Advanced Disk Help Table of Contents

How To ...

The Advanced Disk Library provides Microsoft Visual Basic Applications with the ability to obtain System Disk/File Information. The following Command List details the items available in the **AdvDisk.DLL**. A Visual Basic Code Sample, titled **ADVAPP.MAK,** is also supplied to highlight the use of AdvDisk.

The DLL File **ADVDISK.DLL** is installed in the windows system directory. To un-install the ADVAPP, delete the ADVDISK.DLL in the windows system directory and delete the Visual Basic AdvApp Code installation directory, normally the code example will default to C:\ADVAPP.

Commands

Version

ADVDISK.DLL Version 3.1

Includes

ADVDISK.DLL,	System Information Library
ADVAPP.MAK	ADVDISK.DLL Visual Basic Examples
ADVDISK.HLP	Help File

License

See the License Agreement License, see Registration form, print the form and send it in., see Ordering. which covers Payment options and other information.

Contacts

Advanced Applications	Compuserve	72713,2106
	Internet	72713,2106@compuserve.com
6700 North Tryon Street	Voice	(704) 597-3948
Box 560991.		
Charlotte, NC 28256-0991		
USA		

Copyright notice

owners.

Drive Check command

The **DriveCheck** command determines whether a file or a directory exists and in the case of a file, its mode ability.

Input Value

The string variable determines either a file or a directory which is to be checked.

File	**C:\AUTOEXEC.BAT**
Directory	**C:\DOS**

The possible mode values and their meanings in the **DriveCheck** call are as follows:

Value	**Meaning**
0	Check for existence only.
2	Check for write permission.
4	Check for read permission.
6	Check for read and write permission.

With directories, **DriveCheck** determines only whether the specified directory exists, all directories have read and write mode ability.

Return Value

Value	**Meaning**
0	File has the given mode; Directory does exist.
1	File does not exist or is not accessible in the given mode; Directory does not exist.

Syntax

```
bDriveCheck = DriveCheck(strDrive$, nFileMode%)
```

Declaration

```
Declare Function DriveCheck Lib "advdisk.dll" (ByVal strDriveInfo$, ByVal nCheck%) As Integer
```

Constants

```
Global Const FILE_EXISTS = 0
Global Const FILE_NOT_EXIST = 1
```

Example

```
strDrive$ = C:\AUTOEXEC.BAT
nFileMode% = 0

bDriveCheck% = DriveCheck(strDrive$, nFileMode%)

If bDriveCheck% = FILE_EXISTS Then
    strMessage$ = "Directory/File Exists"
Else
    strMessage$ = "Directory/File Does Not Exist"
End If

MsgBox strMessage$
```

Drive Type command

DriveType augments the Microsoft Windows 16-bit API GetDriveType which presently determines if a drive is a floppy disk, network drive, and a hard disk. Advdisks **DriveType** adds checking for a CD-ROM, a RAM-drive, as well as determining whether the specified drive is compressed or not.

Windows 32-bit API GetDriveType still does not determine whether the specified drive is compressed, but **DriveType** does.

The DriveNumber value specifies the drive for which the drive type is to be determined:

Input Value

Value	Meaning
0	Drive A
1	Drive B
2	Drive C
	(and so on)
25	Drive Z

Return Value

Value	Meaning
-1	Drive requested is not valid.
0	Undetermined Drive.
2	Drive is removable, i.e., floppy Drive.
3	Drive is a fixed Drive.
4	Drive is a remote Drive , i.e., network Drive.
5	Drive is a CD ROM Drive.
6	Drive is a RAM Drive.
1x	Drive is a Compressed Drive, example: 13 = Fixed Drive is Compressed 03 = Fixed Drive is not Compressed

Syntax

```
nDrive% = DriveType(I%)
```

Declaration

```
Declare Function DriveType Lib "advdisk.dll" (ByVal nDrive%) As Integer
```

Constants

```
Global Const DEFAULT_DRIVE = 0
Global Const DRIVE_NOT_VALID = -1
Global Const DRIVE_UNDETERMINED = 0
Global Const DRIVE_REMOVABLE = 2
Global Const DRIVE_FIXED = 3
Global Const DRIVE_REMOTE = 4
Global Const DRIVE_CDROM = 5
Global Const DRIVE_RAM = 6
```

Example

```
' Find the system drives and fill the Drive Combo Box
' Starting with "Drive A" to "Drive Z".
cboDrive.Clear

For I% = 0 To 25
```

```
    nDrive% = DriveType(I%)

    If nDrive% <> DRIVE_UNDETERMINED And nDrive% <> DRIVE_NOT_VALID Then
        If nDrive% < 10 Then
            nCompress% = nDrive%
        Else
        nCompress% = nDrive% - 10
        End If

        Select Case nCompress%
            Case DRIVE_REMOVABLE
                strDrive$ = "[Floppy"
            Case DRIVE_FIXED
                strDrive$ = "[Fixed"
            Case DRIVE_REMOTE
                strDrive$ = "[Network"
            Case DRIVE_CDROM
                strDrive$ = "[CDRom"
            Case DRIVE_RAM
                strDrive$ = "[RAM"
            Case Else
                strDrive$ = ""
        End Select

        If nDrive% > 10 Then
            strDrive$ = strDrive$ & " (Compressed)]"
        Else
            strDrive$ = strDrive$ & "]"
        End If

        strTemp$ = Chr$(65 + I%) & ": "
        strTemp$ = strTemp$ & strDrive$
        cboDrive.AddItem strTemp$
    End If
Next
```

FileExists command

The **FileExists** command determines whether a file or a directory exists.

The string variable determines either a file or a directory is to be checked. This command is a shortcut version of the Drive Type, all that is necessary is to provide the string containing the file or path.

This command will return information about hidden and system files, files normally not displayed using the normal DOS DIR (Directory) command.

Input Value

Value	**Meaning**
C:\IO.SYS	A normally hidden DOS IO System File. Note: IO.SYS is a Microsoft system file. This file may not be available on systems using IBMs PCDOS.
C:\CONFIG.SYS	The system DOS configuration file.

