学步园

SHGetFileInfo 获得文件类型图标

2014年09月05日／综合／共 18345字／字号 小 中 大／评论关闭

Windows编程：Shell 编程 第四章 SHGetFileInfo()

以前，所有文件和目录都有一个确定的属性集：时间，日期，尺寸，以及表示‘只读的’，‘隐藏的，‘存档的’，或‘系统的’状态标志。然而，Windos95(及后来的WindowsNT4.0)出现使这些概念产生了改变，其中最重要的‘文件’变得更加广泛。现在，文件可以是任何Shell部件对象—不一定必须是文件系统的部件。

文件的精确定义是，任何作为Shell命名空间部件的对象称之为文件对象。注意，在定义中所说的‘命名空间’，它不是C++的关键字。‘Shell 命名空间’所指的是实际组成Shell的所有命名项的集合。它们都被显示在探测器的树观察中。

并不是所有文件都是文件系统中的一个实体，比如‘打印机’和‘我的计算机’。一个子文件对象的文件对象成为文件夹对象，文件和目录是最普通的文件对象。

所有这些变化作为微软完全面向对象的操作系统实现的第一步，已经融入到Windows9x和WindowsNT中。

一个文件对象可以有多少属性呢？回答是，它是一个集合，它完全包含MS-DOS一个文件的所有属性以及几个由Windows95 和Windows NT外壳的图形本质要求的属性。Shell API提供了一个复合函数和相当丰富的功能来探索给定文件对象的特征，它可以是一个普通文件，一个目录，甚或一个系统文件夹，一个象打印机或拨号连接那样的系统对象。这个函数就是SHGetFileInfo()。

在这一章中，主要目标是研究这个函数的原形。对于特定的文件对象，重点是：

怎样获取类型名
怎样获取探测器图标的Handle
怎样获取可执行文件的目标平台
怎样读出属性，以确定在探测器下对文件对象有哪些事情是可以做的，哪些事情是不能做的。

对SHGetFileInfo()函数所获得信息的多样性，你会感到奇怪。曾记得，有一个读者询问我，怎样确定一个给定的.EXE文件是16位的还是32位的(不用映射EXE的头结构)。我的答案是SHGetFileInfo()。几天以后，他回来再次问我，怎样获取驱动器的图标，我再次告诉他，使用SHGetFileInfo()。这最终的结果告诉我们仔细研究SHGetFileInfo()函数是理解Shell文件对象的最好方法。

SHGetFileInfo()函数的功能

同以往一样，我们从函数的原形开始，它在shellapi.h中定义。这个函数有五个变量，定义如下：

```
DWORD SHGetFileInfo(    LPCTSTR pszPath,
DWORD dwFileAttributes,
SHFILEINFO FAR* psfi,
UINT cbFileInfo,
UINT uFlags);
```

基本上讲，SHGetFileInfo()函数提供关于文件系统对象的信息。如前面解释的，这个对象可以是文件，文件夹，目录或驱动器根。DWORD的返回是指可能有相当多的返回状态，这与uFlags变量的设置有关。简单地说，使用这个函数，你可以期望：

确定可执行文件的目标平台(Win32，Win16，MS-DOS)

获取各种有特色的文件图标(小的，大的，有关联重叠的，选中的，打开的)

读出其它显示属性，如文件类型(显示在探测器类型列上的简短描述)和显示名(出现在名字列上)

读出任何其它属性，可以是文件特有的，如，是否可以拷贝，移动，删除或重命名，是否它可以形成一个快捷方式，它是否有子文件夹，是否是共享的，是拖拽目标，或有附加的属性页，等等。

SHGetFileInfo()函数的工作原理

为了正确地理解函数具有的功能，使用所有可能的方法强制调用这个函数是十分必要的。首先，让我们查看一下他所要求的变量：

变量名

描述

pszPath

一个包含要取得信息的文件相对或绝对路径的缓冲。它可以处理长或短文件名。

dwFileAttributes

资料上说，这个参数仅用于uFlags中包含SHGFI_USEFILEATTRIBUTES标志的情况。如此，它应该是文件属性的组合：存档，只读，目录，系统等。

Psfi

指向一个接收数据的SHFILEINFO结构的指针。

cbFileInfo

简单地给出上项结构的尺寸。

uFlags

函数的核心变量，通过所有可能的标志，你就能驾驭函数的行为和实际地得到信息。

SHFILEINFO结构定义如下：

```
typedef struct _SHFILEINFO
{
HICON hIcon;
int iIcon;
DWORD dwAttributes;
char szDisplayName[MAX_PATH];
char szTypeName[80];
} SHFILEINFO;
```

此外，这个结构总是用于返回数据到调用程序，并且从不需要初始化。唯一可以包含信息来影响函数行为的是dwAttributes成员，在后面将进一步给出解释。显然，驾驭SHGetFileInfo()函数各种行为的所有兴趣都集中在对uFlags变量值的设置上。绝大多数情况下，信息经由psfi缓冲返回，但也有些情况，回应可以有效地包含在函数的DWORD返回之中。

