

计算机 C/C++ 语言系列丛书

郑雪明 编著

VISUAL C++

基础类库参考大全



学苑出版社

0679971

计算机 C/C++ 语言系列丛书

Visual C++ 基础类库参考大全

郑雪明 编著
亦 鸥 审校

学苑出版社

1994

(京)新登字 151 号

内 容 简 介

本书是 Microsoft Visual C++ 基础类库的参考手册,书中全面、系统地介绍了 Visual C++ 基础类库中的类、宏、全程函数和全程变量。

本书对从事软件设计、开发和应用的专业技术人员具有重要的参考价值。

需要本书的用户,可与北京 8721 信箱联系,邮码 100080,电话 2562329。

计算机 C/C++ 语言系列丛书

Visual C++ 基础类库参考大全

编 著: 郑雪明
审 校: 亦 鸥
责任编辑: 甄国宪
出版发行: 学苑出版社 邮政编码: 100036
社 址: 北京市海淀区万寿路西街 11 号
印 刷: 北京东升印刷厂印刷
开 本: 787×1092 1/16
印 张: 30.125 字数: 705 千字
印 数: 1~5000 册
版 次: 1994 年 5 月北京第 1 版第 1 次
I S B N: 7-5077-0875-6/TP·24
本册定价: 45.00 元

学苑版图书印、装错误可随时退换

第一章 Microsoft 基础类参考

Microsoft 基本类分为两个主要部分:(1)基本类;(2)宏和全程符。若某个函数或变量不是某个类的一个成员,那么它是一个全程函数或变量。

本章先按字母顺序详细介绍基本类,第二章再介绍宏和全程符。

1. CArchive 类

CArchive 类用于以持久的二进制形式(通常是磁盘存储)来存储一个复杂的对象网络。这种存储一直持续到对象被删除。用户可以从存储区中调入对象,然后在内存中重新构造它们。这个使数据持久化的过程称过“串行化”。

用户可以把档案对象理解成一种类型的二进制流。如同一个输入/输出流,一个档案和一个文件相联系。它允许从存储区中有缓冲地读取数据,或把数据写入存储区中。一个输入/输出流处理一个 ASCII 码字符串序列,而一个档案则是以一种有效无冗余的方式处理二进制对象数据。

用户在建立一个 CArchive 对象前,必须建立一个 CFile 对象。另外,还必须保证档案的调入/存储状态与文件的打开模式兼容。一个文件只能对应一个激活了的档案。

用户构造一个 CArchive 对象时,把它连接到一个 CFile(或一个派生类)的对象,这代表着一个打开的文件。用户还必须指定档案是用于调用还是用于存储。一个 CArchive 类对象不仅可以处理原始类型,还可以处理用于串行化的 CObject 派生类的对象。一个可串行化的类必须有一个 Serialize 的成员函数,必须使用 DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏,如在 CObject 类下的介绍。

重载抽取(<>)和插入(<< >>)运算符是方便的档案编程接口,它们支持原始类型和 CObject 派生类。

```
#include <afx.h>
```

参见: CFile, CObject

构造/析构——公共成员

CArchive	建立一个 CArchive 对象。
~CArchive	破坏一个 CArchive 对象,清除未写入的数据。
Close	清除未写入的数据并与 CFile 断开。

基本输入/输出——公共成员

Flush	清除档案缓冲区中未写入的数据。
operator >>	从档案中载入对象及原型。
operator <<	存储对象及原型到档案中。

Read	读原始字节数据。
Write	写原始字节数据。
状态——公共成员	
GetFile	取得一个档案的 CFile 对象指针。
IsLoading	判断档案是否载入。
IsStoring	判断档案是否存储。
对象的输入/输出——公共成员	
ReadObject	为载入调用一个对象的 Serialize 函数。
WriteObject	为存储调用一个对象的 Serialize 函数。

成员函数

(1) CArchive::CArchive

```
CArchive(CFile * pFile,UINT nMode,int nBufSize=512,void FAR * lpBuf=NULL)
throw(CMemoryException,CArchiveException,CFileException);
```

说明:构造一个 CArchive 对象并指定它是用于载入或存储对象。用户建立档案后就不能再改变这项指定。在用户关闭档案前,也不能够运用 CFile 来改变文件的状态。任何这样的操作将破坏档案的完整性。用户可以用以下方式在串行化过程中取得文件指针的位置:

1)从 GetFile 成员函数中取得档案的文件对象。

2)使用 CFile::GetPosition 函数

在取得文件指针的位置前用户还需要调用 CArchive::Flush 函数。

参见:CArchive::Close,CArchive::Flush,CFile::Close

(2)CArchive::~~CArchive

```
~CArchive();
```

说明:析构函数 CArchive 关闭档案。但在调用析构函数前,用户必须调用成员函数 Close。用户使用 CFile 对象归档后,必须关闭和破坏它。

参见:CArchive::Flush,CFile::Close

(3)CArchive::Close

```
void Close()
throw(CArchiveException,CFile Exception);
```

说明:清除缓冲区中剩余的所有数据,关闭档案,从文件中断开档案。这时不允许任何对档案进行的操作。用户关闭档案后,可以再为在文件建立另一档案,也可以关闭文件。成员函数 Close 将保证所有的数据从档案转移到文件之中,同时它使档案不能使用。为完成从文件到存储媒质的转移,用户必须先使用 CFile::Close 并废弃 Cfile 对象。

参见:CArchive::Flush

(4)CArchive::Flush

```
void Flush()  
throw (CFileException);
```

说明:把档案缓冲区中剩余的所有数据写入文件。成员函数 Flush 将保证所有的数据从档案转移到文件中,为完成从文件到存储媒质的转移,用户必须调用 CFile::Close。

参见:CArchive::Close,CFile::Flush,CFile::Close

(5)CArchive::GetFile

```
CFile * GetFile() const;
```

说明:为档案取得 CFile 的对象指针。在使用 GetFile 前用户必须清除档案。

返回:返回一个使用中的 CFile 对象的常量指针。

(6)CArchive::IsLoading

```
BOOL IsLoading() const;
```

说明:判断档案是否在装载数据。这个成员函数由档案类的 Serialize 函数来调用。

返回:若档案当前正在载入数据,返回 TURE;否则,返回 FALSE。

参见:CArchive::IsStoring

(7)CArchive::IsStoring

```
BOOL IsStoring() const;
```

说明:判断档案是否正在存储数据。这个成员函数由档案类的 Serialize 函数所调用。若一个档案的 IsStoring 状态位为 TRUE,它的 IsLoading 状态为 FALSE。另一种情况与此相反。

返回:若档案当前正在存储数据,返回 TURE;否则,返回 FALSE。

参见:CArchive::IsLoading

(8)CArchive::Read

```
UINT Read(void FAR * lpBuf,UINT nMax)  
throw(CFileException);
```

说明:从档案中读入指定字节数的数据。档案对字节并不加以解释。用户可以在 Serialize 函数中使用 Read 成员函数,以读入用户指定对象中包含的普通结构。

返回:一个无符号整数包含实际读入的字节数。若返回值小于需要读的数目,说明文件已结束。

(9)CArchive::ReadObject

```
CObject * ReadObject(const CRuntime Class * pClass)  
throw( CFileException,CArchiveException,CMemoryException);
```

说明:从档案中读入对象数据,并构造一个合适类型的对象。若对象中包含着一个指向其他对象的指针,这些对象也将自动予以建立。这个对象通常被 CArchive 析取运算符(>>)

调用,为 CObject 指针重载而用。反过来,ReadObject 又调用档案类的 Serialize 函数。若用户在 RUNTIME_CLASS 宏中提供了非零的 pClass 参数,函数验证档案对象的运行时间类。这里假定用户已在类的执行部分中使用了 IMPLEMENT_SERIAL 宏。

返回:返回一个 CObject 指针。通过使用 CObject::IsKindOf,它必须确保指向正确的派生类。

参见:CArchive::WriteObject,CObject::IsKindOf

(10)CArchive::Write

```
void Write(const void FAR * lpBuf,UINT nMax)
throw(CFileException);
```

说明:向档案中写个指定字节数的数据。档案并不格式化字节。用户可以在 Serialize 函数中使用 Write 成员函数,以写入用户对象中包含的普通结构。

参见:CArchive::Read

(11)CArchive::WriteObject

```
void WriteObject(const CObject * pObj)
throw(CFileException,CArchiveException);
```

