Under Construction: CORBA Exceptions In Delphi

by Bob Swart

Two months ago, we examined VisiBroker for Delphi 5. We used a CORBA server written in Delphi 5 and showed three possible ways to connect a Delphi 5 CORBA client to it. This time, we'll focus on the use of CORBA exceptions for Delphi.

WebBroker And IE5

Last time, we ended with a WebBroker example to return the *HTTP/1.0 401 Unauthorized* headers, as can be seen in Listing 1.

This code worked well with Netscape Navigator, but shortly afterwards I found out that it didn't quite work in Internet Explorer. For some reason, Internet Explorer just showed the output, but never the login dialog. Strangely enough, the same approach using the plain CGI application worked just fine, so it must have been something specific to WebBroker applications.

Using IntraBob, my ISAPI debugger (which can be found on my website), I finally found that WebBroker applications don't use the `StatusCode` unless a `ReasonString` is specified too. This was not indicated in the help, so I missed it. Adding a statement to Listing 1 to include a `ReasonString` isn't too hard. However, the code still didn't appear to work in Internet Explorer. Careful examination of CGIApp.pas showed the code in Listing 2 inside the procedure `TCGIResponse.SendResponse`.

For some reason, the header field with the `200 OK` or the error code `401 Unauthorized` is not prefixed with `HTTP/1.0` but with `Status:`. And while this works fine with Netscape Navigator, it fails to function correctly inside Internet Explorer. So, I've changed the last line in Listing 2 to:

```
AddHeaderItem(StatusString,
  'HTTP/1.0 %s'#13#10);
```

Which solved my problem with Internet Explorer. Only one puzzle remains, and that's the fact that neither Netscape Navigator nor Internet Explorer are able to show me the name of the `Realm` in the login dialog. And we do pass this information in the `Response.Realm` property. The only problem is that for some reason the procedure `TCGIResponse.SendResponse` doesn't use the `Realm` property. In fact, I found no place at all where this property was used, so we could just as well have left the assignment out of Listing 1.

Obviously, this is yet another bug in WebBroker. In this case, we need to replace the single call to

```
AddHeaderItem(
  WWWAuthenticate,
  'WWW-Authenticate:%s'
  #13#10);
```

with the following two lines:

```
StatusString :=
  Format('WWW-Authenticate: '+
  ' %s realm="%s"'#13#10,
  ['%s', Realm]);
AddHeaderItem(WWWAuthenticate,
  StatusString);
```

The `Format` replaces the realm string, while the `AddHeaderItem` takes care of inserting the `WWWAuthenticate` string at the correct location.

This final change to the `CGIApp` unit resolves the unknown realm problem. Of course, similar modifications are needed in the `ISAPIApp` unit to ensure that the code for authentication/realm works fine for ISAPI DLLs too.

CORBA Exceptions

Back to the main topic for today: CORBA exceptions. In Delphi 4, and Delphi 5 without VisiBroker for Delphi, there was no support for CORBA-specific exceptions. Those of you who tried may have seen the 'catastrophic failure' message, which is about the only indication you get as a CORBA client that something went wrong inside the CORBA server (ie when the CORBA server raises an exception, we always only get a 'catastrophic failure' at the client, without any

```
procedure TWebModule1.WebModule1WebActionItem3Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var Auth: String;
begin
  Auth := Request.Authorization;
  if Pos('Basic ',Auth) = 1 then
    Delete(Auth,1,6);
  Auth := UnBase64(Auth);
  if Pos('bswart',Auth) = 0 then begin
    { any "bswart" may enter }
    Response.StatusCode := 401;
    Response.WWWAuthenticate := 'Basic';
    Response.Realm := '/DrBob';
    Response.SendResponse;
  end else begin
    Response.Content := 'Welcome: ['+Request.Authorization+']=['+Auth+'])'
  end
end;
```