指定输入文件

恢复文件信息的函数首先要求操作文件的名字，pszPath参数就是用于这个目的。然而，有一些观念是需要澄清的。其中之一是，它可以是路径名(正象所期望的)，或是一个PIDL，这在第二章中讨论过了。

如果想要传递一个PIDL，而不是普通的路径名，你就应该设置SHGFI_PIDL标志到uFlags变量中。反之也是如此：如果设置了SHGFI_PIDL，则pszPath就必须指向一个ITEMIDLIST结构(即一个PIDL)。当然，pszPath也可以是文件夹名或驱动器名，此时，你需要把一个 '\'留在路径名的最后。即，你应该指定'c:\'而不是'c:'以避免错误地恢复了驱动器信息。

在SHGetFileInfo()中使用通配符

资料中并没有给出关于在SHGetFileInfo()函数中使用通配符的的任何解释，因此，你可能认为通配符不能识别。然而，我发现，如果传递一个带有通配符的串，然后提供至少一个文件匹配这个模式，函数照样正确工作。下图显示了一个简单程序的输出结果，这个程序将在后面详细讨论：

这个程序让我们选择一个文件名或路径名，然后恢复它的信息。它返回一个图标，显示名，类型名和所有其它属性的列表。另，你也可以询问程序以确定可执行文件的类型。Exe文件类型复选框抑制所有其它的选择。'返回码'标签显示函数的返回码(或它的文字描述)，通过选中'接受任何文件名'复选框，可以强迫函数接受任何东西作为输入文件。

上面显示，程序使用e:\mssdk\doc\misc*.txt路径名，你可以看到程序的响应：图标和类型名是正确的(在我的PC上，文本文件的描述是'Text Document')。奇怪，尽管指定了通配符，我们还是获得了非空的文件属性和显示名。图标和类型名可以从文件的扩展名中得，但是，同样的情况对显示名和属性是不行的—显然那些信息是相对于一个特殊文件

为了检测所发生的事情，我们再使用不同的路径调用函数：e:\mssdk\doc\misc\g*.*。象所看到的，在扩展名中和文件名中都有通配符。对话框的结如下所示：

正如显示所见，我们获得了与前面相同的信息文件，这明显说明，如果传递通配符，SHGetFileInfo()函数取第一个匹配这个串的文件，并对它进行操作。如果没有匹配这个模式的文件，这个函数什么也不做，直接返回0。另一个我们需要检测的情况是传递一个由'*.*'结尾的路径名。如图所示，函数返回相关文件夹的信息：

还要继续检测吗？先暂停一会，让我们来仔细地查看一下函数的输出。在'显示(Display)'字段，你可以看到一个点(.)，就象在老DOS下目录列表一样。这个结果印证了我前面所说的：SHGetFileInfo()函数操作在名字匹配于这个模式的第一个文件对象上。事实上，如果你试着使用*.*来枚举一个文件夹的内容，作为匹配的串，所获得的第一个项是点(.)。如果还不信，选择测试下面的代码：

```
WIN32_FIND_DATA wff;
FindFirstFile("*.*", &wff);
Msg(wff.cFileName);
```

总的来说，即使这个特征没有写进资料，你也可以在函数中使用通配符如果：

指定一个至少匹配一个文件的模式串

知道函数停止在第一个找到的文件上

这可能是SHGetFileInfo()函数的内在代码的某个地方保持了一个WIN32_FIND_DATA结构所致。它由底层文件信息所填写，因此，用在这里就一点也不奇怪了。附带地，这个结构还涉及到另一个Shell函数SHGetDataFromIDList()，他也返回文件对象的信息，这将在后面章节中进行表述。

文件的显示名

查看上面的截图，并在你自己的机器上运行这个程序，你就会注意到它返回的显示名稍微有些不同。在我的机器上，显示名是由文件名加扩展名组成，但是在你的机器上可能只看到文件名。这依赖于探测器的'观察|文件夹选项'对话框的设置，在此，你可以选择'隐藏已知类型的文件扩展名'。

这里，'已知文件类型'是指一个注册的文件类型。我们在第十四章中讨论怎样注册文件类型。现在，知道它就是一个Shell知道怎样处理的文档类型就足够了。如果你双击一个已知类型的文件，偶然地这个资料将由知道怎样处理它的程序打开。要编程取得这个设置，你需要使用SHGetSettings()函数。我们将在下一章讲述。

示例程序

前面，我们已经看到了这个用于测试的示例程序。它是一个基于对话框的应用程序。这次我给它取的工程(project)名为FileInfo。示例的操作部分围绕SHGetFileInfo()函数展开，作为一个通用查询执行器，它查询给定文件或文件夹的状态和属性。下面是程序界面：

正如所见，用户界面由一个编辑框和相关的按钮组成，使你能选择一个文件。不幸地是，这种风格不能选择目录，如果你想传递文件夹名，就必须手动键入。复选检查框允许选择想要加到调用中的标志，如果选中了EXE类型框，所有其它复选框都被禁止。这是因为SHGetFileInfo()函数要求单独指定可执行类型标志。文件图标绘制在一个静态控件上，属性被解析并转换为描述串。

