# MFC

## function

### 获取窗口对象

CWnd \*pWnd = CWnd::FromHandle(hwnd);

### 时间

CTime time(2001,1,1,13,13,13);

//构造时间2001年1月1日 13：13：13

//也可以使用类的其他构造函数构造时间

CTime time=CTime::GetCurrentTime();

CString str=time.Format("%Y-%m-%d %H:%M:%S");

//也可以使用类中的其他函数单独获取年月日等，用作文件名时，不能带冒号

int year=time.GetYear();

//可以让两个时间相减，得到间隔时间

CTimeSpan t = ctime2 - ctime1;

\_\_time64\_t x = t.GetTimeSpan(); //得到两个时间的秒差

t.GetMinutes(); //通过该类函数可以获取两个时间之间年、月、小时等直接的时差

### 调试

TRACE("xx");//可以将CString输出到调试窗口，debug有效

TRACE(\_T("\r\n"));

### 消息盒子

#### 使用

if(IDCANCEL==AfxMessageBox(\_T("close"),MB\_OKCANCEL)) return;

#### 修改标题

重写App类的虚函数DoMessageBox

将函数中内容替换为：

LPCTSTR pOldAppName=m\_pszAppName;

m\_pszAppName=\_T("新标题");

int i=CWinApp::DoMessageBox(lpszPrompt, nType, nIDPrompt);

m\_pszAppName=pOldAppName;

return i;

### 截获消息

在虚函数PreTranslateMessage中：

if(pMsg->message==WM\_KEYDOWN){

//屏蔽关闭按键

if (pMsg->wParam == VK\_ESCAPE || pMsg->wParam == VK\_RETURN) return TRUE;

}

对于其他程序发的消息，只能接收PostMessage发送的。

if (pMsg->message == WM\_SYSKEYDOWN && pMsg->wParam == VK\_F4) return TRUE;

### 右键菜单

菜单资源IDR\_MENU1，使用第一列中的菜单内容

CMenu m\_menu;

m\_menu.LoadMenu(IDR\_MENU1);

动态创建

CMenu menu;

menu.CreatePopupMenu();

menu.AppendMenu(MF\_STRING, 消息id,\_T("cmd1"));//消息id

POINT pt;

GetCursorPos(&pt);

m\_menu.GetSubMenu(0)->TrackPopupMenu(TPM\_LEFTALIGN,pt.x,pt.y,this);

### 屏蔽f10

if (pMsg->message == WM\_SYSKEYDOWN) if (pMsg->wParam == VK\_F10) return TRUE;

### 屏蔽帮助

CXXXApp.cpp中去掉：  
ON\_COMMAND(ID\_HELP, &CWinApp::OnHelp)

## 消息

### 消息映射机制

#### 基本条件

1. 类必须从CCmdTarget派生
2. 类内必须声明宏函数DECLARE\_MESSAGE\_MAP()

afx\_msg .................处理函数原型

1. 类外必须添加实现宏函数

BEGIN\_MESSAGE\_MAP(derived Class, base Class)

1） ON\_MESSAGE(WM\_xxx,处理函数名）

2） ON\_EN(BN)\_XXX(ID,处理函数名)

3) ON\_COMMAND(ID,处理函数名)

4) ON\_WM\_XXXX()

END\_MESSAGE\_MAP()

4.类外实现消息处理函数

#### 处理顺序

Frame，app

当处理同一个消息时，如果frame中有处理函数，则使用

Frame中的处理函数，如果没有则使用app中的处理函数。

### 消息分类

#### windows标准消息

键盘、鼠标、定时器……

ON\_WM\_XXXX()

Exp:

class CMyFrameWnd:public CFrameWnd{

DECLARE\_MESSAGE\_MAP() 类内声明宏

public:

afx\_msg int [OnCreate](mk://_mfc_cwnd.3a3a.oncreate.htm)( LPCREATESTRUCT pcs);

可在msdn中查找到系统消息ON\_WM\_CREATE( )对应的处理函数原型，其中的参数可以不写或者只写其中的个别需要用的参数，但是原来参数的位置顺序不可变

};

BEGIN\_MESSAGE\_MAP(CMyFrameWnd,CFrameWnd)

参数分别为当前类，及其父类

ON\_WM\_CREATE( )

系统消息声明

END\_MESSAGE\_MAP()

int CMyFrameWnd::OnCreate(LPCREATESTRUCT pcs ){

处理函数的实现

AfxMessageBox(\_T("xxxxxxxx"));

return CFrameWnd::OnCreate(pcs);

}

#### 自定义消息

#define WM\_XXXX WM\_USER+n

SendMessage/PostMessage

Exp:

A类发消息，就在A类的头文件中定义消息。如果由B类响应这个消息，那么B类包含A类头文件

#define WM\_MY WM\_USER+10

任意一个函数中发送消息：

::SendMessage(this->m\_hWnd,WM\_MY,1,2);

1,2代表附加信息

BEGIN\_MESSAGE\_MAP(\*\*\*\*,\*\*\*\*)

ON\_MESSAGE(WM\_MY,OnMyMessage)

OnMyMessage为自定义处理函数

END\_MESSAGE\_MAP()

类内声明：

DECLARE\_MESSAGE\_MAP()

afx\_msg LRESULT OnMyMessage(WPARAM

wParam, LPARAM lParam);

类外实现：

LRESULT CMyFrameWnd::OnMyMessage(WPARAM

wParam, LPARAM lParam){

CString str;

str.Format("define %d %d",wParam,lParam);

AfxMessageBox(str);

return 0;

}

#### 命令消息

WM\_COMMAND

一般是菜单项目的响应消息

#### 通知消息

ON\_BN\_CLICKED

ON\_EN\_CHANGE

ON\_通知码

## file

### 删除目录下所有文件

void DeleteDirectory(CString directory\_path)

{

CFileFind finder;

CString path;

path.Format(\_T("%s/\*.\*"),directory\_path);

BOOL bWorking = finder.FindFile(path);

while(bWorking){

bWorking = finder.FindNextFile();

if(finder.IsDirectory() && !finder.IsDots()){//处理文件夹

DeleteDirectory(finder.GetFilePath()); //递归删除文件夹

RemoveDirectory(finder.GetFilePath());

}

else{//处理文件

DeleteFile(finder.GetFilePath());

}

}

}

### 文件状态

CFileStatus status;

CFile::GetStatus("D:\\1.txt",status);

======用CTimeSpan修改时间

CTimeSpan span(7,0,0,0);//天数，时，分，秒

status.m\_mtime-=span;

//将文件修改时间状态信息提前七天

=======用CTime修改时间

CTime time(2001,1,1,13,13,13);

status.m\_ctime=time;

CFile::SetStatus("D:\\1.txt",status);//确认修改

属性修改

status.m\_attribute|=FILE\_ATTRIBUTE\_HIDDEN;

status.m\_attribute|=FILE\_ATTRIBUTE\_READONLY;//设置为只读以后就不可用对status进行修改

CTimeSpan的成员变量（CTime类型）：

m\_ctime （Create）

m\_mtime (last modified)

m\_atime (last accessed for reading)

### 逐行读写

#### 读文件

//CStdioFile file(\_T("D:\\test.csv"), CStdioFile::modeRead|CStdioFile::typeBinary

);//二进制读写和文本读写对\n的解释不同

CStdioFile file;

if(!file.Open(\_T("D:\\test.csv"),CStdioFile::modeRead))

{return;}

CString strline;

while(file.ReadString(strline)) {AfxMessageBox(strline);}

#### 写文件

Unicode下不能写中文

CStdioFile file;

if(!file.Open(\_T("D:\\1.csv"), CStdioFile::modeWrite|CStdioFile::modeCreate)) return;

CString line1(\_T("a,b,c\n"));

file.WriteString(line1);

CString line2(\_T("1,2,3\n"));

file.WriteString(line2);

file.Close();

### 序列化

class Test:public CObject

{

DECLARE\_SERIAL(Test)

public:

int x;

double y;

virtual void Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

ar<<x<<y;

}

else

{

ar>>x>>y;

}

}

};

IMPLEMENT\_SERIAL(Test,CObject,1);

写入：

Test tt;

CFile file(\_T("D:\\tt.archive"),CFile::modeCreate|CFile::modeWrite);

CArchive oar(&file,CArchive::store);

oar<<&tt;

oar.Close();

file.Close();

读取:

Test \*pTT = new Test();

CFile file(\_T("D:\\tt.archive"),CFile::modeRead);

CArchive oar(&file,CArchive::load);

oar>>pTT;

### 文档类

#### 保存功能

视图类显示数据，文档类中的成员负责保存数据，自己定义的类是个中间变量，可以管理某类型的数据

文档类中使用一个数组保存自定义的类的多个对象

1）定义图形类A(以画小圆圈为例)

成员变量:CRect m\_rect;

2）在文档类中添加成员变量

CObArray m\_array;

数组类，可以存任何CObject类及其派生类的数据，和数组用法相似

1. 在视图类的某个消息中确认保存数据到文档类的成员

如鼠标左键消息(鼠标点击一下则画一个小圆圈，同时new出一个A的对象，然后将new出的对象保存到文档类的m\_array中)

OnLButtonDown(UINT nFlags, CPoint point) ：

CRect rect(point.x-10,point.y-10,point.x+10,point.y+10);

CClientDC dc(this);

dc.Ellipse(rect);

A \*pa=new A;

pa->m\_rect=rect;

CADoc\* pDoc = GetDocument();

在onDraw函数中就可以不用写这句，系统已经自动添加过

pDoc->m\_array.Add(pa);

如果将a进行delete操作，则m\_array中的成员消失

视图类cpp中需要包含自定义类A的头文件（包含的头文件必须写在stdafx.h下面一行，不能写在上面）

4）可在OnDraw函数中从m\_array中取出变量，然后重新绘图，这样缩放窗口就不会出现问题

for(int i=0;i<pDoc->m\_array.GetSize();i++){

A \*pa=(A\*)pDoc->m\_array[i];

pDC->Ellipse(pa->m\_rect);

}

5）在文档类中释放new出的对象

用一个函数clear(文档类内的自定义函数)封装以下代码

for(int i=0;i<m\_array.GetSize();i++)

delete m\_array[i];

m\_array.RemoveAll(); 必须存在此句，否则会出错

文档类析构函数中调用clear

C\*\*Doc::OnNewDocument()中也需要调用clear，因为是新建文件

1. 在文档类C\*\*Doc::Serialize(CArchive &ar)虚函数中保存文件：

if (ar.IsStoring())

{

int count=m\_array.GetSize();

ar<<count;

for(int i=0;i<count;i++){

A \*pa=(A\*)m\_array[i];

CRect rect=pa->m\_rect;

ar<<rect.left<<rect.top<<rect.right<<rect.bottom;

}

}

else

{

clear();

int count;

ar>>count;

for(int i=0;i<count;i++){

A \*pa=new A;

CRect &rect=pa->m\_rect;

ar>>rect.left>>rect.top>>rect.right>>rect.bottom;

m\_array.Add(pa);

}

}

在存储数据时，一定要注意数据的存储顺序和变量名，还有流的方向不可写错，否则在打开时会出现空白或者错误数据。

文档类的序列化是在打开和保存文件时系统自动进行调用

自定义类时，不使用类向导(注意先包含stdafx.h)，然后在自己定义的类中进行序列化，这样就可以使用不同版本号，而且因为自定义类已经有了序列化，所以可以使用ar>>pa的这种语法。但是从类向导中弄出的类没有版本号，而且序列化后无法使用ar>>pa语法。

#### 文件类型

字符串表资源第一个IDR\_MAINFRAME

有六个\n,七个成员

在msdn的GetDocString,其中有对七个成员的解释

windowTitle，docName ，fileNewName，filterName，filterExt，regFileTypeId，regFileTypeName

最后两个参数是将文件写入注册表

第四和第五个参数是保存类型和文件名后缀类型

如：/n name(\*.jyl)/n .jyl 第五个参数只能为空，或者写成文件后缀

显示时： 文件名 \*\*.jyl

保存类型 name(\*.jyl)

在新建工程第四步，高级选项中可以设置

#### 撤销/还原

void CDrawDoc::undo()

{

int count=m\_array.GetSize();

if(count){

CMyShap \*shap=(CMyShap \*)m\_array[count-1];

m\_array.RemoveAt(count-1);

m\_temp.Add(shap);

}

}

void CDrawDoc::redo()

{

int count=m\_temp.GetSize();

if(count){

CMyShap \*shap=(CMyShap \*)m\_temp[count-1];

m\_temp.RemoveAt(count-1);

m\_array.Add(shap);

}

}

#### 双击打开文件

在C\*\*App类中的InitInstance函数中加入：

m\_pMainWnd->DragAcceptFiles(); //拖拽打开功能

EnableShellOpen();

RegisterShellFileTypes(TRUE);

需要加在AddDocTemplate后面，否则会报错

## draw

### 绘图处理位置

画图操作不能写在Create消息中

不能在OnDraw函数或者WM\_PAINT中刷新窗口，否则会死循环

#### onDraw函数

(视图类中的虚函数)中用

virtual void OnDraw(CDC \*pDC);

pDC->TextOut(200,200,"xxxx");

