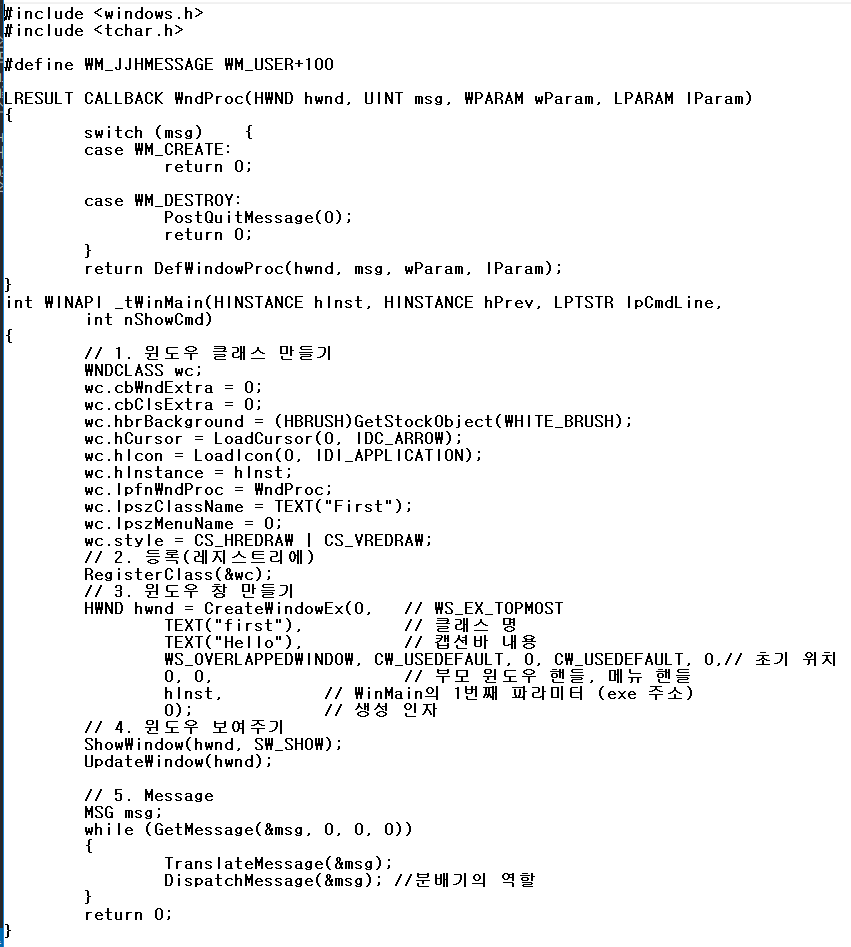
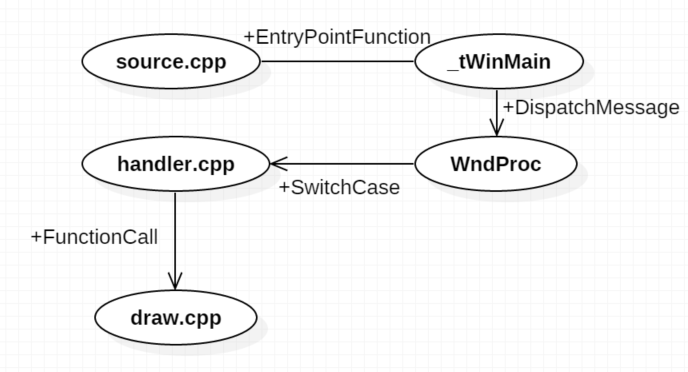
Window API \_Code 정리

* **Skelton Code Pattern**
  + Basic Code Pattern

Source.cpp



* + Bit27기 Code Pattern



* **Window Application System 실행 과정**

1. OS 에서 Application 실행
   1. Application handle이 분배기의 Application Queue에 전달됨
   2. Application에서 Message를 Message Queue에 보냄
   3. 분배기가 Message Queue에서 Message를 추출 후 분석
   4. 분석한 Message 값을 Application Queue에 있는 핸들 리스트에 대조해서 적절한 Application에 메시지 값을 전달함
2. Application 내부 상황
   1. GetMessage를 통해 Message Queue에서 Message를 달라고 요청함
   2. 1.C를 수행한 뒤 Message를 전달받아 MSG 타입의 변수에 저장함
   3. 받은 Message를 자신이 생성한 Application class의 프로시저 함수로 보내기 위해서 DispatchMessage를 이용함 이 함수는 윈도우 프로시저로 메시지를 전달하여 분배기가 메시지를 Application class에 setting한 프로시저에 전달 할 수 있도록 한다.
   4. 프로시저 함수 단에서 Message 값에 맞춰 처리함
   5. 처리하지 못한 Message 값은 DefWindowProc에 전달해서 처리함
   6. 처리가 완료되면 while을 이용한 루프를 만들지 않으면 프로그램이 종료하게된다.