大多数代码都在OnOK()方法中，当用户单击'Go'按钮后执行这段代码。要成功地编译这段代码，记住包含 #include "resource.h"语句，并保存对话框控件的IDs，<shlobj.h>中声明SHGetFileInfo()的原形：

```
void OnOK(HWND hDlg)
{
TCHAR szFile[MAX_PATH] = {0};
TCHAR szBuf[1024] = {0};
// 取得文件名
GetDlgItemText(hDlg, IDC_FILENAME, szFile, MAX_PATH);
////////////////////////////////////////
// 收集标志
//
DWORD dwFileAttributes = 0;
UINT uFlags = 0;
if(IsDlgButtonChecked(hDlg, IDC_FILEICON))
uFlags |= SHGFI_ICON;
if(IsDlgButtonChecked(hDlg, IDC_DISPLAYNAME))
uFlags |= SHGFI_DISPLAYNAME;
if(IsDlgButtonChecked(hDlg, IDC_TYPENAME))
uFlags |= SHGFI_TYPENAME;
if(IsDlgButtonChecked(hDlg, IDC_OTHER))
uFlags |= SHGFI_ATTRIBUTES;
if(IsDlgButtonChecked(hDlg, IDC_WILDCARD))
uFlags |= SHGFI_USEFILEATTRIBUTES;
if(IsDlgButtonChecked(hDlg, IDC_EXETYPE))
uFlags = SHGFI_EXETYPE;
////////////////////////////////////////
// 调用函数
//
SHFILEINFO sfi;
```

```
ZeroMemory(&sfi, sizeof(SHFILEINFO));
DWORD dwRC = SHGetFileInfo(
szFile, dwFileAttributes, &sfi, sizeof(SHFILEINFO), uFlags);
//////////////////////////////////////
// 处理界面显示
//
wsprintf(szBuf, "%d", dwRC);
SetDlgItemText(hDlg, IDC_RETCODE, szBuf);
wsprintf(szBuf, "Icon Index: %d", sfi.iIcon);
SetDlgItemText(hDlg, IDC_ICONINDEX, szBuf);
SetDlgItemText(hDlg, IDC_DISPLAY, sfi.szDisplayName);
SetDlgItemText(hDlg, IDC_TYPE, sfi.szTypeName);
//////////////////////////////////////
// Parse 解析和显示属性
//
DWORD dwAttrib = sfi.dwAttributes;
lstrcpy(szBuf, "");
if(dwAttrib != 0)
{
if(dwAttrib & SFGAO_CANCOPY)
lstrcat(szBuf, "Copy, ");
if(dwAttrib & SFGAO_CANMOVE)
lstrcat(szBuf, "Move, ");
if(dwAttrib & SFGAO_CANDELETE)
lstrcat(szBuf, "Delete, ");
if(dwAttrib & SFGAO_CANRENAME)
lstrcat(szBuf, "Rename, ");
if(dwAttrib & SFGAO_CANLINK)
lstrcat(szBuf, "Link, ");
if(dwAttrib & SFGAO_HASPROPSHEET)
lstrcat(szBuf, "PropSheet, ");
if(dwAttrib & SFGAO_GHOSTED)
lstrcat(szBuf, "Ghosted, ");
if(dwAttrib & SFGAO_SHARE)
lstrcat(szBuf, "Shared, ");
if(dwAttrib & SFGAO_HASSUBFOLDER)
lstrcat(szBuf, "SubFolders, ");
if(dwAttrib & SFGAO_REMOVABLE)
lstrcat(szBuf, "On removable media, ");
if(dwAttrib & SFGAO_FOLDER)
lstrcat(szBuf, "Folder, ");
lstrcat(szBuf, "and more!");
}
SetDlgItemText(hDlg, IDC_ATTRIB, szBuf);
//////////////////////////////////////
// 显示图标
//
HICON hIcon = sfi.hIcon;
SendDlgItemMessage(hDlg, IDI_ICON, STM_SETICON,
reinterpret_cast<WPARAM>(hIcon), 0);
}
```

这段代码足以产生你所看到的函数行为，但是我们还是要逐步加入更多的功能。If 开始是两个最主要块。它做了些什么并不是马上就清楚的。在这章剩余的部分，我们将主要讨论这两个代码段。这一段唯一要实现的是处理浏览(...)按钮。这需要在APP_DlgProc()函数和OnBrowse()函数中添加分枝语句开关，代码如下：

```
void OnBrowse(HWND hDlg)
{
TCHAR szFile[MAX_PATH] = {0};
OPENFILENAME ofn;
ZeroMemory(&ofn, sizeof(OPENFILENAME));
ofn.lStructSize = sizeof(OPENFILENAME);
ofn.lpstrFilter = "All files\0*.*\0";
ofn.nMaxFile = MAX_PATH;
ofn.lpstrFile = szFile;
if(GetOpenFileName(&ofn))
SetDlgItemText(hDlg, IDC_FILENAME, ofn.lpstrFile);
}
```

为了使用通用对话框(此处是Open对话框)，需要连接comdlg32.lib库和添加#include <commdlg.h>语句。下面图象显示了这个程序的典型输出，它被要求提供关于Favorites文件夹的图标，类型，显示名和属性：

