29

FILE DECOMPRESSION AND INSTALLATION

When operating systems were simpler, and the typical user was interested in computers as a hobby, file installation was fairly easy. Most applications simply came with a document that told users to copy all the necessary files to their hard disk, and that was all there was to it.

Todays' Windows 95 environment is considerably more complicated, and the typical user is more interested in accomplishing tasks than in understanding how the computer works. Satisfying these users made application installation programs significantly more difficult to develop. An installation program now must deal with such things as compressed installation files and multiple versions of system libraries. The Win32 API contains functions that allow your application to deal with these situations. In addition, it also contains many functions that help your application work with compressed files. You can use these functions in any situation, but they are used most often in conjunction with application installation packages.

File Installation Overview

Consider, as an example, the problem posed by the existence of multiple versions of the common dialog library COMMDLG.DLL. This library provides common dialog boxes for applications. Imagine that a computer user has installed a new application package that uses the latest version of this library. Everything is working well. The user then purchases a copy of your application that the local software store has had on its shelf for several months. Your application also uses the COMMDLG.DLL library, but since it was packaged some months ago, it includes a rather old copy of this library. If your installation program simply copies the COMMDLG.DLL library to the users system, your application will work fine, but the original application that required the newer version of the library will no longer function correctly. Your installation
program has caused a problem for the user by unintentionally replacing a newer version of the COMMDLG.DLL library with an older version.

Installing the Proper Version

Because of the problem outlined in the previous section, application installation programs should not blindly copy files to the user's system. Instead, an application should determine if a version of the file already exists, and whether or not the existing file is newer than the file about to be installed. Windows provides the VerFindFile() and VerInstallFile() functions just for this purpose. VerFindFile() determines where an application should install a given file. It checks the system for existing copies of the file, and returns information that is then used by VerInstallFile().

Listing 29-1 shows a function that uses the VerFindFile() and VerInstallFile() functions to determine where to install a shared file and to install it.

Listing 29-1. Installing with Version Checking

```
.
.
// Install a shared file with version checking.
//............................................
InstallSharedFile( "COMMDLG.DLL", "C:\\MYAPP" );
.
.
.

DWORD InstallSharedFile( LPCTSTR szFileToInstall, LPCTSTR szDirectory )
{
   TCHAR   sWinDir[_MAX_PATH];             // Holds the windows directory.
   TCHAR   sCurDir[_MAX_PATH];             // Returns path to preexisting file.
   TCHAR   sDestDir[_MAX_PATH];            // Returns recommended path for file.
   TCHAR   sTmpFile[_MAX_PATH];            // Buffer for temporary file name.
   UINT    uCurLen  = MAX_PATH;            // Holds the length of sCurDir.
   UINT    uDestLen = MAX_PATH;            // Holds the length of sDestDir.
   UINT    uTmpLen  = MAX_PATH;            // Holds the length of sTmpFile.
   DWORD   dwStatus;                       // Return value from function.

   // Get the windows directory.
   //............................
   GetWindowsDirectory( sWinDir, sizeof(sWinDir) );

   // Find out if this file exists on the system already and where.
   //...........................................................
   dwStatus = VerFindFile( VFFF_ISSHAREDFILE, szFileToFind, sWinDir, szAppDir,
                           sCurDir, &uCurLen, sDestDir, &uDestLen );

   if ( dwStatus == VFF_FILEINUSE )
       MessageBox( "This file is in use, close all applications and try again.", "Install", MB_OK |
MB_ICONSTOP );
   else
       dwStatus = VerInstallFile( 0, szFileToFind, szFileToFind, "A:\\",
                                  sDestDir, sCurDir, sTmpFile, uTmpLen );
   return( dwStatus );
}
```

The first parameter to VerFindFile() indicates whether or not the file is private to this application. In the example above, VFF_ISSHAREDFILE indicates that the given file is shared among different Windows applications. After the call to VerFindFile(), the buffer sDestDir will contain the location for the new file that Windows recommends. The buffer sCurDir will contain the location of any preexisting version of the file. The return value from VerFindFile() can be one of the values shown in Table 29-4 in the description of this function.

After having called VerFindFile(), and assuming that no errors were encountered, the application can use VerInstallFile() to actually install the file. VerInstallFile() will copy the new file into a temporary file, decompressing it if necessary. It will then check the version information of any preexisting file. If VerInstallFile() finds no problems, the new file will be installed in place of the original file. VerInstallFile() returns a bit mask containing one or more of the values shown in Table 29-3 indicating the success or failure of the installation. Refer to the description of VerInstallFile() at the end of this chapter for more information.