* **Message에 대해서**
  + How to make Custom Message
    - WM\_USER 값 이후의 값들을 Custom Message로 이용 할 수 있음
    - #define 만들고싶은이름 WM\_USER + 양의정수값 으로 메시지 값을 설정 할 수 있다.
    - 보통 IPC 통신을 하려고 할 때 사용된다.
    - ~~그래봐야 숫자 놀이….~~
  + Function

|  |  |  |  |
| --- | --- | --- | --- |
| 함수 이름 | 반환형 | 매개변수 | 내용 |
| GetMessage | BOOL | LPMSG lpMsg,  HWND  hWnd,  UINT  wMsgFilterMin,  UINT  wMsgFilterMax | 메시지 큐에서 메시지를 가져오는 함수,메시지가 없으면 무한 대기 한다.  ipMsg에 MSG 타입의 변수 주소를 넣는다. 나머지는 0으로 해도 좋다. |
| TranslateMessage | BOOL | const MSG \* lpMsg | 가상 키 메시지를 문자 메시지로 변환한다. |
| DispatchMessage | LRESULT | const MSG \*lpmsg | 받은 메시지를 윈도우 프로시저에 전달한다. |
| SendMessage | LRESULT | \_In\_ HWND   hWnd,  \_In\_ UINT   Msg,  \_In\_ WPARAM wParam,  \_In\_ LPARAM lParam | hWnd의 핸들 값을 가진 윈도우에 Msg 값을 전달하는 함수 wParam, lParam에 값을 추가해서 전달 할 수 있다. |
| PostMessage | BOOL | \_In\_opt\_ HWND   hWnd,  \_In\_     UINT   Msg,  \_In\_     WPARAM wParam,  \_In\_     LPARAM lParam | SendMessage와 하는 짓은 같다 하지만 이 값은 bool 값을 성공값을 반환한다. 또 메시지를 던져 놓고 다른 작업을 할 수 있다. |
| PeekMessage | BOOL | \_Out\_    LPMSG lpMsg,  \_In\_opt\_ HWND  hWnd,  \_In\_     UINT  wMsgFilterMin,  \_In\_     UINT  wMsgFilterMax,  \_In\_     UINT  wRemoveMsg | 메시지 큐에 있는 메시지를메세지 lpMsg에 넣는다.  hWnd값이 0이면 현재 윈도우에 맞는 메시지를 검색한다.wMsg…0해도 됨  wRemoveMsg는 PM\_REMOVE는 큐에서 메세지를 제거 PM\_NOREMOVE는 메시지를 제거 안함  이 함수가 true면 메시지가 있다는 의미 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + Window Message 상수 값 리스트
    - System 정의 상수 값
      * 요령

시스템 상태를 전달하는 용도로 사용되는 경우가 많다.

* + - * 리스트 (대표 적인 비 큐 메세지다)

|  |  |
| --- | --- |
| 이름 | 내용 |
| WM\_CREATE | 윈도우 인스턴스가 생성되어 지면 날라오는 값 |
| WM\_DESTROY | 메시지 큐에 있는 WM\_QUIT 값을 가져 오면 실행되는 값 |
| WM\_PAINT | 무효화 영역이 발생하면 OS로부터 날라오는 메시지 계속 날라온다. 진짜 미친 듯이 날라온다.  ,메시지 큐에 좌표 값 만 업데이트 되고 2번 이상 들어오지는 않는다.  즉 2개 이상 메시지가 존재하지 않다는 의미 하지만 메시지는 계속 날라온다. |
|  |  |
|  |  |
|  |  |

* **장치 입력 값 받기**
  + - 키보드 입력
      * 요령

키보드 입력 값의 경우 실제 문자열이 들어오는 것이 아니라. 어셈블리 값이 들어오게 된다. 따라서 아무리 들어오는 메시지 값을 출력하려고 해도 원하는 문자열이 출력되지는 않는다.

또한 문자열 키패드 부분은 일반 KeyDown으로는 처리하기가 곤란하기에 그 처리 과정이 여타 Key들과는 다르다.

기본적으로 키보드 값은 스캔 코드, 가상 키 코드, 아스키 코드 이렇게 3개의 코드 값을 받을 수 있게된다.

스캔 키 값은 제조사마다 다르기 때문에 필요가 없고

가상 키 코드 와 아스키 키 코드 값을 주로 사용하게 된다.

기본적으로 가상 키 코드가 들어오지만 TranslateMessage()함수를 이용해서 번역된 값이 wParam으로 들어오게 끔 해서 WM\_CHAR에서 처리 할 수 있도록 한다.

* + - * Scan Code Value
        + 요령

Int sCode =(lParam & 0x00FF0000)>>16;

이 코드를 이용해서 들어온 스캔 코드 값을 얻을 수 있다.

대소문자 구분하지 않고 값이 들어오게 된다.

* + - * + ㅇ
      * Virtual Key Value
        + 요령

주로 Function key , 방향키 메시지를 처리 하기 위해 사용한다.

Int vCode = (int)wParam;

이 코드를 이용해서 들어온 가상 키 값을 얻을 수 있다.

대소문자 구분하지 않고 값이 들어오게 된다.

