MFC Message Map

HCI Programming 2 (321190)
2008년 가을학기
10/14/2008
박경신

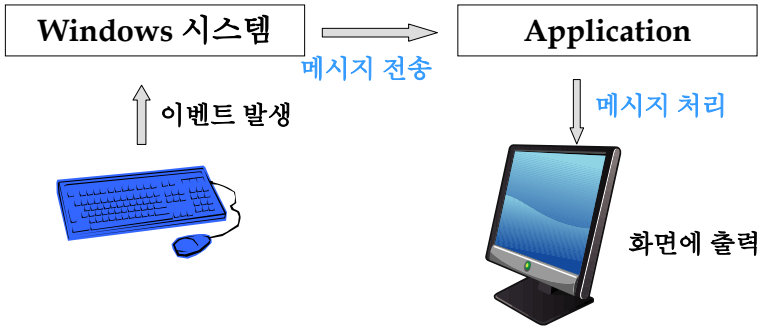
Windows와 C++

- C++의 다형성 (Polymorphsim)
  - 기반클래스의 어떤 멤버 함수를 파생클래스에서 재정의 (overriding)하기 위해서는 기반클래스의 그 멤버함수가 가상함수 (virtual function)로 정의되어야 함
- MFC의 CWnd에서 메시지 핸들러 함수는 가상함수로 정의되어 있지 않고 메시지 맵을 사용
  - 모든 메시지 핸들러 함수가 가상함수로 되어있어 동적바인딩을 하게 된다면 메모리를 많이 차지하게 되는 자원낭비의 문제 발생
  - 윈도우 메시지의 수와 종류가 바뀌고 있으므로 메시지 핸들러 함수를 가상함수로 사용하는 것은 메시지가 변경될 때 코드를 쓸모없게 만듬
  - 그래서, 효율적이고, 확장 가능하고, 컴파일러에 의존적이지 않은 메시지 처리방법인 메시지 맵을 개발
  - 메시지 맵이란 일종의 매크로로, 간단하게 윈도우 메시지와 명령어를 클래스의 멤버함수와 연결(map) 시켜주는 것임

Message Map

- MFC에서는 SDK의 switch문을 대체하는 방법으로 메시지 맵이란 방법을 사용
- 메시지 맵이란 특정 메시지와 그 메시지가 발생했을 경우 취해야 할 동작을 하나의 쌍으로 연결시켜 놓은 테이블
- 미리 정의된 메시지 핸들러를 이용하면 특정 메시지를 받을 때마다 그 메시지에 연결된 핸들러가 호출
- 메시지는 자신을 나타내는 식별자와 함께 추가적인 정보를 메시지 핸들러의 인자로 전달

Message Driven Programming



Message

- 메시지는 프로그램에 변화가 생겼을 때 Windows가 프로그램에게 알리는 정보
- 메시지는 MSG로 정의되는 구조체
  - 윈도우 핸들, 메시지 식별 번호, 추가 정보, 시간, 커서 위치 등 포함

```
typedef struct tagMSG {
        HWND hwnd;          // 메시지가 발생한 윈도우 핸들
        UINT message;       // message id
        WPARAM wParam;      // 추가 정보
        LPARAM lParam;      // 추가 정보
        DWORD time;         // 메시지 발생 시간
        POINT pt;           // 커서 위치
} MSG;
```

Message 종류

- 메시지 발생 주체에 따라
  - 사용자에 의해 발생되는 메시지
    - 사용자가 하는 동작 (e.g. 마우스 클릭, 마우스 이동, 키보드 누름 등)을 윈도우가 이해할 수 있는 메시지로 만들어서 응용프로그램에 전달
  - 시스템에 의해 발생되는 메시지
    - 윈도우에 의해서 메시지가 발생되는 경우 (e.g. 특정 윈도우의 클라이언트 영역을 지울 필요가 있다는 사실을 발견하게 되면 WM_PAINT 메시지 활성)