The Version Resource

A version information resource may be embedded within executable files, libraries, and controls. Create this resource just as you do other windows resources, such as bitmaps and menus – with a resource editor. If an error occurs while installing files, the application can retrieve the version information resource to display useful error messages to the user. An application can retrieve the version information resource for any file by using the GetFileVersionInfoSize(), GetFileVersionInfo(), and VerQueryValue() functions. Listing 29-2 shows an example of a version resource.

Listing 29-2. Version Resource

```
1 VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "Comments", "This is a sample version information header.\0"
            VALUE "CompanyName", "Waite Group Press\0"
            VALUE "FileDescription", "Sample Application\0"
            VALUE "FileVersion", "1.01\0"
            VALUE "InternalName", "VerQuery\0"
            VALUE "LegalCopyright", "Copyright \251 1995\0"
            VALUE "OriginalFilename", "VerQuery.RC\0"
            VALUE "ProductName", "Version Query Example\0"
            VALUE "ProductVersion", "1.0\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END
```

Once you have retrieved the version information resource, you can use VerQueryInfo() to obtain specific values. Refer to the description of VerQueryValue() for details on retrieving information from the version information resource.

File Decompression

The version support functions already discussed can automatically handle compressed files. Sometimes, it is useful for an application to be able to decompress files independent of the version control functions. Windows provides a number of functions for this purpose.

If an application is decompressing a file, it can use the LZCopy() function to accomplish this. Before you can use the LZCopy() function, you must open the source and destination files with LZOpenFile(). This function is similar to the OpenFile() function, and returns file handles that are used by LZCopy(), LZRead(), and LZSeek(). If a file is already opened with the OpenFile() function, use that file handle with the LZInit() function to obtain an equivelent handle as returned by the LZOpenFile() function. Finally, after the files have been copied, the application uses the LZClose() function to close the file handles.

A compressed file may have embedded within it the name of the uncompressed file. An application can use the GetExpandedName() function to retrieve this name. If the compressed file does not have the original file name embedded within it, then this function simply returns the name of the compressed file.

If your application requires a large read-only file, you can save space on the user's system by keeping the file compressed. The Win32 API provides the LZRead() and LZSeek() functions to allow you to read data from a compressed file as if it were uncompressed. The LZRead() and LZSeek() are functionally similar to the ReadFile() and SetFilePointer() functions.

The Win32 API does not provide functions to compress files. Compressing files is done with the COMPRESS.EXE utility, an MS-DOS utility that compresses files using a scheme that is compatible with the functions shown in this chapter.

File Decompression and Installation Function Summary

Table 29-1 summarizes the file decompression and installation functions. A detailed description of each function follows the table.

Table 29-1. File Decompression and Installation Function Summary

Function	Purpose
GetExpandedName	Retrieves the original file name of a compressed file. The file must have been compressed using the /r switch.
GetFileVersionInfo	Obtains the version information resource from a Win32 executable file.
GetFileVersionInfoSize	Returns the size of the version information resource in a Win32 executable file.
LZClose	Closes the file opened with LZOpenFile().
LZCopy	Copies a file. If the source file was compressed using the Microsoft compression utility, then the destination file will be decompressed. Otherwise, a direct copy is made.
LZInit	Initializes the decompression library for multiple file operations.
LZOpenFile	Opens a file for use with other decompression functions.
LZRead	Reads from a file that was opened with LZOpenFile(). If the file is compressed, LZRead() decompresses the data before placing it in the buffer.
LZSeek	Moves the read pointer for a file opened with LZOpenFile(). If the file is compressed, the read pointer is moved based on an expanded image of the data.
VerFindFile	Determines where a file should be installed, searching for existing versions of the file.
VerInstallFile	Installs a file, decompressing as needed.
VerQueryValue	Obtains specific version information from the buffer returned by GetFileVersionInfo().

GETEXPANDEDNAME — WIN32S WINDOWS 95 WINDOWSNT

Description — GetExpandedName() retrieves the original name of a compressed file, if the file was compressed using the Microsoft compression program, COMPRESS.EXE, and the /r switch was specified.

Syntax — INT GetExpandedName(LPTSTR lpszSourceFile, LPTSTR lpszOriginalName)

Parameters

lpszSourceFile — LPTSTR: A pointer to a null terminated string that contains the file name of the compressed file.

lpszOriginalName — LPTSTR: A pointer to a buffer that receives the original name of the uncompressed file. If the compressed file was not compressed with the /r switch, then the source file name will be copied to this buffer.

Returns — INT: If successful, a value of 1. Otherwise, LZERROR_BADVALUE.

