파일 입출력

HCI Programming 2 (321190)
2008년 가을학기
11/4/2008
박경신

Overview

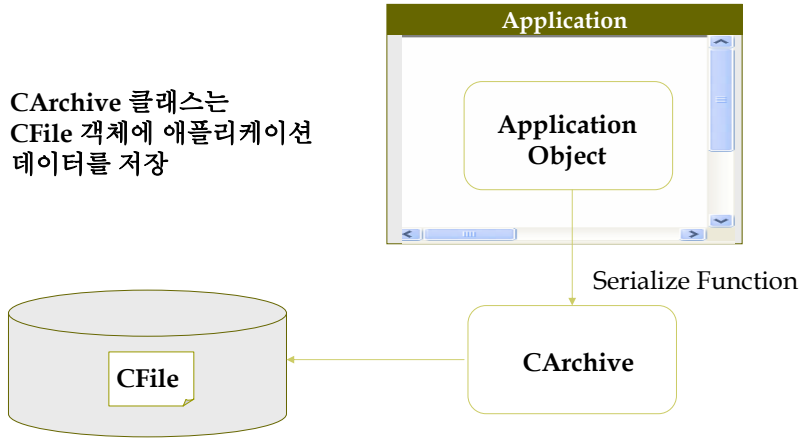
- CFile 클래스를 이용한 파일 입출력 기법
- 도큐먼트/뷰 구조 이해
- CArchive 클래스를 이용한 직렬화 기법

파일 입출력 방법

- 일반 파일 입출력
  - CFile (파생) 클래스
  - Read(), Write() 등의 함수 이용
- 직렬화 (serialization) – 애플리케이션 데이터가 파일의 형태로 시스템 드라이브에 저장될 때
  - CArchive 클래스
  - << 또는 >> 연산자 이용

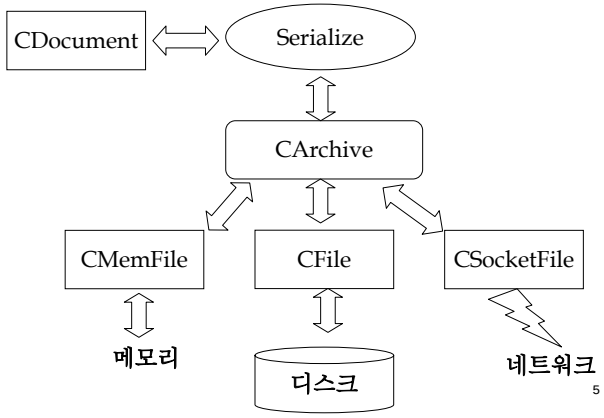
CArchive와 CFile 클래스

CArchive 클래스는 CFile 객체에 애플리케이션 데이터를 저장



CArchive 클래스

- CDocument 클래스와 CFile 클래스를 연결해 주기 위한 클래스
- IsStoring() : 읽기상태 또는 저장상태를 판별



CArchive 중요 멤버 함수

- IsLoading () CArchive 객체가 자료를 불러오는 중인지를 알 수 있음
- IsStoring () CArchive 객체가 자료를 저장하는 중인지를 알 수 있음
- 연산자 >> CArchive 객체로 부터 객체나 기본 데이터형을 읽음
- 연산자 << CArchive 객체에 객체나 기본 데이터형을 저장
- Read() CArchive 객체로부터 byte 단위의 블록을 읽어옴
- Write() CArchive 객체에 바이트 단위의 블록을 저장
- ReadString() CArchive 객체로 부터 문자열을 읽어옴
- WriteString() CArchive 객체에 문자열을 저장

MFC 클래스 계층도

- MFC 클래스 계층도

CFile 클래스 핵심 입출력 연산

- Open() 파일을 열거나 생성한다
- Read() 파일 포인터의 위치에서 데이터를 읽는다
- Write() 파일 포인터의 위치에 데이터를 쓴다
- Seek() 파일 포인터의 위치를 변경한다
- Close() 파일을 닫는다

CFile 클래스 열기와 생성

- 열기와 생성

```
TRY {
    CFile file("mytest.txt", CFile::modeReadWrite); // 방법1
}
CATCH (CFileException, e) {
    e->ReportError();
    e->Delete();
}
END_CATCH
```

```
CFile file;
CFileException e;
if(!file.Open("mytest.txt", CFile::modeReadWrite, &e))  // 방법2
    e.ReportError();
```


CFile 클래스 파일 접근/공유 모드

- 파일 접근/공유 모드

플래그	의미
CFile::modeCreate	파일을 무조건 생성한다. 같은 이름의 파일이 있다면 크기를 0으로 바꾼다.
CFile::modeNoTruncate	| 연산자를 이용하여 CFile::modeCreate 플래그와 더불어 사용하면 같은 이름의 파일이 있을 경우 크기를 0으로 바꾸지 않고 이 파일을 연다.
CFile::modeRead	읽기 전용 모드로 파일을 열거나 생성한다.
CFile::modeReadWrite	읽기 및 쓰기 모드로 파일을 열거나 생성한다.
CFile::modeWrite	쓰기 전용 모드로 파일을 열거나 생성한다.
CFile::shareDenyNone	다른 프로세스에게 파일에 대한 읽기/쓰기를 허용한다.
CFile::shareDenyRead	다른 프로세스에게 파일에 대한 읽기를 금지한다.
CFile::shareDenyWrite	다른 프로세스에게 파일에 대한 쓰기를 금지한다.
CFile::shareDenyExclusive	다른 프로세스에게 파일에 대한 읽기/쓰기를 금지한다.