注意，此时我们是通过物理名而不是PIDL引用文件夹的。在我的PC上，目录是C:\WINDOWS\Favorites，这并没有关系，即使我所寻找的文件在网络驱动器上—如果这样，详细资料也会传送。

SHGetFileInfo()

函数的标志

显然uFlags是SHGetFileInfo()函数的绝对中心。它可以用下列值的几乎任何组合构成，有一些组合在上面的代码中已经看到了：

代码
值
描述

SHGFI_ICON
0x0100
将文件的HICON类型的图标Handle存储到结构SHFILEINFO的hIcon成员中。

SHGFI_DISPLAYNAME
0x0200
将指向文件显示名串的指针存储到结构SHFILEINFO的szDisplayName成员中。

SHGFI_TYPENAME
0x0400
将指向文件类型串的指针存储到结构SHFILEINFO的szTypeName成员中。

SHGFI_ATTRIBUTES
0x0800
将DWORD类型的给定文件所有要恢的属性值存储到SHFILEINFO结构的dwAttributes成员中。

SHGFI_ICONLOCATION
0x1000
将指向包含了Shell正在用于指定对象的图标的文件名串的指针存储到SHFILEINFO结构的szDisplayName中。因此，这个标志不能和SHGFI_DISPLAYNAME标志一起使用。奇怪的是，它似乎仅在指定文件夹时才能工作，指定文件名则总是返回空。

SHGFI_EXETYPE
0x2000
引起函数返回一个表示可执行文件二进制格式和它的目标平台的值。

SHGFI_SYSICONINDEX
0x4000
引起函数返回一个包含图标的系统图像列表Handle。图标的索引存储在SHFILEINFO 结构的iIcon字段中。

通过使用这个测试程序，我们发现一个有趣的现象。似乎在SHGFI_ICON和SHGFI_ATTRIBUTES之间存在一种关系：前者暗含了后者，即，当指定SHGFI_ICON时，SHFILEINFO结构的dwAttributes成员总是被填写，无论是否指定了SHGFI_ATTRIBUTES。
所有上面的标志都告诉函数为程序员执行某种特有的任务。还有其它一些标志，但是它们不是主要的，其中有一些可以修饰上表标志指定的操作：

代码
值
描述

SHGFI_LARGEICON
0x00000
引起函数恢复文件的大图标。

SHGFI_SMALLICON
0x00001
引起函数恢复文件的小图标。

SHGFI_OPENICON
0x00002
对于文件夹，这个设置引起函数恢复它打开时显示的图标。

SHGFI_SHELLICONSIZE
0x00004
引起函数恢复具有一定尺寸的图标，这个图标尺寸在显示控制板的'外观'标签中设置。

SHGFI_SELECTED
0x10000
恢复的图标是一个文件选中时显示的图标(与高亮颜色混合的)。

SHGFI_LINKOVERLAY
0x08000
恢复的图标是一个文件作为快捷方式显示的图标。

这个表中的标志影响到SHGFI_ICON，并且仅在与它相关的时候起作用。就象你可以看到的那样，它能够使你获得非常有个性的图标。

另一个修饰前述标志的是SHGFI_ATTR_SPECIFIED标志，它作用于SHGFI_ATTRIBUTES标志。也就是说SHFILEINFO结构的dwAttributes字段已经由调用者想要SHGetFileInfo()函数恢复的属性初始化了。换句话说，如果dwAttributes设置了特殊的标志，比如SFGAO_SHARE，则在操作的文件上函数必须检查这个标志(仅仅是这个标志)。默认情况下，dwAttributes 包含0xFFFFFFFF，就是说需要检查所有属性。关于文件的属性，在后面我们将进一步说明。

对于完整的标志列表，到现在为止正好还有两个没有讲到：SHGFI_PIDL和SHGFI_USEFILEATTRIBUTES。在下一节中我们集中讨论这两个标志，以及修饰SHGFI_ICON的标志。

如果有一种方法返回数据，则你可以同时指定几个标志。也就是说，你可以一起请求图标，显示名和类型名，但是不能同时有大图标和小图标，因为它们通过同一个缓冲区返回。

获取给定文件的类型信息

如果想要知道系统用于给定文档种类的图标和类型名，你就必须借助于通配符。反之，你可以采用SHGetFileInfo()函数的特性，这是有很好的资料说明的。

由在uFlags参数中设置SHGFI_USEFILEATTRIBUTES标志，可以使函数认为在pszPath参数中传递的文件是存在的，此时，它可以获得扩展名，并且搜索注册表来得到图标和类型名信息。这确实是有趣的特征，它允许你查询给定文件族类的图标，仅需要简单地指定*.ext就可以了。

当然，如果使用了SHGFI_USEFILEATTRIBUTES标志，就不能再有其他标志如SHGFI_EXETYPE, SHGFI_ATTRIBUTES或SHGFI_PIDL标志了，因为它们都是特指文件存的标志。