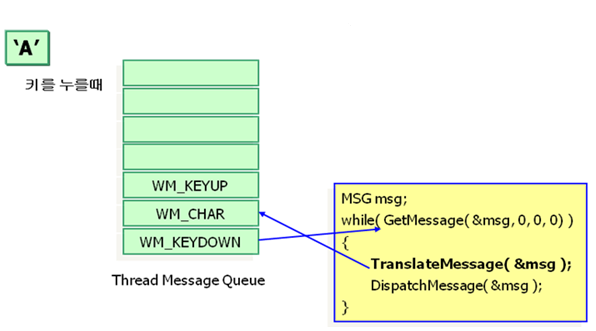
문자 값의 경우 아스키 값과 같은 값을 얻을 수 있다.

* + - * + 가상키 코드표



* + - * 아스키 Value
        + 요령

메시지 루프 영역에서 TranslateMessate 함수를 호출해 wParam에 아스키 값이 들어 올 수 있도록 하자 이렇게 받은 값은 대소문자를 구분하여 사용 할 수 있게 된다.

******

* + - * + ㅇ
    - 마우스 입력

Switch(msg)

{

Case 해당 마우스 버튼 값:

}

=====================================

WM\_MBUTTONDOWN: 가운데 값

WM\_LBUTTONDOWN: 왼쪽 버튼 값

WM\_RBUTTONDOWN: 오른쪽 버튼 값

WM\_MOUSEMOVE : 마우스가 움직일 때 들어오는 값

* + - * 마우스 메시지 종류

|  |  |  |
| --- | --- | --- |
| Client 영역 메시지 | Non client 영역 메시지 | 기타 |
| WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_LBUTTONDBLCLK  WM\_RBUTTONDOWN  WM\_RBUTTONUP  WM\_RBUTTONDBLCLK  WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MBUTTONDBLCLK  WM\_MOUSEMOVE | WM\_NCLBUTTONDOWN  WM\_NCLBUTTONUP  WM\_NCLBUTTONDBLCLK  WM\_NCRBUTTONDOWN  WM\_NCRBUTTONUP  WM\_NCRBUTTONDBLCLK  WM\_NCMBUTTONDOWN  WM\_NCMBUTTONUP  WM\_NCMBUTTONDBLCLK  WM\_NCMOUSEMOVE | WM\_NCHITTEST  WM\_MOUSEWHEEL |

* + - * 더블 클릭 메시지 발생하기

WINDCLASS 를 만들고 setting 할때

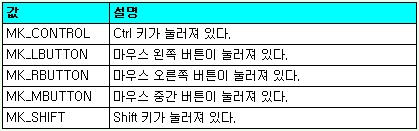
WINDCLASS Wc.style = CS\_DBLCLKS;를 하고 레지스트리에 등록하면 더블 클릭을 할 수 있게 한다.

다만 워낙 성능이 거지같아서 그냥 안쓰는 것이다.

* + - * 마우스 wParam, lParam에 대해서

마우스 위치 값은 lParam에 좌표값으로 저장되어 있다. 상위 16비트는 y값 하위 16트에는 x좌표가 들어있다.

wParam에는 마우스 버튼의 상태와 키보드 조합키의 상태가 전달된다.



* + - * Hit Test Code 클릭한 영역에 대한 정보

마우스가 어떠한 행동을 System은 커서 아래 있는 윈도우에게 WM\_NCHITTEST메시지를 보낸다. 프로시저는 이 메시지를 바로 처리 하지 않는다. 처리하지 않고 DefWindowProc에 바로 보낸다. DefWindowProc은 커서 좌표를 조사해 커서가 현재 윈도우 어디에 있는지 Hit Test Code로 리턴한다.

따라서 DWORD code=DefWindowProc()으로 code 안에 마우스 좌표 값을 받을 수 있다.

* + - 조합키 입력
      * 요령

WM\_KEYDOWN , WM\_CHAR의 내부에 if문으로 IsKeyPress(해당키)를 호출 해 현재 해당 값이 눌려지고 있는지 필터 처리를 해주면 된다.

* + - * \ㅇ
* **GDI 모듈**
* InvalidateArea (무효화 영역)에 대해서
  + 무효화 영역이 필요한 이유

이미 gdi로 그려진 위치에는 다른 그림을 그릴 수 없다.

* + 무효화 영역 발생 시기

윈도우를 옮길 때

윈도우를 제거 했을 때

감추어졌던 윈도우 영역이 보이게 될 때

윈도우의 크기를 조정했을 때

ScrollWindwo(),ScrollDC()함수를 사용하는 경우

InvalidateRect(),InvalidateRgn()함수를 사용하는 경우

윈도우의 일부에 겹친 DialogBox나 메시지 박스가 제거 될 대

메뉴가 나타났다 사라질 때

풍선 도움말이 나타났을 때

* + 구조체

|  |  |  |
| --- | --- | --- |
| 이름 | 멤버 | 내용 |
| PAINTSTRUCT | HDC hdc; // dc handle  BOOL fErase; //배경을지운경우 FALSE, 지우지 않은 경우 TRUE  RECT rcPaint; // 클리핑 영역을 포함하는 최소 사각형  BOOL fRestore; //Reserved; used internally by the system  BOOL fIncUpdate; //Reserved; used internally by the system  BYTErgbReserved[32]; //Reserved; used internally by the system | 이 구조체는 발생된 무효화 영역에 대한 여러가지 정보를 얻어 오기 위해 사용한다. |
|  |  |  |