Return Value

Value	**Meaning**
0	File does exist; Directory does exist.
1	File does not exist; Directory does not exist.

Syntax

```
nReturn% = FileExists(strFile$)
```

Declaration

```
Declare Function FileExists Lib "advdisk.dll" (ByVal strFile$) As Integer
```

Constants

```
Global Const FILE_EXISTS = 0
Global Const FILE_NOT_EXIST = 1
```

Example

```
strFrom$ = C:\CONFIG.SYS

nReturn% = FileExists(strFrom$)

If nReturn% = FILE_EXISTS Then
    MsgBox "File Exists"
Else
    MsgBox "File Does Not Exist"
End If
```

DiskFreeSpace command

The **DiskFreeSpace** command determines the amount of free space left on a selected disk drive. The default Drive is 0 (zero), Drive A is 1, Drive B is 2, and so on.

Input Value

Value	Meaning
0	Default Drive
1	Drive A
2	Drive B
3	Drive C
	(and so on)
26	Drive Z

Return Value

Value	Meaning
lDriveInfo&	Disk Free Space, (long data type).
< 40	Error Codes, see . Disk Errors

Syntax

```
lDriveInfo& = DiskFreeSpace(nDrive%)
```

Declaration

```
Declare Function DiskFreeSpace Lib "advdisk.dll" (ByVal nDrive%) As Long
```

Constants

See Disk Errors Constants.

Example

```
nDrive% = 0              Test the DEFAULT Drive.
lDriveInfo& = DiskFreeSpace(nDrive%)

' Did an error occur?
If lDriveInfo& <= DISK_FULL And lDriveInfo& >= FILE_NOT_FOUND Then
    DisplayError lDriveInfo&
    Exit Sub
End If

' Display the Information
MsgBox Bytes   &   lDriveInfo&
```

Disk Total Space command

The **DiskTotalSpace** command determines the total amount of space on a selected disk drive. The default Drive is 0 (zero), Drive A is 1, Drive B is 2, and so on.

Input Value

Value	**Meaning**
0	Default Drive
1	Drive A
2	Drive B
3	Drive C
	(and so on)
26	Drive Z

Return Value

Value	**Meaning**
lDriveInfo &	Disk Free Space.
< 40	Error Code, see Disk Errors

Syntax

```
lDriveInfo& = DiskTotalSpace(nDrive%)
```

Declaration

```
Declare Function DiskTotalSpace Lib "advdisk.dll" (ByVal nDrive%) As Long
```

Constants

See Disk Errors Constants.

Example

```
nDrive% = 0              Test the DEFAULT Drive.
lDriveInfo& = DiskTotalSpace(nDrive%)

' Did an error occur?
If lDriveInfo& <= DISK_FULL And lDriveInfo& >= FILE_NOT_FOUND Then
    DisplayError lDriveInfo&
    Exit Sub
End If

' Display the Information
MsgBox Bytes   &   lDriveInfo&
```

Disk Errors and Disk Status

The following error and status values are constants assigned to return events of various error and status conditions. The following values may be encountered while finding the disk information commands, Disk Free Space and Disk Total Space, also see General Error Procedure**.**

Return Value

Value	**Meaning**
-1	General File Error.
2	File was not found.
3	Directory was not found.
7	Argument list is too long.
8	Exec format error.
9	Bad File Number
12	Not enough memory.
13	Permission denied.
17	Drive exists.
18	Cross Device Link.
22	Invalid argument.
24	Too many files open.
28	No space left on device.
33	Math argument.
34	Result too large.
36	Resource deadlock would occur.
39	Disk Full

Constants

```
Global Const ARGUMENT_LIST_TOO_LONG = 7
Global Const PERMISSION_DENIED = 13
Global Const BAD_FILE_NUMBER = 9
Global Const RESOURCE_DEADLOCK_WOULD_OCCUR = 36
Global Const MATH_ARGUMENT = 33
Global Const DRIVE_EXISTS = 17
Global Const INVALID_ARGUMENT = 22
Global Const TOO_MANY_OPEN_FILES = 24
Global Const FILE_NOT_FOUND = 2
Global Const DIRECTORY_NOT_FOUND = 3
Global Const EXEC_FORMAT_ERROR = 8
Global Const NOT_ENOUGH_MEMORY = 12
Global Const NO_SPACE_LEFT_ON_DEVICE = 28
Global Const RESULT_TOO_LARGE = 34
Global Const CROSS_DEVICE_LINK = 18
Global Const DISK_FULL = 39
Global Const FILE_ERROR = -1
```

Disk Display Procedure

The following Disk Error Example will handle all general and specific errors as encountered by ADVDISK.

Example

```
Sub DisplayError (lError As Long)
    ' Display the error
    Select Case nError
        Case ARGUMENT_LIST_TOO_LONG
            lError = "Argument list too long"
        Case PERMISSION_DENIED
            lError = "Permission denied"
        Case BAD_FILE_NUMBER
            lError = "Bad file number"
        Case RESOURCE_DEADLOCK_WOULD_OCCUR
            lError = "Resource deadlock would occur"
        Case MATH_ARGUMENT
            lError = "Math argument"
        Case FILE_EXISTS
            lError = "File exists"
        Case INVALID_ARGUMENT
            lError = "Invalid argument"
        Case TOO_MANY_OPEN_FILES
            lError = "Too many open files"
        Case FILE_NOT_FOUND
            lError = "No such file"
        Case DIRECTORY_NOT_FOUND
            lError = "No such directory"
        Case EXEC_FORMAT_ERROR
            lError = "Exec format error"
        Case NOT_ENOUGH_MEMORY
            lError = "Not enough memory"
        Case NO_SPACE_LEFT_ON_DEVICE
            lError = "No space left on device"
        Case RESULT_TOO_LARGE
            lError = "Result too large"
        Case CROSS_DEVICE_LINK
            lError = "Cross-device link"
        Case DISK_FULL
            lError = "Disk Full"
        Case FILE_ERROR
            lError = "File Error"
        Case MOVE_TO_NOT_OPENED
            lError = "(Move) To File could not be opened"
        Case MOVE_TO_NOT_CLOSED
            lError = "(Move) To File could not be closed"
        Case MOVE_FROM_NOT_CLOSED
            lError = "(Move) From File could not be closed"
        Case MOVE_FROM_NOT_REMOVED
            lError= "(Move) From File could not be Deleted"
        Case MOVE_FROM_NOT_EXIST
            lError= "(Move) From File does not exist"
        Case MOVE_TO_DISK_FULL
            lError= "(Move) To Drive has Full Disk"
        Case MOVE_TO_DIR_CREATE_ERROR
            lError= "(Move) To Directory Creation Error"
        Case MOVE_TO_DO_NOT_CREATE_DIR
            lError= "(Move) Did not Create Directory, as requested"
        Case MOVE_DO_NOT_OVER_WRITE
            lError= "(Move) As Requested, did not overright existing file"
        Case MOVE_NEWER_REVISION
            lError= (Move) Did not over write newer revision
        Case Else
            lError= "Unknown Error"
    End Select

    MsgBox "Disk Information Error (" & lError &").", MB_ICONSTOP, "Disk Error"
End Sub
```