Include File — lzexpand.h

Example — The following example retrieves the original file name of the compressed file COMP.DO_.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           TCHAR szLongName[MAX_PATH];

                           if ( GetExpandedName( "COMP.DO_", szLongName ) != LZERROR_BADVALUE )
                              MessageBox( hWnd, "Expanded File name.", szLongName, MB_OK );
                        }
                        break;
         .
         .
         .
```

GETFILEVERSIONINFO WIN32S WINDOWS 95 WINDOWS NT

Description	GetFileVersionInfo() retrieves the version information resource from a Win32 executable file, library, or control. An application must include VERSION.LIB to reference this function. GetFileVersionInfoSize() obtains the size of the buffer needed to hold the version information resource for any given file. After allocating an appropriate buffer, the application can use GetFileVersionInfo() to retrieve the version information resource.
Syntax	BOOL GetFileVersionInfo(LPTSTR lpszFileName, DWORD dwReserved, DWORD dwBufferSize, LPVOID lpBuffer)
Parameters	
lpszFileName	LPTSTR: A pointer to a buffer containing the name of the file from which the version information is to be retrieved.
dwReserved	DWORD: This parameter is ignored.
dwBufferSize	DWORD: On entry, contains the size of the buffer pointed to by lpBuffer. On return, this parameter contains the actual size of the buffer required to contain the version resource. If the buffer is not large enough to contain the version resource, the resource is truncated to fit.
lpBuffer	LPVOID: A pointer to a buffer that will contain the version resource.
Returns	BOOL: TRUE if successful; otherwise, the return value is FALSE.
Include File	winver.h
See Also	GetFileVersionInfoSize(), VerQueryValue()
Example	The following example uses GetFileVersionInfoSize() to retrieve the size of the version resource, then uses GetFileVersionInfo() to retrieve the version information. The CompanyName, FileVersion, and ProductName strings are then retrieved from the StringFileInfo section of the version information. Refer to Listing 29-2 at the beginning of this chapter for the version information in the resource file used with this example.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
static HWND hList;

   switch( uMsg )
   {
      case WM_CREATE :
              hList = CreateWindowEx( WS_EX_CLIENTEDGE, "LISTBOX", "",
                                      LBS_STANDARD | WS_CHILD | WS_VISIBLE,
                                      0, 0, 10, 10, hWnd, (HMENU)101, hInst, NULL );
              break;

      case WM_SIZE :
              MoveWindow( hList, 0, 0, LOWORD( lParam ), HIWORD( lParam ), TRUE );
              break;

      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           DWORD  dwSize;
                           DWORD  dwReserved;
                           LPVOID lpBuffer;

                           dwSize = GetFileVersionInfoSize( "GETVERIN.EXE", &dwReserved );

                           lpBuffer = HeapAlloc( GetProcessHeap(), HEAP_ZERO_MEMORY, dwSize );

                           if ( lpBuffer &&
                                GetFileVersionInfo( "GETVERIN.EXE", 0, dwSize, lpBuffer ) )
                           {
                              LPTSTR lpStr;
                              DWORD  dwLength;
```

```
                  // Retrieve version information.
                  //..............................
                  VerQueryValue( lpBuffer, "\\StringFileInfo\\040904b0\\CompanyName",
                                 &lpStr, &dwLength );
                  SendMessage( hList, LB_INSERTSTRING, (WPARAM)-1, (LPARAM)lpStr );

                  VerQueryValue( lpBuffer, "\\StringFileInfo\\040904b0\\FileVersion",
                                 &lpStr, &dwLength );
                  SendMessage( hList, LB_INSERTSTRING, (WPARAM)-1, (LPARAM)lpStr );

                  VerQueryValue( lpBuffer, "\\StringFileInfo\\040904b0\\ProductName",
                                 &lpStr, &dwLength );
                  SendMessage( hList, LB_INSERTSTRING, (WPARAM)-1, (LPARAM)lpStr );
               }

               if ( lpBuffer )
                  HeapFree( GetProcessHeap(), 0, lpBuffer );
            }
            break;
   .
   .
   .
```

GetFileVersionInfoSize — Win32s Windows 95 Windows NT

Description	GetFileVersionInfoSize() retrieves the size of the version information resource of the specified file. This information allows an application to allocate a buffer of the appropriate size for use with the GetFileVersionInfo() function. An application must include VERSION.LIB to reference this function.
Syntax	DWORD GetFileVersionInfoSize(LPTSTR lpszFileName, LPDWORD lpdwReserved)
Parameters	
lpszFileName	LPTSTR: A pointer to a buffer containing the name of a file. The size of the version information resource within this file is returned.
lpdwReserved	LPDWORD: A pointer to a DWORD value. This function sets the value of this DWORD to zero.
Returns	DWORD: The size, in bytes, of the version information resource contained within the specified file.
Include File	winver.h
See Also	GetFileVersionInfo(), VerQueryValue()
Example	See GetFileVersionInfo() for an example of this function.