其实，在上述过程中，最不可理解的就是这个标志的名称，为什么使用SHGFI_USEFILEATTRIBUTES名呢？这个名字和资料似乎暗示了它与SHGetFileInfo()函数dwFileAttributes变量之间有某种联系：函数的行为就象pszPath指定名的文件存在一样，且属性被设置到dwFileAttributes中。然而，这里文件属性的作用似乎被弱化了。不管wFileAttributes的值如何，上面所写的程序总能很好地运行。

为了查看这个标志的活动，在上面应用中，选中‘接受任何文件名’复选框，并输入*.htm到‘文件名’编辑框。下面是一个辅助函数可以独立的获得任何文件类型的类型名和图标：

```
HICON GetFileTypeIcon(LPCTSTR szFileType, LPTSTR szTypeName)
{
SHFILEINFO sfi;
ZeroMemory(&sfi, sizeof(SHFILEINFO));
SHGetFileInfo(szFileType, 0, &sfi, sizeof(SHFILEINFO),
SHGFI_USEFILEATTRIBUTES | SHGFI_ICON | SHGFI_TYPENAME);
lstrcpy(szTypeName, sfi.szTypeName);
return sfi.hIcon;
}
```

Shell图标尺寸

SHGFI_SHELLICONSIZE标志迫使函数恢复具有‘Shell图标尺寸’值指定尺寸的大图标，‘Shell图标尺寸’在下面的注册表路径上：

HKEY_CURRENT_USER\Control Panel\desktop\WindowMetrics

这些值由控制板显示小程序通过选择外观标签设置。他影响到整个桌面和文件夹内的大图标尺寸：

如果这个键不存在，或没有指定SHGFI_SHELLICONSIZE标志，则由SHGetFileInfo()函数接受到的图标尺寸遵循默认窗口设置，为32x32像素尺寸。每当你改变'Shell图标尺寸'键时，探测器都刷新其内部的图标缓存，它仅是由SHGFI_SYSICONINDEX返回的系统图标列表。要获得所恢复的实际图标尺寸，应该使用ImageList_GetIconSize()函数。

使用PIDL

SHGFI_PIDL标志简单地通知系统所传递的项如果代表了文件名，则实际是一个PIDL，因此需要特殊处理。例如，下面代码说明怎样获得'我的计算机'文件夹的图标：

```
LPITEMIDLIST pidl;
SHGetSpecialFolderLocation(NULL, CSIDL_DRIVES, &pidl);
DWORD dwRC = SHGetFileInfo(reinterpret_cast<LPCTSTR>(pidl),
dwFileAttributes, &sfi, sizeof(SHFILEINFO), uFlags | SHGFI_PIDL);
```

'我的计算机'是一个特殊文件夹，它并不映射到计算机磁盘上的物理目录。相反它是一个由命名空间扩展编码支持的虚拟文件夹。因为这样的文件夹并没有匹配的路径名，因此我们需要用其他的方法在SHGetFileInfo()函数中标识它，显然是PIDL。

从版本4.71之后，Shell API定义了SHGetSpecialFolderLocation()函数，它使用一个符号标识特殊文件夹，并且返回对应的PIDL。对'我的计算机'，其符号是CSIDL_DRIVES。在早期的Shell版本中取得特殊文件夹的PIDL也是可能的，但是操作很复杂。可是，如果我们想要获得不是文件系统的对象的PIDL，或不是特殊文件夹的PIDL，这实际上确实是一种必须由我们自己处理的代码。这样的代码在第五章中给出，在那里我们将写出遍历任何文件夹内容的客户例程。

如果使用上面那三行代码调用SHGetFileInfo()，输出的结果应该是：

获取文件属性

你可以从给定的文件对象中恢复一个很长的属性列表，很多都在OnOK()实现中的第二个 if 块中出现。这些使用SHGetFileInfo()函数所能获得的属性与使用IShellFolder接口的GetAttributesOf()方法恢复的属性是一样的。换句话说，SHGetFileInfo()函数，在这种情况下，封装了IShellFolder接口。你所能读到的属性全部都定义在shlobj.h头文件中。这里就是那些最有可能涉及到的属性的列表：

属性
描述

SFGAO_CANCOPY
文件对象可以通过拖拽或剪裁板进行拷贝

SFGAO_CANDELETE
可以通过Shell删除文件对象

SFGAO_CANLINK
文件对象可以建立快捷方式

SFGAO_CANMOVE
文件对象可以通过拖拽或剪裁板移动

SFGAO_CANRENAME
文件对象可以通过Shell重命名

SFGAO_HASPROPSHEET
文件对象至少有一张树形表单

SFGAO_GHOSTED
问津对象使用精灵图标显示(一般是隐藏文件)

SFGAO_LINK
这个文件对象是一个快捷方式

SFGAO_READONLY
这个文件对象是只读的

SFGAO_SHARE
指定的文件夹是共享的

SFGAO_HASSUBFOLDER
指定的文件夹至少有一个自文件夹

SFGAO_COMPRESSED
文件对象驻留于压缩驱动器上

SFGAO_FILESYSTEM
文件对象是文件系统得部件，不是虚拟文件夹。也就是说它是一个存在的物理对象(驱动器，目录或文件)