Message 종류

- 메시지 처리 주체에 따라
  - 윈도우 (Window) 메시지
    - WM_COMMAND를 제외한 WM_로 시작하는 모든 메시지
  - 통지 (Notification) 메시지
    - 자식 윈도우로부터 부모 윈도우를 향해 알리는 메시지
  - 명령 (Command) 메시지
    - 사용자 인터페이스에서 발생되는 WM_COMMAND 메시지
  - 사용자 정의 메시지

Message 종류

- 윈도우 메시지 (Window Message)
  - WM_로 시작하는 메시지 (WM_COMMAND는 제외)
  - 매개 변수를 통하여 메시지를 어떻게 처리할 것인지를 결정
  - 윈도우 관리 메시지 : 윈도우의 상태가 바뀔 때 발생
    - WM_PAINT, WM_ACTIVE, WM_SIZE, WM_MOVE, WM_CREATE, WM_DESTORY
  - 초기화 메시지 : 응용 프로그램이 대화상자를 시작할 때 발생
    - WM_INITDIALOG
  - 입력 메시지 : 마우스, 키보드로 입력할 때 발생
    - WM_KEYDOWN, WM_CHAR
    - WM_MOUSEMOVE, WM_LBUTTONDOWN, …

Message 종류

- 컨트롤 통지 메시지 (Control Notification Message)
  - Button, Combo Box와 같은 컨트롤 객체나 자식 윈도우에서 부모 윈도우로 보내는 메시지
  - BN_CLICKED, EN_CHANGE, CBN_SELCHANGE, LBN_SELCHANGE, …
- 명령 메시지 (Command Message)
  - 메뉴, 툴바, 엑셀레이터 키와 같은 사용자 인터페이스 객체로부터 발생되는 WM_COMMAND 메시지
  - 명령 메시지는 윈도우 뿐만 아니라 도큐먼트, 도큐먼트 템플릿, 뷰, 다른 애플리케이션 객체에 의해서도 발생가능
  - 어떤 메뉴가 눌렸는지 구별하기 위해서 메뉴의 ID가 윈도우 메시지의 WPARAM을 통해 전달

Message 처리 방식

- SDK 프로그램
  - 들어온 메시지를 switch문을 사용하여 처리
- MFC 프로그램
  - 메시지 처리를 위해 메시지 맵을 사용
  - 메시지 핸들러 함수 구현
- 메시지 맵
  - 메시지 번호와 메시지가 발생하였을 때 호출되는 함수의 포인터 등의 정보를 갖고 있는 테이블
  - 프로그램에 전달된 메시지와 메시지 핸들러 함수를 연결하는데 사용
  - 파생 클래스의 메시지 핸들러 함수가 우선

Message Map 메시지 처리 단계

1. 윈도우 클래스의 멤버 함수로 메시지 핸들러 함수를 선언한다
2. 메시지 맵에 메시지와 메시지 핸들러 함수를 묶는 메시지 맵 매크로를 추가한다
3. 메시지 핸들러 함수의 기능을 구현한다


```
//(1) 메시지 핸들러  함수 선언 (CWinmsgView.h)
class CWinmsgView : public CView
{
  protected:
            //{{AFX_MSG(CWinmsgView)
            afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
            //}}AFX_MSG
            DECLARE_MESSAGE_MAP()
  …
}
//(2) 메시지 맵 매크로 (CWinmsgView.cpp)
BEGIN_MESSAGE_MAP(CWinmsgView, CView)
            //{{AFX_MSG_MAP(CWinmsgView)
            ON_WM_LBUTTONDOWN()
            //}}AFX_MSG_MAP
END_MESSAGE_MAP()
//(3)메시지 처리 함수 구현 ((CWinmsgView.cpp)
void CWinmsgView::OnLButtonDown(UINT nFlags, CPoint point)
{
            AfxMessageBox("마우스 왼쪽 버튼 누름") ; // 메시지 처리 루틴
            CView::OnLButtonDown(nFlags, point);
}
```