* + Function

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환값 | 매개변수 | 내용 |
| BeginPaint | HDC | \_In\_  HWND  hwnd,  \_Out\_ LPPAINTSTRUCT lpPaint | WM\_PAINT 처리 루틴에서 DC를 얻을 때 사용해야 하는 특수 함수  DC를 만들고 UpdateRegion을 DC의 Cliping 영역으로 지정한다.  무효화 영역을 지우기 위해 WM\_ERASEBKGND 메시지를 보낸다  PAINTSTRUCT 구조체를 채운다.  Caret이 있을 경우 그리기 전에 Hide하고 EndPaint()에서 Show한다.  무효화 영역을 유효화 한다.  반드시 해제는 EndPaint()함수를 이용해 해제를 해주어야 한다. |
| EndPaint | BOOL | HWND hwnd  Const PAINTSTRUCT \* lpPaint | BeginPaint를 통해 생성한 DC 값을 해제 하기 위한 함수 |
| GetDC | HDC |  | DC핸들을 얻는 일반적인 방법  매개변수로 0을 주면 바탕화면의 DC를 받는다  ReleaseDC를 통해 해당 값을 해제 해주어야 한다. |
| GetWindowDC | HDC |  | 윈도우 전체를 위한 DC를 얻는다 |
| InvalidateRect | BOOL | \_In\_       HWND hWnd,  \_In\_ const RECT \*lpRect,  \_In\_       BOOL bErase | 일정한 공간 lpRect영역에 강제로 무효화 현상이 일어나게 하는 함수  hWnd 무효화할 해당 윈도우 핸들  lpRect 무효화 영역  bErase는 True면 바로 적용됨  false는 나중에 적용된 왜냐면 메시지 큐에 WM\_PAINT 메시지메 넣고 그 다음에 반영되게 끔 하기 때문이다. |
| InvalidateRgn |  |  | 강제로 무효화 현상이 일어나게 하는 함수 |

* + 클리핑
    - 무효영역 중에서도 화면에 보이는 가시 영역을 말한다. 그려져야 할 부분 중에서도 그리기가 허가된 영역
* 그래픽 출력
  + 요령

모든 GDI 객체는 부모로 HGDIOBJ를 가지고 있다. 그래서 모든 GDI 객체는 HGDIOBJ 데이터 형태로 담겨질 수 있다. 원하는 GDI로 변경하고 싶다면 강제 Casting을 통해 변경 해 줄 수 있다.

* + 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환형 | 매개변수 | 내용 |
| ***SetPixel*** | ***COLORREF*** | ***hdc, nXPos, nYPos, clrref*** | 컬러 값을 반환하는데 그 이유는 잘 모르겠음  Hdc는 그려질 도화지  nXPos는 x좌표  nYPos는 y좌표  clrref는 RGB 컬러값  해당 좌표에 컬러 값을 가지는 픽셀을 출력한다. |
| ***MoveToEx*** | ***DWORD*** | ***hdc, x, y, lpPoint)*** | 지정된 지점으로 현재위치를 업데이트한다. lpPoint에 포인터를 주면 해당 포인터에 이전 위치가 반환된다  정확히는 hdc가 가지고 있는 current Position을 x y.값으로 변경한다.  근데 Rendtagle, Ellipse등의 함수에는 영향을 미치지 않는다. |
| ***LineTo*** | ***BOOL*** | ***hdc, xEnd, yEnd*** | Hdc에 있는 currentPosition값을 시작점으로 x y 좌표 까지 선분을 그린다. |
| ***Rectangle*** | ***BOOL*** | ***hdc, nLeftRect, nTopRect, nRightRect, nBottomRect);*** | Hdc에 맞추어 해당 윈도우에 해당 논리 좌표에 맞추어 사각형을 그린다. 정확히는 (nLeftRect, nRightRect)점부터 (nTopRect,nBottomRect)의 점까지의 사각형을 그린다. |
| ***Ellipse*** | ***BOOL*** | ***hdc, nLeftRect, nTopRect, nRightRect, nBottomRect*** | Hdc에 맞추어 해당 윈도우에 해당 논리 좌표에 맞추어 타원형을 그린다. 정확히는 (nLeftRect, nRightRect)점부터 (nTopRect,nBottomRect)의 점까지의 타원형을 그린다. |
| CreateSolidBrush | HBRUSH | COROLO ref | ref에 RGB를 넣어주고 색상을 지정하면 해당 색상의 펜 객체를 반환한다. |
| CreatePen | HPEN | \_In\_ int      fnPenStyle,  \_In\_ int      nWidth,  \_In\_ COLORREF crColor | fnPenStyle에 따라서 펜 스타일이 결정되고 nWidth 값을 통해 펜의 굵기를 결정한다. crColor는 펜의 색상을 결정하는 매개변수다. 해당 함수는 이렇게 만들어진 HPEN 객체를 반환한다. |
| SelectObject | HGDIOBJ | \_In\_ HDC     hdc,  \_In\_ HGDIOBJ hgdiobj | Hdc 값을 변경하는 함수로 hdiobj를 내부적으로 분석하여 적절한 hdc 안에 있는 파라미터를 변경해준다. 그리고 기존에 hdc 파라미터의 값을 HGDIOBJ형태로 반환한다. |
| DeleteObject | BOOL | \_In\_ HGDIOBJ hObject | hObject를 삭제한다.  파라미터에서 알 수 있듯이 GDI객체를 제거하는 전용 함수다 |
| RGB | COLORREF | Int r, int g,int b | R은 레드값  G는 그린값  B는 블루값  해당 함수는 COLORREF 형태로 파라미터 값을 뭉쳐서 반환한다. |
| SetROP2 | int | HDC hdc , int fnDrawMode | 그림이 겹쳐질 때 어떻게 처리 할 것인지 모드로 결정한다. Hdc의 파라미터 값을 지정해주고 반환 값으로 이전 혼합 모드를 반환한다. |