Move Copy File command

The **MoveCopyFile** command either moves or copies a selected file from one location to another location. The following feature are provided:

The Delete parameter of the **MoveCopyFile** turns the command into a copy command.
File Revision Checking, by date comparison.
Decide if Over Writing the existing file is appropriate or not.
Create the directory if they do not exist or if the directories do not exist, do not move/copy the file.
Check the destination for available disk space.

Input Values

Value	Meaning
strFrom$	String value of the From File..
strTo$	String value of the To File..
nDirectoryCreate	Create the Directory if it does not exist.. 0 = Create directory. 1 = Do not create directory.
nDelete	Delete the From File. 0 = Delete file. 1 = Do not delete file.
nOverWrite	If the To File exists, overwrite it.. 0 = Do not overwrite the To file. 1 = Overwrite the To file.
nNewerRevision	If the To File exists, compare the To files Date to the From files date; 0 = Do not overwrite the To file. 1 = Overwrite the To file.

Return Value

Value	Meaning
0	No Errors.
22	Disk requires formatting
100	The move To File was not opened.
101	The move To File was not closed.
102	The move From File was not closed.
103	The move From File was not deleted.
104	The move From File does not exist.
105	The move To disk drive is full.
106	The move To directory could not be created.
107	The move To directory did not exist, and the directory was not created, as requested.
108	The move To file was not overwritten, because it already existed, as requested.
109	The move To file was not overwritten, because the To file is a newer revision..

Syntax

```
        lReturn& = MoveCopyFile(strFrom$, strTo$, nDirectoryCreate%, nDelete%, nOverWrite%,
nNewerRevision%)
```

Declaration

```
        Declare Function MoveCopyFile Lib "advdisk.dll" (ByVal strFrom$, ByVal strTo$, ByVal nCreate%,
ByVal nDelete%, ByVal nOverwrite%, ByVal nNewRevision%) As Long
```

Constants

```
Global Const MOVE_TO_NOT_OPENED = 100
Global Const MOVE_TO_NOT_CLOSED = 101
Global Const MOVE_FROM_NOT_CLOSED = 102
Global Const MOVE_FROM_NOT_REMOVED = 103
Global Const MOVE_FROM_NOT_EXIST = 104
Global Const MOVE_TO_DISK_FULL = 105
Global Const MOVE_TO_DIR_CREATE_ERROR = 106
Global Const MOVE_TO_DO_NOT_CREATE_DIR = 107
Global Const MOVE_DO_NOT_OVER_WRITE = 108
Global Const MOVE_NEWER_VERSION = 109
Global Const MOVE_DISK_UNFORMATTED = 22

Global Const CREATE_DIRECTORY = 0
Global Const NO_CREATE = 1
Global Const DELETE_FROM_FILE = 0
Global Const NO_DELETE = 1
Global Const OVERWRITE_EXISTING = 0
Global Const NO_OVERWRITE = 1
Global Const DO_NOT_OVERWRITE_NEWER_REVISION = 0
Global Const DO_OVERWRITE_NEWER_REVISION = 1
```

Example

```
Screen.MousePointer = HOURGLASS

strFrom$ = C:\TEST.BAT
strTo$ = C:\TEST\TEST1\TEST2\TEST1.BAT
nDirectoryCreate% = CREATE_DIRECTORY
nDelete% = NO_DELETE
nOverWrite% = OVERWRITE_EXISTING
nNewerRevision% = DO_OVERWRITE_NEWER_REVISION

lReturn& = MoveCopyFile(strFrom$, strTo$, nDirectoryCreate%, nDelete%, nOverWrite%,
nNewerRevision%)

Select Case lReturn&
        Case MOVE_DISK_UNFORMATTED
            lReturn& = DiskFormat()
        Case 0
            DisplayError lReturn&
    End Select
```

Note: In the above example, the **MoveCopyFil**e command will attempt to **Copy:**

1. The File C:\TEST.BAT from the Root Directory of Drive C,
2. creating each level of subdirectories starting with \TEST\ to \TEST2\ if the directories do not exist,
3. renaming the file to TEST1.BAT,
4. if the file already exists, the **FileTo** date is shall be checked to see if it is newer than the file **FileFrom** date. If the **FileTo** date is newer, the copy will not be completed. But on the other hand, if the **FileTo** date has an older date than the **FileFrom** date, then the copy will take place.

To change the above example to a **Move**, change the following variable and rerun the example:

nDelete% = DELETE_FROM_FILE.

Path Split

The **PathSplit** command breaks a full path into one of four components. The path argument points to a buffer which will receive the returned path component. This command is used to break up the path string instead of having to use Visual Basic For/Next search loops.

Input Values

Value	Meaning
strPath$	Full path, i.e., C:\TEST\TEST1\TEST.BAT
szBuffer$	A zero filled buffer, String(255, 0), which will hold the return string.
nValue%	The value determines which of the four possible components that will be returned: 0 - Return the Drive, 1 - Return the Directory, 2 - Return the File Name, 3 - Return the File Extension.