CFile 클래스 닫기

- 닫기: 방법1

```
void CExFileView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    CFile file;
    CFileException e;
    if(!file.Open("mytest.txt", CFile::modeReadWrite|
                                CFile::modeCreate, &e)) {
        e.ReportError();
        return;
    }
    // 생략 ...
} // -> CFile::~CFile() 함수가 호출된다.
```

CFile을 사용하면 객체가 소멸시에 자동으로 파일을 닫으므로 명시적으로 파일을 닫는 함수를 호출할 필요가 없다

CFile 클래스 닫기

- 닫기: 방법2

```
void CExFileView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    CFile file;
    CFileException e;
    if(!file.Open("mytest.txt",
                  CFile::modeReadWrite|CFile::modeCreate|
                  CFile::modeNoTruncate, &e)) {
        e.ReportError();
        return;
    }
    // 생략 ...
    file.Close();
}
```

CFile을 이용하여 여러 개의 파일을 다룰 때

CFile 클래스 Read, Write, Seek

- 읽기와 쓰기

```
UINT CFile::Read (void* lpBuf, UINT nCount) ;
void CFile::Write (const void* lpBuf, UINT nCount) ;
```

- 파일 포인터 위치 변경

```
ULONGLONG CFile::Seek (LONGLONG lOff, UINT nFrom);
```

nFrom	의미
CFile::begin	파일의 처음 위치부터 lOff만큼 파일 포인터 이동
CFile::current	현재의 파일 포인터 위치부터 lOff만큼 파일 포인터 이동
CFile::end	파일의 끝 위치부터 lOff만큼 파일 포인터 이동

CFile 클래스 데이터 입출력

- 파일에 데이터 쓰기

```
int buffer[1000];
CFile file;
file.Open(_("test.dat", CFile::modeCreate | CFile::modeWrite);
file.Write(buffer, 1000 * sizeof(int));
file.Close( );
```

- 파일에서 데이터 읽기

```
CFile file;
file.Open(_T("test.dat"), CFile::modeRead);
int nLength = file.GetLength( );
int *buffer = new BYTE [nLength];
file.Read(buffer, nLength);
file.Close( );
```


CFile 클래스 에러 처리

- 에러의 가능성
  - 지정한 파일이 디스크에 존재하지 않음
  - 다른 프로그램에서 파일을 사용
  - 디스크가 모두 차서 쓸 공간 부족

```
void main() {
   TRY{
      int buffer[1000];
      CFile file;
      file.Open(_T("test.dat"), CFile::modeCreate | CFile::modeWrite);
      file.Write(buffer, 1000 * sizeof(int));
      file.Close( );
   }
   CATCH(CFileException, e){
      e->ReportError( );
   }
   END_CATCH
}
```


CFile 클래스 기타 함수

- CFile::GetLength(), CFile::SetLength()
  - 파일의 현재 크기를 얻거나 변경한다
- CFile::GetPosition()
  - 현재의 파일 포인터 위치를 얻는다
- CFile::LockRange(), CFile::UnlockRange()
  - 파일의 일정 영역을 잠그거나 해제한다. 잠근 영역은 다른 프로세스가 접근할 수 없다
- CFile::GetFilePath(), CFile::GetFileName()
  - 파일의 전체 경로(Full Path)와 이름을 얻는다
- CFile::GetFileTitle()
  - 확장자를 제외한 파일 이름을 얻는다
- CFile::GetStatus(), CFile::SetStatus()
  - 파일이 생성된 시간, 최종 변경된 시간, 크기, 속성등 파일의 상태를 얻는다. 파일의 상태를 지정한다

CMemFile 클래스

- 사용 예

```
void CExFileView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
   CMemFile file;

   // 메모리 파일에 쓰기
   int a = 100;
   file.Write(&a, sizeof(a));

   // 메모리 파일에서 읽기
   file.SeekToBegin();
   int b;
   file.Read(&b, sizeof(b));
   TRACE("b = %d\n", b);
}
```


CStdioFile 클래스

- 사용 예
  - 마우스를 더블클릭하면 test2.txt생성 후 복사
  - ReadString과 WriteString을 제공, Text 파일을 읽고 쓰게 함.

```
void CExFileView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
   CStdioFile file1; // 읽기 전용으로 file1을 open
   CFileException e;
   if(!file1.Open("test1.txt", CFile::modeRead, &e))
   {
      e.ReportError();
      return;
   }

   CStdioFile file2; // 쓰기 전용으로 file2를 open
   if(!file2.Open("test2.txt", CFile::modeWrite|CFile::modeCreate,
   &e))
```


