Lab 13.2: Building A COM Object Client Application

Objectives

After completing this lab, you will be able to:

- Import a COM server's type library.
- Create an interface pointer.
- Invoke methods of an interface using an interface pointer.
- Use the _bstr_t and BSTR data types to pass strings to and from an interface method
- Use structured exception handling to trap exceptions that arise in instantiating a COM object or invoking its methods.

Prerequisites

Completion of Lab 13.1:Building a COM Object Server.

Lab Setup

To see a demonstration for the solution to this lab, see Lab 13.1: Building a COM Object Server.

Estimated time to complete this lab: **25 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Converting a Program to Use a COM Server

In this exercise, you will convert an existing program to use an out-of-process (EXE) COM server.

If you have not done Lab 9.1 before starting this lab, use the 32-bit ODBC driver manager in Control Panel to add a data source using the Microsoft Access Driver. Name the data source PERSONAL. Use the database located in \Labs\C13\Personal.Mdb.

Copy the starting code for this lab from \Labs\C13\Lab02\Baseline. The completed code for these exercises is in \Labs\C13\Lab01\Xxx, where Xxx is the exercise number. If you are running on a Windows 95 system, you must install DCOM for Windows 95 from the Visual C++ installation disk before doing this lab.

Exercise 1: Converting a Program to Use a COM Server

The purpose of this lab is to convert the application developed in **Lab 9.1: Creating an ODBC Database Application** to use methods in a COM out-of-process server. Completion of Lab 9.1 is not a prerequisite for this lab because a modified version of the solution to Lab 9.1 is provided as a starting point for this lab.

Lab 9.1 is a database browser/editor built using ODBC. That lab has two custom DDX and DDV functions. Part of the code from those two custom functions was placed in an out-of-process COM server in Lab 13.1. In this lab, you will invoke the two methods of the object exposed by that server.

➢ **Load the baseline code for this exercise.**

1. Open the project contained in \Labs\C13\Lab2\Baseline.

➢ **Import the server's type library**

1. Open the file Stdafx.H.
2. Add the following statement after #include <afxwin.h>.

```
#import "..\..\lab01\ex01\modayrx.tlb"
```

➢ **Include the server's class identifiers file**

1. Add the following statement after the #import line shown on the previous step.

```
#include "..\..\lab01\ex01\modayrx_i.c"
```

Note In some projects, you must add code to your application object's **InitInstance** function to initialize the OLE libraries. In this application, the call to **AfxEnableControlContainer** initializes the OLE libraries.

➢ **Generate the TLH and TLI Files**

1. Build the project to generate the TLH and TLI files. The compiler automatically generates these two files as it processes the #import statement.

Note If you get warnings about unused variables (HRESULT), disregard them.

➢ **Add a namespace statement to the View class's header file**

1. From the File menu, choose Open. Locate the TLH file in your project's Debug directory and open it.
2. Find the name space symbol and copy it to the clipboard.
3. Open the file EmployeeView.H and add the words using namespace before the class declaration, then paste the symbol from the clipboard, and add a semicolon. This section of the file should now look like this:

```
using namespace MODAYRXLib;

class CEmployeeView : public CRecordView
```

➢ **Invoke MDYfromBSTR from the global function DDX_FieldText**

1. Open EmployeeView.Cpp.
2. Find the // BEGIN LAB and // END LAB comments in the function **DDX_FieldText.** All code added in this step will be placed between those two lines.
3. Create a _bstr_t object from the CString that contains the date

```
_bstr_t bstrDate = strDate.AllocSysString();
```

4. Initialize the TIMESTAMP_STRUCT object to 0:

```
memset(&date, 0, sizeof(date));
```

5. Add a try block. Inside the try block, create and initialize an interface pointer. Invoke the interface method MDYfromBSTR, passing as arguments the BSTR and pointers to the month, day and year members of the TIMESTAMP_STRUCT object. The try block should look like this:

```
try
{
    ImdyPtr pMDY = ImdyPtr(CLSID_mdy);
    hr = pMDY->MDYfromBSTR (bstrDate, (short *)&date.month,
                           (short *)&date.day, &date.year);
}
```

Note The interface pointer class can be found, without the *Ptr*, in the TLH file. The *CLSID* CLSID_mdy can be found in the Lab01\Ex01\Modayrx_i.c file that you included in an earlier step.