Return Value

Value	Meaning
nLength%	The length of the szBuffer$ return by AdvDisk.dll
szBuffer$	Contains the returned component embedded within the zero filled buffer. Extracting the return component is accomplished by: strWord$ = Left$(szBuffer$, nLength%)
strWord$	The returned component determined by the input value: **Drive**: (if nValue% = 0) Contains the drive letter followed by a colon (:) if a drive is specified in path, i.e., C: **Directory**: (if nValue% = 1) Contains the path of subdirectories, if any, including the trailing slash. Forward slashes (/), backslashes), or both may be present in path, i.e., \TEST\TEST1\ **FileName**: (if nValue% = 2) Contains the base filename without any extensions, i.e., TEST **ext**: (if nValue% = 3) Contains the filename extension, if any, including the leading period (.), i.e., .BAT

Syntax

```
nLength% = PathSplit(strPath$, szBuffer$, nValue)
```

Declaration

```
Declare Function PathSplit Lib "advdisk.dll" (ByVal strPath$, ByVal strBuffer$, ByVal nValue%) As Integer
```

Constants

```
    Global Const DRIVE_COMPONENT = 0
    Global Const DIRECTORY_COMPONENT = 1
    Global Const FILENAME_COMPONENT = 2
    Global Const EXTENSION_COMPONENT = 3
```

Example

```
szBuffer$ = String(255, 0)
nValue% = DIRECTORY_COMPONENT

If txtPath.Text = "" Then Exit Sub
strPath$ = txtPath.Text

nLength% = PathSplit(strPath$, szBuffer$, nValue%)

If nLength% = 0 Then
    MsgBox "Cannot Split request", MB_ICONSTOP, "Split Return Error"
    Exit Sub
End If

strWord$ = Left$(szBuffer$, nLength%)
MsgBox Directory is   & strWord$
```

Create Path

The **CreatePath** command creates a new multi-level directory with the specified Directory Name. With Visual Basic, only one directory can be created at a time, but **CreatePath** creates according to the following parameters.

The following is the max multi-level subdirectory which can be created at one time, the max character length is 65 characters. This limit is imposed by the operating system.

```
strPath$ = "c:\testing\test1\test2\test3\test4\test5\test6\test7\test8\test9\"
```

Input Values

Value	**Meaning**
strpath$	The Drive and path required to create, i.e., (either of the following examples are correct, i.e., with or without the trailing (\)): C:\TEST\TEST1\TEST2 C:\TEST\TEST1\TEST2\

Return Value

Value	**Meaning**
0	No Errors.
non zero	Error Code, see Disk Errors

Syntax

```
nReturn% = CreatePath(strPath$)
```

Declaration

```
Declare Function CreatePath Lib "advdisk.dll" (ByVal strPath$) As Integer
```

Constants

```
Global Const NO_ERRORS = 0
```

Example

```
strPath$ = C:\TESTING\TEST1\TEST2\TEST3\TEST4\TEST5\TEST6\TEST7\TEST8\TEST9\
nReturn% = CreatePath(strPath$)

If nReturn% = NO_ERRORS Then
    MsgBox "Directory was created", , Directory
Else
    MsgBox Directory Creation Error Code ( & nReturn% & ), MB_ICONSTOP, "Return Error"
End If
```

Note: If strPath$ had been equal to: C:\TESTING\TEST1\TEST2\TEST3\TEST4 and then you decided that you needed to add another level, all that is required to do is the add the necessary level with the following string value: C:\TESTING\TEST1\TEST2\TEST3\TEST4\TEST5.

Delete Path

The **DeletePath** command deletes a new multi-level directory with the specified Directory Name. With Visual Basic, only one directory can be deleted at a time, but **DeletePath** deletes according to the following parameters.

The following is the max multi-level subdirectory which can be created at one time, the max character length is 65 characters. This limit is imposed by the operating system.

```
strPath$ = "c:\testing\test1\test2\test3\test4\test5\test6\test7\test8\test9\"
```

Input Values

Value	Meaning
strpath$	The Drive and path required to deleted, i.e., (either of the following examples are correct, i.e., with or without the trailing (\)): C:\TEST\TEST1\TEST2 C:\TEST\TEST1\TEST2\

Return Value

Value	Meaning
0	No Errors.
non zero	Error Code, see Disk Errors

Syntax

```
nReturn% = DeletePath(strPath$)
```

Declaration

```
Declare Function DeletePath Lib "advdisk.dll" (ByVal strPath$) As Integer
```

Constants

```
Global Const NO_ERRORS = 0
```

Example

```
strPath$ = C:\TESTING\TEST1\TEST2\TEST3\TEST4\TEST5\TEST6\TEST7\TEST8\TEST9\
nReturn% = DeletePath(strPath$)

If nReturn% = NO_ERRORS Then
    MsgBox "Directory was Deleted", , Directory
Else
    MsgBox Directory Deletion Error Code ( & nReturn% & ), MB_ICONSTOP, "Return Error"
End If
```

Possible Exceptions

1.If while try to delete a level, another application such as File Manager is accessing one of the levels, you will receive an Access Denied Return Code.

2. One of the levels has two directory levels, i.e., level \TEST3\ has not only \TEST4\ but also \SPLITOFF\, you will then delete up to level \TEST4\ but all levels from the ROOT to \SPLITOFF\ will remain.

3. One or more of the levels contains files. The same rule applies here as applies in number two (2) above.

Search List

The **SearchList** command displays a inherited dialog box for the desired path, directory and file. When the user selects the desired file(s) and presses the OK button, these file(s) are returned to your Visual Basic Application.

Input Values

Value	**Meaning**
iAttr%	Search attribute: 0 = Normal file/directory search 1 = Read only and normal file search 2 = Hidden and normal file search 3 = System and normal file search 4 = Directory and normal file search 5 = Archived and normal file search 6 = Include drives with normal file search 7 = Search for only hidden and system files 8 = Search Read only files 9 = Search Hidden files only 10 = Search System files only 11 = Search Directories only 12 = Search Archived files only 13 = Search Drives only 14 = Search Hidden and System files only
iDelimited%	File seperator type: 0 = Space seperator 1 = Comma seperator 2 = Semi colon seperator 3 = Tab seperator 4 = Carriage Return seperator
strPath$	Search path, i.e., C:\*.*
szBuffer$	A zero filled buffer, String(255, 0), which will hold the return string.