CStdioFile 클래스

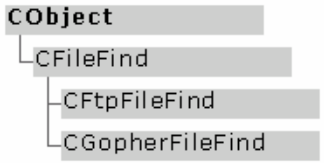
- 사용 예

```
   {
      e.ReportError();
      return;
   }

   CString str; // test1.txt를 string으로 읽어들여 test2.txt에 쓴다
   while(file1.ReadString(str)){
      str.MakeUpper();
      file2.WriteString(str + "\n");
   }
}
```


CFileFind 클래스

- 로컬 디스크에 있는 파일 검색 기능 제공
- 주된 멤버함수
  - FindFile – 어떤 이름을 갖는 파일 찾기
  - IsDirectory – 디렉토리인지 파악
- MFC 클래스 계층도



CFileFind 클래스

- 사용 예
  - 현재 디렉토리에 대한 모든 파일과 디렉토리를 보여주는 예제

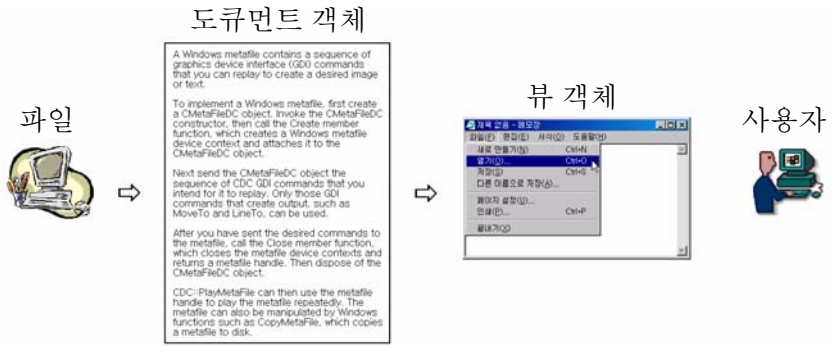
```
void CExFileView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    CFileFind finder;
    BOOL bWorking = finder.FindFile("*.*");    // *.* 파일을 찾는다
    while(bWorking){                           // 찾는 파일이 있으면
        bWorking = finder.FindNextFile();      // 실제파일정보를 받고
        if(finder.IsDirectory())               // 만약 디렉토리이면
            TRACE("[%s]\n", (LPCTSTR)finder.GetFileName());
        else
            TRACE("%s\n", (LPCTSTR)finder.GetFileName());
    }
}
```


도큐먼트/뷰 구조

- 개념
  - 프로그램에서 사용할 데이터를 관리하는 부분과 이 데이터를 실제로 화면에 표시하는 부분을 서로 다른 모듈로 구현한다
- Document
  - 데이터 관리에 해당하는 기능을 구현하는 클래스
- View
  - 데이터를 화면에 표시하는 기능을 구현하는 클래스

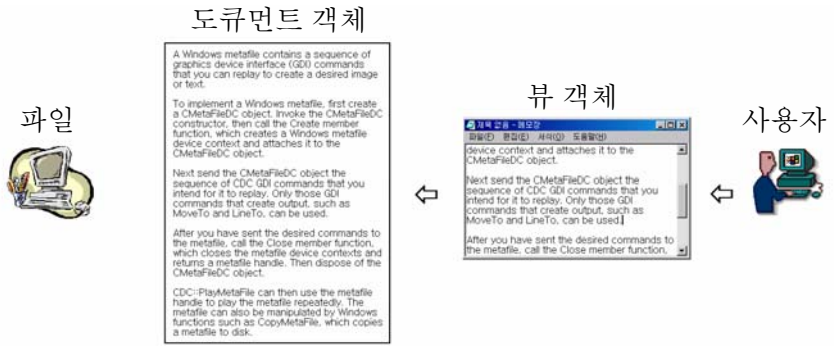
도큐먼트/뷰 구조

- 디스크에 저장된 파일을 읽는 경우
  - 도큐먼트 객체가 파일에 저장된 데이터를 읽은 후
  - 뷰 객체로 하여금 데이터를 화면에 보이게 한다



