Exercise 1 Implementing the Shop with COM

1.1 Overview

This document is to be seen as a guideline for how this exercise can be designed and implemented. Own initiatives are welcomed.

1.1.1 Goals

After completing this exercise you should be able to:

- Give an overview of how to use the COM and ATL technology “in practice”.
- Use C++ with ATL to implement simple COM objects and COM clients.
- Use the ATL database consumer support to write code that uses a database.
- Compare the amount and type of work needed to accomplish the same task in the different technologies

1.1.2 Requirements

Visual Studio 2010 Professional should be installed on the local machine.

1.1.3 Prerequisites

You should be familiar with C++. You should have read introductory documentation on COM.

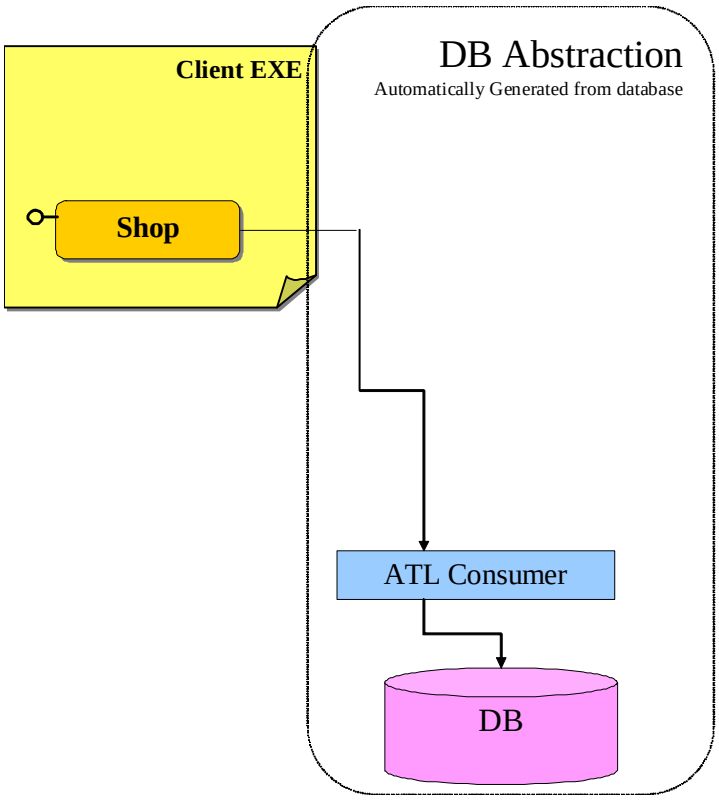
1.1.4 Examination

You must be able to explain all parts of your produced software.

The application shall be zipped and submitted in Blackboard (http://blackboard.mdh.se).

You are allowed to be maximum two persons in each group. Both must know – and be able to answer questions about – all parts of the produced software.

1.2 Functional Description



Since COM is a pretty complex model, I have simplified the architecture a bit. If you are interested you are welcome to implement the full shop but this is not mandatory for the course.

For simplicity we also assume that the shop always has enough fruits and there is no way to cancel bought fruits (since this functionality would not add any new technologies to the exercise.)

The database support is pretty rudimentary in this exercise. It could be improved a lot, but since that is not the focus of the lab it has been set aside.

1.3 Working Steps

1.3.1 The Shop Database

Create an Access database with the following table and columns:

ShopTable : Table

	ProductID	ProductName	ItemsInStock	Price
▶	1	Apple	328	1,00 kr
	2	Orange	449	2,00 kr
	3	Pear	348	1,00 kr
	4	Grape	2991	1,00 kr
	5	Papaya	189	4,00 kr
*	(AutoNumber)			