Message Handler

- 윈도우로부터 애플리케이션에 메시지가 전달될 때 해당 메시지를 처리하는 멤버 함수
- 함수 이름
  - 메시지 핸들러는 윈도우 메시지의 WM_ 대신에 On을 붙여 시작
  - 함수 선언 시 afx_msg는 메시지 핸들러 함수 표시
- 예

```
//{{AFX_MSG(CMyView)
        afx_msg void OnKeyDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```


DECLARE_MESSAGE_MAP

- DECLARE_MESSAGE_MAP 매크로 – afxwin.h

```
#define DECLARE_MESSAGE_MAP() \
private: \
    static const AFX_MSGMAP_ENTRY _messageEntries[]; \
protected: \
    static AFX_DATA const AFX_MSGMAP messageMap; \
    virtual const AFX_MSGMAP* GetMessageMap() const; \
```


DECLARE_MESSAGE_MAP

```
struct AFX_MSGMAP_ENTRY { // 메시지맵 항목을 정의하기 위해 사용
    UINT nMessage;      // 시스템을 통해 들어오는 윈도우 메시지 ID
    UINT nCode;         // 제어 코드나 WM_NOTIFY 코드를 나타냄
    UINT nID;           //  메시지를 생성한 콘트롤 ID
    UINT nLastID;       // 콘트롤 식별자의 범위를 나타내는 엔트리를
                        // 위해서 사용
    UINT nSig;          // 메시지를 다루는 핸들러 함수의 signature를 표시
    AFX_PMSG pfn;       // 메시지를 처리하는 핸들러 함수를 가리킴
};

struct AFX_MSGMAP { // 메시지맵에 등록된 메시지핸들러 호출
    const AFX_MSGMAP* pBaseMap;    // base class의 메시지 맵
    const AFX_MSGMAP_ENTRY* lpEntries;      // 링크 리스트
};
```


Message Map Macro

- 메시지 맵은 BEGIN_MESSAGE_MAP과 END_MESSAGE_MAP 사이에 메시지 엔트리들로 구성

```
BEGIN_MESSAGE_MAP(CMainWnd, CFrameWnd)
        ON_WM_PAINT()
        ON_WM_LBUTTONDOWN()
        ON_WM_KEYDOWN()
END_MESSAGE_MAP()
```

파생클래스

기반 클래스

Message Map Macro

- BEGIN_MESSAGE_MAP(CDerivedClass, CBaseClass)
  - 만약 CDerivedClass의 메시지 맵에서 일치하는 것을 발견하지 못하면 CBaseClass에 지정된 기반 클래스의 메시지 맵에서 찾게 됨
  - 기반 클래스의 메시지 맵에서 일치하는 메시지 핸들러가 발견되지 않는다면 메시지에 대한 디폴트 처리가 수행됨 – 만약 윈도우 메시지라면 메시지는 연결된 디폴트 윈도우 메시지 프로시져로 보내짐
  - 기반 클래스에서 제공된 메시지 핸들러도 결국 파생 클래스에 의해 상속됨 – 바로 이 점이 메시지 핸들러를 가상 함수로 만들지 않고도 가상 함수와 유사한 기능을 하게 만듦

BEGIN/END_MESSAGE_MAP

- BEGIN_MESSAGE_MAP & END_MESSAGE_MAP 매크로

```
#define BEGIN_MESSAGE_MAP(theClass, baseClass) \
    const AFX_MSGMAP* theClass::GetMessageMap() const \
    { \
            return &theClass::messageMap; \
    } \
    AFX_DATADEF const AFX_MSGMAP theClass::messageMap = \
    { &baseClass::messageMap, &theClass::_messageEntries[0] }; \
    const AFX_MSGMAP_ENTRY theClass::_messageEntries[] = \
    { \
```

_messageEntries[]에 메시지 맵 항목을 넣을 수 있게 준비