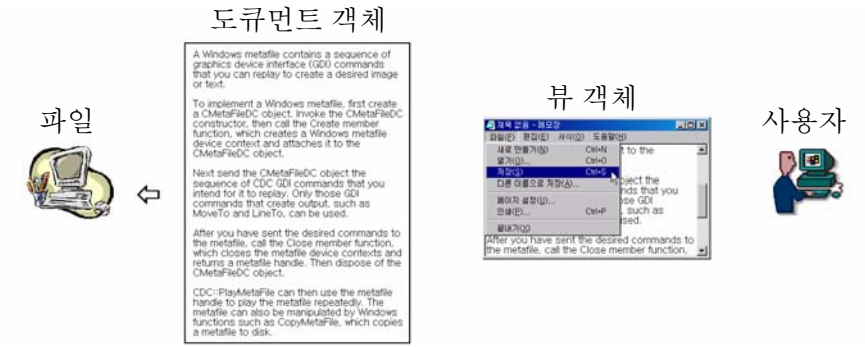
도큐먼트/뷰 구조

- 사용자가 키보드/마우스로 데이터를 입력하는 경우
  - 뷰가 처리한 후
  - 입력된 데이터를 도큐먼트 객체에 저장



도큐먼트/뷰 구조

- 입력된 문서를 디스크 파일로 저장하는 경우
  - 저장된 명령을 내리면
  - 도큐먼트 객체가 자신이 유지하는 데이터를 디스크 파일로 저장



도큐먼트/뷰 구조

- 도큐먼트와 뷰 클래스의 역할

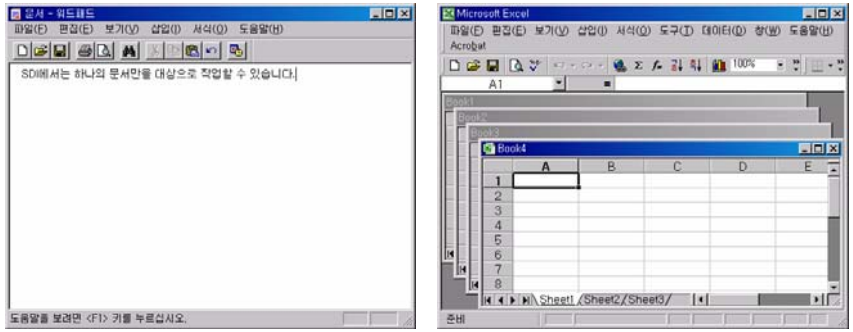
클래스	역할
도큐먼트	데이터를 저장하거나 읽어들인다. 데이터의 변경 사항이 생기면 뷰의 화면을 갱신한다.
뷰	데이터를 화면에 표시한다. 사용자와의 상호 작용을 담당한다.

도큐먼트/뷰 구조

- 도큐먼트/뷰 구조의 장점
  - 서로 다른 기능을 도큐먼트와 뷰 클래스로 분리해서 구현하기 때문에 개념적으로 이해하기 쉽다.
  - 하나의 도큐먼트에 여러 개의 뷰가 존재하는 모델을 구현하기가 쉽다.
    - 예) 비주얼 C++ 편집창
  - MFC에서 도큐먼트/뷰 구조를 위해 제공하는 부가적인 서비스를 이용할 수 있다.
    - 예) 직렬화

도큐먼트/뷰 구조

- SDI (Single-Document Interface)와 MDI (Multiple-Document Interface)
  - 다룰 수 있는 문서의 개수에 따라 구분



도큐먼트/뷰 구조

- 도큐먼트 템플릿
  - 도큐먼트/뷰 구조의 핵심적인 클래스
  - 도큐먼트, 프레임 윈도우, 뷰 클래스 정보를 유지
  - 필요에 따라 해당 객체를 동적으로 생성
- MFC 클래스 계층도

도큐먼트/뷰 구조

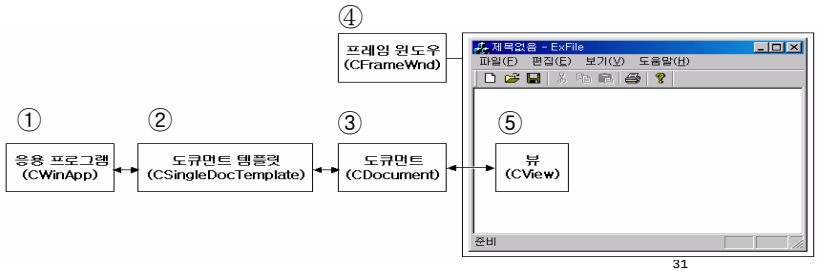
- InitInstance() 함수
  - 템플렛 객체를 동적으로 생성한 후
  - 응용 프로그램에 등록

```
BOOL CExFileApp::InitInstance()
{
   ...
   CSingleDocTemplate* pDocTemplate;
   // 도큐먼트, 프레임, 뷰를 가지고 있는 템플렛 객체 생성
   pDocTemplate = new CSingleDocTemplate(
      IDR_MAINFRAME,
      RUNTIME_CLASS(CExFileDoc),
      RUNTIME_CLASS(CMainFrame),
      RUNTIME_CLASS(CExFileView));
   AddDocTemplate(pDocTemplate); // 응용프로그램에 등록
   ...
}
```