➤ *Listing 1: WebBroker Authorization example.*

➤ *Listing 2*

```
if (ReasonString <> '') and (StatusCode > 0) then
  StatusString := Format('%d %s', [StatusCode, ReasonString])
else
  StatusString := '200 OK';
AddHeaderItem(StatusString, 'Status: %s'#13#10);
```

indication of what went wrong. This has been very frustrating (at least to me), when all that went wrong was an I/O exception or even an unhandled (by the CORBA server) `StrToInt` exception. Even apart from the fact that we don't get any additional information, we don't even get a slight indication what went wrong inside the CORBA server, so it's much harder to pinpoint the exact location.

Fortunately, as I wrote two months ago, VisiBroker for Delphi adds support for CORBA exceptions, although initially only for CORBA clients written in Delphi. We need to wait for a later version of VisiBroker for Delphi for the CORBA server support. In the meantime, however, we can safely communicate with 'foreign' CORBA servers that raise CORBA exceptions.

In Delphi, we use `try..except` to work with ObjectPascal exceptions, while in C++ and Java people use `throw..catch` to work with exceptions. When using CORBA IDL we talk about raising exceptions again, just like ObjectPascal. And although it's not possible to actually raise an exception in IDL (IDL stands for Interface *Definition* Language), we must use IDL to define exception types and specify which exceptions can be raised by a given method.

An IDL exception is a record-like type definition. We can add data members, but no methods. We also don't get inheritance, so you cannot set up an exception hierarchy, which is a shame since that can become quite handy for narrowing down specific error situations. For example, in BobNotes, we had two methods to get and set the `Lines` of a specified `User`. Apart from the `Lines` and `User`, we also needed to pass a `Password` argument. There are at least a number of things that can go wrong here. For example, we can define a `PermissionDenied` exception, which can be raised if either the `User` is unknown or the `Password` doesn't match the user's password. You might consider specifying two sub-exception types, `UserKnown` and `PasswordIncorrect`, (although even if you could sub-class IDL exceptions, I personally wouldn't use those because of the increased security risks involved when giving away too much detail). The IDL-way is to add data members (properties) that hold the specific information, like a `Reason` field of type `String`.

```
module BobNotes
{
  interface ICorBobNotes
  {
    void GetLines(in wstring User, in wstring Password, out wstring Lines);
    void SetLines(in wstring User, in wstring Password, in wstring Lines);
  };
  interface CorBobNotesFactory
  {
    ICorBobNotes CreateInstance(in string InstanceName);
  };
};
```

➤ *Listing 3: BOBNOTES.IDL (no exceptions).*

```
module BobNotes
{
  exception PermissionDenied
  {
    string Reason;
  };
  interface ICorBobNotes
  {
    void GetLines(in wstring User, in wstring Password, out wstring Lines)
      raises (PermissionDenied);
    void SetLines(in wstring User, in wstring Password, in wstring Lines)
      raises (PermissionDenied);
  };
    interface CorBobNotesFactory
  {
    ICorBobNotes CreateInstance(in string InstanceName);
  };
};
```

➤ *Listing 4: BOBNOTES.IDL (exceptions).*

Regardless of our use, the CORBA IDL definition of the above specified exception `PermissionDenied` type with a string field `Reason` is as follows:

```
exception PermissionDenied
{
  string Reason;
};
```

Now, let's consider the IDL file for BobNotes with the interface `ICorBobNotes` as defined earlier. As you can see in Listing 3, we have two methods inside the `ICorBobNotes` interface. CORBA forces us to specify the exceptions that can be raised by each of these methods, and in this case the exception `PermissionDenied` can be raised by both methods (see Listing 4).

Note that the exception `PermisionDenied` is defined outside of the `ICorBobNotes` interface in Listing 4. We could also have embedded the exception definition inside the interface, which would have made it a 'local' exception type, unable to be used by other interfaces defined in the same module.