说明:存储指定的 CObject 到档案中。若对象包含了其他对象的指针,它们将依次被串行化。这个函数通常被 CArchive 插入运算符(<<)所调用,以完成 CObject 重载。反过来,WriteObject 调用了档案类的 Serialize 函数。用户必须使用 IMPLEMENT_SERIAL 宏以能够存档,WriteObject 向档案中写入 ASCII 码类名,这个类名在后面的载入过程中才变成有效。一个特殊的编码规则可以防止类的多个对象不必要的重复复制类名,同时它还防止了对作为多个指针目标的对象的冗余存储。这个对象编码方式(包括 ASCII 码类名的存在方式)在将来的库版本中有可能改变。

注:用户在开始存档前,必须结束对象建立、删除和修改的操作。当对象的修改与存档混合进行时将造成档案的混乱。

参见:CArchive::ReadObject

运算符

(1)CArchive::operator <<

```
friend CArchive &operator << (CArchive& ar,const CObject * pObj)
throw (CArchiveException,CFileException);
CArchive & operator << (BYTE by)
throw (CArchiveException,CFileException);
CArchive & operator << (WORD w)
throw (CArchiveException,CFileException);
CArchive & operator << (LONG l)
throw (CArchiveException,CFileException);
```

```

CArchive & operator << (DWORD dw)
throw ( CArchiveException, CFileException);
CArchive & operator << (float f)
throw ( CArchiveException, CFileException);
CArchive & operator << (double d)
throw ( CArchiveException, CFileException);

```

说明:存储指定对象或原型到档案中。若用户在类实现中使用了 IMPLEMENT_SERIAL 宏,则为 CObject 重载的插入运算符调用保护方式下的 WriteObject。这个函数反过来又调用 Serialize 函数。

返回:返回一个 CArchive 引用,使多个插入运算符可以处于一行。

参见:CArchive::WriteObject, CObject::Serialize

(2) CArchive::operator >>

```

friend CArchive & operator << (CArchive & ar, CObject * pObj)
throw (CArchiveException, CFileException, CMemoryException);
friend CArchive & operator << (CArchive & ar, const CObject * pObj)
throw (CArchiveException, CFileException, CMemoryException);
CArchive & operator >> (BYTE & by)
throw (CArchiveException, CFileException);
CArchive & operator >> (WORD & w)
throw (CArchiveException, CFileException);
CArchive & operator >> (LONG & l)
throw (CArchiveException, CFileException);
CArchive & operator >> (DWORD & dw)
throw (CArchiveException, CFileException);
CArchive & operator >> (float & f)
throw (CArchiveException, CFileException);
CArchive & operator >> (double & d)
throw (CArchiveException, CFileException);

```

说明:从档案中装载指定对象或原型。若用户在类实现中使用了 IMPLEMENT_SERIAL 宏,则为 CObject 重载的析取运算符调用保护方式下的 ReadObject 函数(和一个非零的运行一时间类指针)。这个函数反过来又调用了 Serialize 函数。

返回:返回一个 CArchive 引用,使多个插入运算符可处于一行。

参见:CArchive::ReadObject, CObject::Serialize

2. CArchiveException 类:public CException

一个 CArchiveException 对象代表着一个串行化异常情况。CArchiveException 类包括了代表异常原因的一个公共数据成员。CArchiveException 对象在 CArchive 成员函数中建立和

废弃。用户可以在 CATCH 表达式的范围内访问这些对象。原因代码和操作系统是无关系的。

#include <afx.h>

参见:CArchive,AfxThrowArchiveException

数据成员——公共成员

m_cause 表示了异常原因。

构造/析构——公共成员

CArchiveException 构造一个 CArchiveException 对象。

成员函数

(1)CArchiveException::CArchiveException

CArchiveException(int cause = CArchiveException::none)

说明:构造一个 CArchiveException 对象,在对象中存储 cause 的数值。用户可以在堆阵上建立一个 CArchiveException 对象并自己废弃它或由全程函数 AfxThrow ArchiveException 来管理。用户不必直接使用这个构造器,可以通过调用上述函数来使用。

数据成员

(1)CArchiveException::m_cause

说明:指定异常的原因。这个数据成员是一个整型的公共变量。它的值由一个 CArchiveException 枚举类型所定义。枚举值和其意义如下:

- CArchiveException::none 没有产生错误。
- CArchiveException::generic 未确定的错误。
- CArchiveException::readOnly 试图写入为载入而打开的档案。
- CArchiveException::endOfFile 读取对象已到文件结束处。
- CArchiveException::writeOnly 试图读取为存储而打开的档案。
- CArchiveException::badIndex 非法文件格式。
- CArchiveException::badClass 试图读取对象到一个错误类型的对象中。
- CArchiveException::badSchema 试图读取一个不同版本类的对象。

注:这些 CArchiveException::原因枚举值与 CArchiveException 原因枚举值是显然不同的。

3. CBEEdit 类:public CHEdit

CBEEdit 类封装框状方式下的手写体编辑,或称“bedit”,这是 Microsoft Windows 下的 Pen Computing 功能。CBEEdit 控制器使应用程序的用户以标准的手写编辑姿态进入和修改文本。这和使用 CHEdit 派生类而建立的手写体编辑(或称 CHEdit)控制器还是有所不同的。不同之

处在于前者显示了一个梳,用于提示用户输入字符的位置。“梳”提高了辨认精度,它提供给辨认器有关输入字符的位置信息。

框状编辑控制器下的文本被看作是文本的单一流,为方便辨认,它们位于行单元之中,并在行末换行。

用户可以用 `SetBoxLayout` 成员函数设置 `bedit` 控制器的结构,不设置时默认为缺省值。

从 `CHEdit` 类中可得到以下信息:

- 使用 App studio 建立框状编辑控制器。
- 为 `CBEEdit` 控制器设置字母代码(ALC)方式。
- 为 `CBEEdit` 控制器设置控制方式。
- 标志消息。

如果用户有意处理 `CBEEdit` 控制器发送往它的父类(通常是 `CDialog` 的派生类)的 Windows 标志消息,可以为每个消息都向其父类加上消息图入口和消息处理器函数。

```
#include <afxpen.h>
```

构造/析构——公共成员

<code>CBEEdit</code>	构造一个 <code>CBEEdit</code> 对象。
<code>Creat</code>	建立并显示一个 <code>CBEEdit</code> 控制器。
运算符	
<code>CharOffset</code>	转换 <code>bedit</code> 控制器中一个字符的逻辑字符位置到一个字节偏移量。
<code>CharPosition</code>	转移文本缓冲区中的字节偏移量到 <code>bedit</code> 控制器中的逻辑字符位置。
<code>DefaultFont</code>	把 <code>bedit</code> 控制器中的字模变成缺省字模。
<code>GetBoxLayout</code>	取得框状布局。
<code>SetBoxLayout</code>	设置框状布局。

成员函数

(1) `CBEEdit::CBEEdit`

```
CBEEdit();
```

说明:构造一个 `CBEEdit` 对象。

参见:`CBEEdit::Create`

(2) `CBEEdit::CharOffset`

```
DWORD CharOffset(UINT nCharPosition);
```

说明:在 `bedit` 控制器中字符和单元并不始终保持一一对应的关系。不取得文本缓冲区中给定单元位置的偏移量,或“逻辑”字符位置的偏移量,可使用 `CharOffset`。

返回:若 `nCharPosition` 指定的逻辑位置小于控制器中的逻辑字符总数,返回值的低位字

是字节偏移量,高位字是 0。若 nCharPosition 大于或等于控制器中的逻辑字符总数,低位字包含文本长度的字节数,高位字包含 0xFFF。用户可以使用 LOWORD 和 HIWORD 宏来检查返回值的两个部分。

参见: CEdit::CharPosition, LOWORD, HIWORD, WM_HEDITCTL

(3) CEdit::CharPosition

DWORD CharPosition(UNIT nCharOffset);

说明:在 edit 控制器中字符和单元之间并不始终保持一一对应的关系。为寻找文本缓冲区中给定字节偏移量所对应的单元或“逻辑”字符位置,可使用 CharPosition。

返回:若 nCharOffset 指定的位置小于文本的长度字节数,返回值的低位字包含着逻辑字符位置,高位字是 0。若 nCharOffset 指定的位置大于或等于文本的长度字节数,返回值的低位字包含了返回的控制器中的逻辑字符总数,高位字包含了 0xFFFF。