#### OnPaint

（WM\_PAINT消息）函数中，常用于对话框

CPaintDC dc(this);

dc.TextOut(34,234,"ssf");

写了WM\_PAINT消息会使onDraw函数失效，因为在WM\_PAINT消息响应函数中封装了，将dc传到onDraw函数，所以如果重写了该消息响应函数，会使onDraw得不到调用。

#### 其他函数中

常使用CClientDC这类的DC

CDC\* pDC=GetDC();

pDC->TextOut(300,34,str);

ReleaseDC(pDC);

### 刷新

#### 刷新无效问题

可以通过PostMessage调用函数刷新控件

刷新时先判断控件是否初始化

if(NULL == GetWindow(GW\_CHILD)) return;

OnInitDialog() 初始化后刷新

使用 PostMessage

WM\_SIZE 移动和最大化时刷新

消息处理时刷新 if (nType == SIZE\_MAXIMIZED) 使用 PostMessage

WM\_MOVE 控制移动时刷新

消息处理时刷新

WM\_SYSCOMMAND 最小化还原刷新

使用 PostMessage

#### 刷新显示

InvalidateRect(NULL); //刷新整个范围

#### 减少闪烁

##### 刷新指定位置

CRect rect;

GetDlgItem(ID)->GetWindowRect(rect); //得到该控件在屏幕的位置

this->ScreenToClient(rect);//函数是对话框的，所以转换为以对话框为基准的坐标系

InvalidateRect(rect);

GetWindowRect这种函数都是以屏幕坐标系为基准的

GetClientRect这种函数都是以当前窗口为基准的

##### 擦除背景

如果在对话框类中找不到该消息，可以在类向导class info中设置消息过滤为window则可以看到该消息。

在WM\_ERASEBKGND消息处理中，返回值改为return TRUE,则可以不擦除背景。在返回前进行位图等绘制，可以减少闪烁，但是如果有其他背景绘制，可能会有影响。

##### 内存DC缓冲

可以将绘图内容以位图形式保存到兼容DC中，需要的时候将这个兼容DC中的内容绘制出来（缓冲，减少闪烁）。

#### 内存缓冲

CDC \*pDC= GetDlgItem(ID)->GetDC();

CRect rect;

GetDlgItem(ID)->GetClientRect(&rect);

CDC MemDC;

if (NULL == MemDC.m\_hDC) return;

MemDC.CreateCompatibleDC(pDC);

CBitmap bitmap;

bitmap.CreateCompatibleBitmap(pDC,rect.Width(),rect.Height());

MemDC.SelectObject(&bitmap);

MemDC.BitBlt(0,0,rect.Width(),rect.Height(),pDC,0,0,SRCCOPY);

///////////此处对MemDC进行任意绘图

HDC hDC = MemDC.GetSafeHdc();

。。。。。。。。

///最终进行显示

pDC->BitBlt(0,0,rect.Width(),rect.Height(),&MemDC,0,0,SRCCOPY);

ReleaseDC(&MemDC);

ReleaseDC(pDC);

### DC分类

#### CDC类的对象

父类CObject，封装了win32下所有绘图的API函数，还封装了m\_hDC(保存了绘图设备句柄)

#### CClientDC

封装了在窗口的客户区中绘图的绘图设备(封装了CDC类，在构造时调用了GetDC，析构时调用了ReleseDC)

#### CWindowDC

可在标题栏绘图

CWindowDC dc(AfxGetApp()->m\_pMainWnd);

dc.TextOut(0,0,”xxx”); 可以画到框架窗口栏，因为以框架为句柄

第三个参数可以直接是CString类的对象str

或者用GetParent()

如果使用GetDesktopWindow()则可以在桌面绘图,即在应用程序窗口外部绘图

#### CPaintDC

封装了在WM\_PAINT消息中绘图的绘图设备

只能用于WM\_PAINT也就是OnPaint函数中（该函数也可以用CClientDC），其他地方不管用

不能写在函数的if这类语句中，要写在函数最外层

#### CMetaFileDC

是一个文件dc。在调用这个dc时，使用Ellipse等函数相当于给这个dc发绘图命令，让这个dc存储这些命令，使用时可以直接将所有的绘图命令执行出来。

##### wmf

1）创建

CMetaFileDC fileDC;

fileDC.Create();

2）使用

不需要调用OnDraw这类的函数进行重绘

fileDC.Ellipse(point.x-10,point.y-10,point.x+10,point.y+10);

CClientDC dc(this);

HMETAFILE hMetaFile=fileDC.Close();

dc.PlayMetaFile(hMetaFile);

fileDC.Create(); //再次创建

fileDC.PlayMetaFile(hMetaFile);//将之前的图形在fileDC上面再画一次，相当于创建fileDC后把之前的绘图命令放进去

DeleteMetaFile(hMetaFile);

3）保存

HMETAFILE hMetaFile=fileDC.Close();

CopyMetaFile(hMetaFile,\_T("d:\\f.wmf"));

fileDC.Create();

DeleteMetaFile(hMetaFile);

4）打开

HMETAFILE hMetaFile=GetMetaFile(\_T("d:\\f.wmf"));

fileDC.PlayMetaFile(hMetaFile);

CClientDC dc(this);

hMetaFile=fileDC.Close();

dc.PlayMetaFile(hMetaFile);

fileDC.Create();

fileDC.PlayMetaFile(hMetaFile);

DeleteMetaFile(hMetaFile);

##### emf

1）创建

CMetaFileDC metaDC;

RECT rectHimm,rectPixel;

Init中

HDC hdcRef=GetDC()->m\_hDC;

int iWidthMM=GetDeviceCaps(hdcRef,HORZSIZE);

int iHeightMM=GetDeviceCaps(hdcRef,VERTSIZE);

int iWidthPels=GetDeviceCaps(hdcRef,HORZRES);

int iHeightPels=GetDeviceCaps(hdcRef,VERTRES);

rectHimm.left=0;

rectHimm.top=0;

rectHimm.right=iWidthMM\*100;

rectHimm.bottom=iHeightMM\*100;

rectPixel.left=0;

rectPixel.top=0;

rectPixel.right=iWidthPels;

rectPixel.bottom=iHeightPels;

metaDC.CreateEnhanced(GetDC(),NULL,&rectHimm,NULL);

2）使用

metaDC.Ellipse(point.x-10,point.y-10,point.x+10,point.y+10);

HENHMETAFILE hemf=metaDC.CloseEnhanced();

PlayEnhMetaFile(GetDC()->m\_hDC,hemf,&rectPixel);

metaDC.CreateEnhanced(GetDC(),NULL,&rectHimm,NULL);

metaDC.PlayMetaFile(hemf,&rectPixel);

DeleteEnhMetaFile(hemf);

1. 保存 //打开和保存后无法编辑

HENHMETAFILE hemf=metaDC.CloseEnhanced();

CopyEnhMetaFile(hemf,\_T("d:\\x.emf"));

DeleteEnhMetaFile(hemf);

1. 打开

HENHMETAFILE hemf=GetEnhMetaFile(\_T("d:\\x.emf"));

UINT size=GetEnhMetaFileHeader(hemf,0,NULL);

ENHMETAHEADER \*emHeader=(ENHMETAHEADER \*)malloc(size);

GetEnhMetaFileHeader(hemf,size,emHeader);

RECTL rect1=emHeader->rclBounds;

RECT rect={rect1.left,rect1.top,rect1.right,rect1.bottom};

PlayEnhMetaFile(GetDC()->m\_hDC,hemf,&rect);

DeleteEnhMetaFile(hemf);

#### DC的属性

dc.GetBkColor()可以获得背景颜色

CSize size=dc.GetTextExtent(str); 可以获得CString类对象str的尺寸

TEXTMETRIC tm;

dc.GetTextMetrics(&tm); 可以获得dc中字体的格式（如字体高度）

#### GDI绘图对象

CPaintDC，CClientDC,CWindowDC不能使用delete函数删除。

自己定义的CDC，需要用DeleteDC删除；定义的pen和brush要用DeleteObject进行删除。

CPen/CBrush/CFont/CBitmap

CGdiObject 父类CObject

封装了m\_hObject(保存了和对象绑定在一起的GDI绘图设备句柄)

##### CPen

CClientDC dc(this);

CPen pen(PS\_SOLID,10,RGB(255,0,0));//画刷类型非PS\_SOLID时，画笔宽度1才有效

CPen \*oldpen=dc.SelectObject(&pen);

dc.SelectObject(oldpen);

pen.DeleteObject();

##### CBrush

CBrush brush(RGB(255,0,0));

dc.SelectObject(&brush)

或者CBrush brush(HS\_CROSS,RGB(255,0,0));

或者也可以用CBrush brush(\*bitmap) 位图的画刷

CBrush \*pbrush=

CBrush::FromHandle((HBRUSH)GetStockObject(NULL\_BRUSH));空画刷

设置画刷背景色，阴影画刷时不设置背景色会是白色

dc.SetBkMode(TRANSPARENT);

##### CFont

dc.SetBkMode(TRANSPARENT);可以设置文字背景透明

CFont font;

font.CreatePointFont(120,"方正喵呜体"); //120代表字体大小12号

dc.SelectObject(&font);//画文字时，没有使用pen和brush

COLORREF clr=dc.SetTextColor(RGB(255,0,0));

COLORREF clBk=dc.SetBkColor(RGB(0,0,255));//设置文字背景色

dc.TextOut(0,0,"xxxx"); //用设置的红色绘制字体

dc.SetTextColor(clr);

dc.SetBkColor(clBk);

DrawText(str,rect,DT\_LEFT)可在指定矩形区域用指定方式绘制文字

设置字体大小

void SetPdcFont(CDC \* pDC, int fontHeight)

{

CFont font;

LOGFONT lf;

memset(&lf, 0, sizeof(LOGFONT));

lf.lfHeight = fontHeight;

VERIFY(font.CreateFontIndirect(&lf));

CFont\* def\_font = pDC->SelectObject(&font);

}

### 文字处理

#### 获取系统所有字体名

int CALLBACK GetNameFun(

\_In\_ ENUMLOGFONT \*lpelf,

\_In\_ NEWTEXTMETRIC \*lpntm,

\_In\_ DWORD FontType,

\_In\_ LPARAM lParam

)

{

list<CString> \*pfontName = (list<CString>\*)lParam;

pfontName->push\_back(lpelf->elfFullName);

return TRUE;

}

list<CString> fontName;

::EnumFontFamilies(GetDC()->m\_hDC, (LPTSTR) NULL, (FONTENUMPROC)GetNameFun, (LPARAM)&(fontName));

#### 文字输出

dc.TextOut(x,y,str);

中文字符宽度

int widthText = tm.tmAveCharWidth\*2

#### 创建插入符

在Create消息中

CClientDC dc(this);

TEXTMETRIC tm;

dc.GetTextMetrics(&tm); //获得DC中的字体的信息

CreateSolidCaret(tm.tmAveCharWidth/8,tm.tmHeight); //利用字体信息中的宽度和高度创建插入符

ShowCaret(); //显示插入符

也可以使用位图插入符：

CBitmap bmp; //类内成员变量

bmp.LoadBitmap(IDB\_BITMAP1);

CreateCaret(&bmp);

#### 制作输入文本

在输入文字时，每次都是从输入符（pt的位置）最开始的位置重新刷新显示一下，用户输入的字母不断加入CString中，然后重新显示一遍

CString m\_str; //类内定义一个存储字符串的变量，用来记录一行字符串的值，当换行时，m\_str才需要完全清空

CPoint m\_pt; //类内定义一个存鼠标上一次点击时的位置的变量，相当于字符串开始绘制的位置，当换行时候，m\_pt才会发生变化，变化的值是y值

类的构造函数中，进行初始化：

m\_str=””;

m\_pt=0;

在WM\_LBUTTONDOWN消息中：

SetCaretPos(point); //改变插入符的位置

m\_str.Empty(); //将存放字符串的变量内容清空

m\_pt=point; //将当前鼠标的位置保存下来，用来当输入点

在WM\_CHAR消息中：

CClientDC dc(this);

TEXTMETRIC tm;

dc.GetTextMetrics(&tm);

if(13==nChar){ //13代表回车键（CR）,在msdn中ASCII character

codes可查到字母的ascii码

m\_str.Empty(); //字符串进行清空

m\_pt.y+=tm.tmHeight; //按回车时，输入的坐标点变到下一行

}

else if(8==nChar){ //8代表退格键（BS）

COLORREF clr=dc.SetTextColor(dc.GetBkColor());

dc.TextOut(m\_pt.x,m\_pt.y,m\_str);//用背景色将字符串画出来一遍，相当于原来的字符串消失

m\_str=m\_str.Left(m\_str.GetLength()-1);//如果是多字节，需要判断ASCII码，决定-1还是-2

dc.SetTextColor(clr);

}

else{

m\_str+=nChar; //将字符加入到字符串

}

CSize size=dc.GetTextExtent(m\_str);

CPoint pt; //用来记录新的插入符的位置，此时不可改变

m\_pt的值

pt.x=m\_pt.x+size.cx;

pt.y=m\_pt.y;