SFGAO_FOLDER
指定的对象是一个文件夹

SFGAO_REMOVABLE
文件对象驻留于可移动介质上(典型地软盘)

给出这个表之后，很容易写出函数来测试这些条件。例如，怎样才能知道一个给定的目录是否是共享的。
确定这个问题的最简单的方法是检查SHGetFileInfo()函数返回结构SHFILEINFO的dwAttributes字段相对SHGAO_SHARE位的内容：

```
BOOL IsDirectoryShared(LPCTSTR szDirName)
{
SHFILEINFO sfi;
ZeroMemory(&sfi, sizeof(SHFILEINFO));
SHGetFileInfo(szDirName, 0, &sfi, sizeof(SHFILEINFO), SHGFI_ATTRIBUTES);
return(sfi.dwAttributes & SFGAO_SHARE);
}
```

对于共享的给定文件夹，可能要求函数返回具有‘手捧’样式的图标： ，此后，还要求SHGetFileInfo()返回文件夹被‘选中’的图标，这些要求都不是牵强的。不幸地是这个函数并不支持这些特性，没有解决问题的办法了吗！下面说明解决这个问题需要做的工作。

建立‘手捧’文件夹图标
‘手捧’图标是shell32.dll的第29个图标(0为第一个时，是第28个)：

无需使用设备关联，XOR和AND屏蔽等操作，我们可以充分利用Windows95 通用控件：图像列表的功能。

图像列表表示一个图像(图标和Bitmap)的集合，它以一种非常特殊而有效的方法驻留在内存中：图像以单独的Bitmap形式并排存储，一个Bitmap包含了所有组成图像的要素。可以把它看作图像带或一盘电影胶片。图像列表的基本约束是所有图像都有相同的尺寸，因而，允许系统通过索引快速而容易地访问图像。图像列表一般应用于有大量小图片需要管理的场合，而且，很多Windows95和WindowsNT通用控件(列表观察和树观察等)要求通过图像列表提供图标。

从编程的角度讲，图像列表是不可视控件，有它自己的消息和风格集，且有一个特殊的Handle(HIMAGELIST)。图像列表有一个非常丰富的编程接口，以及管理列表 (抽取，添加，拷贝)操作的函数，用以支持拖拽和绘制。更多关于图像列表的信息，可以参考Platform SDK | User Interface
Services | Shell and Common Controls | Image Lists.

关于图像列表我们感兴趣的是它内建对图标与小Bitmap图像组合和重叠操作的支持。探测器本身使用图像列表来显示一定类型文件对象的复合图像，例如，快捷方式和共享文件夹：首先取得基本图标，然后，在必要时，以下述方式处理它：

混合以高亮色(选中状态)

混合以灰色(精灵状态或隐藏文件状态)

与另外的图标重叠，如连接或'手捧'

如果你想要做的全部就是简单地产生输出，则ImageList_SetOverlayImage()函数就是做这个操作的函数。它与ImageList_Draw()函数一道工作，组合给定设备关联的两个图标。下面是例子：

```
HICON hiFolder;
HICON hiHand;
// 装入图标
// hiFolder = ...;
// hiHand = ...;
// 取得要绘制的设备DC
HDC hdc = GetDC(GetFocus());
// 建立图像列表
HIMAGELIST himl = ImageList_Create(32, 32, ILC_MASK, 1, 0);
// 添加图标
ImageList_AddIcon(himl, hiFolder); // Icon index of 0
ImageList_AddIcon(himl, hiHand); // Icon index of 1
// 图标1(手捧图标)是要重叠的，设置为一号屏蔽
ImageList_SetOverlayImage(himl, 1, 1);
// 图标0(文件夹)必须由一号屏蔽重叠
ImageList_Draw(himl, 0, hdc, 0, 0, INDEXTOOVERLAYMASK(1));
// 释放图标
DestroyIcon(hiFolder);
DestroyIcon(hiHand);
// 清理和退出
ReleaseDC(GetFocus(), hdc);
ImageList_Destroy(himl);
```

这个源码段采用了几个图像列表的API函数。特别是ImageList_Create()，给出具有指定尺寸(32x32)图像的新列表标识。最终操作完成后，使用ImageList_Destroy()函数销毁它。与名字的意义一样ImageList_AddIcon()函数添加图标到指定的图像列表。这些函数以及其他函数在VC++在线资料中均有说明。

在上面的代码中是调用ImageList_SetOverlayImage()函数实现图像列表中第二个图标(索引1，'手捧'图标)作为重叠修饰#1，与索引为0的图标(即文件夹图标)在hdc指定的关联设备上重叠。注意，图标与修饰图标的索引是不同的—前者为0，后者为1。

尽管这段代码可以很好地运行并且也能完成所要求的操作，然而，如果能返回新图标的标识到调用者就更好了，因此，我们需要寻找象ImageList_Draw()样的函数，建立一个可以由HICON handle 标识的图标，然后返回这个handle给调用者。事实上，我们并不需要做太多的工作，ImageList_Merge()函数正好有能满足我们需要的功能。

```
HIMAGELIST ImageList_Merge(HIMAGELIST himl1,
int i1,
HIMAGELIST himl2,
int i2,
int dx,
int dy);
```