用户可以使用 LOWORD 和 HIWORD 宏来检查返回值的两个部分。

参见: CEdit::CharOffset, LOWORD, HIWORD, WM_HEDITCTL

(4) CEdit::Create

BOOL Create(DWORD dwStyle, const RECT & rect,
CWnd * pParentWnd, UINT nID);

说明:用户构造一个 CEdit 对象需要两个步骤。首先,构造一个 CEdit 对象,然后调用 Create。

Create 将建立一个 edit 控制器并连接到 CEdit 对象上。为扩充处理中的消息,由 CEdit 派生出一个类,向新类附加消息图,并重载适当的消息处理成员函数。例如,重载 OnCreate 以完成新类所需的初始化。

返回:若初始化成功,返回非零值;否则,返回零值。

参见: CEdit::Create, CEdit::CEdit, CEdit::SetInflate, WM_HEDITCTL

(5) CEdit::DefaultFont

void DeFaultFont(BOOL bRepaint);

注释 用户若已调用了 SetFont,可能想使 edit 控制器仍以原先建立的字模来显示。Default font 能致使 edit 控制器选择这一缺省的字模,可选择性地重新调色。

参见: CWnd::SetFont, WM_HEDITCTL

(6) CEdit::GetBoxLayout

void GetBoxLayout(LPBOXLAYOUT lpBoxLayout);

说明:使用 GetBoxLayout 来检索一个 BOXLAYOUT 的结构,它描述了控制器中 edit 框状的安排方式。用户可以使用 GetBoxLayout 和 SetBoxLayout 一起来修改框状布局的某一方面。

注释 使用 BOXLAYOUT 结构与 GetBoxLayout 和 SetBoxLayout 函数一起来定制用户的 edit 控制器。

参见: CEdit::SetBoxLayout, WM_HEDITCTL

(7) CEdit::SetBoxLayout

BOOL SetBoxLayout(LPBOXLAYOUT lpBoxLayout);

说明: 使用 SetBoxLayout 可以改变 edit 控制器的框状布局的缺省值。用户可以使用 GetBoxLayout 填充 BOXLAYOUT 结构, 在需要时可以只改变部分成员。

返回: 若改变成功, 返回非零; 否则, 返回零值。

参见: CEdit::GetBoxLayout, WM_HEDITCTL

4. CBitmap 类: public CGdiObject

CBitmap 类封装一个 Windows 的图形设备接口 (GDI) 位图, 并为管理位图提供成员函数。要使用 CBitmap 对象, 可以先构造一个对象, 再使用一个初始化成员函数在对象中设置一个句柄 (handle), 然后调用对象的成员函数。

#include <afxwin.h>

构造/析构——公共成员

CBitmap 构造一个 CBitmap 对象。

初始化——公共成员

LoadBitmap 从应用程序的可执行文件中调入命名的位图资源以初始化对象, 并连接位图到对象上。

LoadOEMBitmap 调入预定义过的 Windows 位图以初始化对象, 并连接位图到对象上。

CreateBitmap 初始化对象。此对象具有从属于设备并具有指定宽度、高度和图型位图。

CreatBitmapIndirect 初始化映象。此对象具有在 BITMAP 结构中给定宽度、高度和图形的位图。

CreatCompatibleBitmap 初始化一个具有位图的对象, 以与指定设备兼容。

CreatDiscardableBitmap 初始化一个具有可舍弃位图的对象, 以与指定设备兼容。

运算符——公共成员

FromHandle 当给 Windows HBITMAP 位图一个句柄时, 返回一个指向 CBitmap 对象的指针。

SetBitmapBits 设置位图的指定位。

GetBitmapBits 拷贝指定位图的某些位列指定缓冲区。

SetBitmapDimension 赋予位图宽度和高度值, 单位 0.1 mm。

GetBitmapDimension 返回位图的宽度和高度值。这里假定已使用 SetBitmapDimension 成员函数设置了高度与宽度。

成员函数

(1) CBitmap::CBitmap

CBitmap();

说明:构造一个 CBitmap 对象。必须用初始化成员函数之一来对构造的结果进行初始化。

参见:CBitmap::LoadBitmap, CBitmap::LoadOEMBitmap, CBitmap::CreateBitmap, CBitmap::CreateBitmapIndirect, CBitmap::CreateCompatibleBitmap, CBitmap::CreateDiscardableBitmap