LZClose — Win32s Windows 95 Windows NT

Description	LZClose() closes the file handle that was obtained by using the LZOpenFile() function.
Syntax	VOID LZClose(INT hFile)
Parameters	
hFile	INT: Identifies the file to be closed. This value was returned by a previous call to LZOpenFile().
Returns	VOID: This function does not return a value.
Include File	lzexpand.h
See Also	LZOpenFile()
Example	See LZCopy() for an example of this function.

LZCopy — Win32s Windows 95 Windows NT

Description	LZCopy() copies a source file to a destination file. If the source file is compressed, the destination file will be decompressed.

Syntax	LONG LZCopy(INT hSourceFile, INT hDestFile)
Parameters	
hSourceFile	INT: The handle identifying a file opened with the LZOpenFile() function. This file may be compressed.
hDestFile	INT: The handle identifying a file opened with the LZOpenFile() function. If the source file is a compressed file, the destination file will be decompressed.
Returns	LONG: If successful, the size, in bytes, of the destination file; otherwise, a value less than zero is returned, which specifies an error condition. Table 29-2 gives specific error values.

Table 29-2. LZ Function Error Codes

Return Value	Description
LZERROR_BADINHANDLE	The parameter specified for hSourceFile was not valid.
LZERROR_BADOUTHANDLE	The parameter specified for hDestFile was not valid.
LZERROR_BADVALUE	One of the input parameters is not valid.
LZERROR_GLOBALLOC	An error occurred allocating memory for the decompression process. There is currently a limit of 16 open compressed files.
LZERROR_GLOBLOCK	An error occurred locking the memory used for the decompression buffer.
LZERROR_READ	The source file could not be read. This could be due to specifying a corrupt compressed file.
LZERROR_WRITE	There is insufficient space for the output file.

Include File	lzexpand.h
See Also	LZClose(), LZInit(), LZOpenFile()
Example	This example copies the compressed file COMP.DO_ to an uncompressed file “Sample Document.DOC” when the user presses the Test! menu item..

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_TEST :
                        {
                           INT      hSource;
                           INT      hDestination;
                           LONG     lRet;
                           OFSTRUCT OfStruct;

                           // Open the source and destination.
                           //.................................
                           hSource      = LZOpenFile( "COMP.DO_", &OfStruct, OF_READ );
                           hDestination = LZOpenFile( "Sample Document.DOC", &OfStruct, OF_CREATE );

                           // Decompress the document.
                           //........................
                           lRet = LZCopy( hSource, hDestination );

                           // Close the files.
                           //................
                           LZClose( hSource );
                           LZClose( hDestination );

                           if ( lRet > 0 )
                              MessageBox( hWnd, "File copied.", lpszTitle, MB_OK | MB_ICONINFORMATION );
                           else
                              MessageBox( hWnd, "File NOT copied!", lpszTitle, MB_OK | MB_ICONASTERISK );
                        }
                        break;
```

.
.
.

LZINIT

WIN32S WINDOWS 95 WINDOWS NT

Description	LZInit() converts a file handle obtained by the OpenFile() function into a file handle that can be used with the various decompression functions. If the file is compressed, then LZInit() allocates the appropriate buffers to allow decompression.
Syntax	INT LZInit(INT hFile)
Parameters	
hFile	INT: The file handle of a file opened with OpenFile().
Returns	INT: A new file handle that can be used with the file decompression library.
Include File	lzexpand.h
See Also	LZRead(), LZSeek(), LZOpenFile(), OpenFile()
Example	The following code segment opens a file using the normal OpenFile() function. It then uses LZInit() to convert the file handle into one usable by the decompression library, and performs a read and a seek on the file with the LZSeek() and LZRead() functions.

```
HFILE      hNormalFile;
OPENSTRUCT OpenStruct;
LONG       SeekLocation;
INT        hCompressedFile;

//  First, open the file as a normal file.
//......................................
hNormalFile = OpenFile(lpszFileName, &OpenStruct, OF_READ);
.
.
.
//  We have determined that this is a compressed file. Switch to the
//    file decompression library routines.