这个函数从两个图像列表中分别取得两个图像(可以是同一个列表)，用第二个落在第一个上的绘制方式合并图像。新图像存储在新的图像列表中，它是由在提交的两个图像上使用OR操作获得结果图像。资料上并没有完全清楚地说明dx,dy 参数的作用，但是有一些试验解释说它们表示重叠图像绘制在第一个图像上的相对位置。这个偏移从左上角像素开始计算，然而在绝大多数情况下，它们应该设置为0。

下面是函数GetSharedFolderIcon()的源码，它携带一个HICON，和返回一个新图标Handle，这是一个出示图像与'手捧'图像重叠的图标。

```
HICON GetSharedFolderIcon(HICON hiFolder)
{
HICON hiShared;
HICON hiHand;
// 取得'手捧'图标
ExtractIconEx("shell32.dll", 28, &hiHand, NULL, 1);
// 建立一个用于合并文件夹图标与手捧图标的图像列表
HIMAGELIST himl = ImageList_Create(32, 32, ILC_MASK, 1, 0);
// 添加图标到列表
ImageList_AddIcon(himl, hiFolder);
ImageList_AddIcon(himl, hiHand);
// 合并图标到新的图像列表
HIMAGELIST himlNew = ImageList_Merge(himl, 0, himl, 1, 0, 0);
// 抽取新图像列表中的图标
hiShared = ImageList_ExtractIcon(0, himlNew, 0);
// 释放'手捧'图标，并不释放'文件夹'图标
// 因为它是从调用者传递来的.
DestroyIcon(hiHand);
// 清理图像列表，并退出
ImageList_Destroy(himl);
ImageList_Destroy(himlNew);
return hiShared;
}
```

这个函数接收一个调用者要求建立共享标志的图标，所以，必须做的第一件事就是取得'手捧'图标，它存储在shell32.dll中的第28个索引处。使用ExtractIconEx()函数抽取图标，这不是惟一可用的方法(ExtractIcon()也能做的很好)，但是它有更多灵活的选择：可以同时抽取多种尺寸的多个图标。

```
ExtractIconEx("shell32.dll", 28, &hiHand, NULL, 1);
```

这一行指令仅仅从shell32.dll中装入第29个大尺寸图标(记住，索引从0开始)。我们既不想要小图标也不想要多个图标。关于ExtractIconEx()函数更多的细节参见VC++ 在线帮助资料

对这两个图标，我们建立一个图像列表，并添加这两个要组合的图标到列表中。然后合并图标，存储结果到另一个新列表中，这个列表仅有一个图标—ImageList_Merge()函数的语法要求你通过图像列表和索引对来标识图标。调用了ImageList_Merge()之后，得到了一

个新的，可以从其中抽取合成图标的图像列表标识。现在来修改OnOK()代码，如果SHGetFileInfo()函数操作的对象是一个共享对象，则这个新函数就被调用。

```
//////////////////////////////////////
// 显示图标
//
HICON hIcon;
if(dwAttrib & SFGAO_SHARE)
hIcon = GetSharedFolderIcon(sfi.hIcon);
else
hIcon = sfi.hIcon;
SendDlgItemMessage(hDlg, IDI_ICON, STM_SETICON, reinterpret_cast<WPARAM>
(hIcon), 0);
```

下面显示了函数执行的结果：

可执行文件的二进制格式

SHGetFileInfo()函数另一个特征是它能够返回可执行文件的二进制格式。通过设置正确的标志，你就可以知道一个给定.exe是32位还是16位模块，即使它仅需要最小的Windows平台。需要这个功能的典型情况是：

在编写一个系统范围的例程，分析窗口进程，或扫描文件时，你可能想要说明建立进程或窗口的程序是16位的还是32位的。

- 编程检测是否你的客户已经把工具从16位版本升级到32位版本
- 编写底层工具检测系统和文件
- 实现进程内通讯，这也需要可执行程序的类型知识。

如果是这些情况，惟一的方法就是了解Windows(或许还包括DOS)的可执行文件的二进制格式，并且手动查看二进制代码以查找它们的标识。幸运地是SHGetFileInfo()函数从必须查找二进制码的工作中拯救了我们。为了确定给定程序设计在那个Windows平台时代，你只需要指定SHGFI_EXETYPE标志就可以了。注意，这个标志不能与任何其它标志进行组合，否则不能正常执行。

恢复可执行文件的格式信息有几种情况，你必须分析函数的返回码来推断结果。SHGetFileInfo()返回DWORD值，此时低字表示可执行文件的签名，下面表中给出解释：

文件签名	Hex码	意义
PE	0x4550	Win32可执行格式，由微软所有32位操作系统采用。
NE	0x454E	Windows 3.x新的可执行格式，典型地16位窗口程序
MZ	0x5A4D	DOS 可执行格式，如果查询．com 或 ．bat也返回这个值。