SetCaretPos(pt); //将插入符设置到新的位置

dc.TextOut(m\_pt.x,m\_pt.y,m\_str); //绘制出字符串

#### 字体逐渐变色效果

在类内定义一个变量m\_width,用来存放渐变的宽度大小

已经用TextOut函数画出文字

CString str("xxxxxxxxxxxxxxxxxxxxxxxxxxxxx");

dc.TextOut(0,10,str);

设置一个定时器，在定时器消息处理中：

CClientDC dc(this);

TEXTMETRIC tm;

dc.GetTextMetrics(&tm);

dc.SetTextColor(RGB(255,0,0));

m\_width+=3;

CRect rect(0,10,m\_width,10+tm.tmHeight);

CString str("xxxxxxxxxxxxxxxxxxxxxxxxxxxxx");

dc.DrawText(str,rect,DT\_LEFT);

如果使用DT\_RIGHT则会出现文字从最后一个字母开始出现，相当于文字从左向右移动

可以使用

CSize size=dc.GetTextExtent(str);

if(m\_width>size.cx)

m\_width=0; 进行重置

### 图形绘制

#### 绘制点

dc.SetPixel(CPoint(100,100),RGB(255,0,0));

#### 直线绘制

CPoint pt(12,12);

pt取代MoveTo等函数中x,y两个代表点的参数

dc.MoveTo(pt1);

dc.LineTo(pt2);

#### 矩形/椭圆

画椭圆(Ellipse)或者矩形(Rectangle)时，都使用了四个坐标点

dc.Ellipse(rect);

(x1,y1)

(x2,y2)

CRect rect(x1,y1,x2,y2);

rect可作为LPCRECT类型参数传入

#### 画刷/边框绘图

FillRect函数相当于Rectangle函数没有画笔，只有画刷

dc.FillRect(CRect(20,20,100,100),&CBrush(RGB(100,0,0)));

FrameRect相当于Rectangle函数只有画刷，没有画笔

dc.FrameRect(CRect(200,200,400,400),&CBrush(RGB(0,0,100)));

#### 限制dc输出区域

对dc的输出区域进行设置后，dc输出时候只在指定区域绘图

CClientDC dc(this);

//设置输出区域为五角星

POINT point[5]={{0,200},{600,200},{100,600},{300,0},{500,600}};

HRGN hrgn=CreatePolygonRgn(point,5,WINDING);

::SelectClipRgn(dc.m\_hDC,hrgn);

#### 曲线

贝塞尔曲线：至少需要四个点，第一个点和最后一个点是首尾，其他点是中间控制点。

CClientDC dc(this);

POINT pt[4]={{0,0},{100,50},{100,200},{0,100}};

dc.PolyBezier(pt,4);

Polyline(pt,num); 可以将num个点连接起来

绘制sin曲线：

#include <math.h>

#define PI2 2\*3.14

#define NUM 1000

CClientDC dc(this);

POINT pt[NUM];

for(int i=0;i<NUM;i++)

{

int width=400;

int height=300;

pt[i].x=i\*width/NUM;

pt[i].y=(int)(height/2\*(1-sin(i\*PI2/NUM)));//sin的结果有正负，所以要用1去减，全部转化为正数

}

dc.Polyline(pt,NUM);

#### 不规则图形

1. 利用CRgn对象调用一系列CreateXXX函数创建基本规则图形
2. 调用CRgn::CombineRng合并规则图形（按照指定规则合并）

3)调用CDC::FillRgn给合并的图形填充颜色

4)调用CDC::FrameRgn给合并的图形设置线条颜色

CClientDC dc(this);

1) CRgn rgn1;

rgn1.CreateEllipticRgn(0,0,500,500);

CRgn rgn2;

rgn2.CreateEllipticRgn(0,0,250,250);

2) rgn1.CombineRgn(&rgn1,&rgn2,RGN\_AND);

3) CBrush brushFill(RGB(255,0,0));

dc.FillRgn(&rgn1,&brushFill);

4) CBrush brushFrame(RGB(0,255,0));

dc.FrameRgn(&rgn1,&brushFrame,2,2);最后两个参数代表线条的水平方向和垂直方向粗细

两圆形区域的交界处即为合成后的图形

（250，250）

（500，500）

::SetROP2(R2\_NOT)

#### 画区域路径

（让两个图在同一区域画图时进行运算）

CString str("xxxxxx");

pDC->TextOut(100,100,str);

CSize size=pDC->GetTextExtent(str); //得到字体区域的大小

pDC->BeginPath(); //开始画选择区域

pDC->Rectangle(100,100,size.cx+100,size.cy+100);

//将文字区域放入路径中

pDC->EndPath(); //结束画选择区域

pDC->SelectClipPath(RGN\_DIFF);

//使下面画的圆圈不会覆盖上面字体的区域，也可用

RGN\_AND，则圆圈只画区域相交部分

pDC->Ellipse(0,0,300,300);

#### 铅笔绘图

CPoint ptStart;

BOOL btnDown=FALSE;

void C\*\*View::OnLButtonDown(UINT nFlags, CPoint point)

{

ptStart=point;

btnDown=TRUE;

CView::OnLButtonDown(nFlags, point);

}

void C\*\*View::OnMouseMove(UINT nFlags, CPoint point)

{

CClientDC dc(this);

if(btnDown){

dc.MoveTo(ptStart);

dc.LineTo(point);

ptStart=point; //模仿铅笔绘图

}

CView::OnMouseMove(nFlags, point);

}

void void C\*\*View::OnLButtonUp(UINT nFlags, CPoint point)

{

btnDown=FALSE;

CDialogEx::OnLButtonUp(nFlags, point);

}

#### 获取颜色

COLORREF ref=dc.GetPixel(CPoint(100,100));

### 图像绘制

#### DIB

宽度可以非4的倍数,

点阵的内存必须是4的倍数

int channel = 3;

int Width = 301;

int Height = 300;

int val = Width\*channel;

int widstep = val%4 == 0 ? val: val + 4 - val%4;

int dibBufSize = widstep\*Height;

char chBmpBuf[2048] = { 0 };

BITMAPINFO \*pBmpInfo = (BITMAPINFO \*)chBmpBuf;

pBmpInfo->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);

pBmpInfo->bmiHeader.biWidth = Width;//位图的宽度像素为单位

pBmpInfo->bmiHeader.biHeight = Height;//位图的高度

pBmpInfo->bmiHeader.biPlanes = 1;//必须为1

pBmpInfo->bmiHeader.biBitCount = 8 \* channel;//每个像素的位数(必须是8的倍数)

pBmpInfo->bmiHeader.biCompression = BI\_RGB;//压缩方式，一般为0或者BI\_RGB（未压缩）

//info.bmiHeader.biSizeImage 以字节为单位的图像大小（仅用于压缩位图）

//info.bmiHeader.biXPelsPerMeter 以目标设备每米的像素来说明位图的水平分辨率

//info.bmiHeader.biYPelsPerMeter 以目标设备每米的像素来说明位图的垂直分辨率

//info.bmiHeader.biClrUsed 颜色表的颜色数，如果是0,则使用最大颜色数

//info.bmiHeader.biClrImportant 重要颜色数目，如果是0，表示所有颜色都重要

//// 黑白图像需要初始化调色板

for (int i = 0;i<256;i++)

{

pBmpInfo->bmiColors[i].rgbBlue = i;

pBmpInfo->bmiColors[i].rgbGreen = i;

pBmpInfo->bmiColors[i].rgbRed = i;

pBmpInfo->bmiColors[i].rgbReserved = i;

}

//BYTE\* PDIBBuffer = NULL;

//HBITMAP hDib = CreateDIBSection(NULL, pBmpInfo, DIB\_RGB\_COLORS, (void\*\*)&PDIBBuffer, NULL, 0);//HDC仅在 usage参数为DIB\_PAL\_COLORS才使用

int bufSize = pBmpInfo->bmiHeader.biBitCount\*pBmpInfo->bmiHeader.biWidth\*pBmpInfo->bmiHeader.biHeight;

BYTE\* PDIBBuffer = new BYTE[bufSize];

memset(PDIBBuffer, bufSize, 0);

//从左下方开始，将图像左下角一半变成红色

for (int row = 0; row <= 150; ++row)

{

for (int col = 0; col <= 150; ++col)

{

int pos = widstep \* row + col \* channel;

PDIBBuffer[pos++] = 0; //B

PDIBBuffer[pos++] = 0; //G

PDIBBuffer[pos++] = 255; //R

}

}

CRect wrect;

GetClientRect(wrect);

CDC \*pdc = GetDC();

HDC hdc = pdc->GetSafeHdc();

::SetStretchBltMode(hdc, COLORONCOLOR);

::StretchDIBits(hdc,

0, wrect.Height(), wrect.Width(), -wrect.Height(),

0, 0, pBmpInfo->bmiHeader.biWidth, pBmpInfo->bmiHeader.biHeight,

PDIBBuffer, pBmpInfo,

DIB\_RGB\_COLORS, SRCCOPY);//0, wrect.Height(), wrect.Width(), -wrect.Height()可以进行反转显示

ReleaseDC(pdc);

//DeleteObject(hDib);

#### 显示BMP

CClientDC dc(this);

CBitmap bmp;

//bmp.LoadBitmap(IDB\_BITMAP1);

HBITMAP hBitmap = (HBITMAP)::LoadImage(

AfxGetInstanceHandle(), \_T("D:\\1.bmp"),

IMAGE\_BITMAP, 0, 0,

LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);

bmp.Attach(hBitmap);

//得到图片的尺寸信息

BITMAP BitInfo;

bmp.GetBitmap(&BitInfo);

int width = BitInfo.bmWidth;

int height = BitInfo.bmHeight;

CDC memDC;

memDC.CreateCompatibleDC(&dc);

CBitmap \*oldbmp = memDC.SelectObject(&bmp);

//dc.BitBlt(100, 100, 800, 700, &memDC, 0, 0, SRCCOPY);

//100，100代表图片左上角的坐标位置，800，700代表显示图片的框架的大小(这四个坐标相当于进行图像裁剪)

//伸缩显示

//可以将图片从本来的width,height变成客户区大小显示

CRect rect;

GetClientRect(&rect);

dc.StretchBlt(0, 0, rect.right, rect.bottom, &memDC, 0, 0, width, height, SRCCOPY);

memDC.SelectObject(oldbmp);

memDC.DeleteDC();

bmp.DeleteObject();

#### 合成图

在画完位图后，用同一个dc设置位置再画一个位图，则会覆盖在前一个位图上面。

CClientDC dc(this);

CBitmap bmp;

bmp.LoadBitmap(IDB\_BITMAP1);

CDC memDC;

memDC.CreateCompatibleDC(&dc);

CBitmap \*oldbmp=memDC.SelectObject(&bmp);

dc.BitBlt(500,0,800,700,&memDC,0,0,SRCCOPY);

bmp.DeleteObject();

//位图bmp中的缓存需要删除；不能删除memDC，后面需要用；oldbmp一直保存着原来的内容，结束后在进行还原

bmp.LoadBitmap(IDB\_BITMAP2);

memDC.SelectObject(&bmp);

dc.BitBlt(700,200,800,700,&memDC,0,0,SRCCOPY);

memDC.SelectObject(oldbmp);

memDC.DeleteDC();

bmp.DeleteObject();

#### 去图片底色

原图 255 255 0

掩码 255 0 0

掩码求反 0 255 255

与原图位与 255 255 0

结果 0 255 0

CClientDC dc(this);

CBitmap bmp;

bmp.LoadBitmap(IDB\_BITMAP1);

CDC memDC;

memDC.CreateCompatibleDC(&dc);

memDC.SetBkColor(RGB(0,0,0));//代表要去掉的颜色

CBitmap \*oldmap=memDC.SelectObject(&bmp);

CDC maskDC;//掩码DC

maskDC.CreateCompatibleDC(&memDC);

CBitmap maskbmp;

maskbmp.CreateBitmap(400,400,0,0,NULL); //创建掩码位图，两个400代表掩码图的图像大小（设置成和位图一样的大小,假设位图为400\*400）

CBitmap \*oldmask=maskDC.SelectObject(&maskbmp);

maskDC.BitBlt(0,0,400,400,&memDC,0,0,SRCCOPY);

dc.BitBlt(0,0,400,400,&memDC,0,0,SRCINVERT);

dc.BitBlt(0,0,400,400,&maskDC,0,0,SRCAND);

dc.BitBlt(0,0,400,400,&memDC,0,0,SRCINVERT);

memDC.SelectObject(oldmap);

memDC.DeleteDC();

bmp.DeleteObject();

maskDC.SelectObject(&oldmask);

maskDC.DeleteDC();

maskbmp.DeleteObject();

#### 获取剪贴板图像

if (OpenClipboard())

{

HBITMAP handle = (HBITMAP) GetClipboardData(CF\_BITMAP);

if (handle!=NULL)

{

CBitmap\* bmp = CBitmap::FromHandle(handle);

CClientDC dc(this);

CDC memDC;

memDC.CreateCompatibleDC(&dc);

memDC.SelectObject(bmp);

BITMAP BitInfo;

bmp->GetBitmap(&BitInfo);

int width=BitInfo.bmWidth;

int height=BitInfo.bmHeight;

dc.StretchBlt(0,0,width,height,&memDC,0,0,width,height,SRCCOPY);

}

CloseClipboard();