// Convert handle to LZ handle and read data.
//.........................................
hCompressedFile = LZInit( hNormalFile );
LZRead( hCompressedFile, &SeekLocation, sizeof(SeekLocation) );
LZSeek( hCompressedFile, SeekLocation, 0 );
```

LZOPENFILE

WIN32S WINDOWS 95 WINDOWS NT

Description	LZOpenFile() prepares files for use by other decompression functions, deletes files, and creates new uncompressed files.
Syntax	INT LZOpenFile(LPTSTR lpFileName, LPOFSTRUCT lpOpenStruct, WORD wStyle)
Parameters	
lpFileName	LPTSTR: A pointer to a buffer containing the name of the file to open.
lpOpenStruct	LPOFSTRUCT: A pointer to an OFSTRUCT structure. This function will fill the values of the OFSTRUCT structure the first time the file is opened. This structure then can be used by subsequent calls to LZOpenFile(). Refer to the description of OpenFile() in Chapter 17 for more information.
wStyle	WORD: Specifies a bit mask of various actions. Refer to the description of OpenFile() in Chapter 17 for more information.
Returns	INT: An integer value identifying the open file if successful. Otherwise, it returns a value less than zero, specifying an error condition. Refer to Table 29-2 for possible return values.
Include File	lzexpand.h

See Also	LZClose(), LZCopyFile()
Example	See the example of the LZCopy() function.

LZREAD WIN32S WINDOWS 95 WINDOWS NT

Description	LZRead() reads data from a file opened with LZOpenFile() or LZInit(). From the point of view of the application, this function is identical to a normal file read. LZRead() handles all required decompression.
Syntax	INT LZRead(INT hFile, LPTSTR lpBuffer, INT nByteCount)
Parameters	
hFile	INT: A file handle retrieved by using LZOpenFile() or LZInit().
lpBuffer	LPTSTR: A pointer to the buffer where the data is to be placed.
nByteCount	INT: Specifies the number of bytes to read. This value is in reference to uncompressed data.
Returns	INT: If successful, the number of bytes actually read; otherwise, the return value is an error code. Refer to Table 29-2 for possible error codes.
Include File	lzexpand.h
See Also	LZInit(), LZSeek()
Example	LZInit() gives an example of this function.

LZSEEK WIN32S WINDOWS 95 WINDOWS NT

Description	LZSeek() moves the read pointer of the specified file.
Syntax	LONG LZSeek(INT hFile, LONG lOffset, INT iFrom)
Parameters	
hFile	INT: A file handle retieved by using LZOpenFile() or LZInit().
lOffset	LONG: Specifies the distance, in bytes, that the read pointer is to be moved. This value is in reference to uncompressed data. This value may be negative to move backward in the file.
iFrom	INT: A flag specifying how the read pointer is to be moved. Refer to Table 29-3 for specific values for this parameter.

Table 29-3. Valid Values for the IFrom Parameter of LZSeek()

Value	Meaning
0	Moves the read pointer to the position indicated, starting at the beginning of the file.
1	Moves the read pointer the specified number of bytes from the current position.
2	Moves the read pointer the specified number of bytes from the end of the file.

Returns	LONG: If successful, the new absolute position of the read pointer; otherwise, the return value is an error code. Refer to Table 29-2 for possible error codes.
Include File	lzexpand.h
See Also	LZInit(), LZRead()
Example	LZInit() gives an example of this function.

VERFINDFILE WIN32S WINDOWS 95 WINDOWS NT

Description	VerFindFile() recommends the proper location that an application should use when installing a specific file. This recommendation is based on whether a version of the file already exists on the

user's system. The values returned by this function should be used in a subsequent call to VerInstallFile() to actually install the file. An application must include VERSION.LIB to reference this function.

Syntax DWORD VerFindFile(DWORD dwFlags, LPTSTR szFileName, LPTSTR szWindowsDirectory, LPTSTR szApplicationDirectory, LPTSTR szCurrentDirectory, LPDWORD lpdwCurrentLength, LPTSTR szDestinationDirectory, LPDWORD lpdwDestinationLength)

Parameters

dwFlags DWORD: If this parameter is VFFF_ISSHAREDFILE, then the file is considered to be a shared file, i.e. used by multiple applications. If this parameter is zero, then the file is considered to be private to this application.

szFileName LPTSTR: A pointer to the name of the file. This should include only a file name and extension. It should not include a drive letter or path.

szWindowsDirectory LPTSTR: A pointer to the name of the directory where Windows is running or will be run. You can use the GetWindowsDirectory() function to obtain this value.