Return Value

Value	**Meaning**
nLength%	The length of the szBuffer$ return by AdvDisk.dll, if nLength% = 0, then user did not select any files from the search list.
szBuffer$	Contains the returned component embedded within the zero filled buffer. Extracting the return component is accomplished by: strWord$ = Left$(szBuffer$, nLength%)

Syntax

```
nLength% = SearchList(strPath$, iAttr%, iDelimited%, szBuffer$)
```

Declaration

```
Declare Function SearchList Lib "advdisk.dll" (ByVal strPath$, ByVal iAttr%, ByVal iDelimited%, ByVal strBuffer$) As Integer
```

Constants

```
Global Const SEARCH_NORMAL = 0
Global Const SEARCH_READONLY = 1
Global Const SEARCH_HIDDEN = 2
Global Const SEARCH_SYSTEM = 3
Global Const SEARCH_DIRECTORY = 4
Global Const SEARCH_ARCHIVED = 5
Global Const SEARCH_INCLUDE_DRIVES = 6
Global Const SEARCH_HIDDEN_SYSTEM = 7
Global Const SEARCH_ONLY_READONLY = 8
Global Const SEARCH_ONLY_HIDDEN = 9
Global Const SEARCH_ONLY_SYSTEM = 10
Global Const SEARCH_ONLY_DIRECTORY = 11
Global Const SEARCH_ONLY_ARCHIVED = 12
Global Const SEARCH_ONLY_INCLUDE_DRIVES = 14
Global Const SEARCH_ONLY_HIDDEN_SYSTEM = 13

Global Const SEARCH_SPACE = 0
Global Const SEARCH_COMMA = 1
Global Const SEARCH_SEMI_COLON = 2
Global Const SEARCH_TAB = 3
Global Const SEARCH_CR = 4
```

Example

```
szBuffer$ = String(255, 0)

    strPath$ = "C:\*.*"
    iAttr% = SEARCH_NORMAL
    iDelimited% = SEARCH_SEMI_COLON

    nLength% = SearchList(strPath$, iAttr%, iDelimited%, szBuffer$)
    DoEvents

    If nLength% = 0 Then
        MsgBox "Search Item(s) not returned", , "Search List Return Error"
        Exit Sub
    End If

    strWord$ = Left$(szBuffer$, nLength%)
    MsgBox "Search Item Returned: " & strWord$, , "Search List Return Error"
    DoEvents
```

Disk Format

The **Disk Format** command allows the user to format either Drive A or Drive B from your Visual Basic code. This is accomplished by using Microsofts File Manager. You will notice that the File Manager main window is not displayed, only the format dialogs appear.

Input Values

None.

Return Value

Value	Meaning
-1	No Errors.
0 (or) non zero	Error. See Below Constants which details any errors

Syntax

```
nReturn% = DiskFormat()
```

Declaration

```
Declare Function DiskFormat Lib "advdisk.dll" () As Long
```

Constants

```
Global Const COMMAND_OK = -1      ' No Errors.
Global Const COMMAND_ERR00 = 0   ' System was out of memory, executable file was corrupt, or relocations were invalid.
Global Const COMMAND_ERR01 = 1   ' System command not available
Global Const COMMAND_ERR02 = 2   ' File was not found.
Global Const COMMAND_ERR03 = 3   ' Path was not found.
Global Const COMMAND_ERR05 = 5   ' Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
Global Const COMMAND_ERR06 = 6   ' Library required separate data segments for each task.
Global Const COMMAND_ERR08 = 8   ' There was insufficient memory to start the application.
Global Const COMMAND_ERR10 = 10 ' Windows version was incorrect.
Global Const COMMAND_ERR11 = 11 ' Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
Global Const COMMAND_ERR12 = 12 ' Application was designed for a different operating system.
Global Const COMMAND_ERR13 = 13 ' Application was designed for MS-DOS 4.0.
Global Const COMMAND_ERR14 = 14 ' Type of executable file was unknown.
Global Const COMMAND_ERR15 = 15 ' Attempt was made to load a real-mode application (developed for an earlier version of Windows).
Global Const COMMAND_ERR16 = 16 ' Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.
Global Const COMMAND_ERR19 = 19 ' Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
Global Const COMMAND_ERR20 = 20 ' Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
Global Const COMMAND_ERR21 = 21 ' Application requires Microsoft Windows 32-bit extensions.
```

Example

```
lReturn& = DiskFormat()
```

Disk Copy

The **Disk Copy** command allows the user to copy either Drive A or Drive B from your Visual Basic code. This is accomplished by using Microsofts File Manager. You will notice that the File Manager main window is not displayed, only the format dialogs appear.

Input Values

None.

Return Value

Value	Meaning
-1	No Errors.
0 (or) non zero	Error. See the below constants which denote any errors.