}

## dialog

### 外观

#### 全屏

ModifyStyle(WS\_BORDER, WS\_OVERLAPPED);或者Border属性设置为None

//ShowWindow(SW\_SHOWMAXIMIZED);

int cx = GetSystemMetrics(SM\_CXSCREEN);

int cy = GetSystemMetrics(SM\_CYSCREEN);

MoveWindow(0,0,cx,cy);

#### 背景

可以把背景色刷成和背景图片的背景颜色一样，然后把背景图片放到屏幕中间。

注意设置背景色时，是在OnPaint函数的else里面，必须将CDialog::OnPaint();注释掉。

CBrush brush(RGB(0,0,22));

CPaintDC dc(this);

CRect rectfill;

GetClientRect(rectfill);

dc.FillRect(rectfill,&brush);

也可以使用WM\_CTLCOLOR消息中返回画刷来改变背景色

#### 控件位置

OnSize函数中调节控件位置要进行判断

if (GetWindow(GW\_CHILD) != NULL)

{

GetDlgItem(IDC\_BUTTON1)->MoveWindow(0, 0, cx / 2, cy / 2);

}

int cx=::GetSystemMetrics(SM\_CXSCREEN);

int cy=::GetSystemMetrics(SM\_CYSCREEN);

CRect rect;

m\_btn.GetClientRect(rect);//GetWindowRect获取关于屏幕坐标位置

m\_btn.MoveWindow(.............);

xxx.SetWindowPos(&m\_ctl, 0, 0, rect.Width(), rect.Height(), 0);

第一个参数代表插入到哪个窗口位置后面，如果是null，则放到顶层。

Ctrl+d 编辑时调整控件z序列，数字小控件越顶层，只能点击数字大的，让其变小

#### 颜色和透明背景

除了静态文本，此功能只能用于多字节

可用于对话框，也可用于对话框上的控件（只要是窗体就可以）

添加对话框的WM\_CTLCOLOR消息响应函数

在return hbr上面加入语句进行控制

(设置字体时要注意大小)

CFont font; //成员变量，不能在CTLCOLOR中反复创建对象

switch(pWnd->GetDlgCtrlID())

{

case IDC\_TEXT: //静态文本框的ID，也可用于输入文本及其他控件

{

font.DeleteObject();

font.CreatePointFont(180,"方正喵呜体"); //180代表18号字体

pDC->SelectObject(&font); //此处只选入，不能在return前选回旧的

pDC->SetTextColor(RGB(255,0,0)); //设置文字颜色

pDC->SetBkMode(TRANSPARENT); //透明化文字背景

return (HBRUSH)::GetStockObject(NULL\_BRUSH);

// HBRUSH b=CreateSolidBrush(RGB(135,206,235)); //成员变量，不能在CTLCOLOR中反复创建对象

}

break;

default :

break;

}

如果有某个函数调用时会改变文字（编辑框是EN\_CHANGE里面）

那么在那个函数中加入

CRect rect;

GetDlgItem(ID)->GetWindowRect(rect);

ScreenToClient(rect);

InvalidateRect(rect); //只刷新控件的显示

不能加入到CTLCOLOR中

Center Image属性选择True文本则居中显示

#### 标题栏重绘制拖动

处理NCHITTEST消息

LRESULT CtestmfcDlg::OnNcHitTest(CPoint point)

{

CRect rect;

GetClientRect(rect);

ClientToScreen(rect);

//使用某个控件的顶部位置限制标题栏检测底边

CRect rt;

GetDlgItem(IDC\_LIST1)->GetWindowRect(rt);

rect.bottom = rt.top;

return rect.PtInRect(point) ? HTCAPTION: CDialog::OnNcHitTest(point);

}

OnPaint()函数else中设置标题栏颜色

CWnd \*pwnd = GetDlgItem(IDC\_LIST1);

if (NULL != pwnd)

{

CBrush brush(RGB(0,200,122));

CPaintDC dc(this);

CRect rectfill;

GetClientRect(rectfill);

CRect rt;

pwnd->GetWindowRect(rt);

ScreenToClient(rt);

rectfill.bottom = rt.top;

dc.FillRect(rectfill,&brush);

}

#### 鼠标指针

HCURSOR m\_hCurSor;

m\_hCurSor = LoadCursorFromFile(\_T("skin\\curFile.cur"));

//LoadCursor( NULL  , IDC\_CROSS ) ;获取系统定义的鼠标

SETCURSOR消息处理

BOOL Dlg::OnSetCursor(CWnd\* pWnd, UINT nHitTest, UINT message)

{

::SetCursor(m\_hCurSor);

return TRUE;

}

App类最后一行

::DestroyCursor(dlg.m\_hCurSor);

#### 窗口显示状态获取

WINDOWPLACEMENT lwndpl;

GetWindowPlacement(&lwndpl);

if (SW\_SHOWMAXIMIZED == lwndpl.showCmd)

#### 获取控件在客户区位置

CRect rect;

GetDlgItem(IDC\_PIC)->GetWindowRect(rect);

ScreenToClient(rect);

### 子对话框

插入一个对话框资源，假设ID为：IDD\_DIALOG1

双击对话框，创建一个类Cdlg

类内变量

Cdlg \*pdlg;

pdlg=new CFL();

pdlg->Create(IDD\_DIALOG1);

pdlg->ShowWindow(SW\_SHOW);

### 可变大小

设置勾选属性Minimize box和maximize box

border属性选择resizing

### 控件功能扩展

每个控件能响应的消息和其中的变量是固定的，如果需要扩展，可以继承这个控件类，然后增加函数和变量。

以按钮为例，增加一个新的类class CBtnX : public CButton，在使用向导为按钮绑定控件型变量时，不选CButton，选择CBtnX即可（注意在对话框类绑定变量，需要包含CBtnX的头文件）。

此时可以为CBtnX增加变量以及各种消息响应，这样按钮就可以响应别的消息以及拥有其他功能。

### 数据交互

不能在线程中对控件进行调整，只能在线程中对控件状态进行判断。

对话框与外界数据交互：可以为对话框加入一个GetData函数，用指针变量成员（m\_p）接收外界的变量的指针(m\_pData)，然后在对话框中利用这个m\_p就可以修改外界的数据

CDlgwidth dlg;

dlg.GetData(&m\_penwidth);

dlg.DoModal();

### 获取主窗口

AfxGetMainWnd()->m\_hWnd

### 消息传递

对话框的控件在收到点击消息时，会拦截消息，使消息在当前控件处理后不再被其他控件或者对话框响应，如果要增加其他的响应，需要在控件中向其他窗口发消息。

### CtoolTipCtrl

可以为指定控件（窗口）增加提示，当鼠标移动到控件上时，弹出提示。

类内变量:

CToolTipCtrl m\_tt;

初始化:

EnableToolTips(TRUE);

m\_tt.Create(this);

m\_tt.Activate(TRUE);

m\_tt.AddTool(GetDlgItem(ID),"提示内容");

PreTranslateMessage:

m\_tt.RelayEvent(pMsg);

### 菜单

创建菜单资源

CMenu m\_menu;

m\_menu.LoadMenu(IDR\_MENU1);

SetMenu(&m\_menu);

选择属性添加事件处理函数

### 按钮

双击可进入代码，编辑按钮点击的响应函数

GetDlgItem(IDC\_BUTTON1)->EnableWindow(FALSE);

IsWindowEnabled()

判断按钮是否可用

### 文本框

#### 基本操作

编辑框的文本属性样式可以使用密码模式

静态文本如果需要响应点击消息，需要将style中勾选notify，同时修改原来的IDC\_STATIC

CString str=\_T("");

//获取和设置编辑框内容

GetDlgItemText(IDC\_EDIT1,str); //所有控件都可以使用这个获得控件上显示的文字

SetDlgItemText(IDC\_EDIT1,str); //也可用于设置其它控件的文本

//可以将控件属性设置为数字，则只可以输入数字

int i=GetDlgItemInt(IDC\_EDIT1);

SetDlgItemInt(IDC\_EDIT1,i);

#### 静态文本控制

静态文本绑定变量control型m\_text

类内变量：CFont m\_Font;

初始化时：

m\_Font.CreatePointFont(300, "宋体", NULL);

m\_text.SetFont(&m\_Font,true);

#### 粘连选择

//复选编辑框内容，相当于鼠标粘连选择，注意设置后需要设置焦点，否则焦点不在编辑框，而看不见效果。setSel函数第一个参数代表起始位置，第二个代表结束位置（如果-1，代表到结尾）

((CEdit\*)GetDlgItem(IDC\_EDIT1))->SetSel(0,-1);

GetDlgItem(IDC\_EDIT1)->SetFocus();

#### 回车键换行

Want return属性true，在编辑框中按下Enter键时即可换到下一行而不关闭对话框。

#### 多行显示

multiline 和 vertical scroll属性（可以设置Read Only属性为ture）

编辑框绑定变量：数值型m\_show，控件型m\_control

假设要加入的数据是str

if(m\_show==\_T("")){ //如果是第一次添加，就不换行

m\_show+=str;

}

else{

m\_show+=\_T("\r\n")+str;

}

this->UpdateData(FALSE);

#### 编辑框日志显示

##### 属性

Multiline, Want return,read only勾选

vertical scroll

##### 加入数据

static CString log;

log += strLine;

log += \_T("\r\n");

m\_logCtl.SetWindowText(log);

m\_logCtl.LineScroll(m\_logCtl.GetLineCount());

UpdateData(FALSE);

##### 清空

m\_logCtl.SetWindowText(\_T(""));

m\_logCtl.LineScroll(0);

UpdateData(FALSE);

### Check Box

绑定变量或使用IsDlgButtonChecked(m\_hWnd, IDC\_CHECK1)进行勾选判断

方型，打勾选择

//GetCheck判断勾选状态，SetCheck设置勾选

if(1==((CButton\*)GetDlgItem(IDC\_CHECK1))->GetCheck()){

((CButton\*)GetDlgItem(IDC\_CHECK1))->SetCheck(0);

}

else{

((CButton\*)GetDlgItem(IDC\_CHECK1))->SetCheck(1);

}

======代码创建CheckBox

CButton \*pBtn = new CButton();

RECT rect = {x,y,x+width,y+height};

pBtn->Create(\_T("标题"),WS\_TABSTOP|WS\_VISIBLE|WS\_CHILD|BS\_AUTOCHECKBOX,rect,this,id);

### Radio Button

圆形，打点选择（互斥，只能选中一个）

默认以有组属性的单选按钮和该按钮之后连续的按钮为一组，直到遇到下一个组属性的按钮。

ctrl+d 调出layout，改变先后编号

单选按钮多个时，利用layout排序，将顺序弄连续，然后将编号小的控件属性选择组属性，即可分组（可用分组框进行修饰）

分组后，类向导中会有组属性的控件ID，绑定成员变量，通常使用int型。值为-1，代表没有选择任何选项。

### Combo Box

支持用户的输入和选择，可下拉

（最好在加入时将高度弄高，修改时需要点击下拉按钮，然后把鼠标放到控件下方中间点才可以修改）

在属性中有数据项目，可以输入项目名称，按ctrl+回车，切下一行（VS中用;写表示换行）

如果属性里样式类型改为下拉列表（drop list），就只能选择，不能接受输入，可勾选分类，让其自动分类

Control,CcomboBox

m\_combox.EnableWindow(BOOL) 设置组合框是否可用

int nSel=m\_combox.GetCurSel();//得到当前选择的位置，如果没有选择则为-1

m\_combox.GetLBText(nSel,str);//获取选择位置的数据

### 图片控件

类型可以选择位图，图标等

先在资源中导入图片，然后在图片控件的图像中找到选择图片的ID，则图片控件会显示图片

图片控件还可以选择框架类型，颜色选最后一种，则可做出一条直线

（刻在窗口上的线条）

动态加载:

CBitmap bitmap;

bitmap.LoadBitmap(IDB\_BITMAP1);

HBITMAP hBmp = (HBITMAP)bitmap.GetSafeHandle();

pic.SetBitmap(hBmp); //图片控件绑定变量

或者：

HBITMAP hBmp=(HBITMAP)::LoadImage(

AfxGetInstanceHandle(),\_T("d:\\1.bmp"),

IMAGE\_BITMAP,0,0,

LR\_LOADFROMFILE|LR\_CREATEDIBSECTION); pic.SetBitmap(hBmp);

### 时间控件

CTime time;

DWORD dwResult = m\_dateCtl.GetTime(time);

if (dwResult != GDT\_VALID)

{

AfxMessageBox(\_T("Time not set!"));

}

CString strDate = time.Format(\_T("%Y-%m-%d"));

### List Box

绑Control类型

样式的选择里，可设置单个或者多个的选择，无法通过属性添加数据

listbox.SetItemHeight(index,30);//设置列表高度

if(LB\_ERR==m\_list.FindString(-1,cstring)){

//搜索判断类表中所有项目名称，如果没有名称cstring则执行

int nIndex=m\_list.AddString(cstring);

CString \*pPath=new CString;

//为list添加附加数据（比如添加地址，后面可以取出）

//如果添加的数字是数字，就可以直接加入。。不用这个方式

\*pPath=m\_strFilePath;

m\_list.SetItemData(nIndex,(DWORD)pPath);