szApplicationDirectory LPTSTR: A pointer to the name of the directory where the application is being installed.

szCurrentDirectory LPTSTR: A pointer to a buffer where this function returns the directory if any existing version of the file is located. If there is no existing version of the file, this buffer will contain a zero-length string.

lpdwCurrentLength LPDWORD: A pointer to a DWORD value. On entry to this function, this DWORD should be set to the length of the szCurrentDirectory buffer. On return from this function, this DWORD will be set to the length of the string returned. If szCurrentDirectory is too small to contain the entire path, then this parameter is set to the number of bytes required to hold the path name.

szDestinationDirectory LPTSTR: A pointer to a buffer where Windows will place the directory name of the location that it recommends be used to install the file.

lpdwDestinationLength LPDWORD: A pointer to a DWORD value. On entry to this function, this value should be set to the length of the szDestinationDirectory buffer. On return from this function, this value will be set to the length of the string returned. If szDestinationDirectory is too small to contain the entire path, then this parameter is set to the number of bytes required to hold the path name.

Returns DWORD: The return value is a bit mask indicating various results. It can be one or more of the values shown in Table 29-4.

Table 29-4. VerFindFile() Return Flags.

Bit Value	Description
VFF_CURNEDEST	A previous version of the file exists, but is not in the recommended location.
VFF_FILEINUSE	A previous version of the file exists, and is currently being used by Windows or another application. The file may not be deleted or replaced.
VFF_BUFFTOOSMALL	Either lpdwCurrentLength or lpdwDestinationLength specified that the respective buffer was too small to contain the entire path name.

Include File winver.h

See Also VerInstallFile()

Example Refer to Listing 29-1 in the introduction of this chapter for an example of this function.

VERINSTALLFILE WIN32S WINDOWS 95 WINDOWS NT

Description VerInstallFile() installs a specific file. The file is decompressed if necessary, and version information is checked against any preexisting file. An application must include VERSION.LIB to reference this function.

Syntax	DWORD VerInstallFile(DWORD dwFlags, LPTSTR szSrcFileName, LPTSTR szDestFileName, LPTSTR szSrcDirectory, LPTSTR szDestDirectory, LPTSTR szExistingDirectory, LPTSTR szTempName, LPDWORD lpdwTempLength)
Parameters	
dwFlags	DWORD: Various flags that may be used to alter the operation of this function. Table 29-5 gives valid values for this parameter.

Table 29-5. VerInstallFile()dwFlags.

Bitmask	Meaning
VIFF_FORCEINSTALL	Forces VerInstallFile() to install the new file regardless of any version or type mismatch with an existing file. If VIFF_DONTDELETEOLD is not specified, and the previous version of the file exists in a different directory, it will be deleted. If the previous version of the file exists in the destination directory specified, it will be overwritten regardless of the setting of VIFF_DONTDELETEOLD.
VIFF_DONTDELETEOLD	If a previous version of the file exists in a directory other than the recommended destination directory, using this flag will keep VerInstallFile() from removing the old file.

szSrcFileName	LPTSTR: The name of the file to be installed. This can only include the file name and extension. No path or drive information should be included.
szDestFileName	LPTSTR: The name this file should be given once installed. This can include only the file name and extension. No path or drive information should be included. This parameter allows the file to be renamed during the installation process.
szSrcDirectory	LPTSTR: The drive and directory where the source file exists.
szDestDirectory	LPTSTR: The drive and directory where the file is to be installed. This is usually the szDestinationDirectory parameter of VerFindFile().
szExistingDirectory	LPTSTR: The location where a preexisting version of the file can be found. This is usually the szCurrentDirectory parameter of VerFindFile().
szTempName	LPTSTR: A buffer where this function will store the temporary name used during the file-installation process.
lpdwTempLength	LPDWORD: The size of the szTempName buffer. On return, this parameter will indicate the actual size of the szTempName string.
Returns	DWORD: A bit mask that indicates various exception conditions during the installation process. One or more of the values defined in Table 29-6 may be included.

Table 29-6. VerInstallFile() Return Codes.