* + GDI 객체 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| GDI 오브젝트 | 핸들 타입 | 설명 | 디폴트 |
| 펜 | HPEN | 선을 그릴 때 사용 | 검정색의 가는 실선 |
| 브러쉬 | HBRUSH | 면을 채울 때 사용 | 흰색 |
| 폰트 | HFONT | 문자 출력에 사용 | 시스템 글꼴 |
| 비트맵 | HBITMAP | 비트맵 이미지 | 선택되지 않음 |
| 팔레트 | HPALETTE | 팔레트 | 선택되지 않음 |
| 리전 | HRGN | 화면상의 영역 | 선택되지 않음 |

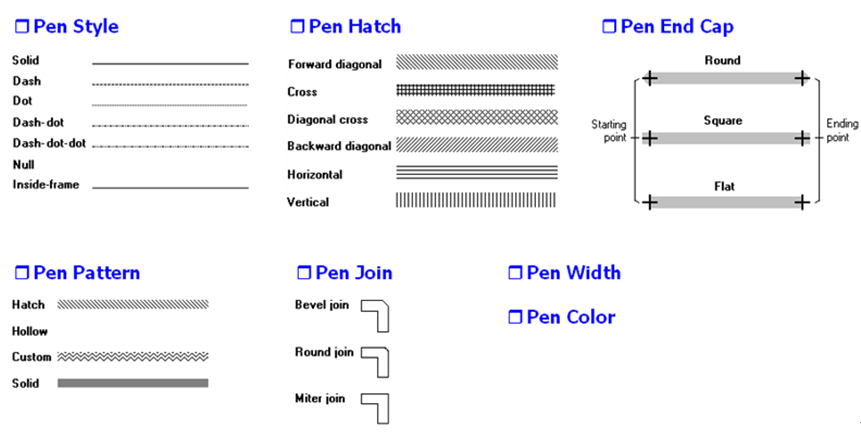
위의 객체들을 이용해 만든 새로운 객체들은 반드시 DeleteObject를 통해 삭제를 해야 메모리 상에 더 이상 남지 않게 된다.

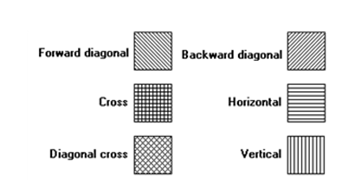
* + 스톡 GDI 객체 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 오브젝트 상수값 | 설명 | 오브젝트 | 설명 |
| BLACK\_BRUSH | 검정색 브러쉬 | GRAY\_BRUSH | 회색 브러쉬 |
| NULL\_BRUSH | 투명 브러쉬 | WHITE\_BRUSH | 흰색 브러쉬 |
| DKGRAY\_BRUSH | 짙은 회색 브러쉬 | LTGRAY\_BRUSH | 옅은 회색 브러쉬 |
| DC\_BRUSH | 색상 브러쉬 |  |  |
| BLACK\_PEN | 검정색 펜 | WHITE\_PEN | 흰색 펜 |
| NULL\_PEN | 투명 펜 | DC\_PEN | 색상 펜 |
| ANSI\_FIXED\_FONT | 고정폭 폰트 | ANSI\_VAR\_FONT | 가변폭 폰트 |
| DEFAULT\_PALETTE | 시스템 팔레트 |  |  |

HGDIOBJ GetStockObject(int fnObject)를 이용해 스톡 GDI객체를 가져 올 수 있다.

이 값들은 DeleteObject 할 필요가 없다.





* + 문자형 그림
    - 요령

잘써라

* + - 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환값 | 매개변수 | 내용 |
| ***TextOut*** | ***BOOL*** | ***HDC hdc,***  ***int nXStart,***  ***int nYStart,***  ***LPCTSTR lpString,***  ***int cbString*** | 문자열을 출력하는 함수  Hdc는 출력할 대상 윈도우의 핸들  nXStart는 문자열이 출력시작 될 X좌표  nYStart는 문자열이 출력 시작 될 Y좌표  lpString은 출력할 문자열의 포인터  cbString을 출력할 문자열의 길이 |
| ***SetTextAlign*** | ***UINT*** | ***HDC hdc,***  ***UINT fMode*** | 문자열을 정렬하는 함수  Hdc는 정렬할 대상 윈도우의 핸들  fMode는 정렬방법을 나타내는 메크로 |
| ***DrawText*** | ***int*** | ***HDC hdc,***  ***LPCTSTR lpString,***  ***int nCount,***  ***LPRECT lpRect,***  ***UINT uFormat*** | 포맷된 형태로 문자열을 사각형 안에 출력하는 함수  Hdc는 출력할 대상 윈도우의 핸들  lPString은 출력할 문자열의 포인터  nCount는 출력할 문자열의 길이 -1을 입력하면 널 종료 문자열을 검색해서 문자열을 전부다 출력해준다  lpRect는 문자열을 출력할 사각형의 값을 나타낸다  uFormat은 어떤식으로 출력할 것인지 정렬과 출력 방식을 OR 연산을 통해 줄 수 있다. |