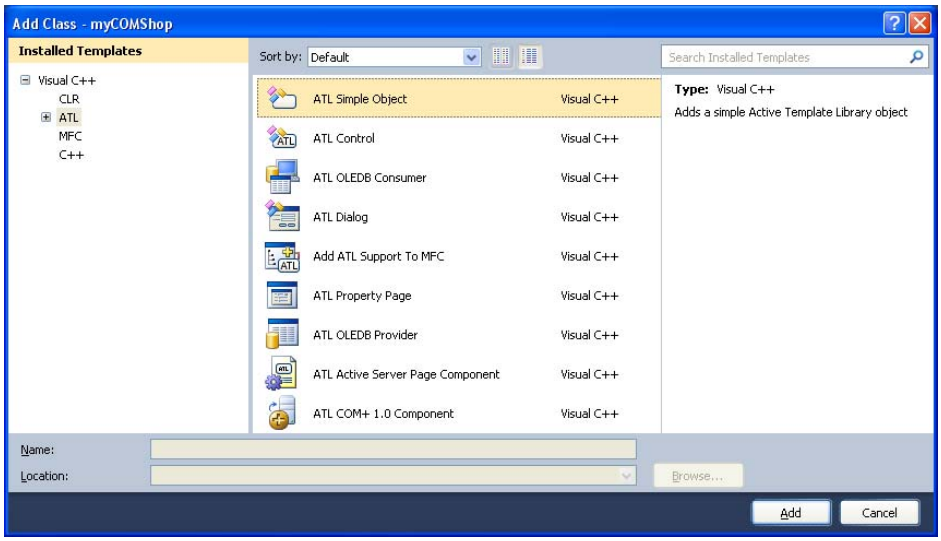
1.3.2 The Shop COM Server

First we will create the server application for the shop object in Visual Studio:

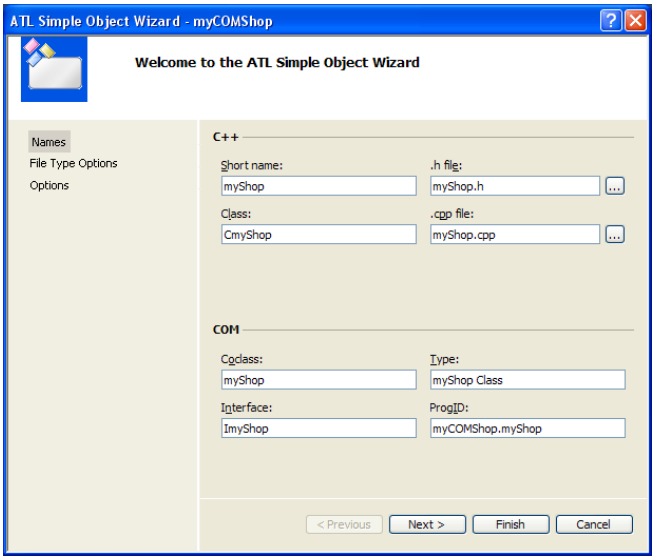
1. Select *File* | *New* | *Project...* | *Other Languages* | *Visual C++* | *ATL* | *ATL Project*.
2. Name the project *myCOMShop* and click *OK*.
3. In the *Applications Setting* page:
    a. Make sure that *Executable (EXE)* is selected.
4. Now the wizard has created all code necessary for hosting COM objects. Take some time to look at the produced code so that you can identify the code added when we add the COM classes.

Now we will create the Shop COM class within myCOMShop project:

5. Select *Project* | *Add class...*



6. Select to add an *ATL | Simple Object*. This is a type of object that has no GUI.
7. Name the new class *myShop* and put *myCOMShop.myShop* for ProgID. The rest of the values will be added automatically:



8. Do not change the options. Click *Finish*.

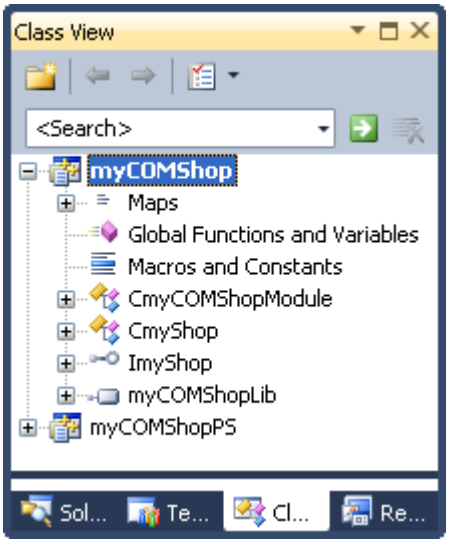
Take a look at the code:
`CComObjectRootEx` and `CComCoClass` implement the `IUnknown` interface, while `IDispatchImpl` provides the implementation of the `IDispatch` interface, which is used by e.g. scripting languages and Visual Basic.

