Exercise 13 Introduction to COM+

In this exercise you configure, write and work with COM+ applications and components.

Step 1 - Getting Started

Open the workspace ***QABank*** in the following folder

COM Programming\Exercises\Introduction to COM+

You will find three projects, as follows:

QABankCreator contains code to prepare a SQL Server database for the rest of this exercise.

QABankData contains the code for a component that can adjust the balance of a bank account record. This code will work as is, although we will add support for an Object Constructor String later.

QABankTransfer contains the code for a component that is used to transfer money from one account to another. This project is not currently complete, and you will add the code to support this process.

There is also a Visual Basic test harness that you can use when testing your code.

Step 1 – Preparing the Database

Before we can do any work on the code, we must first build the database. To do this, build and run the ***QABankCreator*** project. You can verify the output of this program by start SQL Server Enterprise Manager, expand the tree until you can see the Databases folder, and you should see a database called ***QABank***.

Open this database, and you will find a table called ***Accounts***. Right click on this table, select ***Open Table -> Select All Rows...*** and you will see information on the five holders of bank accounts at the QA Bank.

Step 2 – Implementing the first phase of the Transfer function

It is time to transfer some money between the accounts. For this step, you will be working in the ***QABankTransfer*** project

Add code to the Transfer() method on the CTransfer class to create an instance of the ***QABankData's QABankAccount*** object and call its ***UpdateAccount()*** method twice to add and subtract money from two accounts.

The code might look something like this:

```
HRESULT CTransfer::Transfer( [in] double amount, [in] BSTR fromAccount,
                             [in] BSTR toAccount )
{
   QABANKDATALib::IQABankAccountPtr spQBA;
   short rowsAffected = 0;

   try
   {
        spQBA.CreateInstance(
                __uuidof( QABANKDATALib::QABankAccount ) );

        // Subtract money from one account, and add it to the other
        if( !spQBA->UpdateAccount( fromAccount, -amount ) ||
           !spQBA->UpdateAccount( toAccount, amount ) )
        {
           _com_issue_errorex( E_FAIL, this, __uuidof( ITransfer ) );
        }
    }
    catch(  _com_error ce  )
    {
    }

    return S_OK;
}
```

Build and test the code using the supplied Visual Basic project. Note that you will need to set the reference to the ***QABankTransfer*** type library in the Visual Basic project before it will run.

Enter the names John Smith and Jane Doe, and the amount of 1000. Click the transfer button, and money moves one from account to another! Check this out using SQL Enterprise Manager, by viewing the table's contents as before.

Step 3 – Adding support for transactions

What happens if you enter the name John Smith for the first account and Arthur Daley for the second? Simply stated, the money will be removed from John Smith's account, but as there is no Arthur Daley account, the money has disappeared. What we need is transaction support, which we will add now.

The first part of this process is to support transactions within the code. By default, COM+ assumes that a successful reply indicates that the transaction vote is to commit. Let's add some code that will check to see if the component is running in a transaction, and if it is and the call has failed, abort the transaction.

Within the catch() block of the Transfer() method, declare a variable of type ***IObjectContext ** ** and use the ***CoGetObjectContext()*** API to obtain a reference to the context object.

Assuming that this call is successful, use the ***IsInTransaction()*** method to check if you are running in a transaction. If it is, use the ***SetAbort()*** method to kill the transaction.

The code might look something like this:

```
catch(  _com_error ce )
{
        IObjectContext *pContext = 0;
        CoGetObjectContext( IID_IObjectContext, (void **) &pContext );

        if( pContext )
        {
                if( pContext->IsInTransaction() )
                {
                        pContext->SetAbort();
                }

                pContext->Release();
        }
}
```

Build the component.

Before you can run the component with its new transaction features, you have to configure it to use COM+ services.

Step 4 – Configuring the QABankTransfer.Transfer Component

Start the Component Services snap-in, and select COM+ Applications. Create a new application called ***QABank***. Choose an application type of ***Server***.

Add a new component (right click on the Components folder within the ***QABank*** application and select *New, Component*). Choose to insert an existing component, and select the ***QABankTransfer.Transfer.1*** component from the list provided.

Set the properties for this component so that it's Transaction attribute is set to ***Required*.** Remember, right click on the component to bring up its properties dialog, and choose the Transaction tab.

The component should now be configured and ready to run under COM+. Try transferring money between accounts. If both accounts are valid, money should move freely – if one of them is incorrect, no money should move.

Step 5 – Adding support for an Object Constructor String

Open the ***QABankData*** project, and then open the ***CQABankAccount*** class header. Locate the TODO statement in the FinalConstruct() method, and then comment out the assignment into the ***m_connectString*** member variable.

Add code to the Construct() method of ***CQABankAccount*** to obtain the constructor string from COM+ instead.

You need to call the ***get_ConstructString()*** method of the ***IObjectConstructString*** interface, a reference to which you can obtain by using ***QueryInterface()*** on the IDispatch pointer that is passed in.

Set the value of ***m_connectString*** to the returned BSTR.

Your code should look something like this:

```
STDMETHODIMP CQABankAccount::Construct(IDispatch *pDisp)
{
    CComPtr<IObjectConstructString> pocs = 0;

    if( !pDisp )
        return E_UNEXPECTED;

    HRESULT hr = pDisp->QueryInterface( IID_IObjectConstructString,
                                        (void **) &pocs
                                      );

    if( SUCCEEDED( hr ) && pocs != 0 )
    {
        BSTR b;
        pocs->get_ConstructString( &b );
        m_connectString.Assign( b );
    }
    else
    {
        hr = E_UNEXPECTED;
    }

    return hr;
}
```

Build the component.

To make use of object constructor strings, you must configure the component with COM+. Add the QABankAccount component to the application you prepared in step 4. On the component's *Activation* tab, enable the constructor string and set it to

```
Provider=SQLOLEDB;Trusted_Connection=yes;Initial Catalog=QABank
```

The VB test harness should continue to operate correctly.