(2) CBitmap::CreateBitmap

BOOL CreateBitmap(int nWidth, int nHeight, UINT nPlanes, UINT nBitcount, const void FAR * lpBits);

说明:初始化具有指定宽度、高度和图形的从属设备的内存位图。对于彩色位图, nPlanes 或 nBitcount 参数应该设置为 1。若所有的参数设置为 1, CreateBitmap 将建立一个单色位图。用户不能为显示设备直接选择位图, 可以采用下述方式: 使用 CDC::SelectObject 为内存设备描述表选择一个作为当前位图, 然后使用 CDC::BitBlt 函数拷贝它到任何兼容的设备描述表中。

用户想结束用 CreateBitmap 函数建立的 CBitmap 对象时, 首先从设备描述表中选择出位图, 然后删除 CBitmap 对象。

返回:若建立成功, 返回非零值; 否则, 返回零。

参见:CDC::SelectObject, CGdiObject::DeleteObject, CDC::BitBlt::CreateBitmap

(3) CBitmap::CreateBitmapIndirect

BOOL CreateBitmapIndirect(LPBITMAP lpBitmap);

说明:格式化一个在 lpBitmap 指向的结构中指定了宽度、高度和图形的位图。用户不能为显示设备直接选择位图, 可以采用下述方式: 使用 CDC::SelectObject 为内存设备描述表选择一个位图做为当前位图, 并使用 CDC::BitBlt 或 CDC::StretchBlt 函数拷贝它到任意相容的设备描述表中。

若 lpBitmap 参数指向的 BITMAP 结构已经通过使用 GetObject 函数进行了填充, 位图的一些位并未指定, 这时位图是未初始化的。为进行初始化, 应用程序中可以使用函数如 CDC::BitBlt, 或::SetDIBits, 把位从 CGdiObject::GetObject 的第一个参数所标识的位图拷贝到 CreateBitmapIndirect 建立的位图中。

用户想结束用 CreateBitmapIndirect 建立的 CBitmap 对象时, 首先从设备描述中选择出它, 然后进行删除。

返回:若建立成功, 返回非零; 否则, 返回零。

注:当前使用的位图方式可能是单色或彩色的。单色位图使用一位、一维平面方式。每一次扫描都是 16 位的倍数。

对于高度为 n 的单色位图扫描的组织方式如下:

```
scan 0  
scan 1  
⋮  
scan n-2  
scan n-1
```

单色设备的像素是黑色或白色的。若位图中相应位为 1, 像素开启(白色); 若为 0, 像素断开(黑色)。

所有的设备都支持具有 RC- BITBLT 位的位图, 这一位在 GetDeviceCaps 成员函数的索引 RASTERCAPS 中设置。

每一种设备都有它自身独特的彩色方式。为从一种设备向另一种设备转换位图, 可以使用 GetDIBits 和 SetDIBits Windows 函数。

参见: CDC::SelectObject, CDC::BitBlt, CGdiObject::DeleteObject, CGdiObject::GetObject, ::CreateBitmapIndirect

(4) CBitmap::CreateCompatibleBitmap

```
BOOL CreateCompatibleBitmap(CDC * pDc, int nWidth, int nHeight);
```

说明:初始化与 pDC 指定设备相兼容的位图。此位图具有与指定设备描述表相同数目的彩色平面或相同的每像素位方式, 并可以被选择作为与 pDC 指定的设备相兼容任何内存设备的当前位图。若 pDC 是一个内存设备描述表, 返回的位图与在设备描述表中选择的位图具有相同的格式。一个“内存设备描述表”是一组代表着显示表面的内存区, 它可用于在内存中准备图象, 然后再拷贝到相容设备的实际显示表面。内存设备描述表建立后, GDI 将自动为它选择单色存储位图。

由于一个彩色内存设备描述表可以选择彩色或单色位图, 由 CreateCompatibleBitmap 所返回的位图格式并不始终相同。但一个非内存的设备描述表相容位图格式却始终是设备所具有的格式:

用户要结束用 CreateCompatibleBitmap 函数建立的 CBitmap 对象时, 首先, 从设备描述表中选择出位图, 然后进行删除。

返回:若建立成功, 返回非零; 否则, 返回零。

参见: ::CreateCompatibleBitmap, CGdiObject::DeleteObject

(5) CBitmap::CreateDiscardableBitmap

```
BOOL CreateDiscardableBitmap(CDC * pDC, int nWidth, int nHeight);
```

说明:初始化一个与 pDC 标识的设备描述表相兼容的可删除的位图, 此位图与指定的设备描述表具有相同数目的彩色平面或相同的每像素位格式。应用程序可选择它作为与 pDC 指定设备相兼容内存设备的当前位图。Windows 可以删除本函数建立的位图, 这只有在应用程序未选择它进入显示文本的情况下, 若 Windows 在位图未被选择时删除了它, 而应用程序过后又试图选择本位图, 则 CDC::SelectObject 函数将返回 Null。

用户想结束 CreateDiscardableBitmap 函数建立的 CBitmap 对象时, 首先在设备描述表中选择出位图, 然后进行删除。

返回:若建立成功,返回非零;否则,返回零。

参见: ::CreateDiscardableBitmap, CGdiObject::DeleteObject

(6) CBitmap::FromHandle

```
static CBitmap * PASCAL FromHandle(HBITMAP hBitmap);
```

说明:当给 Windows GDI 位图一个句柄时,返回一个指向 CBitmap 对象的指针。若 CBitmap 对象未连接到句柄上,将建立一个临时的 CBitmap 对象并连接句柄。这个临时的 CBitmap 对象将持续有效到应用程序有空暇时间在它自身的事件循环中,这时所有临时的图形对象都将被删除。也就是说临时对象只有在处理窗口消息时才是有效的。

返回:成功时返回一个指向 CBitmap 对象的指针;否则返回 Null。

(7) CBitmap::GetBitmapBits

```
DWORD GetBitmapBits(DWORD dwCount, LPVOID lpBits) const;
```

说明:拷贝 CBitmap 的位类型到 lpBits 指向的缓冲区中。dwCount 参数指定了要拷贝的字节数。使用 GetObject 来决定正确的 dwCount 数值。

返回:返回位图中实际的字节数。出现错误时返回零。

参见: CGdiObject::GetObject, ::GetBitmapBits

(8) CBitmap::GetBitmapDimension

```
CSize GetBitmapDimension() const;
```

说明:返回位图的宽度和高度值。这里假定高度和宽度已由 SetBitmapDimension 成员函数设置。

返回:返回位图的宽度和高度值,单位 0.1mm。高度在 CSize 对象的 cy 成员之中,宽度在 cx 成员之中。若位图宽度和高度未使用 SetBitmapDimension 设置过,返回零值。

参见: CBitmap::SetBitmapDimension, ::GetBitmapDimension

(9) CBitmap::LoadBitmap

```
BOOL LoadBitmap(LPCSTR lpszResourceName);
```

```
BOOL LoadBitmap(UINT nIDResource);
```

说明:载入位图资源。此位图名为 lpszResourceName,或从应用程序可执行文件的 nIDResource 中 ID 数字来标识。载入的位置连接到 CBitmap 对象上。若 lpszResourceName 标识的位图不存在,或无足够内存来载入位图,函数返回零。应用程序必须调用 CGdiObject::DeleteObject 函数来删除 LoadBitmap 函数载入的位图。

只适于 Windows 3.1:增加了下列新位图:

OBM_UPARROWI

OBM_DNARROWI

OBM_RGARROWI

OBM_LFARROWI

这些位图不存在于先前版本 Windows 的设备驱动器中。读者若想得到完整的位图清

单,可查阅 Windows 3.1 的软件开发工具的《程序员指南》。

返回:若载入成功,返回零;否则,返回非零。

参见:CBitmap::LoadOEMBitmap, ::LoadBitmap, CGdiObject::DelectObject

(10)CBitmap::LoadOEMBitmap

BOOL LoadOEMBitmap(UINT nIDBitmap);

说明:调入 Windows 使用的经过预定义的位图。以 OBM_ 开头的位图名代表着由 Windows 3.0 以前版本所使用。注意常数 OEMRESOURCE 必须在 include WINDOWS.H 语句之前进行定义,以能够使用任何 OBM_ 常数。