VisiBroker Exceptions

VisiBroker already contains a number of pre-defined exceptions that are available to raise for every interface method. These exceptions have pre-defined ObjectPascal type definitions, derived from `SystemException` (which, in turn, is derived from `Exception`). Table 1 over the page shows a list of pre-defined exceptions, which are described in more detail in the documentation that comes with VisiBroker for Delphi 5 and the new CORBA.PAS unit.

Even more interesting, however, are the user-defined exceptions, which are derived from the ObjectPascal type `UserException` (which is also derived from `Exception`). This is the base class for the ObjectPascal equivalent of our `PermissionDenied` exception, which we shall see after we run the IDL2PAS utility on the new BOBNOTES.IDL file.

UNKNOWN	The unknown exception
BAD_PARAM	An invalid parameter has been passed
NO_MEMORY	Dynamic memory allocation has failed
IMP_LIMIT	Implementation limit violated
COMM_FAILURE	Communication failure
INV_OBJREF	Invalid object reference
NO_PERMISSION	No permission for this operation
INTERNAL	Internal ORB error
MARSHAL	Error while marshalling parameter (or result)
INITIALIZE	ORB initialization error
NO_IMPLEMENT	Implementation not available
BAD_TYPECODE	Bad TypeCode
BAD_OPERATION	Invalid operation
NO_RESOURCES	Insufficient resources for request
NO_RESPONSE	Response to request not (yet) available
PERSIST_STORE	Persistent storage failed
BAD_INV_ORDER	Routine invocations out of order
TRANSIENT	Transient failure
FREE_MEM	Unable to free memory
INV_IDENT	Invalid identifier syntax
INV_FLAG	Invalid flag specified
INTF_REPOS	Error accessing interface repository
BAD_CONTEXT	Error processing context object
OBJ_ADAPTER	Failure detected by Object Adapter
DATA_CONVERSION	Data conversion error
OBJECT_NOT_EXIST	Object does not exist

➤ *Table 1: CORBA System Exceptions.*

C++Builder Exceptions

Before we use IDL2PAS on the BOBNOTES.IDL file (to produce Client Stubs for a Delphi 5 CORBA client), we should first see if we can produce Server Skeletons for a CORBA server, written in any language except Delphi 5. About a year ago we saw JBuilder CORBA servers, so this time I'll use a C++Builder CORBA server. One warning: if you want to connect C++Builder CORBA servers to Delphi CORBA clients, make sure you stick with VisiBroker 3.3, included with C++Builder 4 Enterprise. C++Builder 5 Enterprise ships with VisiBroker 3.4, which is newer, but also breaks Delphi 5 CORBA code. After I installed C++Builder 5 Enterprise (and VisiBroker 3.4), I was no longer able to run any CORBA server or client written in Delphi 5 because of DLL import routine mismatches. A clear case of *DLL Hell*. Reinstalling VisiBroker 3.3 (and VisiBroker for Delphi 5 just to be sure) fixed things, so the following C++ code has been compiled using C++Builder 4 Enterprise. I haven't been able to get an official response from VisiGenic (previously called Inprise) on this, but I did hear similar reports from other Delphi/C++Builder 5 users, so I guess it's not merely an oversight on my part here.

Using IDL2CPP we can generate Server Skeletons for the IDL file, resulting in a `ICorBobNotesImpl` class with two important methods: `GetLines` and `SetLines`. To test the fact that we can raise a CORBA exception inside these methods, I just entered the following line inside each of them (doing nothing at all, except for raising the exception):

```
throw
  BobNotes::PermissionDenied(
  "Exception raised by BCB4");
```

The ORB will intercept the exception and pass it on to the CORBA client, which will turn it into an ObjectPascal native exception and raise it there. Note that this functionality was not available before VisiBroker for Delphi.

Delphi Exceptions

Using IDL2PAS we can 'compile' the BOBNOTES.IDL file to two new units: BobNotes_C.pas and BobNotes_I.pas. The first of these, BobNotes_C.pas, contains the Client Stub definitions, including the type definition of the `EPermissionDenied` exception (which suddenly has an E-prefix).