}

找到列表框IDC\_LIST的LBN\_DBLCLK（双击消息），然后进入相应的处理函数取出pPath：

int nSel=m\_list.GetCurSel();

if(LB\_ERR==nSel) return;

else{

CString \*pPath=(CString \*)m\_list.GetItemData(nSel);

}

new出的pPath需要考虑释放，使用m\_list.GetCount()统计出数量，在for循环中delete掉从GetItemData中拿出的附加数据

int count=m\_list.GetCount();

for(int i=0;i<count;i++){

CString \*pPath=(CString \*)m\_list.GetItemData(i);

delete pPath;}

m\_list.GetText(nSel,str);可获得列表框文本

m\_list.DeleteString(nSel); 将列表框中某行删除

### 调节数值的控件

（可以为控件增加相应的消息处理函数）

设置/获取控件表示的数值(位置)范围：（可以不设置，默认从0到100）

SetRange(int nLower, int nUpper )/GetRange(&nLower, &nUpper )

获取/设置控件的当前位置:

GetPos() /SetPos(int)

响应事件使用对话框的WM\_HSCROLL

或者控件的NM\_RELEASEDCAPTURE

#### 旋转按钮（Spin）

通常与编辑框控件一起使用(与编辑框序号连续，且编辑框序号在前，勾选旋转按钮样式Auto Buddy【自动结伴】，SetBuddy integer【设置结伴整数】，左边排列方式选择靠右。这样在运行时两个控件就会合成一体)，用来完成某个数值的增加或者减少，对应控件类型CSpinButtonCtrl

设置控件的增量/步长：

UDACCEL accel;

accel.nSec=1; //改变数值前所经过的秒数（可用于接收）

accel.nInc=2; //每次数值改变的大小

m\_spin.SetAccel(1,&accel); //第一个参数用来表示有几个accel，一般写1

#### 进度条（Progress）

通常用于安装程序或者复制文件时进度显示，对应CProgressCtrl

m\_progress.SetStep(5); //设置每次的增量（满值是100，所以100/5

就是需要多少次才到满值）

m\_progress.StepIt(); //执行此语句时进度条增加

int nPos=m\_progress.GetPos(); //得到当前进度值

#### 滑块（Slider）

通常用于滑动调节某些数值，对应CSliderCtrl

m\_slider.SetRange(0,255); //设置变化范围0-255

NM\_RELEASEDCAPTURE 鼠标拖动后弹起事件

int pos = m\_slider.GetPos(); //获取位置

### 滚动条

在创建窗口时包含窗口样式WS\_VSCROLL和WS\_HSCROLL即可拥有垂直滚动和水平滚动条。

SetScrollRang函数可以设置卷动列的范围，默认是0到100。如果在这个函数后又使用了其他影响绘制的函数，需要在这个函数的最后一个参数bRedraw设置为FALSE。

SetScrollPos函数可以设置卷动列上面的滑块的当前位置。

用鼠标拖动滑块或者点击卷动列时，会产生WM\_VSCROLL和WM\_HSCROLL消息。每个动作会产生两个消息，一个是按下时发生的，一个是释放时发生的。

其中的wParam表示了对卷动列的操作，低位是SB\_THUMBTRACK时，高位代表拖动时目前的位置，低位是SB\_THUMBPOSITION时，高位是释放后的最终位置。

对话框属性Vertical ScrollBar选择TRUE，则可以产生垂直滚动条。在对话框的WM\_VSCROLL消息中可以进行控制。

class ScrollBarCtl

{

int m\_ScrollType;//类型SB\_VERT或者SB\_HORZ

int m\_curScrol;//当前位置

int m\_ScrollMin;//最小值

int m\_ScrollMax;//最大值

int m\_Scrolldelta;//滚动变化量

CWnd \*m\_pScroll;

public:

ScrollBarCtl(int \_ScrollType):m\_ScrollType(\_ScrollType),m\_curScrol(0),m\_ScrollMin(0),m\_ScrollMax(100),m\_pScroll(NULL),m\_Scrolldelta(1){}

void Init(int \_ScrollMin,int \_ScrollMax,int \_Scrolldelta,CWnd \*\_pScroll)

{

m\_Scrolldelta = \_Scrolldelta;

m\_curScrol = 0;

m\_ScrollMin = \_ScrollMin;

m\_ScrollMax = \_ScrollMax;

m\_pScroll = \_pScroll;

if (NULL == m\_pScroll) return;

m\_pScroll->SetScrollRange(m\_ScrollType,0,m\_ScrollMax);

}

void OnScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

{

if (NULL == m\_pScroll) return;

SCROLLINFO vSI;

m\_pScroll->GetScrollInfo(m\_ScrollType,&vSI);

switch (nSBCode)

{

case SB\_LINEUP: //点击滚动条最上面

vSI.nPos = 0;

break;

case SB\_LINEDOWN: //点击滚动条最下面

vSI.nPos = m\_pScroll->GetScrollLimit(m\_ScrollType);

break;

case SB\_PAGEUP:

break;

case SB\_PAGEDOWN:

break;

case SB\_THUMBPOSITION:

break;

case SB\_THUMBTRACK: //拖动滚动条

vSI.nPos = nPos;

break;

case SB\_ENDSCROLL: //结束拖动滚动条

break;

case SB\_TOP:

vSI.nPos = 0;

break;

case SB\_BOTTOM:

vSI.nPos = INT\_MAX;

break;

}

m\_curScrol = vSI.nPos;

m\_pScroll->SetScrollInfo(m\_ScrollType,&vSI);

}

void ProcMOUSEWHEEL(int delta)

{

if (NULL == m\_pScroll) return;

if (delta > 0 )

m\_curScrol -= m\_Scrolldelta;

else

m\_curScrol += m\_Scrolldelta;

if (m\_curScrol<0) m\_curScrol = 0;

if (m\_curScrol>m\_ScrollMax) m\_curScrol=m\_ScrollMax;

m\_pScroll->SetScrollPos(m\_ScrollType,m\_curScrol);

}

int GetCurrentScrol()

{

return m\_curScrol;

}

};

ScrollCtl m\_ScrollVert(SB\_VERT);

ScrollCtl m\_ScrollHORZ(SB\_HORZ);

void C\*\*Dlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

{

m\_ScrollHORZ.OnScroll(nSBCode,nPos,pScrollBar);

CDialogEx::OnHScroll(nSBCode, nPos, pScrollBar);

}

void C\*\*Dlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar\* pScrollBar)

{

m\_ScrollVert.OnScroll(nSBCode,nPos,pScrollBar);

m\_ScrollImgCtl.Show(m\_ScrollVert.GetCurrentScrol());

CDialogEx::OnVScroll(nSBCode, nPos, pScrollBar);

}

BOOL C\*\*Dlg::PreTranslateMessage(MSG\* pMsg)

{

UINT msg = pMsg->message;

if (msg == WM\_MOUSEWHEEL)

{

int delta =GET\_WHEEL\_DELTA\_WPARAM(pMsg->wParam);

m\_ScrollVert.ProcMOUSEWHEEL(delta);

m\_ScrollImgCtl.Show(m\_ScrollVert.GetCurrentScrol());

}

return CDialogEx::PreTranslateMessage(pMsg);

}

### 列表控件（List Control）

相关控件：CListCtrl 控件类

CListView 视图类，列表为视图

CLiseView=CView+CListCtrl

列表控件属性View中可以选图标、小图标、列表、报告。选择报告可以显示更多信息。

#### 图标设置

使用于ICON，SMALL ICON 和 LIST风格

加入两个位图资源：

1 2

40 20

IDB\_NORMAL IDB\_SMALL

类内定义：

CImageList m\_normal;

CImageList m\_small;

初始化：

m\_normal.Create(IDB\_BIG,40,1,RGB(255,255,0));

m\_small.Create(IDB\_SMALL,20,1,RGB(255,255,0));

40代表一次移动的大小，1代表图片使用时移动一格

RGB代表不显示的颜色（需要去掉的颜色）

m\_listCtl.SetImageList(&m\_normal,LVSIL\_NORMAL);

m\_listCtl.SetImageList(&m\_small,LVSIL\_SMALL);

#### 修改样式

m\_listCtl.ModifyStyle(LVS\_REPORT, LVS\_SMALLICON);

//第一个参数代表需要移除的风格，第二个代表要加入的风格

#### 插入数据

ICON与SMALL ICON风格

只有行的概念，没有列的概念，每个图像代表一行(排列的时候一行会排几个)

LIST风格

只有行的概念，像图标一样，有图和对应文字，相比List Box，只是多了一个图标

m\_listCtl.InsertItem(0,"file1",1);

//加入图标（只要每次调用这个函数使用不同的数字[最好按顺序写]，就按照函数排列的位置进行顺序插入）。标题为”file1”，使用位图中第二个图标

m\_listCtl.SetItemText(1,0,"x");

//修改图标对应的文字

//没有列概念，第二个参数总是0,1指代第二个图像，与图像排列位置相关

#### Report风格

m\_listOptKey.ModifyStyle(LVS\_ICON, LVS\_REPORT);

m\_listOptKey.SetExtendedStyle(LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES);

m\_listOptKey.InsertColumn(0, \_T("功能"), LVCFMT\_LEFT, 120);

m\_listOptKey.InsertColumn(1, \_T("快捷键"), LVCFMT\_LEFT, 100);

m\_listOptKey.SetTextBkColor(RGB(10,10,10));

m\_listOptKey.SetBkColor(RGB(100,100,100));

for (int i = 0; i < itemCount; i++)

{

int icount = m\_listOptKey.GetItemCount();

m\_listOptKey.InsertItem(icount, val\_key[i][0]);

m\_listOptKey.SetItemText(icount, 1, val\_key[i][1]);

}

##### 设置风格

ModifyStyle(LVS\_ICON, LVS\_REPORT);

##### 插入字段

有行和列的概念，先设计好每列的条目，然后再加入行

m\_CtrList.SetExtendedStyle(LVS\_EX\_FULLROWSELECT |

LVS\_EX\_GRIDLINES); //设置为可以行选

添加最上层的条目说明

m\_CtrList.InsertColumn(0, \_T("序号"), LVCFMT\_CENTER, 80);

m\_CtrList.InsertColumn(1, \_T("姓名"), LVCFMT\_CENTER, 100);

m\_CtrList.InsertColumn(1, \_T("年龄"), LVCFMT\_CENTER, 100);

第一个参数代表索引号，最后一个代表字段宽度。

##### 修改字段

//设置第一行第二列的字段

int row = 0;

int col = 1;

SetItemText(1,2,\_T("xx"));

或者：

LVCOLUMN col;

m\_CtrList.GetColumn(row ,&col);

col.pszText=\_T("XX");

m\_CtrList.SetColumn(col ,&col);

##### 插入数据

int icount=m\_CtrList.GetItemCount(); //得到当前条目数，作为新的数据的插入位置

m\_CtrList.InsertItem(icount,strNumber); //插入数据

m\_CtrList.SetItemText(icount,1,strName); //和插入数据同一行的其他数据

m\_CtrList.SetItemText(icount,2,strAge);

##### 显示最后一行

m\_CtrList.EnsureVisible(m\_CtrList.GetItemCount() -1, FALSE);

##### 选择控制

m\_CListCtrl.Create(WS\_CHILD | WS\_VISIBLE|LVS\_SHOWSELALWAYS, rectDummy, pParent, 1); //LVS\_SHOWSELALWAYS, 只在创建时有效

m\_list.SetItemState(-1,0,LVIS\_SELECTED); //取消所有选择状态

m\_CListCtrl.SetItemState(curIndex,LVNI\_SELECTED|LVNI\_FOCUSED,LVNI\_SELECTED|LVNI\_FOCUSED);

m\_CListCtrl.EnsureVisible(curIndex,FALSE);

##### 设置显示选择行

SetExtendedStyle(LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES|LVS\_SHOWSELALWAYS);

##### 获取选择行

int row = m\_bzListCtl.GetSelectionMark();

##### 得到选中的多个位置

CListCtrl\* pListCtrl = (CListCtrl\*)GetDlgItem(IDC\_LIST1);

POSITION pos = pListCtrl->GetFirstSelectedItemPosition();

if (pos != NULL)

{ while(pos){

int nItem = pListCtrl->GetNextSelectedItem(pos);

//利用nitem对选中数据进行操作，这里pos是引用

CString str=pListCtrl->GetItemText(0,1);

}

}

可以添加列表控件的LBN\_DBLCLK消息处理，实现在双击某项获取数据并操作(数据为列表中的项目）

##### 右击消息取数据

NM\_LISTVIEW\* pNMListView = (NM\_LISTVIEW\*)pNMHDR;

if(pNMListView->iItem != -1)

{

DWORD dwPos = GetMessagePos();

CPoint point( LOWORD(dwPos), HIWORD(dwPos) );

m\_CtrList.ScreenToClient(&point);

LVHITTESTINFO lvinfo;

lvinfo.pt = point;

lvinfo.flags = LVHT\_ABOVE;

int nItem = m\_CtrList.SubItemHitTest(&lvinfo);

if(nItem != -1)

{ //单击的是第lvinfo.iItem行第lvinfo.iSubItem列

//取出数据

CString str=m\_CtrList.GetItemText(lvinfo.iItem,lvinfo.iSubItem);