Note the `OBJECT_ENTRY_AUTO` at the bottom of your *myShop.h*. This line tells the server implementation how to create objects of this class.

Also check how the COM class is described in the IDL file before you go on.

Next we will create the interface methods:

9. Select *View | Class View* to get the following window:



10. Right-click the *ImyShop* interface and add the following:

– Properties:
  - *Count* – Deselect the *Put function* checkbox. Select *LONG* as *Property type*.

– Methods:
  - *getNameByIndex* – Parameters:
    - Select *LONG* as *Parameter type*, type *index* in the *Parameter name* textbox, tick the *in* checkbox, and click *Add*.
    - Select *BSTR** as *Parameter type*, type *name* in the *Parameter name* textbox, tick *out* and *retval* (not *in*), and click *Add*.
    - Click *Finish* and repeat the procedure with the three methods below.
  - *getCostByIndex([in] LONG index,[out, retval] LONG* pVal)*
  - *getCountByIndex([in] LONG index,[out, retval] LONG* pVal)*
  - *buy([in] LONG index)*

Take a look in the files *myShop.h*, *myShop.cpp* and *myCOMShop.idl* and see what was added. Note the *get_Count* function.

11. Right-click the project *myCOMShop* and select *Properties*. Expand *Configuration Properties* and *Build Events*. Select *Post-Build Event* and change *Command Line* to the following:

    - "$(TargetPath)" /RegServerPerUser

    Compile the project!

Now we will add database support:

12. Add a new class to the project of type *ATL OLEDB Consumer*.
13. Click on the *Data Source* button.
14. Select *Microsoft Jet 4.0 OLE DB Provider* and click the *Next* button.
    Tip: If you are using Microsoft Access 2003 or newer you will have to select *Microsoft Office 12 Access Database Engine OLE DB Provider*
15. Browse to your Access database and click the *Test Connection* button (leave the *User name* and *Password* settings as they are).
16. Select the table *ShopTable* and click *OK*.
17. Select *Table* and to support *Change*, *Insert* and *Delete*.
18. Untick the *Attributed* checkbox

The wizard has now created a class for you that represents a row in the database table. Visual Studio 2010 generates all the code for the class `CShopTable` in the header file *ShopTable.h*, which means that the class is not compiled (and visible in the class browser) until this file is included in a *.cpp* file. The generated code is furthermore not possible to compile without making the following modifications:

a. Add the following lines at the beginning of *ShopTable.h* (below `#pragma once`)
```
#include <atldbcli.h>
using namespace ATL;
```

b. Find and comment out the following line:
```
#error Security Issue: The connection string may contain a password
```

To make the class even easier to use, we can add code in `FinalConstruct` to open the database and in `FinalRelease` to close it. We also cache the current position (index) in the table. This exercise is not on SQL so we will entirely rely on the row index in the database for accessing the items in the shop.

Open *myShop.h* and add the following line before the definition of `CmyShop`:

```
#include "ShopTable.h"
```

Then, add the following private block to the class:

```
private:
	CShopTable m_db;
	int m_currentIndex;
	bool windToIndex(int index);
```

Change `FinalConstruct` and `FinalRelease` to the handle the database:

```
HRESULT FinalConstruct()
{
	HRESULT hr = m_db.OpenAll();
	m_currentIndex = -1;
	return hr;
}

void FinalRelease()
{
	m_db.CloseAll();
}
```

Implement the interface methods. Sample solution code:

```
// CmyShop

// This private help function winds the recordset to the given index.
// To increase performance, we cache the current index.
// Another improvement would be to move from currentIndex to index
// without MoveFirst.
// Of course, best is to select the row with Some SQL statement but
// that is not the goal of this exercise.
bool CmyShop::windToIndex(int index)
{
	If (m_currentIndex == index)// Nothing to do.
		return true;
	if (index < 0)
		return false; // index too small.

	int tmpIndex = index;
	m_db.MoveFirst();
	while (tmpIndex-- > 0 && m_db.MoveNext() == S_OK) {/*no op*/}
	if (tmpIndex != -1) // index too large. Reset cached index.
	{
		m_currentIndex = -1;
		return false;
	}
	m_currentIndex = index;
	return true;
}
```