도큐먼트/뷰 구조

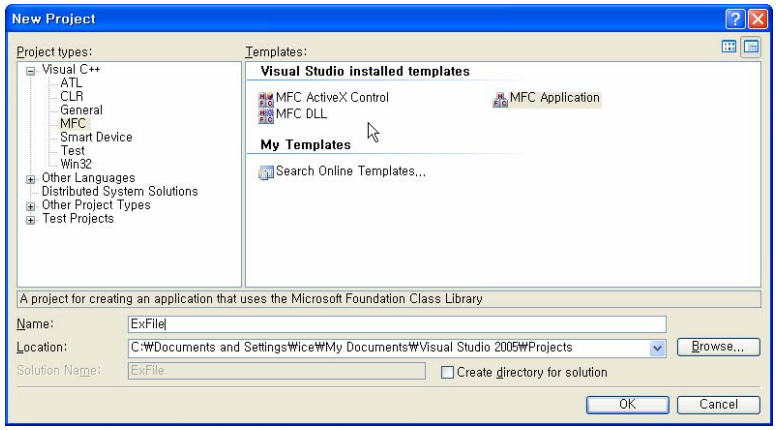
- 주요 객체의 관계

생성 주체	생성되는 것
①응용 프로그램 객체	②도큐먼트 템플릿 객체
도큐먼트 템플릿 객체	③도큐먼트 객체, ④프레임 윈도우 객체
프레임 윈도우 객체	⑤뷰 객체



도큐먼트/뷰 구조 응용 프로그램 예제 작성

- 프로젝트 생성



도큐먼트/뷰 구조 응용 프로그램 예제 작성

- 1~6단계 옵션 설정

단계	변경 사항
1	'Single document'를 선택한다.
2	Compound Document Support 변경 사항 없음
3	DB 변경 사항 없음
4	UI 변경 사항 없음
5	Advanced Features 'ActiveX Controls' 선택을 해제한다.
6	Generated Classes 변경 사항 없음

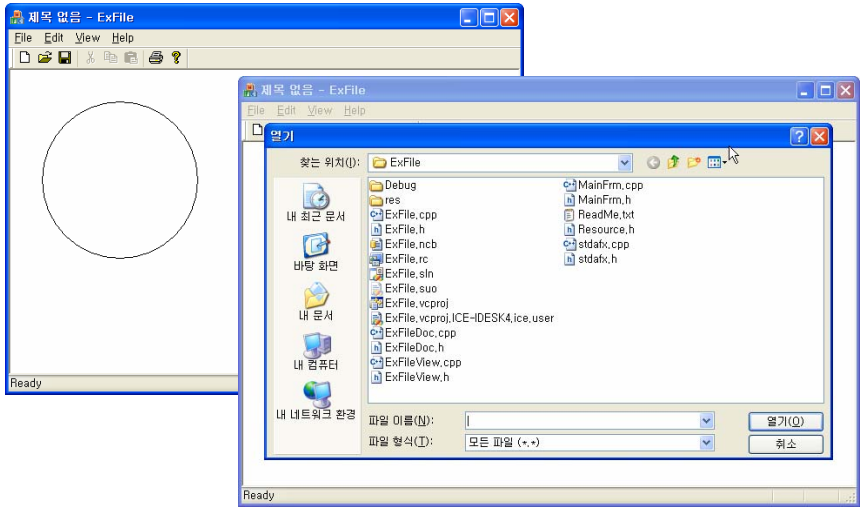
도큐먼트/뷰 구조 응용 프로그램 예제 작성

- 코드 추가

```
void CExFileView::OnDraw(CDC* pDC)
{
    CExFileDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->SetMapMode(MM_LOMETRIC);          // 그림을 추가한다
    pDC->Ellipse(100, -100, 600, -600);
}
```

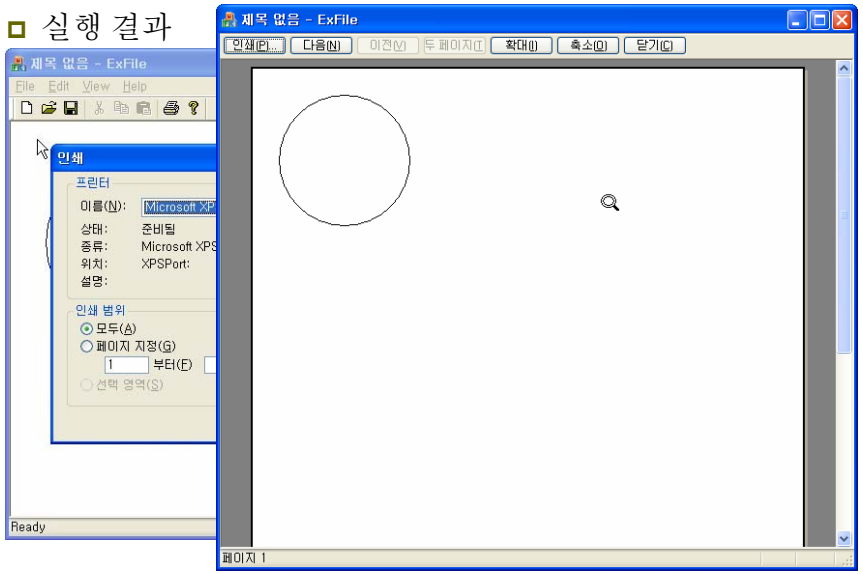

도큐먼트/뷰 구조 응용 프로그램 예제 작성

- 실행 결과



도큐먼트/뷰 구조 응용 프로그램 예제 작성

- 실행 결과



도큐먼트/뷰 구조 응용 프로그램 예제 분석

- 응용 프로그램 클래스

① 메뉴 명령 핸들러 [File->New, Open, Print 등]

```
BEGIN_MESSAGE_MAP(CExFileApp, CWinApp)
   //{{AFX_MSG_MAP(CExFileApp)
   ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
   //}}AFX_MSG_MAP
① ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
   ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
   ON_COMMAND(ID_FILE_PRINT_SETUP,
                                   CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

BOOL CExFileApp::InitInstance()
{
② CSingleDocTemplate* pDocTemplate;
   pDocTemplate = new CSingleDocTemplate(
```