}

}

##### 清空数据

pListCtrl->DeleteAllItems();

##### 属性修改

LONG lStyle;

lStyle = GetWindowLong(m\_CtrList.m\_hWnd, GWL\_STYLE);//获取当前窗口style

lStyle &= ~LVS\_TYPEMASK; //清除显示方式位

lStyle |= LVS\_REPORT; //设置style

SetWindowLong(m\_CtrList.m\_hWnd, GWL\_STYLE, lStyle);//设置style

DWORD dwStyle = m\_CtrList.GetExtendedStyle();

dwStyle |= LVS\_EX\_FULLROWSELECT;//选中某行使整行高亮（只适用与report风格的listctrl）

dwStyle |= LVS\_EX\_GRIDLINES;//网格线（只适用与report风格的listctrl）

m\_CtrList.SetExtendedStyle(dwStyle); //设置扩展风格

##### 设置列宽

m\_CtrList.SetColumnWidth(列编号,列宽度);

##### 设置行不可选

void \*\*::OnLvnItemchanging(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMLISTVIEW pNMLV = reinterpret\_cast<LPNMLISTVIEW>(pNMHDR);

if ((pNMLV->uChanged & LVIF\_STATE) && (pNMLV->uNewState & LVNI\_SELECTED)) \*pResult = 1;

else \*pResult = 0;

}

### Tree

#### 操作说明

风格设置为has Buttons和Lines at root

HTREEITEM item; //存储插入数据后返回的节点句柄

item=m\_tree.InsertItem(“节点”); //只有一个数据代表插入节点数据

item=m\_tree.InsertItem(“数据”,item); //代表在指代的节点下插入数据

m\_tree.SetItemData(item,1234);//加入附加数据，需要数字，可以用(DWORD)pName加入一个指向堆对象的CString

m\_tree.DeleteAllItems();//删除所有节点

m\_tree.SelectItem(item);

HTREEITEM hChildItem = m\_tree.GetChildItem(item); 树控件如果没有底层目录了，值就为空

#### 取出附加数据

为树控件增加NM\_CLICK消息处理

HTREEITEM hSelectItemOld = m\_tree.GetSelectedItem();

CPoint point;

GetCursorPos(&point);

m\_tree.ScreenToClient(&point);

UINT flag = TVHT\_ONITEM ;

HTREEITEM hItem = m\_tree.HitTest(point);

if(hItem != NULL && hSelectItemOld != hItem)

m\_tree.SelectItem(hItem);

HTREEITEM childTree= m\_tree.GetChildItem(hItem);

if (!childTree)

{

//附加数据是字符串时

CString \*str=(CString\*)m\_tree.GetItemData(hItem);

//附加数据是数字时

int i=(int)m\_tree.GetItemData(hItem);

}

#### 循环插入实例

##### 假设数据

假设有如下向量，每个数据带有两个CString的结构，且内部数据是按照根相同的数据连续放置。

#include <vector>

using namespace std;

typedef struct Data

{

CString type;

CString name;

}Data;

typedef vector<Data> VectorData;

VectorData v;

Data d1;

d1.type="蔬菜";

d1.name="白菜";

v.push\_back(d1);

Data d2;

d2.type="蔬菜";

d2.name="土豆";

v.push\_back(d2);

Data d3;

d3.type="水果";

d3.name="葡萄";

v.push\_back(d3);

Data d4;

d4.type="零食";

d4.name="方便面";

v.push\_back(d4);

##### 插入树

HTREEITEM root; //指代根节点，用来判断是否需要插入新的根

CString rootRepeate=\_T(""); //用来存储每次根节点数据，用来判断数据是否和刚才插入的根相同

for(int i=0;i<v.size();i++) //循环加入数据

{

//取出数据

Data data=v[i];

CString name=data.name;

CString type=data.type;

/////////为树插入节点和数据

if (type!=rootRepeate) //type是每次取出的代表根的数据，如果和上次节点不同，就插入新的节点，如果相同，就在上次节点后继续插入（需要取出的数据按规律排）

{

root=m\_tree.InsertItem(type); //如果插入新根节点，就将root指向新的节点

}

HTREEITEM leaf=m\_tree.InsertItem(name,root); //在根节点下插入数据

CString \*pName=new CString;

\*pName=name;

m\_tree.SetItemData(leaf,(DWORD)pName);//附加数据，如果是数字，可以直接写

rootRepeate=type;

}

### TABcontrol

#### 设置

控件需要的页面的对话框如：IDD\_DIALOG1，IDD\_DIALOG2

设置属性：Border none ，Style Child

建立对话框类

dlg1，dlg2

主对话框使用时注意包含页面对话框的头文件

绑定变量

CTabCtrl m\_Table;

类内变量

dlg1 page1;

dlg2 page2;

#### 初始化

##### 设置页面文字

m\_Table.InsertItem(0,\_T("页面1"));

m\_Table.InsertItem(1,\_T("页面2"));

##### 设置控件属性（非必须）

m\_Table.MoveWindow(30,30,200,200);

m\_Table.SetMinTabWidth(30); // 设置标签项的最小宽度

m\_Table.SetPadding(CSize(30,8)); // 设置标签项和图标周围的间隔

##### 创建页面

page1.Create(IDD\_DIALOG1,&m\_Table);

page2.Create(IDD\_DIALOG2,&m\_Table);

##### 页面显示位置

将其覆盖在Tab控件的相应位置上

只能在创建页面后使用

CRect rect;

m\_Table.GetClientRect(&rect);

rect.top+=20;

rect.bottom-=5;

rect.left+=5;

rect.right-=5;

page1.MoveWindow(&rect);

page2.MoveWindow(&rect);

##### 设置选择

page1.ShowWindow(SW\_SHOW); //初始化选择

m\_Table.SetCurSel(1);

#### 选择消息处理

为Tab控件的TCN\_SELCHANGING消息添加ON\_NOTIFY的相应消息处理

在函数中为不同页签的选择添加相应显示

int CurSel=m\_Table.GetCurSel();

switch(CurSel)

{

case 0:

page1.ShowWindow(SW\_SHOW);

page2.ShowWindow(SW\_HIDE);

break;

case 1:

page1.ShowWindow(SW\_HIDE);

page2.ShowWindow(SW\_SHOW);

break;

default: ;

}

### CPropertySheet

#### 数据交互

可以在sheet中声明一些变量，在page中对这些变量进行赋值，在sheet的DoModal之后用sheet的对象取出这些数据。

#### 设置

class CMyPropertySheet : public CPropertySheet

//属性表单，用来联系属性页

建立两个对话框(可以选择IDD\_PROPPAGE\_LARGE这种类型的对话框资源[注意资源语言的选择]，而不使用默认的)

为两个对话框建立对话框类，注意继承自CPropertyPage

class CPage1 : public CPropertyPage

class CPage2 : public CPropertyPage

#### 初始化

注意包含头文件

CMyPropertySheet sheet(\_T("xx"),this);

CPage1 p1;

CPage2 p2;

sheet.AddPage(&p1);

sheet.AddPage(&p2); //按顺序加入，代表向导中顺序

sheet.SetWizardMode();//设置为向导模式

//如果不设置向导模式，可以和TABcontrol有相同显示效果

#### 显示

sheet.SetActivePage(&p2); //设置要激活显示的页面

sheet.DoModal();

///如果是Wizard模式，DoModal的返回值为ID\_WIZFINISH 或者IDCANCEL.

#### 修改属性

##### 修改整体属性

需要为CMyPropertySheet添加OnInitDialog等函数

//隐藏掉帮助按钮

GetDlgItem (IDHELP)->ShowWindow (FALSE);

//隐藏掉取消按钮

GetDlgItem(IDCANCEL)->ShowWindow(FALSE);

##### 修改单页面属性

每个页面包含的按钮以及按钮的可用性不同，需要在CPage1这种页面的类中（注意包含头文件）实现虚函数OnSetActive。

第一个页面中：

((CMyPropertySheet\*)GetParent())->SetWizardButtons(PSWIZB\_NEXT);

最后一个页面中：

((CMyPropertySheet\*)GetParent())->SetWizardButtons(PSWIZB\_FINISH|PSWIZB\_BACK);

中间步骤页面中：

((CMyPropertySheet\*)GetParent())->SetWizardButtons(PSWIZB\_BACK|PSWIZB\_NEXT);

##### 阻止进入下一页

为需要的page页增加虚函数OnWizardNext

if(\*\*\*){ \*\*\* return -1;}

### 文件打开/另存对话框

CFileDialog(打开或另存，文件扩展名，默认文件名，窗口风格，文件类型的过滤，父窗口)

第一个参数TRUE为打开对话框，FALSE为另存对话框

文件类型过滤字符串的格式：

1. 如果是多行，每行以 | 分隔，整个字符串以 || 结束
2. 每行有两部分（显示和后缀）组成，这两部分以 | 分隔

CString fliter = \_T("图像(\*.jpg)|\*.jpg|所有文件(\*.\*)|\*.\*||");

CFileDialog dlg(TRUE, NULL, NULL, OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, fliter);

if (IDCANCEL == dlg.DoModal()) return;

dlg.GetPathName();

dlg.GetFileName();

### 颜色对话框

CColorDialog dlg; //可使用dlg(colorref)，构造初始选择颜色

if(IDCANCEL==dlg.DoModal()) return;

(COLORREF) m\_color=dlg.GetColor();

可通过变量m\_color接收选择的颜色信息

颜色对话框的成员变量m\_cc是一个CHOOSECOLOR结构体，可以通过修改该结构体而改变颜色对话框的属性。

dlg.m\_cc.Flags|=CC\_FULLOPEN;//可以将颜色对话框自定义颜色展开

dlg.m\_cc.Flags|=CC\_RGBINIT;

dlg.m\_cc.rgbResult=RGB(0,255,0);//可以设置初始选择颜色（必须设置CC\_RGBINIT属性才有效）

### 字体对话框

CFontDialog dlg;

CFont font; //用来接收选择的字体

if(IDOK!=dlg.DoModal()) return;

CString fontName = dlg.m\_lf.lfFaceName;

int fontSize = dlg.m\_cf.iPointSize/10;

### 键盘消息响应

对话框不直接相应键盘消息，要在对话框类的PreTranslateMessage中加入：

if(pMsg->message==WM\_KEYDOWN){

UINT iKey=(UINT)pMsg->wParam;

if(iKey==13){//按了回车}

}

### 模式和非模式对话框

模式对话框在弹出时会阻塞，也就是在对话框不关闭时，其他窗口无法通过点击激活

CDialog 父类CWnd

CCommonDialog 通用对话框类，包括字体对话框，颜色对话框，文件打开、保存对话框，打印对话框，打印设置对话框

#### 对话框资源使用

对话框资源中，双击对话框资源，可以创建对话框类（如类名CDlg）

可以在菜单项处理函数中：（需包含CDlg的头文件Dlg.h）

1）创建模式对话框

CDlg dlg;

dlg.DoModal();

1. 创建非模式对话框

CDlg \*dlg=new CDlg;

dlg->Create(IDD\_DIALOG1);

dlg->ShowWindow(SW\_SHOW);

dlg->UpdateWindow();

为对话框类添加WM\_CLOSE的处理函数，用以非模式对话框的销毁

CDialog::OnClose();

DestroyWindow(); 必须写在下面一行

右击对话框类，添加虚函数PostNcDestroy：

CDialog::PostNcDestroy();

delete this; 必须要写在下面这一行

或者可以使用（同一对话框只能创建一个）：

在类内定义成员变量：CDlg dlg;

在某个函数内创建非模式对话框：

dlg.DestroyWindow(); //先销毁对话框类，然后再Create

dlg.Create(IDD\_DIALOG1);

dlg.ShowWindow(SW\_SHOW);

dlg.UpdateWindow();

#### 模式对话框

1. 模式对话框的创建显示：

CDialog::DoModal();

1. 对话框的初始化函数，在对话框显示前调用

CDialog::OnInitDialog()

1. 对话框的关闭

CDialog::OnOK/OnCancel()

EXP:

CDialog dlg(IDD\_DIALOG1);

第二个参数是CWnd\*默认是NULL，可以写成this

dlg.DoModal();

#### 非模式对话框

非模式对话框与一般窗口类似

非模式对话框需要用户自己处理关闭进程(因为要释放new出的对象)

1. 需要重写CDiglog::OnOK()和OnCancel(),在函数中调用DestroyWindow()函数销毁窗口（OK为大写）
2. 重写PostNcDestroy()，在函数中delete this；这个函数在1）中的函数执行时被调用

exp:

void CMyDlg::OnOK(){

CDialog::OnCancle();

DestroyWindow();}

void CMyDlg::PostNcDestroy(){

CDialog::PostNcDestroy();

delete this;}

CMyDlg \*pdlg=new CMyDlg(IDD\_DIALOG1,m\_pMainWnd);

m\_pMainWnd与pdlg是等价的

m\_pMainWnd->ShowWindow(SW\_SHOW);

m\_pMainWnd->UpdateWindow();

### DDX

不可用于线程中

将对话框的控件与对话框类的成员变量绑定，可以通过访问成员变量方式去操作对话框控件

virtual void DoDataExchange(CDataExchange \*pDX)