Syntax

```
nReturn% = DiskCopy()
```

Declaration

```
Declare Function DiskCopy Lib "advdisk.dll" () As Long
```

Constants

```
Global Const COMMAND_OK = -1      ' No Errors.
Global Const COMMAND_ERR00 = 0   ' System was out of memory, executable file was corrupt, or
relocations were invalid.
Global Const COMMAND_ERR01 = 1   ' System command not available
Global Const COMMAND_ERR02 = 2   ' File was not found.
Global Const COMMAND_ERR03 = 3   ' Path was not found.
Global Const COMMAND_ERR05 = 5   ' Attempt was made to dynamically link to a task, or there was a
sharing or network-protection error.
Global Const COMMAND_ERR06 = 6   ' Library required separate data segments for each task.
Global Const COMMAND_ERR08 = 8   ' There was insufficient memory to start the application.
Global Const COMMAND_ERR10 = 10 ' Windows version was incorrect.
Global Const COMMAND_ERR11 = 11 ' Executable file was invalid. Either it was not a Windows
application or there was an error in the .EXE image.
Global Const COMMAND_ERR12 = 12 ' Application was designed for a different operating system.
Global Const COMMAND_ERR13 = 13 ' Application was designed for MS-DOS 4.0.
Global Const COMMAND_ERR14 = 14 ' Type of executable file was unknown.
Global Const COMMAND_ERR15 = 15 ' Attempt was made to load a real-mode application (developed
for an earlier version of Windows).
Global Const COMMAND_ERR16 = 16 ' Attempt was made to load a second instance of an executable file
containing multiple data segments that were not marked read-only.
Global Const COMMAND_ERR19 = 19 ' Attempt was made to load a compressed executable file. The file
must be decompressed before it can be loaded.
Global Const COMMAND_ERR20 = 20 ' Dynamic-link library (DLL) file was invalid. One of the DLLs
required to run this application was corrupt.
Global Const COMMAND_ERR21 = 21 ' Application requires Microsoft Windows 32-bit extensions.
```

Example

```
lReturn& = DiskCopy()
```

SystemWindow

The **System Window** command allows the user to select one of the three following items:

1. To Reboot the complete system,
2. To Restart Windows, or
3. To goto a DOS Prompt.

Restarting Windows or Rebooting Windows can be very helpful when you have setup a new system and you are required to Reboot/Restart the system so that the new changes will take effect.

Input Values

Value	**Meaning**
0	Reboot computer system..
1	Restart Windows.
2	.Goto to a DOS Prompt.

Return Value

Value	**Meaning**
-1	No Errors.
0 (or) non zero	Error. See the below constants which denote any errors.

Syntax

lReturn& = SystemWindow(nType%)

Declaration

Declare Function SystemWindow Lib "advdisk.dll" (nType% As Integer) As Long

Constants

Global Const REBOOT = 0
Global Const RESTART = 1
Global Const PROMPT = 2

Global Const COMMAND_OK = -1 ' No Errors.
Global Const COMMAND_ERR00 = 0 ' System was out of memory, executable file was corrupt, or relocations were invalid.
Global Const COMMAND_ERR01 = 1 ' System command not available
Global Const COMMAND_ERR02 = 2 ' File was not found.
Global Const COMMAND_ERR03 = 3 ' Path was not found.
Global Const COMMAND_ERR05 = 5 ' Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
Global Const COMMAND_ERR06 = 6 ' Library required separate data segments for each task.
Global Const COMMAND_ERR08 = 8 ' There was insufficient memory to start the application.
Global Const COMMAND_ERR10 = 10 ' Windows version was incorrect.
Global Const COMMAND_ERR11 = 11 ' Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
Global Const COMMAND_ERR12 = 12 ' Application was designed for a different operating system.
Global Const COMMAND_ERR13 = 13 ' Application was designed for MS-DOS 4.0.
Global Const COMMAND_ERR14 = 14 ' Type of executable file was unknown.
Global Const COMMAND_ERR15 = 15 ' Attempt was made to load a real-mode application (developed for an earlier version of Windows).
Global Const COMMAND_ERR16 = 16 ' Attempt was made to load a second instance of an executable file

containing multiple data segments that were not marked read-only.
Global Const COMMAND_ERR19 = 19 ' Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
Global Const COMMAND_ERR20 = 20 ' Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
Global Const COMMAND_ERR21 = 21 ' Application requires Microsoft Windows 32-bit extensions.

Example

```
lReturn& = SystemWindow(RESTART)
```

No Help Available

No help is available for this area of the window.

Software Ordering

AdvDisk is distributed as for evaluation for thirty days (License). Please remember that using unlicensed shareware past the evaluation period is unethical and illegal. To register and receive the latest version of AdvDisk, please complete the order form and include **US$30.00**. Foreign orders add **US$5.00** for international postage and handling.

Any questions about the status of shipment, refunds, registration options, product details, technical support, volume discounts, dealer pricing, site licenses, etc., must be directed to Advanced Applications directly (see below). Multiple, network and site licenses are available. Contact Advanced Applications for terms and conditions. Prices subject to change without notice.

Contacts

Advanced Applications	Compuserve	72713,2106
6700 North Tryon Street	Voice	(704) 597-3948
Box 560991.		
Charlotte, NC 28256-0991		
USA		

Ordering Information

Select Registration form, and then after printing the form, send it in.

License Agreement

This software is licensed shareware. You may distribute at no charge, except for media or connect time charges, unmodified copies of the file **ADVDISK.DLL** and **ADVAPP.***. Select Registration form, and then after printing the form, send it in.

AdvDisk is supplied "as is". The author disclaims all warranties, expressed or implied, including, without limitation, the warranties of merchant ability and of fitness for any purpose. The author assumes no liability for damages, direct or consequential, which may result from the use of AdvDisk. You also agree to not hold Advanced Applications liable for any damages, direct or consequential which may result from the use of AdvDisk. By purchasing, using, distributing AdvDisk, you are agreeing to all of these conditions.

AdvDisk is provided free of charge for a thirty day evaluation period. Any use beyond this period requires that AdvDisk be registered with the author, Advanced Applications. This registration fee will license one copy of AdvDisk for use on a single computer. Multi-user, network and site licenses are available; contact Advanced Applications for terms.

As purchaser of this software, you are granted a royalty-free license to distribute executable files generated using the AdvDisk.dll as well as distributing the AdvDisk.dll provided you accept the conditions of the License Agreement. Remember, AdvDisk must be registered prior to distributing with your applications, those persons, companies, business, etc., will be fully prosecuted for distrubuting AdvDisk without proper registration.

Government users: This software and documentation are subject to restrictions set forth in **The Rights in Technical Data and Computer Software clause at 252.227-7013 and elsewhere.**

If you believe AdvDisk is valuable and useful, please give it to anyone else you think would be interested, and encourage them to register their copy. Copy registration also provides notification of Revision Updates, these upgrades will be supplied at a upgrade price.