```
#define END_MESSAGE_MAP() \
            {0, 0, 0, 0, AfxSig_end, (AFX_PMSG)0 } \
    };
```

메시지 엔트리 배열 닫기

Message Map Entries

- 메시지 매크로들은 ON_으로 시작하는 것을 제외하고는 표준 윈도우 메시지의 이름과 동일
  - WM_LBUTTONDOWN 메시지에 해당하는 메시지 매크로는 ON_WM_LBUTTONDOWN() 임
  - 한가지 예외로는 WM_COMMAND 메시지는 ON_COMMAND라는 매크로를 사용

```
#define ON_WM_LBUTTONDOWN() \
 { WM_LBUTTONDOWN, 0, 0, 0, AfxSig_vwp, \
 (AFX_PMSG)(AFX_PMSGW)(void (AFX_MSG_CALL CWnd::*)(UINT, CPoint))&OnLButtonDown },
#define ON_WM_LBUTTONUP() \
 { WM_LBUTTONUP, 0, 0, 0, AfxSig_vwp, \
 (AFX_PMSG)(AFX_PMSGW)(void (AFX_MSG_CALL CWnd::*)(UINT, CPoint))&OnLButtonUp },
```


Message Map Entries

윈도우 메시지		ON_WM_XXXX
명령 메시지		ON_COMMAND
명령 범위 메시지		ON_COMMAND_RANGE
사용자 정의 메시지		ON_MESSAGE
통지 메시지		ON_CONTROL
일반적인 컨트롤	버튼	ON_BN_XXX
	에디트	ON_EN_XXX
	리스트	ON_LBN_XXX
	콤보박스	ON_CBN_XXX

윈도우 관리 메시지와 메시지 핸들러

메시지 유형	발생 상황	메시지 핸들러 함수
WM_CREATE	윈도우가 생성될 때	OnCreate()
WM_ACTIVE	윈도우가 활성화 될 때	OnActive()
WM_PAINT	윈도우가 다시 그려질 때	OnPaint(), OnDraw()
WM_SIZE	윈도우 크기가 변경될 때	OnSize()
WM_MOVE	윈도우가 움직일 때	OnMove()
WM_TIMER	설정된 타이머 시간이 됐을 때	OnTimer()
WM_DESTROY	윈도우가 종료될 때	OnDestroy()


```
//메시지 핸들러 함수 선언( CWinmsgView.h)
class CWinmsgView : public CView {
protected:
        //{{AFX_MSG(CWinmsgView)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnDestroy();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP() …
}
//메시지맵 매크로 (CWinmsgView.cpp)
BEGIN_MESSAGE_MAP(CWinmsgView, CView)
        //{{AFX_MSG_MAP(CWinmsgView)
        ON_WM_CREATE()
        ON_WM_SIZE()
        ON_WM_DESTROY()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
//메시지 핸들러 (CWinmsgView.cpp)
void CWinmsgView::OnDraw(CDC* pDC) {
  CWinmsgDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  // 윈도우 크기를 나타내는 문자열을 윈도우 중앙에 출력
  CRect rectView;  GetClientRect(&rectView);
  pDC->DrawText(m_strWindowSize, rectView, DT_SINGLELINE | DT_CENTER |
DT_VCENTER);
}
```

```
//메시지 핸들러 (CWinmsgView.cpp)
int CWinmsgView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CView::OnCreate(lpCreateStruct) == -1)
                return -1;
        // 윈도우가 생성될 때 메시지 박스 출력
        AfxMessageBox("뷰윈도우가 생성되었습니다.");
        return 0;
}

void CWinmsgView::OnSize(UINT nType, int cx, int cy)
{
        CView::OnSize(nType, cx, cy);
        // 윈도우 크기가 변경될 때 윈도우 크기를 나타내는 문자열 생성
        m_strWindowSize.Format("윈도우 크기는 넓이 %d, 높이 %d입니다.", cx, cy);
        Invalidate();  // 화면 갱신