---

6. Add the following catch block. You can copy this from the solution located in \Labs\C13\Lab02\Ex01\EmployeeView.Cpp.

```
catch (_com_error &ex)
{
    _bstr_t bstrSource(ex.Source());
    _bstr_t bstrDescription(ex.Description());
    char szTemp[1024];
    CString csMsg = "Oops - hit an error!\n";
    wsprintf(szTemp, "Code = %08lx\n", ex.Error());
    csMsg += szTemp;
    wsprintf(szTemp, "Code meaning = %s\n", ex.ErrorMessage());
    csMsg += szTemp;
    wsprintf(szTemp, "Source = %S\n", (wchar_t*)bstrSource);
    csMsg += szTemp;
    wsprintf(szTemp, "Description = %S\n",
            (wchar_t*)bstrDescription);
    csMsg += szTemp;
    AfxMessageBox (csMsg);
    SysFreeString(bstrSource);
    SysFreeString(bstrDescription);
}
```

7. If the method returned S_OK, exit the function. Otherwise, display a message, and invoke the **CDataExchange::Fail** function as follows.

```
if (S_OK == hr)
    return;

AfxMessageBox( "Use date format:\nm/d/yyyy",
        MB_OK | MB_ICONEXCLAMATION );
pDX->Fail( );
```

➢ Invoke ValidateMDY from the global function DDV_DATE

1. Find the // BEGIN LAB and // END LAB comments in the function **DDV_DATE**. All code added in this step will be placed between those two lines.
2. Create a CString object 255 bytes long. The initializing character is an "x," but that is unimportant:

```
CString strError('x', 255);
```

3. Create a BSTR object from the CString:

```
BSTR bstrError = strError.AllocSysString();
```

4. Add a try block. Inside the try block, create and initialize an interface pointer. Then invoke the interface's method ValidateMDY, passing as arguments the BSTR by reference and the month, day and year members of the TIMESTAMP_STRUCT object. The try block should look like this:

```
try
{
    ImdyPtr pMDY = ImdyPtr(CLSID_mdy);
    hr = pMDY->ValidateMDY (&bstrError, date.month,
                            date.day, date.year);
}
```

5. Add the same catch block you used in Step 6 of the previous instruction.

6. If the method returned S_OK, free the BSTR object and exit the function as follows.

```
if (S_OK == hr)
{
    ::SysFreeString(bstrError);
    return;
}
```

7. Create a _bstr_t object from the BSTR object, display a message, invoke **CDataExchange::Fail**, and free the BSTR object.

```
_bstr_t bst(bstrError, false);
AfxMessageBox((char *)bst, MB_OK | MB_ICONEXCLAMATION );
pDX->Fail( );

::SysFreeString(bstrError);
```

Note CString and _bstr_t objects are correctly freed as they go out of scope.

➢ **Build the project**

1. Save the workspace
2. Build and run Employee.Exe.

The application has the same functionality as Employee.Exe from Lab 9.1, except that when you modify the employee's birth date and move to another record to make the change take effect, the code for DDX and DDV is run in another process.

If you have time, consider making the interface pointer used by these two functions into a global object, initializing it only once when it is first needed. This will greatly increase the speed of the subsequent uses of the out-of-process object. Interface pointers can be set to NULL. For example, you can initialize a global interface pointer in the view class's constructor. The first time you need an interface pointer, initialize that pointer if its value is NULL.

The completed code for this exercise is in \Labs\C13\Ex02.