Note that `get_Count` will only work with a database with at least one entry. This will do for this exercise (otherwise you would have to do an SQL query and handle the result which is more complex and outside the topic of the exercise).

```
STDMETHODIMP CmyShop::get_Count(LONG* pVal)
{
	m_currentIndex = -1;
	m_db.MoveFirst();
	int count = 0;
	while (m_db.MoveNext() == S_OK) count++;
	*pVal = count + 1;
	return S_OK;
}

STDMETHODIMP CmyShop::getNameByIndex(LONG index,BSTR* name)
{
	if (!windToIndex(index)) return E_FAIL;
	CComBSTR nameVal = m_db.m_ProductName;
	return nameVal.CopyTo(name);
}

STDMETHODIMP CmyShop::getCostByIndex(long index, long* pVal)
{
	if (!windToIndex(index)) return E_FAIL;
	*pVal = m_db.m_Price.Lo;
	return S_OK;
}

STDMETHODIMP CmyShop::getCountByIndex(long index, long* pVal)
{
	if (!windToIndex(index)) return E_FAIL;
	*pVal = m_db.m_ItemsInStock;
	return S_OK;
}

//Buys one piece of the item at a time
STDMETHODIMP CmyShop::buy(LONG index)
{
	if (!windToIndex(index)) return E_FAIL;
	if (m_db.m_ItemsInStock <= 0) return E_FAIL;
	m_db.m_ItemsInStock--;
	return m_db.SetData();
}
```

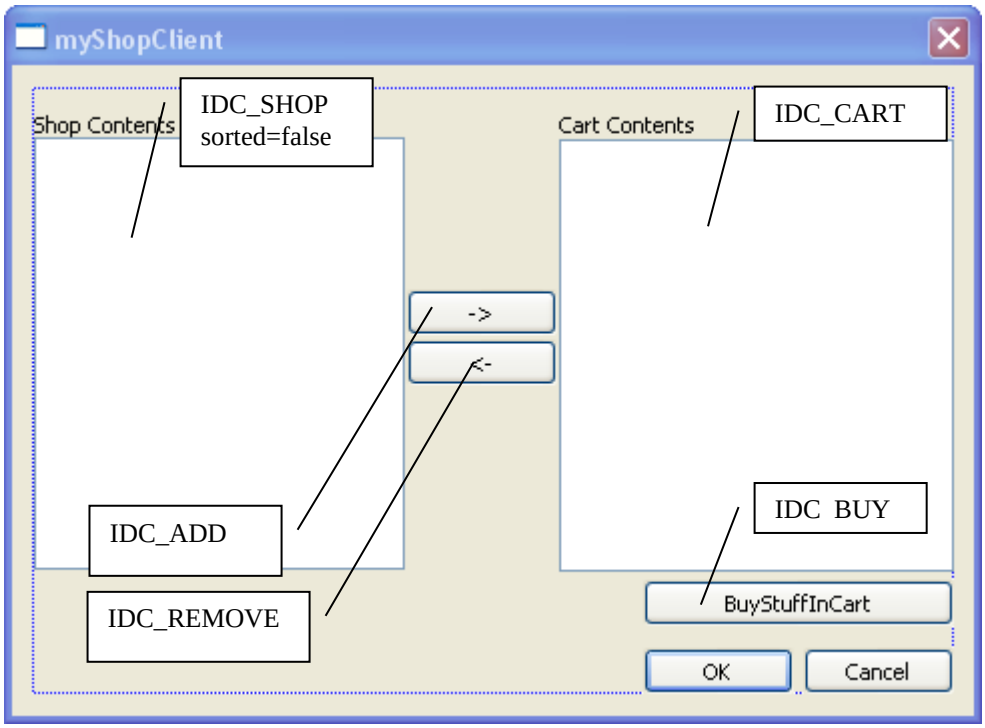
19. Build the project.

1.3.3 The Shop Client

Now we will create a client for the shop. We will use MFC to test if the shop works as it should.

