직렬화

- 객체의 상태를 저장한 후 나중에 그 상태를 읽어 복원시키는 것을 MFC에서는 직렬화 (Serialization)이란 용어로 부름
- DECLARE_SERIAL/IMPLEMENT_SERIAL 매크로를 추가하고, 개발자가 정의한 멤버 변수를 read/write 하도록 CObject::Serialize()를 재정의 (overriding) 해야 함
- 만약 개발자가 직렬화를 사용하게 된다면 개발자는 file의 형식 (format)에 대한 아무런 걱정 없이 read/write 할 수 있음 – 단지 객체를 read/write하는 순서만 맞추어주면 됨
- 직렬화 매크로를 사용하면 DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 과 DECLARE_DYNCREATE/IMPLEMENT_DYNCREATE 을 사용할 필요 없음 – 즉 직렬화된 데이터를 읽어 들일 때 저장된 데이터에 맞는 클래스가 동적 생성되는 기능을 사용할 수 있는 것임

직렬화

- DECLARE_SERIAL/IMPLEMENT_SERIAL 매크로

```
#define DECLARE_SERIAL(class_name) \
    DECLARE_DYNCREATE(class_name) \
    AFX_API friend CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb);

#define IMPLEMENT_SERIAL(class_name, base_class_name, wSchema) \
    CObject* PASCAL class_name::CreateObject() \
            { return new class_name; } \
    IMPLEMENT_RUNTIMECLASS(class_name, base_class_name, wSchema, \
            class_name::CreateObject) \
    AFX_CLASSINIT _init_##class_name(RUNTIME_CLASS(class_name)); \
    CArchive& AFXAPI operator>>(CArchive& ar, class_name* &pOb) \
        { \
            pOb = (class_name*) ar.ReadObject(RUNTIME_CLASS(class_name)); \
            return ar; \
        } \
```


직렬화

```
CObject* CArchive::ReadObject(const CRuntimeClass* pClassRefRequested)
{
    UINT nSchema;
    DWORD obTag;
    CRuntimeClass* pClassRef
                = ReadClass(pClassRefrequested, &nSchema, &obTag);
    // check to see if tag to already loaded object
    CObject* pOb;
    pOb = pClassRef->CreateObject();
    // Serialize the object
    UINT nSchemaSave = m_nObjectSchema;
    m_nObjectSchema = nSchema;
    pOb->Serialize(*this);
    m_nObjectSchema = nSchemaSave;
    return pOb;
}
```


직렬화

- Serialization 매크로에 연산자 <<가 포함되어 있지 않음
- 연산자 <<는 WriteObject을 호출하고, WriteObject는 WriteClass를 호출하고, 그런 다음 WriteObject는 그 객체의 Serialize()를 호출한다
- WriteObject는 CRuntimeClass information을 필요로 하지 않기 때문에 CObject를 위한 연산자 << 하나면 충분하다
- 따라서 CObject 파생물을 위한 연산자 <<는 선언될 필요도 구현될 필요도 없다

```
CArchive& operator<<(CArchive& ar, const CObject* pOb)
{
    ar.WriteObject(pOb);
    return ar;
}
```

직렬화 사용 예

```
// MyClass.h
class CMyClass : public CObject
{
   DECLARE_SERIAL(CMyClass)
public:
   CMyClass();
   WORD m_MyVar1
   DWORD my_MyVar2;
   virtual void Serialize (CArchive& ar);   // 가상함수를 재정의
};
// MyClass.cpp
#include "MyClass.h"
IMPLEMENT_SERIAL(CMyClass, CObject, 1)
void CMyClass::Serialize (CArchive& ar)
{
   // CObject가 제공하는 가상 함수인 Serialize() 함수를 재정의한다.
}
...
```


Serialize 함수

- 애플리케이션이 파일을 읽거나 쓰려 할 때 도큐먼트 객체의 Serialize( ) 함수가 호출되는데 이때 파일을 읽거나 쓰는데 사용되는 CArchive 객체가 인자로 넘겨짐
- IsLoading() 이나 IsStoring() 함수 호출해서 읽기 또는 쓰기에 있는 지 결정

```
void CMyClass::Serialize(CArchive &ar) {
    if(ar.IsStoring( )) { // 아카이브가 현재 쓰기 상태에 있나?
        // 그렇다. 이제, 변수를 순서에 맞춰 아카이브에 저장한다.
        ar << m_MyVar1 << m_MyVar2;
    }
    else { // 아니다. 변수를 순서에 맞춰 읽은다.
        ar >> m_MyVar1 >> m_MyVar2;
    }
}
```


직렬화 가능한 클래스 만들기

- 각각의 레코드를 캡슐화하는 클래스를 만들고 이들 레코드를 배열에 담아 처리
- 클래스는 CObject 클래스에서 파생하고 뷰 클래스에 추가한 컨트롤에 물려둔 변수에 해당하는 데이터 필드를 멤버 변수로 가짐
- 멤버 변수를 액세스 하기 위한 멤버 함수를 가짐
- 이 클래스의 직렬화를 위한 매크로와 Serialize() 함수를 가져야 함

직렬화 가능한 클래스 만들기

- Point3D.h

```
class CPoint3D : public CObject
{
public:
   DECLARE_SERIAL(CPoint3D)
   int m_nX, m_nY, m_nZ;
   CPoint3D();
   CPoint3D(int x, int y, int z);
  ~ CPoint3D();
   void Serialize(CArchive& ar);
};
```

직렬화 가능한 클래스 만들기

- Point.cpp

```
#include "Point3D.h"
IMPLEMENT_SERIAL(CPoint3D, CObject, 1)
CPoint3D::CPoint3D( ) {
    m_nX = m_nY = m_nZ = 0;
}
CPoint3D::CPoint3D(int x, int y, int z) {
    m_nX = x; m_nY = y; m_nZ = z;
}
void CPoint3D::Serialize(CArchive& ar) {
    CObject::Serialize(ar);
    if(ar.IsStoring())        // 쓰고 읽는 변수의 순서가 동일해야 함
        ar << m_nX << m_nY << m_nZ;
    else
        ar >> m_nX >> m_nY >> m_nZ;
}
```


직렬화 가능한 클래스 만들기

- main.cpp

```
void main()
{
    TRY { // 쓰기
        CPoint3D point(10, 20, 30);
        CFile file;
        file.Open(_T("File.dat"), File::modeCreate | CFile::modeWrite);
        CArchive ar(&file, CArchive::store);
        ar << &point;
    }
    CATCH(CFileException, e)  {
        e->ReportError();
    }
    END_CATCH
```