

## Using COM Interfaces with OrbixCOMet

Generating MIDL from OMG IDL is the first step in writing a COM client to contact a CORBA server.

Next, the MIDL must be compiled by the MIDL compiler. This produces C/C++ interface definitions to be used within the application, and a proxy/stub DLL to marshal the custom interface. This procedure is standard practice when writing COM applications. `Ts2idl.exe` provides a useful labour-saving `-p` switch that can produce a makefile for building the proxy/stub DLL.

For example, the following command produces a `grid.mk` file in addition to the `grid.idl` file shown earlier:

```
[c:\] ts2idl -p -f grid.idl grid
```

The `grid.mk` file contains information on how to build the DLL and also how to register it. (Visual C++ 6.0 is required in order to build this marshalling DLL.)

By this stage, you are ready to write your COM client code. The basic operation of the client will be to:

1. Create an instance of an object that implements `ICORBAFactory` (which is the COM version of the interface encountered earlier when writing clients in Visual Basic, PowerBuilder, and so on).
2. Call `GetObject()` to return an `IUnknown*`.
3. Call `QueryInterface` to obtain a pointer to the custom interface (`Igrid` in this example) and call the methods.

The following sections take each of these steps in turn and describe how to write a C++ COM client.

Refer to the *`%ORBIXCOMET%`*`\demos\com\grid` directory for a complete code listing.

### Step 1: Creating the CORBA Factory

You can obtain an `ICORBAFactory*` by using `CoCreateInstanceEx()` as normal. Once again, you have the option to load the bridge in-process, or have it launched as a local server. (It is also possible to launch it on a remote machine. This simple demonstration does not show this but it is simply a matter of passing a filled-in `COSERVERINFO` parameter to `CoCreateInstanceEx()`.) In this example,

the choice is made at runtime, depending on how the client was started. The CORBA server to be contacted is called `grid` and it is registered on the machine `advice.iona.com`. For example:

```
HRESULT         hr = NOERROR;
IUnknown       *pUnk = NULL;
ICORBAFactory  *pCORBAFact = NULL;
DWORD           ctx;
   // our custom interface
Igrid       *pIBasic = NULL;
MULTI_QI        mqi;

// Call to CoInitialize(), some error handling
// and so on omitted for clarity

memset (&mqi, 0x00, sizeof (MULTI_QI));
mqi.pIID = &IID_ICORBAFactory;

if(bOutOfProc)
   ctx = CLSCTX_LOCAL_SERVER;
else
   ctx = CLSCTX_INPROC_SERVER;
hr = CoCreateInstanceEx (IID_ICORBAFactory, NULL,
                         ctx, NULL, 1, &mqi);
CheckHRESULT("CoCreateInstanceEx()", hr, FALSE);
pCORBAFact = (ICORBAFactory*)mqi.pItf;
```

## Step 2: Calling GetObject()

The call to `GetObject()` looks similar to that used from Visual Basic:

```
hr = pCORBAFact->GetObject("grid:grid:advice.iona.com",&pUnk);
if(!CheckErrInfo(hr, pCORBAFact, IID_ICORBAFactory))
{
    pCORBAFact->Release();
    return;
}
pCORBAFact->Release();
```

**Note:** `CheckErrorInfo()` is a utility function used by the demonstrations to check the thread's `ErrorInfo` object after each call. This is useful for obtaining information about, for example, a CORBA system exception that was raised during the course of a call.