1. Create a new project of type *Visual C++ | MFC | MFC Application* (add it to the current solution).
2. Name it *myShopClient*.
3. In the tab *Application Type* select *Dialog Based* and deselect *Use Unicode libraries*. Leave the other choices as they are.

4. Make sure that the resource language is *English* and click *Finish*.
5. In the resource view of the new project, double click *IDD_MYSHOPCLIENT_DIALOG*.
6. Add two list boxes and three buttons like this:



7. Use the properties dialog of each dialog item to change the ID to the suggested IDs in the picture above.

To make it easier to use the list boxes, we will connect a help class to them. We will use the class wizard for this.

8. Right click on the *IDC_SHOP* and choose *Add Variable*
9. Name the new variable *m_shopList* (do not change the other fields).
10. Repeat the procedure for *IDC_CART* and *m_cartList*.

To be able to use the COM component we created before, we need to import its type library.

11. In *myShopClientDlg.h*, add the following statement (on a single line) before the class definition:
```
#import "progid:myCOMShop.myShop.1" raw_interfaces_only, raw_native_types, no_namespace, named_guids
```

This will create and include the necessary header files for using the *ImyShop* interface. But before we can use the interface and create any COM objects, we need to initialize the COM library. We can do this by cheating.

12. Add a new class of type *ATL | ATL Simple Object* to the shop client project. Answer yes, to add ATL support, but click cancel before any actual ATL object is created. By now you have all code needed for using other ATL components.
13. Add a member variable `CComPtr<ImyShop> m_pShop` to `CmyShopClientDlg`.

14. Create a private help function that fills `m_shopList` with the contents of the shop and call it from `onInitDialog`. Example solution code:

```
void UpdateShopList(void)
{
	m_shopList.ResetContent();
	long count;
	if (!m_pShop)
		m_pShop.CoCreateInstance(L"myCOMSHop.myShop.1");
	HRESULT hr = m_pShop->get_Count(&count);
	USES_CONVERSION;
	for (long i = 0; i < count; i++)
	{
		CComBSTR name;
		HRESULT hr = m_pShop->getNameByIndex(i,&name);
		if (FAILED(hr))
		{
			m_shopList.AddString("Failed to read from Shop");
			break;
		}
		long itemCount;
		hr = m_pShop->getCountByIndex(i,&itemCount);
		if (FAILED(hr))
		{
			m_shopList.AddString("Failed to read from Shop");
			break;
		}
		char buf[200];
		sprintf(buf,"%s %d ", W2A(name),itemCount);
		m_shopList.AddString(buf);
		// The list is not sorted and the items in DB
		// are numbered 0 through n for simplicity.
	}
}
```

Compile and test the project before you go on!

Now it is time to add the cart functionality:

15. In the resource view, double click on the buttons and implement the shop behaviour. Here is a sample solution:

```
void CmyShopClientDlg::OnBnClickedAdd()
{
	int curSel = m_shopList.GetCurSel();
	if (curSel<0) return;
	CString selectedProduct;
	m_shopList.GetText(curSel,selectedProduct);
	char buf[200]; // Extract product name
	int count;     // Extract product count
	sscanf(selectedProduct,"%s %d",buf,&count);
	int cartIndex = m_cartList.AddString(buf);
	// curSel is also database key since the
	// m_shopList is not sorted.
	m_cartList.SetItemData(cartIndex,curSel);
}

void CmyShopClientDlg::OnBnClickedRemove()
{
	int curSel = m_cartList.GetCurSel();
	if (curSel<0) return;
	m_cartList.DeleteString(curSel);
}

void CmyShopClientDlg::OnBnClickedBuy()
{
	int count = m_cartList.GetCount();
	for (int i =0; i < count; i++)
	{
		long productId = m_cartList.GetItemData(i);
		HRESULT hr = m_pShop->buy(productId);
	}
	m_cartList.ResetContent();
	UpdateShopList();
}
```

Compile and test out the shop. When it works, you may try to use DCOM to reach the objects on another computer, or implementing the GUI in for example Visual Basic. Note that DCOM may not be available in the computer rooms and may have to be tested on your own computer.