② document template 객체를 동적으로 생성 후 응용프로그램 객체에 등록

도큐먼트/뷰 구조 응용 프로그램 예제 분석

```
      IDR_MAINFRAME,
      RUNTIME_CLASS(CExFileDoc),
      RUNTIME_CLASS(CMainFrame),
      RUNTIME_CLASS(CExFileView));
   AddDocTemplate(pDocTemplate);

③ CCommandLineInfo cmdInfo;
   ParseCommandLine(cmdInfo);

④ if (!ProcessShellCommand(cmdInfo))
      return FALSE;

⑤ m_pMainWnd->ShowWindow(SW_SHOW);
   m_pMainWnd->UpdateWindow();

   return TRUE;
}
```

③ 명령행 인자를 분석 후 결과를 cmdInfo 에 저장

④ cmdInfo 에 따라 처리 [document, frame, view 객체가 생성된다]

⑤ Frame window가 화면에 보이게 한 후 UpdateWindow를 호출하여 WM_PAINT 가 호출되게 함

도큐먼트/뷰 구조 응용 프로그램 예제 분석

- 프레임 윈도우 클래스

```
// 헤더 파일
class CMainFrame : public CFrameWnd
{
...
protected:
① DECLARE_DYNCREATE(CMainFrame)
...
}

// 구현 파일
...
② IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
...
```


도큐먼트/뷰 구조 응용 프로그램 예제 분석

- 뷰 클래스

```
// 헤더 파일
class CExFileView : public CView
{
protected:
   CExFileView();
① DECLARE_DYNCREATE(CExFileView)  ① 동적 객체 생성기능을 지원

public:
② CExFileDoc* GetDocument();
...
}  ② ③ 데이터를 읽어오거나 저장하기 위하여 document에 접근해야함

#ifndef _DEBUG
③ inline CExFileDoc* CExFileView::GetDocument()
   { return (CExFileDoc*)m_pDocument; }
#endif
```

- CDocument type의 포인터
- 언제든지 뷰에서 document를 참조할 수 있게 함

도큐먼트/뷰 구조 응용 프로그램 예제 분석

④ 동적 객체 생성기능을 지원

```
// 구현 파일
④ IMPLEMENT_DYNCREATE(CExFileView, CView)

BEGIN_MESSAGE_MAP(CExFileView, CView)
   //{{AFX_MSG_MAP(CExFileView)
   //}}AFX_MSG_MAP
⑤ ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_PREVIEW,
    CView::OnFilePrintPreview)
END_MESSAGE_MAP()
...
```

⑤ 인쇄, 미리 보기 기능을 기본적으로 제공

도큐먼트/뷰 구조 응용 프로그램 예제 분석

⑥ OnPaint() 대신 여기서는 OnDraw()사용
여기서는 Device Context 객체를 생성할 필요가 없다. 화면 출력뿐아니라, 인쇄/미리보기에 사용

```
⑥ void CExFileView::OnDraw(CDC* pDC)
{
   CExFileDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   pDC->SetMapMode(MM_LOMETRIC);
   pDC->Ellipse(100, -100, 600, -600);
}
...
⑦ CExFileDoc* CExFileView::GetDocument()
{
   ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CExFileDoc)));
   return (CExFileDoc*)m_pDocument;
}
```

⑦ 디버그 버전으로 컴파일할 때 사용됨

도큐먼트/뷰 구조 응용 프로그램 예제 분석

- 도큐먼트 클래스

```
// 헤더 파일
class CExFileDoc : public CDocument
{
protected:
   CExFileDoc();
① DECLARE_DYNCREATE(CExFileDoc)

   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CExFileDoc)
   public:
②  virtual BOOL OnNewDocument();
③  virtual void Serialize(CArchive& ar);
   //}}AFX_VIRTUAL
```


도큐먼트/뷰 구조 응용 프로그램 예제 분석

```
public:
   virtual ~CExFileDoc();
#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif

   DECLARE_MESSAGE_MAP()
};

// 구현 파일
④ IMPLEMENT_DYNCREATE(CExFileDoc, CDocument)

BEGIN_MESSAGE_MAP(CExFileDoc, CDocument)
END_MESSAGE_MAP()
```

도큐먼트/뷰 구조 응용 프로그램 예제 분석

```
CExFileDoc::CExFileDoc() { }

CExFileDoc::~CExFileDoc() { }

⑤ BOOL CExFileDoc::OnNewDocument()
{
   if (!CDocument::OnNewDocument())
      return FALSE;

   return TRUE;
}

⑥ void CExFileDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring()) { }
```