### Step 3: QueryInterface and Method Calls

Finally, you can obtain a pointer to the custom interface `Igrid` via a `QueryInterface()` and make calls to set or get values in the grid. For example:

```
short width, height;
Igrid *pIF= 0;
hr = pUnk->QueryInterface(IID_Igrid,
                                  (PPVOID)& pIF);
if(!CheckErrInfo(hr, pUnk, IID_Igrid))
{
    pUnk->Release();
    return;
}
hr = pIF->_get_width(&width);
CheckErrInfo(hr, pIF, IID_Igrid);
cout << "width is " << width << endl;
hr = pIF->_get_height(&height);
CheckErrInfo(hr, pIF, IID_Igrid);
cout << "height is " << height << endl;
pIF->Release();
```

# DCOM Trouble-Shooting

In *%ORBIXCOMET%*\dcomapp there are two directories called `testDll` and `testExe`. These are pure DCOM applications that are completely independent of Orbix and OrbixCOMet. Their purpose is to allow verification of a DCOM installation on a given machine. Because they are pure DCOM only, they remove one variable from the equation when trouble-shooting is in operation.

Each application has a simple server written using ATL, and an associated Visual Basic client.

## testExe

First consider the application in *%ORBIXCOMET%*\dcomapp\testExe.

The directory should look something like the following:

```
20/02/98  20:01          <DIR>     client
21/02/98  16:29          <DIR>     server
20/02/98  20:01          <DIR>     vbclient
```

The server directory contains an ATL server, the binary for which can be found is *%ORBIXCOMET%*\bin\IT_DcomApp.exe. (You can build the server from scratch in the server directory, if you so wish. The source is provided.) Register the server with the following command:

```
[c:\iona\comet\bin] IT_DcomApp /regserver
```

A simple Visual Basic client for the application can be found in the vbclient directory. When you run the client, if the window shown in Figure 1 appears on the screen, the test is successful. If, as is likely, you intend to use OrbixCOMet with clients and servers on different machines, you should run these tests between those machines.

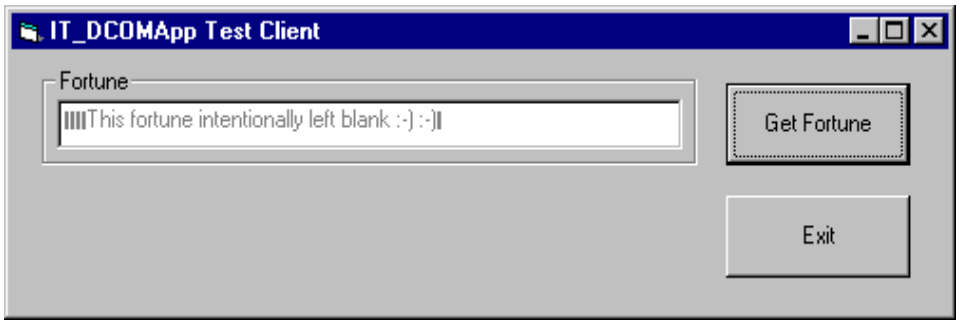


**Figure 1:** *IT_DCOMApp Test Client—Successful Operation*

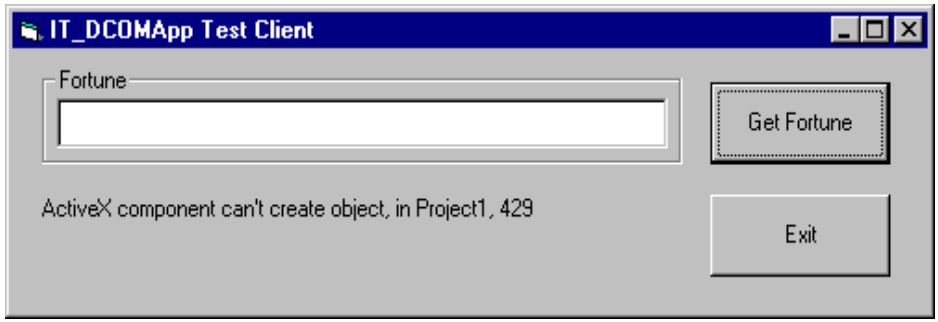


**Figure 2:** *IT_DCOMApp Test Client—Error Launching Server*

If the window shown in Figure 1 on page 20 does not appear, or if an error occurs as shown in Figure 2, refer to "Miscellaneous Configuration Tips" on page 22.