Bit Mask	Meaning
VIF_TEMPFILE	Indicates that a temporary file has been left in the destination directory due to some installation error. Other bits will be set indicating the specific error. An application should remove the temporary file when it is no longer needed.
VIF_MISMATCH	The new file and the preexisting file differ in some manner.
VIF_SRCOLD	The new file is older than the preexisting file.
VIF_DIFFLANG	The new file and the preexisting file specify different language or code page values.
VIF_DIFFCODEPG	The new file and the preexisting file specify different code page values.
VIF_DIFFTYPE	The new file has a different type, sub-type, or operating system identifier than the preexisting file.
VIF_WRITEPROT	The preexisting file has been write-protected.
VIF_FILEINUSE	The preexisting file is currently in use by Windows.
VIF_OUTOFSPACE	Insufficient disk space on the destination drive for the new file.
VIF_ACCESSVIOLATION	Installation of the new file failed due to an access violation.

VIF_SHARINGVIOLATION	Installation of the new file failed due to a sharing violation.
VIF_CANNOTCREATE	The temporary file could not be created.
VIF_CANNOTDELETE	The destination file could not be deleted.
VIF_CANNOTDELETECUR	The destination file exists in a different directory, VIFF_DONTDELETEOLD was not specified, and the existing version of the file could not be deleted.
VIF_CANNOTRENAME	The preexisting file was deleted, but the temporary file could not be renamed.
VIF_OUTOFMEMORY	There is not enough memory to complete the installation. This is generally caused by running out of memory while trying to decompress a file.
VIF_CANNOTREADSRC	The new file could not be read.
VIF_CANNOTREADDST	The preexisting file could not be read, therefore no version information could be obtained.
VIF_BUFFTOOSMALL	The temporary file name buffer is too small to contain the name of the temporary file.

Include File	winver.h
See Also	VerFindFile()
Example	Refer to Listing 29-1 in the introduction of this chapter for an example of this function.

VERQUERYVALUE WIN32S WINDOWS 95 WINDOWS NT

Description	VerQueryValue() retrieves specific information from the version information resource that was obtained using GetFileVersionInfo(). An application must include VERSION.LIB to reference this function.
Syntax	BOOL VerQueryValue(LPVOID lpVerInfo, LPTSTR lpszKey, LPVOID lpvPointerToData, LPUINT lpdwDataLength)
Parameters	
lpVerInfo	LPVOID: A pointer to the version information resource. This data is typically obtained by a call to GetFileVersionInfo().
lpszKey	LPTSTR: Identifies which version information value to return. Table 29-7 gives details.
lpvPointerToData	LPVOID: A pointer to a buffer where this function will place a pointer to the requested version information.
lpdwDataLength	LPUINT: A pointer to a UINT that this function will set to the length of the data pointed to by lpvPointerToData.

Table 29-7. Rules for the lpszKey Parameter of VerQueryValue()

String Form	Description
\	Specifies the fixed version information. Using this key returns a pointer to a VS_FIXEDFILEINFO structure within the version information resource. See the definition of the VS_FIXEDFILEINFO structure below.
\VarFileInfo\Translation	Specifies the language/character set translation table. A pointer to the translation table is returned. An application will use the information in this table to build the strings needed to access the language-specific information in the version information resource. The translation table consists of an array of two WORD entries. The first word in each entry is the language ID, and the second word in each entry is the character set.
\StringFileInfo\<LanguageAndCharacter set>\<String>	Specifies an entry in the language-specific section of the version information resource. The <LanguageAndCharacterSet> string should be an Ascii string specifying the language ID and character set ID obtained from the language/character set translation table. This must be specified as a hexadecimal string. The <String> string specifies which string to retrieve. The following pre-defined strings may be used: CompanyName, FileDescription, FileVersion, InternalName, LegalCopyright, OriginalFilename, ProductName, ProductVersion. As an example, the following string would retrieve the "Company Name" from language ID 1033 character set 1252: "\\StringFileInfo\\040904E4\\CompanyName".

Returns	BOOL: The return value will be TRUE if this function is successful; otherwise, the return value will be FALSE.
Include File	winver.h
See Also	GetFileVersionInfo()

VS_FIXEDFILEINFO Definition