도큐먼트/뷰 구조 응용 프로그램 예제 분석

```
   else { }
}

#ifdef _DEBUG
void CExFileDoc::AssertValid() const
{
   CDocument::AssertValid();
}

void CExFileDoc::Dump(CDumpContext& dc) const
{
   CDocument::Dump(dc);
}
#endif //_DEBUG
```


직렬화 기초

- 직렬화
  - 영속적인 저장 매체에 객체의 내용을 저장하거나 읽어오는 과정

직렬화 기초

- 데이터 쓰기 - 일반 파일 입출력

```
CFile file;
CFileException e;
if(!file.Open("test.dat", CFile::modeReadWrite|CFile::modeCreate, &e))
{
   e.ReportError();
   return
}

int a = 100;
int b = 200;
file.Write(&a, sizeof(a));
file.Write(&b, sizeof(b));
```

직렬화 기초

- 데이터 쓰기 - 직렬화

```
CFile file;
CFileException e;
if(!file.Open("test.dat", CFile::modeReadWrite|CFile::modeCreate, &e))
{
   e.ReportError();
   return;
}

int a = 100;
int b = 200;
CArchive ar (&file, CArchive::store);
ar << a << b;
```


직렬화 기초

- 데이터 읽기 - 일반 파일 입출력

```
CFile file;
CFileException e;
if(!file.Open("test.dat", CFile::modeRead, &e))
{
   e.ReportError();
   return;
}

int a, b;
file.Read(&a, sizeof(a));
file.Read(&b, sizeof(b));
TRACE("a = %d, b = %d\n", a, b);
```


직렬화 기초

- 데이터 읽기 - 직렬화

```
CFile file;
CFileException e;
if(!file.Open("test.dat", CFile::modeRead, &e))
{
   e.ReportError();
   return;
}

int a, b;
CArchive ar (&file, CArchive::load);
ar >> a >> b;
TRACE("a = %d, b = %d\n", a, b);
```


직렬화 기초

- CArchive 클래스 생성자

```
CArchive::CArchive (CFile* pFile, UINT nMode, int nBufSize = 4096,
   void* lpBuf = NULL) ;
```

  - pFile
    - CFile 객체의 주소
  - nMode
    - CArchive::load 또는 CArchive::store
  - nBufSize
    - 내부에서 사용할 버퍼 크기
  - lpBuf
    - 사용자 정의 버퍼의 주소

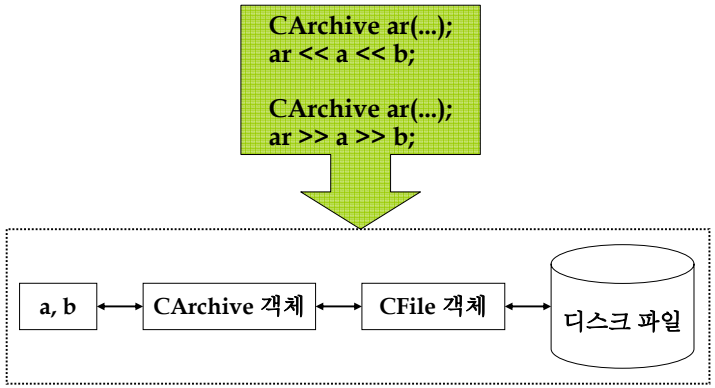
직렬화 기초

- 직렬화 가능한 데이터 타입

구분	데이터 타입
기본형	BYTE, WORD, LONG, DWORD, float, double, int, short, char, wchar_t, unsigned, bool, ULONGLONG, LONGLONG
비 기본형	RECT, POINT, SIZE, CRect, CPoint, CSize, CString, CTime, CTimeSpan, COleVariant, COleCurrency, COleDateTime, COleDataTimeSpan