## testDll

The purpose of this application is to verify that surrogates work correctly on your machine. (You should test this if you want to use OrbixCOMet out-of-process.)

To do this:

1. Using `OLEVIEW`, launch the `IT_DcomTestDLL` class. This opens the **OLE/COM Object Viewer** screen.
2. From the **Object** pulldown menu, select `CoCreateInstance` flags of `CLXCTX_INPROC_SERVER`.
3. If this test fails, refer to "Miscellaneous Configuration Tips" on page 22.

## Miscellaneous Configuration Tips

1. Verify that the server is actually registered, using `OLEVIEW` if possible.
2. If `OLEVIEW` is available, try launching the application from within `OLEVIEW`, and specify `CoCreateInstance` flags of `CLSCTX_LOCAL_SERVER`.
3. If you are using the surrogate process, use `dcomcnfg` to ensure that the **Default Authentication Level** is set to `Connect` and the **Default Impersonation Level** is set to `Identify`.
4. On Windows NT, use the `\winnt35\system32\eventvwr.exe` event viewer to look for logged DCOM events. Figure 3 on page 23 shows a typical example of a logged error.
5. You can find the OrbixCOMet Knowledge Base at:

   `http://www.iona.com/support/kb/OrbixCOMet`
6. You can find the DCOM mailing list archive at:

   `http://microsoft.ease.lsoft.com/archives.index.html`
7. You can look up frequently asked questions about COM security at:

   `http://support.microsoft.com/support/kb/articles/q158/5/08.asp`

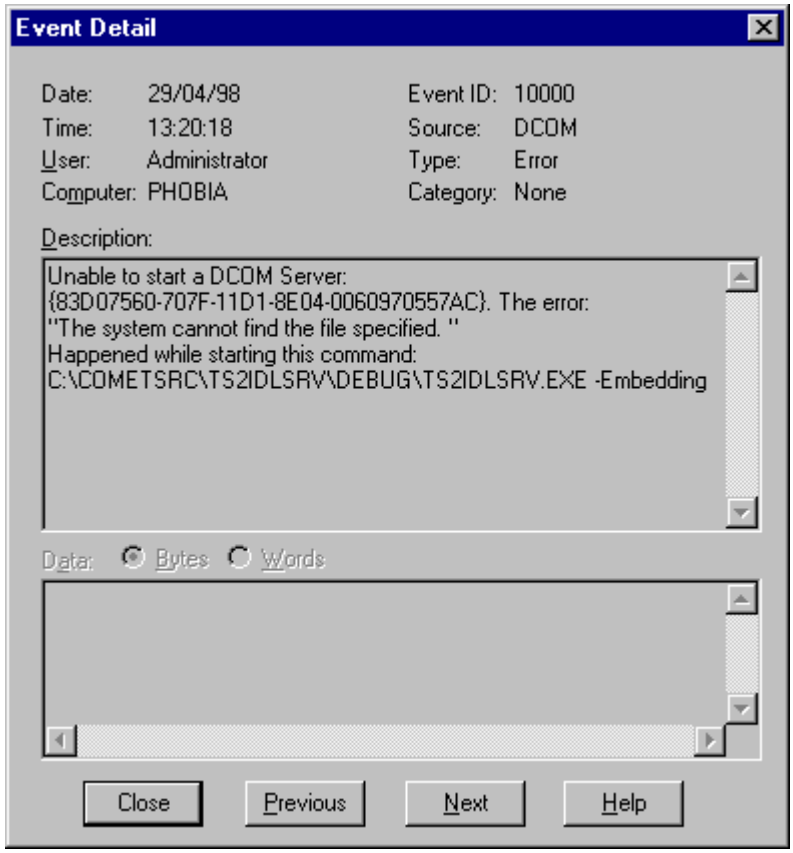


**Figure 3:** *Typical Example of a Logged Error*