对应的Hex码实际是文件签名列的字符码。例如 0x50 对应 P ， 0x45 对应 E 等。

高位字的两个字节包含了运行要求的最小操作系统版本号，如果你的目标只是要知道是否给定模块是16位的还是32位的，这个信息就不是确实必须的了，但是你会发现，对于老的Windows3．2程序，它是0x030A，而对于所有其它32位平台，它是0x0400。惟一的例外是程序指定的目标平台是WindowsNT3.5。此时这个值小于0x0400，即使它是32位程序—在这种情况下，这个值是0x0350。另一种可能是高位字为零，这说明，你查看了一个Win32控制台应用程序。

所以，当SHGetFileInfo()给出你想要知道的关于文件的所有信息时，可以看到它的编程接口还有相当大的改进余地。例如，给定一个文件名，检测它是否为32位还是16位或是DOS程序的过程还相当复杂。调用这个函数仅仅做了一半的工作，你必须检查结果来确定需要做些什么。

下面通过实现涉及'Exe类型'框被选中的代码来结束这个讨论。这需要在FileInfo.cpp的前头定义三个不同类型文件的常量，然后用上述行为测试SHGetFileInfo()函数的返回值。然后用测试结果修改这个应用的输出：

```
// 常量
const int PE_SIGN = 0x4550;
const int NE_SIGN = 0x454E;
const int MZ_SIGN = 0x5A4D;

//////////////////////////////////////
// 涉及到界面UI
//
if(uFlags == SHGFI_EXETYPE)
{
if(dwRC == 0)
lstrcpy(szBuf, "Not an executable file.");
else
lstrcpy(szBuf, "");
if(LOWORD(dwRC) == PE_SIGN)
{
lstrcat(szBuf, "32-bit");
if(HIWORD(dwRC))
lstrcat(szBuf, " Windows executable");
else
lstrcat(szBuf, " Console executable");
}
else if(LOWORD(dwRC) == NE_SIGN)
lstrcat(szBuf, "16-bit executable");
else if(LOWORD(dwRC) == MZ_SIGN)
lstrcat(szBuf, "DOS executable");
}
else
wsprintf(szBuf, "%d", dwRC);
```

下面图像说明查询Explorer.exe所发生的情况：

奇怪的是，SHGetFileInfo()函数并不认为DLL 或VxD 是可执行文件，也不返回二进制格式。因此没有办法知道DLL的二进制格式。这就是说上述解释工作仅仅是针对独立可执行文件(具有.EXE扩展)的。事实上这个函数对屏幕保护文件，无论有否.scr扩展，都失败。这可能是一个Bug。

SHGetFileInfo()函数的返回值

如果函数返回0，则某个地方发生了错误。在大多数情况下，是因为传递了不合理的文件名或PIDL，或指定了矛盾的标志组合。与前两个相比，后面一个更有可能。

除非指定的标志告诉它做指定的操作，如果每一个操作都顺利完成，这个函数返回1。一个例外是，当SHGFI_EXETYPE标志设置的时候，就象前一节所讨论的。另一个使返回码包含更多意义的情况是SHGFI_SYSICONINDEX标志被设置。此时，函数返回一个系统图像列表Handle，它包含了指定文件或文件夹的图标。

有趣的是SHGetFileInfo()函数甚至可以用于成功地恢复关联与CD-ROM的图标。对于其它驱动器而言这个几乎总是标准的图标的结果是由探测器依据autorun.inf文件的内容显示的。有一个函数能够正确地返回这个图标，对于编程而言是一个极大的帮助，无论你是否需要这个图标。

小结

SHGetFileInfo()并没有任何明显的Bug，但是他遭受到资料残缺不全的危害。如果你花费一点时间来研究资料里有什么和测试所有可能的标志组合，你可能偶然地会发现你所需要的东西，关键点在于—好的资料必须明显地说明函数能做什么和不能做什么。至少有三个问题需要SHGetFileInfo()函数来回答，但是发现这些问题远不是容易解决的。

为了修补这些漏洞，我们在这一章讨论了：

怎样取得相关于给定文件或文件夹的各种图标
怎样知道可执行文件(．EXE)的二进制格式
怎样确定给定文件或文件夹可能具有的系统属性
怎样使用图像列表合并两个图标，不用XOR修饰与设备关联操作

返回

【上篇】从CString到char []，怎么做安全
【下篇】引用和const引用（笔记）

作者：13134756222

- 该日志由 13134756222 于3年前发表在综合分类下，最后更新于 2014年09月05日.
- 转载请注明：SHGetFileInfo 获得文件类型图标 | 学步园 +复制链接
-

抱歉!评论已关闭.

- Hadoop I/O 上 SequenceFile 类
- Boost - Function 分析
- 作业的提交和监控（二）
- 作业的提交和监控（一）

随便看看

- 对gridview
- 回收站无法删除文件
- 一些有趣的代码
- 有线无线同时上
- 中文处理 get post
- 用flash做背景
- 值类型与引用类型关系
- 远程桌面由于以下
- 自动化测试案例
- 自己动手写操作系统 清晰

- web前端
- 数据库
- 编程语言
- 搜索技术
- 关于本站

返回首页