}
void CWinmsgView::OnDestroy()
{
        CView::OnDestroy();
        // 윈도우가 종료될 때 메시지 박스 출력
        AfxMessageBox("뷰윈도우가 종료되었습니다.");
}
```


Timer 메시지

- 지정된 시간마다 WM_TIMER 메시지를 발생
- 타이머 설정 함수 - SetTimer

```
UINT SetTimer( UINT nIDEvent, UINT nElapse,
    void (CALLBACK EXPORT* lpfnTimer) (HWND, UINT, UINT, WORD) );
```

  - nIDEvent : 타이머 ID
  - nElapse : WM_TIMER 메시지를 발생시킬 시간 간격
    (단위: mili second = 1/1000초, 1초 = 1000 ms)
  - lpfnTimer() : 설정된 시간 마다 호출되는 함수명, NULL로 설정되면 OnTimer()함수가 호출

- 예

```
SetTimer(0, 1000, NULL);  // 1초당 WM_TIMER 메시지 발생
                          //메시지 처리를 위해 OnTimer()함수 호출
```

Timer 메시지

- 타이머 해제 함수 - KillTimer

  UINT KillTimer(UINT nIDEvent)
  - nIDEvent : SetTimer에서 설정된 Timer의 ID.
- 타이머 message handler - OnTimer
  - WM_TIMER 메시지 처리함수

```
// WM_TIMER 메시지 발생에 대한 처리 내용
void CWinmsgView::OnTimer(UINT nIDEvent)
{
        CClientDC dc(this); //현재윈도우의 클라이언트 영역의 DC얻음
        CTime timer;                    //시간객체 생성
        timer = CTime::GetCurrentTime(); //현재시간 얻음
        CString strTimer;               //출력 문자열 선언
        strTimer = timer.Format("현재시간 < %H:%M:%S >");
                                        //현재시간을 "시:분:초" 형식으로 문자열에 대입
        dc.TextOut(10, 10, strTimer);  //클라이언트 영역에 현재시간을 출력
        CView::OnTimer(nIDEvent);
}
```


사용자 정의 Message

- MFC가 제공하지 않는 윈도우 메시지
  - 만약 WM_MYMESSAGE라는 새로운 메시지를 만들고자 한다면 #define문을 이용하여 WM_USER이후의 값을 선언함

    #define WM_MYMESSAGE WM_USER+1
  - 해당 메시지에 대한 메시지 핸들러의 원형을 헤더부에 설정함

    afx_msg void OnMyMessage(WPARAM wParam, LPARAM lParam);
  - BEGIN_MESSAGE_MAP/END_MESSAGE_MAP 사이에 ON_MESSAGE 매크로를 이용하여 정의한 메시지명과 함수명을 연결

    ON_MESSAGE(WM_MYMESSAGE, OnMyMessage)
  - 메시지 핸들러 함수를 구현함

```
void CMyView::OnMyMessage(WPARAM wParam, LPARAM lParam)
{
  CString p = (LPCSTR) lParam;
  // .....
}
```


사용자 정의 Message

- SendMessage를 사용해서 사용자정의 메시지를 발생
  - 사용자가 만든 메시지는 사용자가 직접 메시지를 발생시킴
  - WM_MYMESSAGE를 생성하기 위해서 SendMessage()를 사용함

    SendMessage(WM_MYMESSAGE, 0, 0);

```
LRESULT SendMessage ( UINT message, WPARAM wParam = 0,
                      LPARAM lParam = 0);
Message : 보내고자 하는 Message ID
wParam : 메시지 정보와 함께 보내주는 WPARAM 인자
lParam : 메시지 정보와 함께 보내주는 lParam 인자
```