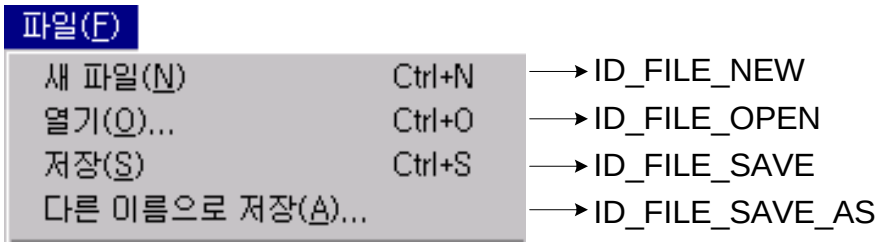
직렬화 기초

- 직렬화 원리

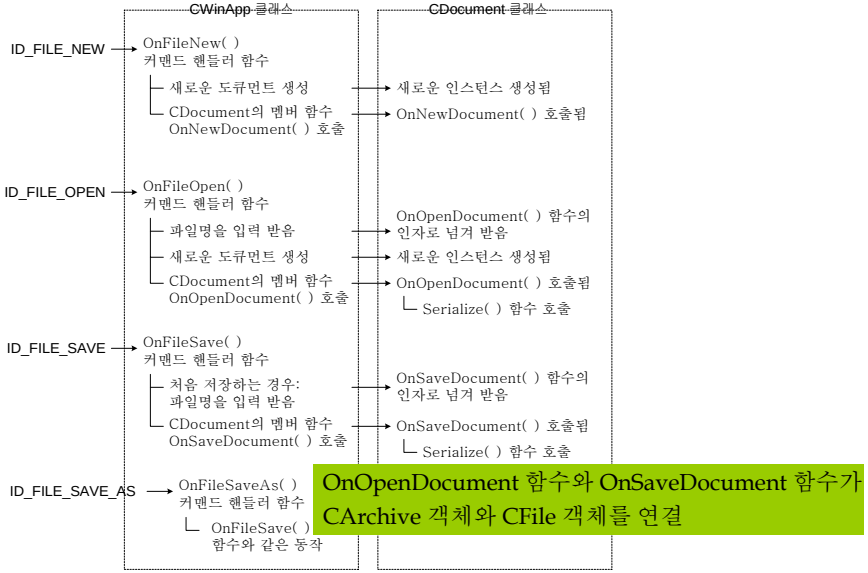


도큐먼트/뷰 구조 직렬화 - 파일 메뉴 처리 기능

- ID_FILE_NEW, ID_FILE_OPEN, ID_FILE_SAVE, ID_FILE_SAVE_AS 는 자동으로 처리
- CWinApp에서 전처리를 하고 CDocument 클래스로 넘긴다



도큐먼트/뷰 구조 직렬화 - 파일 메뉴 처리 기능



도큐먼트/뷰 구조와 직렬화

- [파일]->[열기...] 메뉴를 선택한 경우

```
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
```

```
BOOL CDocument::OnOpenDocument(LPCTSTR lpszPathName)
{
   // CFile 객체 생성. pFile은 CFile 객체의 주소값이다.
   ...
   CArchive ar(pFile, CArchive::load|CArchive::bNoFlushOnDelete);
   ...
   Serialize(ar)
   ...
}
```


도큐먼트/뷰 구조와 직렬화

- [파일]->[저장] 또는 [다른 이름으로 저장...] 메뉴를 선택한 경우

```
ON_COMMAND(ID_FILE_SAVE, OnFileSave)
ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
```

```
BOOL CDocument::OnSaveDocument(LPCTSTR lpszPathName)
{
   // CFile 객체 생성. pFile은 CFile 객체의 주소값이다.
   ...
   CArchive ar(pFile, CArchive::store|CArchive::bNoFlushOnDelete);
   ...
   Serialize(ar)
   ...
}
```


직렬화 클래스 구현

- 다음과 같은 사용자 정의 클래스가 있는 경우

```
class CMyData
{
public:
   CString m_str;
   COLORREF m_color;
public:
   CMyData(CString &str, COLORREF &color)
   {
      m_str = str; m_color = color;
   }
   virtual ~CMyData();
};
```


직렬화 클래스 구현

- 단순히 CExFileDoc 클래스에서 멤버변수 m_data (CMyData 타입)을 다음과 같이 직렬화할 수 **"없음"**

```
void CExFileDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      ar << m_data;
   }
   else
   {
      ar >> m_data;
   }
}
```

직렬화 클래스 구현

- 직렬화를 사용하기 위해 사용자 정의 클래스 변경해야 함

```
// 클래스 선언부
class CMyData : public CObject ①
{
   DECLARE_SERIAL(CMyData) ②
public:
   CString m_str;
   COLORREF m_color;
public:
   CMyData() { } ③
   CMyData(CString &str, COLORREF &color) {
      m_str = str; m_color = color;
   }
   virtual ~CMyData();
   virtual void Serialize(CArchive& ar); ④
};
```


직렬화 클래스 구현

- 사용자 정의 클래스 변경

```
// 클래스 구현부
CMyData::~CMyData()
{
}

IMPLEMENT_SERIAL(CMyData, CObject, 1) ⑤

void CMyData::Serialize (CArchive& ar) ⑥
{
   CObject::Serialize(ar);
   if(ar.IsStoring())
      ar << m_str << m_color;
   else
      ar >> m_str >> m_color;
}
```


직렬화 클래스 구현

- 직렬화 ⇨ 사용자 정의 클래스의 Serialize를 사용해야 함

```
void CExFileDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      m_data.Serialize(ar);
   }
   else
   {
      m_data.Serialize(ar);
   }
}
```


CFileDialog 클래스

- CFileDialog 클래스의 인스턴스를 선언하고, DoModal 함수를 호출

```
char szFilter[] = "Image (*.BMP, *.GIF, *.JPG)|*.BMP;*.GIF;*.JPG |All Files(*.*)|*.*||";

CFileDialog dlg(TRUE, NULL, NULL, OFN_HIDEREADONLY, szFilter);
if(IDOK == dlg.DoModal())
{
   CString strPathName = dlg.GetPathName();
}
```