包含一系列的绑定函数（使用参数pDX），绑定过程可自动生成

绑定函数：以DDX开头

DDX\_Text(......) 值类型的绑定

DDX\_Control(.....) 控件类型的绑定

UpdateData(BOOL) 在控件与成员变量进行数据交换时调用

不能在线程中用这个函数，线程中只能用SendMessage方式让主线程刷新。

TRUE 将用户在控件中输入的值传递给变量，得到数据（默认，一般可不写）

FALSE 将变量的值显示到控件上（编辑框用）

控件绑定的变量不能写在对话框类的初始化函数或者WM\_CREATE里面，只能写在InitDialog函数或者之后的执行函数里面。

对话框类内：

控件类型的绑定

1）类内定义控件类

CButton m\_btnOK; //或者写成CWnd m\_btnOK;

2）在DoDataExchange函数中绑定

DDX\_Control(pDX,IDOK,m\_btnOK);

DoDataExchange中传入 与ok按钮绑定 链接到类内成员

3）使用绑定的控件进行操作（比如在InitDialog函数中）

if(!CDialog::OnInitDialog())

return FALSE;

m\_btnOK.SetWindowText(“xxx”);

m\_btnOK.MoveWindow(0,0,100,100);通过变量修改控件

return TRUE;

值类型绑定：

1. 类内定义变量CString str; 也可以绑定int类型

2）DoDataExchange函数中：

DDX\_Text(pDX,IDC\_EDIT1,str);

与编辑框绑定

【在InitDialog函数中可用：

str="xxxx";

UpdateData(FALSE); 显示到控件

或者运行时改变编辑框的内容，会自动改变str的值，使用前注意先更新数据】

3）类内重写virtual void OnOK();

重写ok按钮的处理

UpdateData(TRUE);

可将编辑框输入的内容取出到str中。TRUE为默认值，可以缺省

AfxMessageBox(str);

CDialog::OnOK();

## MFC线程

CWinThread\* pThread=AfxBeginThread(ThreadProc,NULL);

UINT ThreadProc(LPVOID pParam){return 0;}/

::WaitForSingleObject(pThread->m\_hThread, INFINITE);可用于析构函数中等待线程结束。

//启动线程，初始为挂起状态

pThread = AfxBeginThread(ThreadProc, &m\_ThreadParam, THREAD\_PRIORITY\_ABOVE\_NORMAL, 0, CREATE\_SUSPENDED);

//线程结束时不自动撤销

pThread->m\_bAutoDelete = FALSE;

//恢复线程运行

pThread->ResumeThread();

结束线程并不能析构线程中分配的堆对象。

if(pThread!=NULL){

TerminateThread(pThread->m\_hThread,0);

pThread=NULL;

}

=============CEvent

对API函数进行了封装

CEvent event(FALSE,TRUE);

WaitForSingleObject(event.m\_hObject, INFINITE);

event.ResetEvent();

event.SetEvent();

## SDI

### 绘图消息

ON\_WM\_PAINT()

void CMyFrameWnd::OnPaint(){

CPaintDC dc(this);

dc.TextOut(34,234,"ssf");

}

### 鼠标消息

ON\_WM\_MOUSEMOVE()

void CMyFrameWnd::OnMouseMove(UINT nKey,CPoint pt){

...................

InvalidateRect(NULL,TRUE);

}

### 键盘消息

ON\_WM\_CHAR

afx\_msg void OnChar( UINT nChar, UINT nRepCnt, UINT nFlags ){

if(97==nChar){

AfxMessageBox("a");}

}

### 最近打开文档

#### 添加

theApp.AddToRecentFileList(filePathName);

#### 使用/删除

获取点击的最近文档，返回NULL删除该项目

CDocument\* C\*\*\*App::OpenDocumentFile(LPCTSTR lpszFileName)

{

return NULL;

return CWinAppEx::OpenDocumentFile(lpszFileName);

}

### 视图类滚动条

将基类改为CScrollView（修改其他消息对应代码基类）

OnInitialUpdate中设置滚动条

CSize size(0,0);

SetScrollSizes(MM\_TEXT,size);

该代码可以在其他位置调用进行设置

### 菜单

1. 相关问题

Win32 HMENU

MFC CMenu 类对象

1. 相关类

CMenu 封装了m\_hMenu和API

#### 添加菜单资源

File，new，Resource Script，进入资源选项卡，右击资

源文件夹，insert，可选择加入各种类型的资源。

添加菜单项目时，如使用f&ile则i就变成热键，即使用

alt+i 可以选中该项目。

文件中需包含头文件

#include “resource.h”（不要写在预编译头文件StdAfx.h中，否则会无法修改菜单id）

#### 设置菜单

1.在CFrameWnd：：Create参数中设置菜单

pFrame->Create(NULL,”name”,WS\_OVERLAPPEDWINDOW,CFrameWnd::rectDefault,NULL,MAKEINTRESOURCE(IDR\_MENU1));

后面有缺省参数，可以不写，CFrameWnd：：rectDefault代表四个参数。

2.在框架窗口的ON\_WM\_CREATE()消息处理中：

CMenu menu;

menu.LoadMenu(IDR\_MENU1);

SetMenu(&menu);

menu.Detach();如果menu是局部变量，用这一句可以将句柄与menu对象断开，防止发生问题

#### 处理菜单项目

ON\_COMMAND(ID,OnNew)

参数为菜单的id及自定义的处理函数

afx\_msg void OnNew();

#### 设置菜单状态

在框架类的构造函数中改变变量，取消自动检测更新菜单可用性，这样才可以自由设置菜单项目的可用性

this->m\_bAutoMenuEnable=FALSE;

ON\_WM\_INITMENUPOPUP()

菜单被激活，即将显示时（下拉出菜单的项目时)

afx\_msg void OnInitMenuPopup(CMenu \*pop,UINT

n,BOOL i);

pop->EnableMenuItem(id,

MF\_DISABLED|MF\_GRAYED);

pop->CheckMenuItem（id，MF\_CHECKED）

或者使用;

::CheckMenuItem(pop->m\_hMenu,id,MF\_CHECKED);

MF\_BYCOMMAND代表第一个参数使用ID，如果使用另一个参数MF\_BYPOSITION则第一个参数使用从0开始的相对位置

pop->SetDefaultItem(ID);使菜单项目变粗体，但是只能设置一个菜单项目

##### 设置图像菜单

CBitmap bmp1,bmp2; 在Frame类内定义

bmp1.LoadBitmap(IDB\_BITMAP1);

bmp2.LoadBitmap(IDB\_BITMAP2);

CMenu \*pop=GetMenu();

pop->GetSubMenu(0)->SetMenuItemBitmaps(1,MF\_BYPOSITION,&bmp1,&bmp2);

第一个bmp代表MF\_UNCHECKED时显示的位图，第二个代表MF\_CHECKED时的显示位图

GetSystemMetrics(SM\_CXMENUCHECK)可以获得标记菜单位图应该设置的大小

（注意菜单挂在Frame类中，则只能在Frame中的函数里用GetMenu获取菜单的指针）

SetMenu(NULL); 可以去掉菜单

#### 简易方式设置菜单项目状态

在类向导中，可以为菜单项目id添加COMMAND消息处理，还有一个就是UPDATE\_COMMAND\_UI

这个消息发生在程序发生变化的时候，也就是有程序自动调用，实时根据情况刷新菜单项目

可以为类增加一个变量x，用x的值进行判断，来决定怎么改变菜单项目：

if(x>0){pCmdUI->Enable();//就可以设置是否可用}

else{pCmdUI->Enable(FALSE);}

pCmdUI->m\_nID保存了当前菜单项目对应的ID

#### 右键菜单

##### 使用方式

ON\_WM\_CONTEXTMENU()

(使用屏幕坐标系，不需要转换,ClientToScreen可以把客户区坐标转换成屏幕坐标)

afx\_msg void OnContextMenu(CWnd \*pWnd,CPoint pt);

CMenu menu;

menu.LoadMenu(IDR\_MENU1);

CMenu \*pPopup=menu.GetSubMenu(0);

菜单资源，取第一个顶层菜单项目下的子菜单

pPopup->TrackPopupMenu(TPM\_LEFTALIGN,pt.x,pt.y,this);

this也用右键菜单处理函数的第一个参数pWnd

或者选择project里的add to project里的components and controls在里面c++文件夹直接选择popupmenu加入

##### 常用方式

CMenu menu;

menu.CreatePopupMenu();

menu.AppendMenu(MF\_STRING,IDM\_MENU1,"名称");

//需要为IDM\_MENU1定义宏以及命令消息处理

CPoint pt;

GetCursorPos(&pt);//pt为当前鼠标坐标点

menu.TrackPopupMenu(TPM\_LEFTALIGN,pt.x,pt.y,this);

menu.DestroyMenu();

#### 动态添加菜单

CMenu menu;

menu.CreatePopupMenu(); //创建一个顶层菜单

GetMenu()->InsertMenu(1,MF\_BYPOSITION|MF\_POPUP,(UINT)menu.m\_hMenu,"顶层菜单"); //在菜单第2个位置插入一个顶层菜单

GetMenu()->AppendMenu(MF\_POPUP,(UINT)menu.m\_hMenu,"顶层菜单"); //为菜单栏添加一个顶层菜单

menu.AppendMenu(MF\_STRING,100,"菜单项2"); //添加一个菜单项目，100为项目id

menu.InsertMenu(0,MF\_STRING|MF\_BYPOSITION,101,"菜单项1"); //在指定位置插入一个菜单项目

menu.Detach();//menu为局部变量时使用

GetParent()->DrawMenuBar();在view类中动态添加菜单后，需要这句重绘菜单栏

DeleteMenu(0,MF\_BYPOSITION）

可以删除菜单顶层菜单或者菜单项目

### 工具栏

1. 相关类

CToolBarCtrl

父类CWnd，封装了对工具栏控件的操作

CToolBar

父类CControlBar，封装了工具栏和框架窗口之间关系（停靠位置），另外还包括工具栏的创建

1. 使用

1）添加资源

2）创建工具栏 CToolBar：：Create/CreateEx

3) 加载工具栏资源 CToolBar：：LoadToolBar

工具栏资源创建时，将某项的id设置为菜单某项目的id，可与其进行绑定，也可以不绑定菜单。和菜单的项目处理方式相同，删除某项时，用鼠标将小图直接拖出即可。

<afxext.h>

CToolBar toolbar；在CMyFrame类内定义一个对象

ON\_WM\_CREATE()

afx\_msg int OnCreate(LPCREATESTRUCT pcs);

toolbar.CreateEx(this,TBSTYLE\_FLAT,

WS\_CHILD|WS\_VISIBLE|CBRS\_ALIGN\_TOP|CBRS\_GRIPPER|CBRS\_TOOLTIPS|CBRS\_FLYBY|CBRS\_SIZE\_DYNAMIC);

this之后的有缺省值，可以不写后面的

CBRS\_GRIPPER 可以使工具栏有把手，可移动

CBRS\_TOOLTIPS 使鼠标划过时显示提示信息

CBRS\_FLYBY 使鼠标划过时在状态栏显示相应提示信息

CBRS\_SIZE\_DYNAMIC 使工具栏移动后大小可变

CBRS\_ALIGN\_TOP的作用：

1）后面可以不使用DockControlBar函数，但使用把手

移动的风格失效

2）使用DockControlBar时第二个参数可使用默认值0；

toolbar.LoadToolBar(IDR\_TOOLBAR1)；

加载资源后即可显示

工具栏的停靠（船坞化）

1）工具栏准备停靠的位置

CToolBar：：EnableDocking

2)框架窗口允许停靠的位置

CFrameWnd：：EnableDocking

3）框架窗口确定停靠的位置

CFrameWnd：：DockControlBar

1），2）必须有交集

toolbar.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&toolbar)；设置开始时停靠的位置（第二个参数，可省略）写完此三句后才可拖拽工具栏。

AFX\_IDW\_DOCKBAR\_TOP

AFX\_IDW\_DOCKBAR\_BOTTOM

AFX\_IDW\_DOCKBAR\_LEFT

AFX\_IDW\_DOCKBAR\_RIGHT

toolbar.SetWindowText(“xx”);

在工具栏资源中，选中资源，按回车键可进入属性，在底下的提示内可写提示信息(需要加入风格CBRS\_TOOLTIPS)

提示信息格式：

aaa\nbbb

aaa会在状态栏显示

bbb会在鼠标滑过工具栏项目时提示

工具栏的显示和隐藏

CFrameWnd：：ShowControlBar（&toolbar，TRUE，FALSE）

第二个参数为是否显示

第三个参数为是否延迟，一般用FALSE

toolbar.IsWindowVisible()

判断是否处于显示状态

Mousemove中如果有invalidateRect操作，会影响工具栏的显示

### 状态栏

小格为指示器

CStatusBar 封装了关于状态栏的操作，以及包括创建状

态栏

<afxext.h>

#### 创建

CStatusBar statusbar; CMyFrameWnd类内定义

Create消息处理中创建：

statusbar.CreateEx(this);

#### 设置指示器

CStatusBar：：SetIndicators

1)添加字符串资源（如IDS\_POS,IDS\_TIME）

多文档中，指示器不能直接用系统定义好的字符串资源，将其删除，自己定义字符串资源

2)全局数组

UINT g\_hIndicators[]={0,IDS\_POS,\*\*\*};