* + - 매크로 리스트

|  |  |  |
| --- | --- | --- |
| 값 | 용도 | 설명 |
| TA\_TOP | Align fMod Macro | 지정한 좌표가 상단 좌표가 된다. |
| TA\_BOTTOM | Align fMod Macro | 지정한 좌표가 하단 좌표가 된다. |
| TA\_CENTER | Align fMod Macro | 지정한 좌표가 수평 중앙 좌표가 된다. |
| TA\_LEFT | Align fMod Macro | 지정한 좌표가 수평 왼쪽 좌표가 된다. |
| TA\_RIGHT | Align fMod Macro | 지정한 좌표가 수평 오른쪽 좌표가 된다. |
| TA\_UPDATECP | Align fMod Macro | 지정한 좌표대신 CP를 사용하며 문자열 출력 후에 CP를 변경한다. |
| TA\_NOUPDATECP | Align fMod Macro | CP를 사용하지 않고 지정한 좌표를 사용하며 CP를 변경하지 않는다. |

|  |  |  |
| --- | --- | --- |
| 값 | 용도 | 설명 |
| DT\_LEFT |  | 수평 왼쪽 정렬한다. |
| DT\_RIGHT |  | 수평 오른쪽 정렬한다. |
| DT\_CENTER |  | 수평 중앙 정렬한다. |
| DT\_BOTTOM |  | 사각영역의 바닥에 문자열을 출력한다. |
| DT\_VCENTER |  | 사각영역의 수직 중앙에 문자열을 출력한다. |
| DT\_WORDBREAK |  | 사각영역의 오른쪽 끝에서 자동 개행되도록 한다. |
| DT\_SINGLELINE |  | 한 줄로 출력한다. |
| DT\_NOCLIP |  | 사각영역의 경계를 벗어나도 문자열을 자르지 않고 그대로 출력한다. |

* + - ㅇ
  + 색상 데이터(비트 마스킹을 통해 rgb를 분리 저장하고 있다 8바이트 간격으로 있다)
    - 요령

COLORREF라는 데이터형을 사용한다.

Typedef DWORD COLORREF; 로 정의 되어 있다.

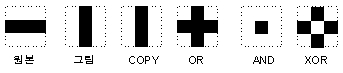
해당 값을 지정하기 위해서는 RGB 매크로 함수를 사용해야 한다.

* + - 전용 매크로 리스트

|  |  |  |
| --- | --- | --- |
| 형태 | 축약 내용 | 하는 짓 |
| GetRValue(rgb) | ((BYTE)(rgb)) | RGB의 레드 값 만을 반환한다 |
| GetGValue(rgb) | ***((BYTE)(((WORD)(rgb))>>8))*** | RGB를 WORD형태로 변화시키고 8비트 쉬프트를 해서 그린 값을 반환한다. |
| GetBValue(rgb) | ***((BYTE)((rgb)>>16))*** | RGB를 16비트 쉬프트 시키고 남은 값을 바이트 화 해서 반환한다. |

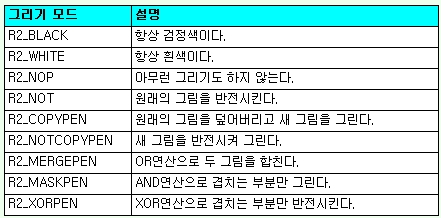
* + 그리기 모드
    - 요령

화면에 뭔가 그릴 때 겹칠 경우 어떻게 처리 할 것인지를 정의하는 모드다



SetROP2(HDC hdc, 매크로 상수); 이 함수를 이용해 해당 hdc의 파라미터 값을 변경 할 수 있다.

* + - fnDrawMode 매크로 상수 리스트



* + 윈도우 좌표 변경
    - 요령

해상도가 변해도 그린 그림이 찌그러지거나, 커지거나 작아지거나 변화 하지 않도록 하기 위해서 사용한다.

GDI 객체들은 논리 좌표를 이용해서 그림을 그리고

컴퓨터들은 입력 값을 물리적 좌표로 전달한다.

지금까지 두 값들이 충돌 없이 멀쩡하게 돌아가고 있었던 이유는 단순히 둘의 단위가 픽셀 단위였기 때문이다.