返回:若载入成功返回非零;否则返回零。

参见:CBitmap::LoadBitmap, ::LoadBitmap

(11)CBitmap::SetBitmapBits

DWORD SetBitmapBits(DWORD dwCount, const void FAR * lpBits);

说明:设置位图的位到 lpBits 所给定的位值。

返回:返回设置位图的位时所使用的字节数;若函数调用失败,返回零。

参见:::SetBitmapBits

(12)CBitmap::SetBitmapDimension

CSize SetBitmapDimension(int nWidth, int nHeight);

说明:给位图的宽度和高度赋值,单位 0.1mm。GDI 并不使用这些数值,只有在应用程序调用了 SetBitmapDimension 成员函数返回这些值时例外。

返回:返回当前位图的幅度。高度是 CSize 对象的 cy 成员变量,宽度是 cx 成员变量。

参见:CBitmap::GetBitmapDimension, ::SetBitmapDimension

5. CBitmapButton 类:public CButton

使用 CBitmapButton 类能够生成以位图图标为标志的按钮控制器,用来代替文本标志。CBitmapButton 对象最多包括四个位图,即包含了按钮在下列四种不同状态下的图标:弹起(或正常)、按下(或被选择)、聚焦和失效。只有第一种位图是必须的,其他都是可选的。

位图按钮图标包括图标本身和图标的边界。边界在显示按钮状态时起到了显著的作用。例如:聚集状态下的位图和弹起状态通常是相似的,但前者有一个嵌在边界上的虚线矩形或边界上的粗线。失效状态下的位图和弹起状态通常相同,但具有较低的对比度。

这些位图可以是任意尺寸的,但一般当作它们和弹起状态下的位图大小相同。

不同的应用需要位图图标的不同组合:

Up	Down	Focused	Disabled	Application
x				位图
x	x			不具有 WS_TBASTOP 风格的按钮
x	x	x	x	所有状态都具备的对话框按钮
x	x	x		具有 WS_TABSTOP 风格的按钮

在窗口的用户区建立位图按钮控制器的步骤如下：

1. 为按钮建立 1—4 个位图图标。
2. 构造 CBitmapButton 对象。
3. 调用 Create 函数来建立 Windows 按钮控制器，并连接到 CBitmapButton 对象上。
4. 构造好位图按钮之后，调用 LoadBitmaps 成员函数来载入位图资源。

为在对话框中包括位图按钮控制器，可按以下步骤：

1. 为按钮建立 1—4 个位图图标。
2. 建立一个对话框模块，其中包括用户指定位置的按钮，按钮的大小是无关紧要的。
3. 设置按钮的标题为一定数值，比如“MYIMAGE”；并定义按钮的符号比如 IDC_MYIMAGE。
4. 在用户源程序文本中，给为按钮生成的图标赋予一个 ID 值，这可以通过附加字母“U”，“D”，“F”或“X”（分别代表 up、down、focused、disabled）到按钮标题的字符串后来完成。例如：对于按钮标题“MYIMAGE”，IDs 可以是“MYIMAGEU”、“MYIMAGED”、“MYIMAGEF”和“MYIMAGEX”。

5. 在应用程序的对话框类中（由 CDialog 派生），增加一个 CBitmapButton 成员对象。

6. 在 CDialog 对象的 OnInitDialog 例程中，调用 CBitmapButton 对象的 AutoLoad 函数。使用按钮的控制 ID 号和 CDialog 对象的 this 指针。

用户若想处理 Windows 的注意消息，比如 BN_CLICKED，可以在 CDialog 派生类对象上为每个消息增加一个消息图入口和消息处理成员函数。这类消息通常是由位图按钮控制器送往它的父类（通常是 CDialog 的派生类）。由 CBitmapButton 对象送出的消息与 CButton 对象送出的消息是相同的。

CToolBar 类接近位图按钮的方式与上面是不相同的。可参见 CToolBar 类的有关内容。

```
#include <afxext.h>
```

参见：CButton, CBitmapButton::AutoLoad, CToolBar

构造/析构——公共成员

CBitmapButton	构造一个 CBitmapButton 对象。
LoadBitmaps	从应用程序源文件中载入一个或多个指定的位图资源，初始化对象，并把位图连接到对象上。
AutoLoad	把对话框中的按钮与 CBitmapButton 类中的对象相连接，调用指定名称的位图，改变按钮的大小以与位图相匹配。