```
void CUserTestView::OnLButtonDown(UINT nFlags, Cpoint point)
{
    CString temp = "사용자 메시지 전송";
    SendMessage(WM_MYMESSAGE, 0, (LPARAM)(LPCSTR) temp);
    CView::OnLButtonDown(nFlags, point);
}
```


자주 사용되는 Message Handler

메시지	메시지 핸들러
WM_CREATE	afx_msg int OnCreate(LPCREATESTRUCT);
WM_CLOSE	afx_msg void OnClose();
WM_CHAR	afx_msg void OnChar(UINT, UINT, UINT);
WM_KEYDOWN	afx_msg void OnKeyDown(UINT, UINT, UINT);
WM_LBUTTONDOWN	afx_msg void OnLButtonDown(UINT, CPoint);
WM_MOUSEMOVE	afx_msg void OnMouseMove(UINT, CPoint);
WM_MOVE	afx_msg void OnMove(int, int);
WM_PAINT	afx_msg void OnPaint();
WM_SETFOCUS	afx_msg void OnSetFocus(CWnd*);
WM_SIZE	afx_msg void OnSize(UINT, int, int);
WM_TIMER	afx_msg void OnTimer(UINT);
WM_ERASEBKGND	afx_msg BOOL OnEraseBkgnd(CDC*);
WM_HSCROLL	afx_msg void OnHScroll(UINT, UINT¿¤CWnd*);

메시지 박스

- AfxMessageBox() 함수
  - 사용자에게 간단한 메시지를 출력하는데 사용되는 대화상자
  - 함수 원형
    - Int AfxMessageBox(LPCTSTR lpszText, UINT nType = MB_OK, UINT nIDHelp = 0)
      - lpszText : 출력하고자 하는 문자열
      - nType : 대화상자에 설정되는 버튼
      - nIDHelp : 도움말(F1)을 실행하였을 때의 도움말 ID
  - 디폴트 메시지 박스 스타일과 아이콘
    - 메시지 박스 : MB_OK
    - 아이콘 : MB_ICONEXCLAMATION

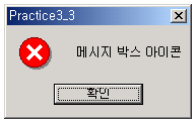
메시지 박스

- 메시지 박스 스타일과 반환 값

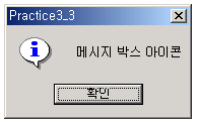
메시지 박스 스타일	사용 가능한 버튼	반환 값
MB_OK	확인	IDOK
MB_OKCANCEL	확인, 취소	IDOK,IDCANCEL
MB_YESNO	예, 아니오	IDYES,IDNO
MB_YESNOCANCEL	예, 아니오, 취소	IDYES,IDNO, IDCANCEL
MB_RETRYCANCEL	재시도, 취소	IDRETRY,IDCANCEL
MB_ABORTRETRYIGNORE	취소, 재시도, 무시	IDABORT,IDRETRY,IDIGNORE

메시지 박스

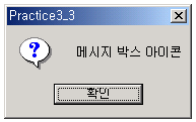
- 아이콘 스타일



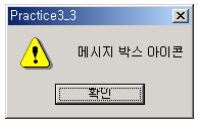
[위험] MB_ICONHAND
또는 MB_ICONSTOP



[정보] MB_ICONINFORMATION
또는 MB_ICONASTERISK



[물음] MB_ICONQUESTION



[경고] MB_ICONEXCLAMATION

메시지 박스

```
//메시지박스 사용 예
int rv;
rv = AfxMessageBox("확인하시겠습니까?", MB_YESNOCANCEL |
                                        MB_ICONQUESTION);
switch(rv){
  case IDYES:
        AfxMessageBox("예를 눌렀습니다."); break;
  case IDNO:
        AfxMessageBox("아니오를 눌렀습니다."); break;
  case IDCANCEL:
        AfxMessageBox("취소를 눌렀습니다."); break;
}
```