* + - 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환값 | 매개변수 | 내용 |
| ***DPtoLP*** | BOOL | HDC     hdc,  LPPOINT lpPoints,  int     nCount | 물리 좌표를 논리 좌표로 변환한다 |
| ***LPtoDP*** | BOOL | HDC     hdc,  LPPOINT lpPoints,  int     nCount | 논리 좌표를 물리 좌표로 변환한다 |
| ***ClientToScreen*** | BOOL | HWND    hWnd,  LPPOINT lpPoint | 클라이언트 좌표를 스크린 좌표로 |
| ***ScreenToClient*** | BOOL | HWND    hWnd,  LPPOINT lpPoint | 스크린 좌표를 클라이언트 좌표로 |
| ***SetMapMode*** | Int | ***HDC hdc, int fnMapMode*** | Hdc의 맵핑 모드 파라미터를 변경한다. fnMapMode로 |
| ***GetMapMode*** | int | ***HDC hdc*** | Hdc의 현재 맵핑 모드 파라미터를 가져온다. |

* + - 맵핑 모드 매크로 리스트



* **UI 모듈**
  + Define 문
* **Timer**
  + 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환값 | 매개변수 | 내용 |
| GetLocalTime | Void | ***LPSYSTEMTIME lpSystemTime*** | 현재 지역 시간을 구한다 |
| ***GetSystemTime*** | void | ***LPSYSTEMTIME lpSystemTime*** | 현재 세계 표준 시간을 구한다 |
| ***GetTimeFormat*** | Int | ***LCID Locale,***  ***DWORD dwFlags,***  ***CONST SYSTEMTIME \*lpTime,***  ***LPCTSTR lpFormat,***  ***LPTSTR lpTimeStr,***  ***int cchTime*** | 시간 값을 문자열로 변환한다 |
| ***GetDateFormat*** | int | ***LCID Locale,***  ***DWORD dwFlags,***  ***CONST SYSTEMTIME \*lpTime,***  ***LPCTSTR lpFormat,***  ***LPTSTR lpTimeStr,***  ***int cchTime*** | 날짜 값을 문자열로 변환한다 |
| ***SystemTimeToTzSpecificLocalTime*** | BOOL | …생략 | 세계시간을 지역 시간으로 변환 |
| ***TzSpecificLocalTimeToSystemTime*** | BOOL |  | 지역시간을 세계 시간으로 변환 |
| ***GetFileTime*** | BOOL |  | 특정 파일의 시간을 구하는 함수 |
| ***FileTimeToSystemTime*** | BOOL |  | FILETIME을 SYSTEMTIME으로 변경하는 함수 |
| ***SetTimer*** | UINT\_PTR | HWND hWnd,  UINT\_PTR  nIDEvent,  UINT      uElapse,  TIMERPROC lpTimerFunc | 주기적으로 발생되는 Timer를 생성한다.  hWnd는 타이머를 메세지를 받을 윈도우  nlDEvent는 설정할 타이머 번호  uElapse는 주기 1/1000초  lpTimerFunc는 타이머 메시지가 발생할 때마다 호출될 함수를 지정 NULL일경우 WM\_TIMER를 반복 호출한다  리턴값은 타이머를 소유하는 윈도우 없이 타이머가 만들어졌을 경우만 의미가 있다. 즉 hWnd값 이 NULL일 경우에만 특별하게 사용된다 |
| ***KillTimer*** | BOOL | ***HWND hWnd,***  ***UINT uIDEvent*** | 설치된 타이머를 제거한다  hWnd은 타이머를 소유한 윈도우 핸들  ulDEvent는 지울 타이머의 ID값 |
|  |  |  |  |

* + 매크로 리스트

|  |  |  |
| --- | --- | --- |
| 이름 | 용도 | 내용 |
| WM\_TIMER | System message | SetTimer함수 내부적으로 불려지는 함수 |

* **Resource**
  + 요령

리소스를 만들고 싶다면 visual studio 리소스 추가를 이용하자

추가한 리소스는 resource.h에 매크로로 ID값들이 정의 되어진다.

보통 클래스 setting 부에서 MAKEINTRESOURCE매크로를 이용해서 넣어주고 클래스를 생성해주게 된다.

생성 한 뒤에는 Load… 등의 함수를 이용해 해당 값을 가져온 뒤에 set…등의 함수로 바꿀 윈도우의 핸들과 함께 값을 넘기면 바뀌게 된다.

* + ICON
    - 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환형 | 매개변수 | 내용 |
| LoadCursor | HCURSOR | HWND hwnd,? | 리소스에서 커서를 불러온다 |
| LoadICon | HICON |  | 리소스에서 아이콘을 불러온다 |
| LoadCursorFromFile | HCURSOR | TEXT str | 해당 경로에 있는 리소스 값을 커서 객체로 만든다 |
| SetCursor |  |  | 커서를 변경하는 함수 |

* + - 매크로 리스트

|  |  |  |
| --- | --- | --- |
| 이름 | 용도 | 내용 |
| MAKEINTRESOURCE | Function | 매개변수에 들어온 값에 따라 리소스를 출력한다. |
| WM\_SETCURSOR | System Message |  |

* + Menu
    - 요령

비쥬얼 스튜디오 내 에서 잘 만들 수 있다.

메뉴 디자인 할 때 (&문자)를 작성하면 새로만들기 (문자)로 출력 할 수 있다.

각 항목의 속성 값에 들어가면 다양한 값을 변경 할 수 있다.

Context Menu를 만들기 위해서는 먼저 System Menu처럼 디자인 툴에서 만들어 줄 필요가 있다. 이후 context menu 처리를 하면 System menu에서 해당 메뉴가 빠지고 context menu로 출력이 된다. ~~진자 좋 같은 시스템이다~~.