Registration Form

To:	Advanced Applications 6700 North Tryon Street Box 560991 Charlotte, NC 28256-0991 Attn.: AdvDisk.DLL, Revision 2.0
Fm:	(Name) ______________________
	(Company) ______________________
	(Address) ______________________
	______________________
	(City, State) ______________________
	(Country) ______________________
	(ZIP/Post) ______________________
	(Phone) ______________________
	(Fax) ______________________
	(Software) **AdvDisk System Information**

Price	**$30.00**	
Copies	(Number of copies).	
Subtotal $	($30.00 x Copies).	
Tax	$	(6% North Carolina State).
Shipping	$ **5.00**	
Total	$	(Latest Release/updates).

(Do not write below this line)

Receive Date:	______________________
Serial Number:	______________________
Check/MO Number:	______________________
Notification Sent:	______________________
Sent by/Date:	______________________

Visual Basic Sample Code

This text discusses the supplied Visual Basic Code used to illustrate **AdvDisk.dll**. Each topic highlights each command as it is used in the sample **AdvApp.mak** application.

AdvApp.mak File Listing

Listing	**Description**
ADVAPP.FRM	Advanced Application Sample
C:\WINDOWS\SYSTEM\THREED.VBX	Visual Basic Control
ADVAPP.BAS	Basic Declaration and Variable File
FRMMOVE.FRM	Move File Form
FRMPATHW.FRM	Path Word Form
FRMCREAT.FRM	Create and Delete Path Form
ProjWinSize=87,84,248,215	Project Window
ProjWinShow=2	Project Window
IconForm="frmAdvancedApplications"	Project
Title="ADVAPP"	Project
ExeName="ADVDISK.EXE"	Project

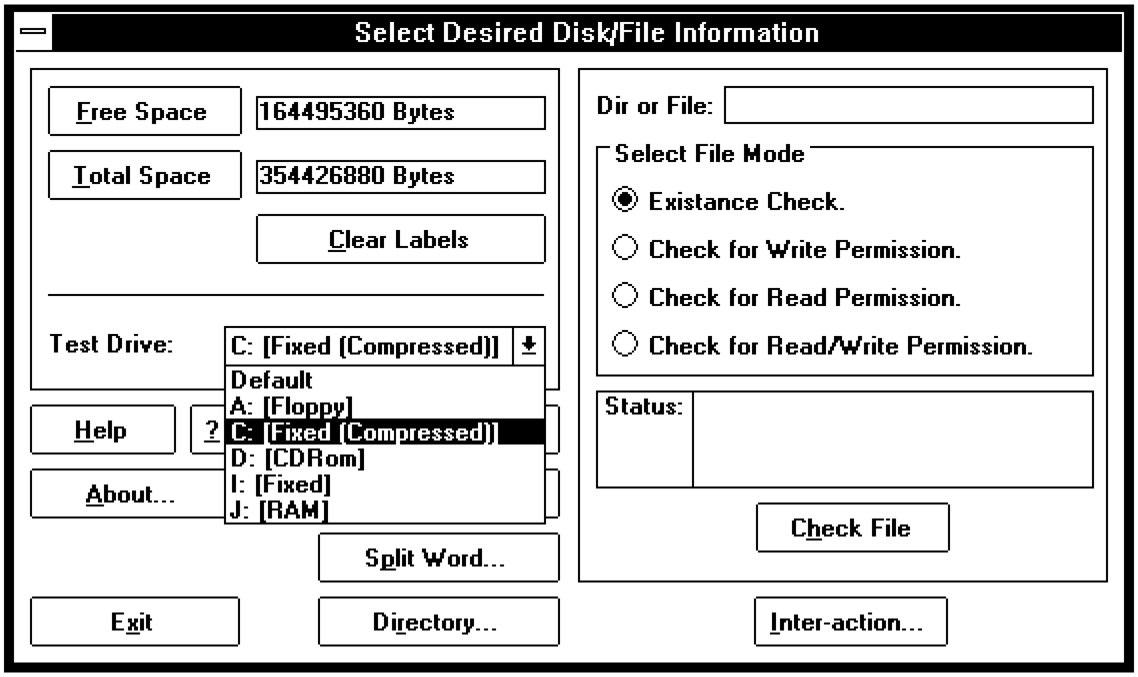
Sections

Declaration List
Constants
About Box

Test Drive

When the Visual Basic application loads, the TEST DRIVE combo box has tested all possible system drives and labeled the drives found with the drive type, see figure 1 below.   The system drives are tested for the following:

Floppy Drive,
Fixed Drive,
Network Drive,
CDROM Drive,
RAM Drive, and
if the drive is a Compressed Drive.



Free Space: 164495360 Bytes
Total Space: 354426880 Bytes
Test Drive: C: [Fixed (Compressed)]
Dir or File:
Select File Mode: [✓ Existance Check.] Check for Write Permission. Check for Read Permission. Check for Read/Write Permission.
Status:

Figure 1 - Application Load.

The code which tests and fills the TEST DRIVE combo box is:

```
' Find the system drives and fill the Drive Combo Box
' Starting with "Drive A" to "Drive Z".
cboDrive.Clear
cboDrive.AddItem "Default"

For I% = 0 To 25                      0 = Drive A, 1 = Drive B, and so on
    nDrive% = DriveType(I%)                   The AdvDisk.dll call

If nDrive% <> DRIVE_UNDETERMINED And nDrive% <> DRIVE_NOT_VALID Then
        If nDrive% < 10 Then
            nCompress% = nDrive%
        Else
        nCompress% = nDrive% - 10
        End If

        Select Case nCompress%
            Case DRIVE_REMOVABLE
                strDrive$ = "[Floppy"
            Case DRIVE_FIXED
                strDrive$ = "[Fixed"
            Case DRIVE_REMOTE
                strDrive$ = "[Network"
            Case DRIVE_CDROM
                strDrive$ = "[CDRom"
            Case DRIVE_RAM
                strDrive$ = "[RAM"
            Case Else
                strDrive$ = ""
        End Select
```

```
            If nDrive% > 10 Then
                strDrive$ = strDrive$ & " (Compressed)]"
            Else
                strDrive$ = strDrive$ & "]"
            End If

            strTemp$ = Chr$(65 + I%) & ": "
            strTemp$ = strTemp$ & strDrive$
            cboDrive.AddItem strTemp$
        End If
    Next
```

Disk Free and Disk Total Space

To test the FREE SPACE and/or TOTAL SPACE of a desired drive, first select the drive from the TEST DRIVE combo box. Then press either the FREE SPACE or the TOTAL SPACE buttons. The labels on the right of each button will display in bytes the drive space, see figure 2 below.