```
typedef struct _VS_FIXEDFILEINFO
{
    DWORD dwSignature;
    DWORD dwStrucVersion;
    DWORD dwFileVersionMS;
    DWORD dwFileVersionLS;
    DWORD dwProductVersionMS;
    DWORD dwProductVersionLS;
    DWORD dwFileFlagsMask;
    DWORD dwFileFlags;
    DWORD dwFileOS;
    DWORD dwFileType;
    DWORD dwFileSubtype;
    DWORD dwFileDateMS;
    DWORD dwFileDateLS;
} VS_FIXEDFILEINFO;
```

dwSignature DWORD: This member contains the value 0xFEEFO4BD. This is used searching a file for the VS_FIXEDFILEINFO structure.

dwStrucVersion DWORD: The binary version number of this structure. The high-order word of this member contains the major version number, and the low-order word contains the minor version number. This value must be greater than 0x00000029.

dwFileVersionMS DWORD: The most significant 32 bits of the file's binary version number. This member is used with *dwFileVersionLS* to form a 64-bit value used for numeric comparisons.

dwFileVersionLS DWORD: The least significant 32 bits of the file's binary version number. This member is used with *dwFileVersionMS* to form a 64-bit value used for numeric comparisons.

dwProductVersionMS

DWORD: The most significant 32 bits of the binary version number of the product with which this file was distributed. This member is used with *dwProductVersionLS* to form a 64-bit value used for numeric comparisons.

dwProductVersionLS

DWORD: The least significant 32 bits of the binary version number of the product with which this file was distributed. This member is used with *dwProductVersionMS* to form a 64-bit value used for numeric comparisons.

dwFileFlagsMask DWORD: A bitmask that specifies the valid bits in *dwFileFlags* member. A bit is set only if its corresponding value was defined when the file was created.

dwFileFlags DWORD: A bitmask that specifies the Boolean attributes of the file. This member can include one or more of the values listed in Table 29-8.

Table 29-8. VS_FIXEDFILEINFO dwFileFlags Values

Flag	Description
VS_FF_DEBUG	The file contains debugging information or is compiled with debugging features enabled.
VS_FF_INFOINFERRED	The file's version structure was created dynamically. Some of the members in this structure may be empty or incorrect. This flag should never be set in a file's VS_VERSION_INFO data.
VS_FF_PATCHED	The file has been modified and is not identical to the original shipping file of the same version number.
VS_FF_PRERELEASE	The file is a development version, not a commercially released product.
VS_FF_PRIVATEBUILD	The file was not built using standard release procedures. If this flag is set, StringFileInfo should contain a PrivateBuild entry.

VS_FF_SPECIALBUILD	The file was built by the original company using standard release procedures but is a variation of the normal file of the same version number. If this flag is set, StringFileInfo should contain a SpecialBuild entry.

dwFileOS DWORD: The operating system for which this file was designed. This member can be one of the values listed in Table 29-9.

Table 29-9. VS_FIXEDFILEINFO dwFileOS Values

Value	Description
VOS_DOS	The file was designed for MS-DOS.
VOS_DOS_WINDOWS16	The file was designed for 16-bit Windows running on MS-DOS.
VOS_DOS_WINDOWS32	The file was designed for Win32 API running on MS-DOS.
VOS_OS216	The file was designed for 16-bit OS/2.
VOS_OS216_PM16	The file was designed for 16-bit Presentation Manager running on 16-bit OS/2.
VOS_OS232	The file was designed for 32-bit OS/2.
VOS_OS232_PM32	The file was designed for 32-bit Presentation Manager running on 32-bit OS/2.
VOS_PM16	The file was designed for 16-bit Presentation Manager.
VOS_PM32	The file was designed for 32-bit Presentation Manager.
VOS_NT	The file was designed for Windows NT.
VOS_NT_WINDOWS32	The file was designed for the Win32 API running on Windows NT.
VOS_UNKNOWN	The operating system for which the file was designed is unknown to Windows.
VOS_WINDOWS16	The file was designed for 16-bit Windows.
VOS_WINDOWS32	The file was designed for the Win32 API.

dwFileType DWORD: The general type of file. This member can be one of the values listed in Table 29-10. Other values may be defined an not listed in this table because they are reserved for future use by Microsoft.

Table 29-10. VS_FIXEDFILEINFO dwFileType Values

Value	Description
VFT_APP	The file contains an application.
VFT_DLL	The file contains a dynamic-link library (DLL).
VFT_DRV	The file contains a device driver. If *dwFileType* is VFT_DRV, *dwFileSubtype* contains a more specific description of the driver.
VFT_FONT	The file contains a font. If *dwFileType* is VFT_FONT, *dwFileSubtype* contains a more specific description of the font file.
VFT_STATIC_LIB	The file contains a static-link library.
VFT_UNKNOWN	The file type is unknown to Windows.
VFT_VXD	The file contains a virtual device.

dwFileSubtype DWORD: The specific function of the file. Normally this member is set to zero except for values of *dwFileType* VFT_DRV, VFT_FONT, and VFT_VXD. If *dwFileType* is VFT_DRV, *dwFileSubtype* can be one of the values listed in Table 29-11. If *dwFileType* is VFT_FONT, *dwFileSubtype* can be one of the values listed in Table 29-12. If *dwFileType* is VFT_VXD, *dwFileSubtype* contains the virtual device identifier included in the virtual device control block. All *dwFileSubtype* values not listed here are reserved for future use by Microsoft.

Table 29-11. VS_FIXEDFILEINFO dwFileSubtype Values for VFT_DRV Types

Value	Description
VFT2_UNKNOWN	The driver type is unknown by Windows.
VFT2_DRV_PRINTER	The file contains a printer driver.
VFT2_DRV_KEYBOARD	The file contains a keyboard driver.

VFT2_DRV_LANGUAGE	The file contains a language driver.
VFT2_DRV_DISPLAY	The file contains a display driver.
VFT2_DRV_MOUSE	The file contains a mouse driver.
VFT2_DRV_NETWORK	The file contains a network driver.
VFT2_DRV_SYSTEM	The file contains a system driver.
VFT2_DRV_INSTALLABLE	The file contains an installable driver.
VFT2_DRV_SOUND	The file contains a sound driver.

Table 29-12. VS_FIXEDFILEINFO dwFileSubtype Values for VFT_FONT Types

Value	Description
VFT2_UNKNOWN	The font type is unknown by Windows.
VFT2_FONT_RASTER	The file contains a raster font.
VFT2_FONT_VECTOR	The file contains a vector font.
VFT2_FONT_TRUETYPE	The file contains a TrueType font.

dwFileDateMS DWORD: The most significant 32 bits of the file's creation date and time stamp. This member is used with *dwFileDataLS* member to form a 64-bit binary value.

dwFileDateLS DWORD: The least significant 32 bits of the file's creation date and time stamp. This member is used with *dwFileDataMS* member to form a 64-bit binary value.

Example See GetFileVersionInfo() for an example of this function.