Ministry of Agriculture & Land Reclamation
Agricultural Research Center
Central Lab for Agricultural Expert Systems

Writing a Versionable Application Using Visual C++6

TRICLAES/233/2002.3

By

Dr.Samhaa
Eng.Mohammed El Helly
Eng. Mohammed Said
Eng Ahmed Fouad

March, 2002

Table Of Contents

Introduction to Serialization

"Serialization" is the process of writing or reading an object to or from a persistent storage medium, such as a disk file. MFC supplies built-in support for serialization in the class **CObject**. Thus, all classes derived from **CObject** can take advantage of **CObject**'s serialization protocol.

Five main steps are required to make a class serializable. They are listed below:

1. Deriving your class from **CObject** (or from some class derived from **CObject**).
2. Overriding the Serialize member function.
3. Using the **DECLARE_SERIAL** macro in the class declaration.
4. Defining a constructor that takes no arguments.
5. Using the **IMPLEMENT_SERIAL** macro in the implementation file for your class.

One of Serialization's problems is a multiple version " that is, files written with different versions of the application" which will be discussed here and we will try to provide appropriate solution to it.

The versioning has two levels:

1- Versioning at the Object level.
2- Versioning at the Document

Handle Versioning at The Object Level

Handle Version On The Object Level required that class of such object must be serializable .

Consider a class that is derived from **CObject** and has two member variables, of types **`Cstring`** .

The following class declaration fragment shows the member variables and the declaration for the overridden **Serialize** member function:

```
class CPerson : public CObject
{
public:
    DECLARE_SERIAL( CPerson )
    // empty constructor is necessary
    CPerson(){};

    CString name;
    CString addr;
    void Serialize( CArchive& archive );

    // rest of class declaration
};
```

CPerson class declaration (**version1**)

The implementation of **Serialize** method for the `CPerson` class which is declared above:

```
IMPLEMENT_SERIAL( Person, CObject, 1 );

void CPerson::Serialize( CArchive& archive )
{
    // call base class function first
    // base class is CObject in this case
    CObject::Serialize( archive );

    // now do the stuff for our specific class
    if( archive.IsStoring() )
        archive << name << addr;
    else
        archive >> name >> addr;
}
```

Serialize body for **(verion1)**

With the growth of system, we need to add additional feature to person class,

Suppose we have a person class (version 2) and declared as follow

```
class CPerson : public CObject
{
public:
    DECLARE_SERIAL( CPerson )
    // empty constructor is necessary
       CPerson(){};
       CString name;
       CString addr;
       Cstring birth_date;//new feature
       void Serialize( CArchive& archive );
       // rest of class declaration
};
```

CPerson class declaration (**version1**)

and the serialize body become

```
IMPLEMENT_SERIAL( Person, CObject, 2 );

void CPerson::Serialize( CArchive& archive )
{
      // call base class function first
      // base class is CObject in this case
      CObject::Serialize( archive );

      // now do the stuff for our specific class
      if( archive.IsStoring() )
          archive << name << addr << birth_date;
      else
          archive >> name >> addr >> birth_date;
}
```

Serialize body for **(version2)**

Assume that we use version1 and after that we try to open files written by version1 with new version error will be occur because in our case two classes have different structure. To handle this problem we update **Serialize** member function to be able to read multiple versions as follow:

1-We use the value **VERSIONABLE_SCHEMA** as an argument to the **IMPLEMENT_SERIAL** macro .

Version1:

IMPLEMENT_SERIAL(Person, CObject, **VERSIONABLE_SCHEMA |** 1);

Version2:

IMPLEMENT_SERIAL(Person, CObject, **VERSIONABLE_SCHEMA |** 2);

2-At serialize implementaion we use **GetObjectSchema** member function of class **Carchive** to know the version of object and according to that version we take appropriate action.

In version 2: -

```
void CPerson::Serialize( CArchive& archive )
{
    CObject::Serialize( archive );
    if( archive.IsStoring() )
      {
          archive << name << addr << birth_date;
      }
    else
       {
           UINT nShema=ar.GetObjectSchema ();
           switch(nShema)
                 {
           case 1:
                 archive >> name >> addr;
                  break;
           case 2:
                  archive >> name >> addr >> birth_date;
                  break;
           default:
                  AfxMessageBox ("Error: Invalid Format");
                 break;

                  }
           }
}
```

Serialize body after update

Handle Versioning at Document Level

The problem in the document class occurred when we add additional member variables to Document class. In this case we cannot open the old version of the document.

This example shows the problem:

In my Document say(CTestDoc) class there is one member variable of type MyMap. Here is the definition of MyMap

```
typedef CTypedPtrMap<CMapStringToOb,CString,CPerson*>MyMap;
class CtestDoc
{
        public:
                MyMap per;
        ….
        ….
        ….
};
```

The class definition

The variable **(per)** want to be serialize. The serialize method of document class can be written as

```
void CTestDoc::Serialize(CArchive& ar)
{
                per.Serialize(ar);
}
```

In the next version of the application we add member variable(s) in the Document class to be serialized.

```
typedef CTypedPtrMap<CMapStringToOb,CString,CPerson*>MyMap;
typedef CTypedPtrMap<CMapStringToOb,CString,computer*>MyMap2;
class CtestDoc
{
        public:
                MyMap per;
                MyMap2 com; //computer is another class
        ….
        ….
};
```

The new version of class

and the implementation of the serialize method as the following:

```
void CTestDoc::Serialize(CArchive& ar)
{
                per.Serialize(ar);
                com. Serialize(ar);
}
```

But the new version cannot read files that be created in the older version of the system. To solve this problem we add member variable in Document class refer to the version number, and in serialize method we check in this variable.

If the number refers to old version we read the file with the same way of the old version and if this number refers to new version we read the file with new version. The following example shows the solution

In the document class of the new version we declare new variable

```
class CtestDoc
{
        public:
                MyMap per;
                MyMap2 com; //computer is another class
        private:
                int version; // number of version
        ….
        ….
};
```

The updated code of new version class

The implementation of the serialize method as the following: -

```
void CTestDoc::Serialize(CArchive& ar)
{
	version =2; //new version
	if(ar.IsStoring())
	{
		ar<<version;
		per.Serialize(ar);
		com.Serialize(ar);
		asd.Serialize(ar);
	}
	else
	{
		ar>>version;
		switch(version) //read the old version file
		case 1:
			per.Serialize(ar);
			com.Serialize(ar);
			break;
		case2: //version 2
			per.Serialize(ar);
			com.Serialize(ar);
			asd.Serialize(ar);
			count_asd=asd.GetCount();
			break;
		default:
			AfxMessageBox("Unknown document version")
	}
```