数组内第一个元素为0，显示的是工具栏提示信

息aaa/nbbb中的aaa（会自动加入字符串资源）

（需为工具栏加入风格CBRS\_FLYBY），第二个和第三个

参数显示的是字符串资源中的内容。

3）statusbar.SetIndicators(g\_hIndicators,

sizeof(g\_hIndicators)/sizeof(UINT));

##### 设置指示器宽度和风格

statusbar.SetPaneInfo(1,IDS\_POS,SBPS\_NORMAL,200);

1代表数组内的第二个元素,第二个参数貌似没啥

用。第三个参数代表设置的风格，第四个参数为指

示器宽度。

##### 设置/修改指示器的文本内容：

statusbar.SetPaneText(1,”文字”);

1,代表第一个位置。0位置用于最左边动态显示工具栏提示。右边开始可以从1开始数指示器位置。

可用于动态设置指示器内容，如mousemove消息中：

CString str;

str.Format(“%d,%d”,pt.x,pt.y);

statusbar.SetPaneText(1,str);

（鼠标移动消息的处理和状态栏类的声明同在类

CMyFrameWnd中）

#### 设置状态栏高度

StatusBar.GetStatusBarCtrl().SetMinHeight( 30 ) ;

#### 指示器位置

CRect rect;

m\_wndStatusBar.GetItemRect(1,rect);

可以得到状态栏中第2格的指示器的位置，该位置是以状态栏为基准的位置，rect.top一般都是2。

该函数不能在OnCreate函数中调用，要等到状态栏创建完成后才能调用。

#### 对话框中添加指示器

需要添加代码：

CRect rect;

GetClientRect(&rect); 参数也可以写成rect

statusbar.MoveWindow(0,rect.bottom-20,rect.right,20);

### 加速键

资源：id 加速键

id可对应菜单项目的id

HACCEL ：：LoadAccelerators

BOOL ：：TranslateAccelerator

CWinApp类的虚函数

virtual BOOL PreTranslateMessage（MSG \*pMsg）;

HACCEL hAccel=::LoadAccelerators(

AfxGetInstanceHandle(), MAKEINTRESOURCE(IDR\_ACCELERATOR1));

::TranslateAccelerator(this->m\_pMainWnd->m\_hWnd,

hAccel,pMsg);

return CWinApp::PreTranslateMessage(pMsg);

第一个参数得到的是CMyFrameWnd类窗口的窗口句柄，因为菜单消息在CMyFrameWnd中处理

### 光标

光标资源

ON\_WM\_SETCURSOR()

鼠标移动过程中并且在没有捕获鼠标消息情况下连续出现的消息

在设置光标资源的消息处理函数中处理(或者在mousemove中):

afx\_msg void OnSetCursor(CWnd \*pWnd,UINT nHitTest,UINT message)

if (NULL != m\_hcursor)

{

::SetCursor(m\_hcursor);

return TRUE;

}

return CView::OnSetCursor(pWnd, nHitTest, message);

m\_hcursor= LoadCursorFromFile(\_T("skin\\curFile.cur"));

m\_hcursor = LoadCursor( NULL , IDC\_CROSS );

### ribbon

#### 隐藏右键菜单和系统功能

class CMFCRibbonBarEx:public CMFCRibbonBar

{

public:

void RemoveQAT(){ m\_QAToolbar.RemoveAll();}

virtual BOOL OnShowRibbonContextMenu(CWnd\* pWnd, int x, int y, CMFCRibbonBaseElement\* pHit){return TRUE;}

};

#### 移除面板

m\_wndRibbonBar.GetActiveCategory()->RemovePanel(7);

### 自定义窗口

::CreateWindowEx(0,"button","ok",WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON,400,100,100,40,this->m\_hWnd,(HMENU)1001,AfxGetInstanceHandle(),NULL);

创建一个按钮(在有视图时，创建的按钮应放在视图窗口上,且应该写在WM\_CREAT消息中，写在Create函数中无效)

或者使用：

CButton \*btn=new CButton; 也可以为类内增加一个btn对象成员

btn->Create("ok",WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON,CRect(0,0,100,40),this,1001);

ON\_BN\_CLICKED(1001,OnClicked)当按钮被点击时，可对应自定义处理函数OnClicked，其中1001为按钮id，虽然参数为菜单形式，但不是菜单。

::CreateWindowEx(0,"EDIT","",WS\_CHILD|WS\_VISIBLE|WS\_BORDER,100,100,200,200,this->m\_hWnd,(HMENU)1002,AfxGetInstanceHandle(),NULL);

创建一个编辑框

ON\_EN\_CHANGE(1002,OnEnChange)当编辑框内容发生变化时，可对应自定义处理函数OnEnChange，其中1002为编辑框id。

Edit和button是系统窗口类名，可以不区分大小写

### 设置窗口样式

在MainFrame类的虚函数PreCreateWindow中可以修改结构体CREATESTRUCT& cs，达到改变窗口样式的目的。

该结构体就是CreateWindowEx函数中12个参数组成的结构。

cs.cx=100;

cs.cy=100;

cs.style | =WS\_\*\*\*; //增加窗口风格

cs.style &=~WS\_\*\*\*;//去掉窗口风格

对话框不响应该函数，对话框经过OnInitDialog()

在View类WM\_CREATE的OnCreate函数中，可以通过代码，在窗口创建后改变窗口样式。

SetClassLong(m\_hWnd,GCL\_HBRBACKGROUND,(LONG)GetStockObject(BLACK\_BRUSH));//设置黑色背景

SetClassLong(m\_hWnd,GCL\_HBRBACKGROUND,(LONG)CreateSolidBrush(RGB(255,0,0))); //设置红色背景

第二个参数可以是GCL\_HICON等，通过定时器动态调用该函数，可以做到窗口样式的动画效果。

### 标题改变

显示格式：文档名 —窗口名

#### 文档名

每次打开文件时，文档名变化，是自动进行的

在文档类中OnNewDocument函数中加入：

SetTitle("文档");

这样可以在新建文件时显示新的文字，而不是未命名

#### 窗口名

资源中修改String Table中的IDR\_MAINFRAME的值。第一个就是窗口名（多文档中此资源只有1项），修改即可。

### 修改窗口风格

可以在框架类的PreCreateWindow函数里修改风格

cs.style&=~FWS\_ADDTOTITLE; //去除标题条中的文档名

### 获取其它窗口

#### 对话框句柄获取

HWND hwnd=::FindWindow(NULL,"对话框标题");

可以获得对话框的句柄

#### 获取父级指针

GetParent(); 在视图类获得框架类的指针

#### 获取应用程序实例

AfxGetInstanceHandle（）

可以拿到winMain的第一个参数g\_hInstance

#### 获取视图类指针

C\*\*View \*p=(C\*\*View \*)GetActiveView(); 在框架类中获得视图类的指针

### 框架类处理命令消息

在框架类中，可以重写虚函数OnCommand(WPARAM wParam, LPARAM lParam),可以使用，用来截获命令消息，：

int id=LOWORD(wParam);

if(ID\_NEW==id){

..........

return TRUE;

}

### 显示/隐藏工具栏/状态栏

void CMainFrame::OnShowToolBar()

{

if (m\_wndToolBar.GetStyle() & WS\_VISIBLE)

{

//隐藏工具栏

ShowControlBar(&m\_wndToolBar, FALSE, FALSE);

}

else

{

//显示工具栏

ShowControlBar(&m\_wndToolBar, TRUE, FALSE);

}

}

void CMainFrame::OnUpdateShowToolBar(CCmdUI\* pCmdUI)

{

if (m\_wndToolBar.GetStyle() & WS\_VISIBLE)

{

pCmdUI->SetCheck(TRUE);

}

else

{

pCmdUI->SetCheck(FALSE);

}

｝

### 屏蔽拖动标题栏功能

WM\_NCLBUTTONDOWN

void CMainFrame::OnNcLButtonDown(UINT nHitTest, CPoint point)

{

if (HTCAPTION == nHitTest) return;

CFrameWndEx::OnNcLButtonDown(nHitTest, point);

}

### 单文档只有第一次打开时最大化问题

int CRoadProcApp::ExitInstance()

{

AfxOleTerm(FALSE);

CleanState();

return CWinAppEx::ExitInstance();

}

最大化设置

ParseCommandLine(cmdInfo);

m\_nCmdShow=SW\_SHOWMAXIMIZED;

### 去除标题

App::InitInstance

::SetWindowText(AfxGetMainWnd()->GetSafeHwnd(),\_T(""));

CMainFrame::PreCreateWindow

cs.style &= ~ FWS\_ADDTOTITLE;

## ActiveX

### 控件的导入

如果系统没有注册控件，而程序中存在控件，会出现点击程序后无任何反应。

控件为dll或者ocx，安装了某些软件后会自动生成一些控件。有些控件依赖于软件的存在，有些控件只需要一个dll或者ocx注册一下即可。

在VC6.0中可以通过“project”-> “Add To Project”-> “Components and Controls”导入ActiveX控件。

VS系列中可以通过“project”-> “Add Class”->“MFC Class From ActiveX ”，可以在下拉框中找到控件名，然后增加到“Generated classes”中，点完成即可。

### 显示处理

OnDraw函数中可以将原来绘图的代码注释掉，写入自定义的显示代码。

调用Invalidate();或者InvalidateControl();可以让窗口重新刷新显示。

### 消息处理

可以为C\*\*Ctrl类加入消息响应，如WM\_CREATE中初始化，WM\_TIMER处理定时器消息（注意在写SetTimer(1,1000,NULL);这种函数时候，第一个参数其实已经包含，可能会出现代码提示帮助错误）

### 与外界交互

#### 属性页

在C\*\*Ctrl的cpp中有关于属性页的绑定函数：

BEGIN\_PROPPAGEIDS(C\*\*\*Ctrl, 1)

PROPPAGEID(C\*\*\*PropPage::guid)

END\_PROPPAGEIDS(C\*\*\*Ctrl)

如果要增加属性页中的页面，可以在这里加入绑定，加入后数量从原来的1，变为2。

如增加颜色属性页：

PROPPAGEID(CLSID\_CColorPropPage)

ClassWizard内Automation页面中，选择Add Property，这个页面中的External name可以输入一个名称定义一个属性，也可以选择增加一些常规的属性。

##### 常规属性

Implementation选择Stock表示选择的属性是控件的标准常规属性。添加后可以在控件的属性页面中看到这个属性并且可以设置。程序中可以通过函数获取这个设置的属性，然后对控件进行响应属性改变。

增加常规属性BackColor，后可以在属性设置页面中设置这个值，在程序中通过：

COLORREF ref=TranslateColor(GetBackColor());

CBrush brush(ref);

pdc->FillRect(rcBounds,&brush);

可以获取颜色，然后使用设置的颜色设置控件背景色。

pdc->SetBkMode(TRANSPARENT)可以使上面的文字背景色透明。;

##### 自定义变量/属性

Implementation选择Member variable，表示增加一个变量，同时会生成相关函数（如果选择Get/Set Methods只能生成函数，还需要自己加入成员变量来保存该属性变量）

External name是提供给外部程序调用时使用的变量名。如x;

Type可以自定义选择变量类型，如short。

Variable name是控件内部程序对于这个变量使用的名字。如m\_x;

Notification function中对应的是函数名，该函数在外部程序改变这个变量时自动调用。在该变量设置后，外部使用这个控件会看到关于该属性的设置和获取的函数。

##### 属性值保存

在对话框中通过属性页设置了属性后，可以保存下来，下次打开工程时，查看控件的属性时可以是修改后的值。

在CXXXCtrl的DoPropExchange函数中：

PX\_Short(pPX,"x",m\_x,100); //初始值为100

##### 属性值通知

自定义了属性值，在改变后，需要让容器也知道属性值改变了，以便让容器的属性面板中的相应值进行改变。

在属性值改变的响应函数中加入代码：

BoundPropertyChanged(0x01);

数字代表在\_D\*\*中自动生成的变量id号码。如 [id(1)] short x;

##### 运行状态

使用if(AmbientUserMode()){...}可以判断控件是在运行时还是在容器中处于设计状态。可以使有些代码的运行只在程序运行后执行。

##### 属性页对话框

可以在属性页的对话框资源中加入编辑框等控件，然后绑定变量（和C\*\*\*PropPage绑定的），这时有一个Option Property name，可以关联到常规属性或者自定义属性。这样在属性页中可以修改自定义或者常规的属性。

#### 与外界连通的函数

ClassWizard内Automation页面中选择Add Message。可以设置函数的名称，返回值以及参数列表（参数的类型有限制）。确认后会在\_D\*\*\*中生成相应外部函数接口代码。在C\*\*Ctrl类中，则是函数的声明和实现。

#### 事件

可以为控件增加某些消息事件，会在\_D\*\*\*Events中生成相应的代码。增加后，控件就可以响应该消息，并将该消息发送给容器。比如鼠标点击控件，则对话框可以收到控件被点击的消息。

可以增加标准事件，也可以增加自定义事件，自定义事件后，会生成一个函数（而且会为控件生成一个消息名称，在外部可以响应这个消息）。在控件中调用该函数时候，就可以向外部发出消息。