Note that if we had defined the `PermissionDenied` exception inside the `ICorBobNotes` interface definition (in the IDL file shown in Listing 4), then we would have had another type name for the exception here, namely one with the interface name followed by an underscore embedded as well. In our case, the `PermissionDenied` embedded in the `ICorBobnotes` interface would result in the `EICorBobNotes_PermissionDenied` ObjectPascal exception type. Note that the VisiBroker for Delphi documentation is slightly wrong on this topic, as it says that when an exception is defined within an

➤ *Listing 5 : EPermissionDenied.*

```
EPermissionDenied = class(UserException)
private
  FReason : AnsiString;
protected
  function  _get_Reason : AnsiString; virtual;
public
  property  Reason : AnsiString read _get_Reason;
  procedure Copy(const _Input : InputStream); override;
end;
```

```
unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TForm3 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
  public
  end;
var
  Form3: TForm3;
implementation
{$R *.DFM}
uses
  CORBA, OrbPas30, CorbaObj, BobNotes_I, BobNotes_C;
procedure TForm3.Button1Click(Sender: TObject);
var
  Factory: CorBobNotesFactory;
  Client: ICorBobNotes;
  Lines: WideString;
begin
  Factory := TCorBobNotesFactoryHelper.Bind('CorBobNotes');
  Client := Factory.CreateInstance('CorBobNotes');
  Client.GetLines('Bob','swart',Lines);
  Memo1.Lines.Add(Lines);
  Client := nil;
  Factory := nil
end;
procedure TForm3.Button2Click(Sender: TObject);
var
  Factory,Client: TAny;
  Lines: WideString;
  User,Pass: WideString;
begin
  Factory :=
    Orb.Bind('IDL:BobNotes/CorBobNotesFactory:1.0');
  Client := Factory.CreateInstance('CorBobNotes');
  User := 'Bob';
  Pass := 'swart';
  try
    Client.GetLines(User,Pass,Lines);
  except
    on E: EICorBobnotes_PermissionDenied do
      Memo1.Lines.Add(E.Reason)
  end;
  Memo1.Lines.Add(Lines);
  Client := unassigned;
  Factory := unassigned;
end;
end.
```

➤ *Listing 6: EICorBobNotes_PermissionDenied.*

interface, the ObjectPascal class name is prefixed with a leading underscore, the interface name, followed by another underscore, so in our case that would become `_ICorBobNotes_PermissionDenied` (obviously, the first underscore is in fact replaced by the letter `E`).

The code to handle the exception can be written using a regular ObjectPascal `try...except` block, as can be seen in Listing 6 of this article.

Note that Listing 6 was made using the embedded IDL exception definition (that is, `EICorBobnotes_PermissionDenied`). Also note that this is currently the only CORBA exception that I check for. In real life, however, one should also check for CORBA system exceptions, especially in the `Button1Click` method where a connection to the CORBA Server is made, and all kinds of possible errors (system exceptions) can occur.

Conclusions

VisiBroker for Delphi adds IDL2PAS for statically linked CORBA clients. It also adds the ability for records and exceptions, which finally brings the CORBA implementation in Delphi to a professional level. I still can't wait until the IDL2PAS for Delphi CORBA servers becomes available, probably first in Kylix and at a later date in Delphi 6 Enterprise.

Next Time

Next month, we will examine more enhancements made to MIDAS 3, the Multi-tier Distributed Application Services for multi-tier applications. We'll see *what, why, how* and more, including the new lower-than-ever deployment licence model. *So stay tuned...*

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is an @-Consultant for TAS Advanced Technologies, co-founder of the TAS-AT DOC Delphi Oplossings-Centrum (www.tas-at.com/doc), as well as a freelance technical author with numerous articles and some book sections to his name.