* + - 함수 리스트

|  |  |  |  |
| --- | --- | --- | --- |
| 이름 | 반환형 | 매개변수 | 내용 |
| LoadMenu | HMUNU | 출력할 윈도우 핸들, id값 | 리소스에서 메뉴 값을 불러온다 |
| SetMenu |  |  | 해당 윈도우의 메뉴를 매개변수 값으로 변경한다 |
| GetMenu | HMENU | HWND hwnd | Hwnd에 해당하는 윈도우의 메뉴 핸들을 반환한다 |
| GetSubMenu | HMENU | HMENU hMenu, int nPos | hMenu에서 서브 메뉴의 핸들 값을 반환한다. nPos에 의해 가져오는 서브 메뉴의 위치가 달라진다. 메인 메뉴 왼편부터 0 오른쪽으로 가면서 1,2,3… 이런식으로 증가 해 간다. |
| TrackPopupMenu | BOOL | HMENU hSubMenu ,  UINT uFlags ,  Int x , int y,  Int nReserved  HWND hwnd  RECT \* rc | Hwnd의 윈도우에 x, y좌표에 rc크기의 hSubMenu 팝업창을 띄우는 함수  uFlasgs는 커서 기준으로 어느 방향으로 출력될 것인지를 나타낸다  rc가 0일 경우 적당한 크기로 출력된다 |
| AppendMenu | BOOL | HMENU hMenu,  UINT uFlags,  UINT\_PTR uIDNewItem,  LPCTSTR lpNewItem | hMenu이 메뉴 핸들에 uFlags 모드를 반영해서 새로운 메뉴를 추가한다.  ulDNewItem은 해당 메뉴 값의 ID값이고  lpNewItem은 해당 메뉴의 이름을 나타낸다.  이 함수는 메뉴를 추가하는 것이지 드롭다운 메뉴를 추가하는 것은 아니다. 따라서 이 메뉴로 만들어진 메뉴 밑에는 서브 메뉴가 붙을 수가 없다 |

* + - 메크로 리스트

|  |  |  |
| --- | --- | --- |
| 이름 | 용도 | 내용 |
| WM\_INITMENUPOPUP | System Message | 드롭다운 메뉴가 활성화 될 때 발생되는 메시지. 즉 최상단 메뉴들을 아무거나 선택해서 밑으로 메뉴가 주르륵 생기면 발생하는 메시지 또한 주르륵 생기는 메뉴는 CONTEXTMENU도 해당한다. 따라서 내부적으로 CONTEXTMENU를 실행 시키면 메뉴가 주르륵 생기는데 그때도 발생되는 메시지다.  wParam에는 드롭다운 메뉴가 펼쳐지고 난 뒤에 선택한 메뉴의 코드 값이 들어 있다. (HMENU)로 강제 캐스팅 하면 읽을 수 있다. |
| WM\_COMMAND | System Message | 사용자가 메뉴를 선택하거나 컨트롤이 부모 창으로 Notify 메시지를 보내거나 액셀러레이터 키 입력이 변환 될 때 발생한다, 메뉴 탑에 있는 이름들은 아이디 코드들이 없다. 그러나 아이디 코드가 있는 메뉴값을 선택할 경우 발생되는 메시지다  wParam에 하위 ResourceID 즉 메뉴의 ID값이 들어온다.  이 메시지에 각 메뉴 이벤트에 대해서 case문으로 작성하면 된다.  wParam에 상위 자식이 보낸 Notify 알림 문구가 들어온다. |
| WM\_CONTEXTMENU | System Message | 화면에 마우스 오른쪽 버튼을 클릭 한 윈도우에 날리는 메시지  이 메시지는 내부적으로 INITMENUPOPUP 메시지를 발생시킨다.  이 메시지에 각 메뉴의 활성화 조건문을 작성하면 된다 |
| **TPM\_LEFTALIGN** | Flags for TrackPopupMenu | 팝업창을 커서 왼쪽으로 출력한다 |
| **TPM\_RIGHTALIGN** | Flags for TrackPopupMenu | 팝업창을 커서 오른쪽으로 출력한다. |
| **TPM\_CENTERALIGN** | Flags for TrackPopupMenu | 팝업창을 커서 가운데 밑으로 출력한다 |

* + 컨트롤
    - 요령

WM\_CREATE 메시지 에 control 들을 생성한다

자식 control에서 통지 알림을 받을 때

WM\_COMMAND 메시지에서 자식 control이 보내는 notify 메세지를 처리한다. 하위 wParam으로 어떤 자식 control들이 들어 오는지 필터링한다

WM\_COMMAND 메시지에서 상위 wParam으로 control이 보내는 Notify code가 무엇인지 필터링 한다.

부모가 자신 control에게 메시지를 보낼 때

SendMessage(대상 control , 보낼 메시지 , wParam, lParam) 함수를 이용해서 자식 control에게 메세지를 보내고 반환값을 받는다.

* + - 기본 제공 System Class 리스트
    - 각 매크로 리스트
    - ㅇ
* **Custom Function Library**
  + 윈도우
  + 글자
  + GDI

ㅇㅇㅇㅇㅇㅇㅇ