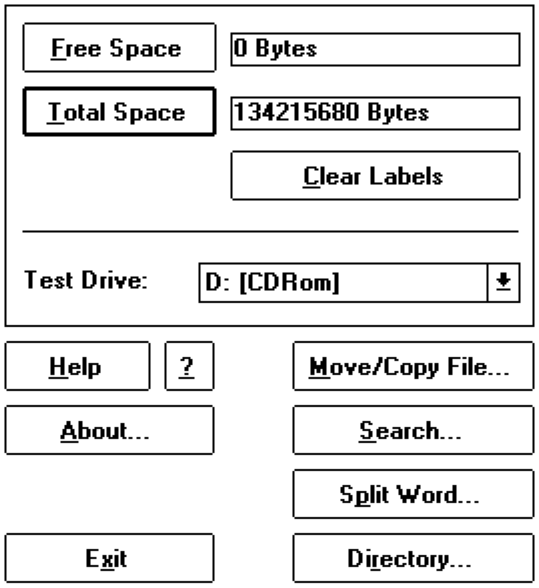


Figure 2 - Disk Free and Total Space

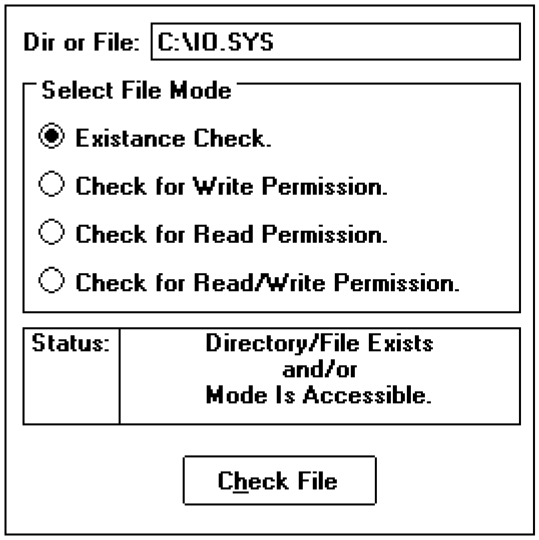
In the example case in Figure 2, The test drive is Drive C or DEFAULT and the FREE SPACE equals 164 megs and the TOTAL SPACE is 355 megs. Pressing the CLEAR LABELS will clear both space labels. Below lists the code to perform the drive space tests.

```
    Sub cmdDiskSpace_Click (Index As Integer)
        If cboDrive.Text = "Default" Then
            nDrive% = 0
        Else
            strDrive$ = Left(cboDrive.Text, 1)
            nDrive% = Asc(strDrive$) - 64
        End If

        Select Case Index
            Case FREE_SPACE
                lblFreeSpace.Caption = ""
                DoEvents
                Screen.MousePointer = HOURGLASS

               ' Find the Free Space for the Selected Drive
                '
                lDriveInfo = DiskFreeSpace(nDrive%)

                Screen.MousePointer = DEFAULT

                ' Did an error occur?
                If lDriveInfo <= DISK_FULL And lDriveInfo >= FILE_NOT_FOUND Then
                    DisplayError lDriveInfo
                    Exit Sub
                End If

                ' Display the Information
                lblFreeSpace.Caption = lDriveInfo & " Bytes"

            Case TOTAL_SPACE
                lblTotalSpace.Caption = ""
                DoEvents

                Screen.MousePointer = HOURGLASS

                ' Find the Total Space for the Selected Drive
                '
                lDriveInfo = DiskTotalSpace(nDrive%)

                Screen.MousePointer = DEFAULT

                ' Did an error occur?
                If lDriveInfo <= DISK_FULL And lDriveInfo >= FILE_NOT_FOUND Then
                    DisplayError lDriveInfo
                    Exit Sub
                End If

                ' Display the information
                lblTotalSpace.Caption = lDriveInfo & " Bytes"
        End Select
    End Sub
```

File Existance

The next step of the example illustrates how to check for the existence of a file. With this test, you can test for the following items:

The Existance of a Directory or a File.
Check for the read and write capabilities of a file.
Directories always have read/write capabilities.


Dir or File: C:\IO.SYS

Select File Mode

[✓ Existance Check.]

Check for Write Permission.

Check for Read Permission.

Check for Read/Write Permission.

Status: Directory/File Exists and/or Mode Is Accessible.

Figure 3 - File Existance

In the case above, the system was checked for the existance of the IO.SYS file. This is a hidden file. The EXISTANCE CHECK shows that the file exists. If your computer has this file as part of its system, you will find that EXISTANCE CHECK and CHECK FOR READ PERMISSION will show that the file exists and is accessible. But checking for CHECK FOR WRITE PERMISSION and CHECK FOR READ/WRITE PERMISSION will show that the file is not accessible.

The following code illustrates how this checking is accomplished.

```
Sub cmdCheckFile_Click ()
    strDrive$ = Trim(txtFile.Text)         ' Must be a String

     ' Check the Drive.
     bDriveCheck = DriveCheck(strDrive$, nFileMode)

    If bDriveCheck = FILE_EXISTS Then
        strMessage$ = "Directory/File Exists"
        strMessage$ = strMessage$ & Chr$(KEY_RETURN)
        strMessage$ = strMessage$ & "and/or"
        strMessage$ = strMessage$ & Chr$(KEY_RETURN)
        strMessage$ = strMessage$ & "Mode Is Accessible."
    Else
        strMessage$ = "Directory/File Does Not Exist"
        strMessage$ = strMessage$ & Chr$(KEY_RETURN)
        strMessage$ = strMessage$ & "and/or"
        strMessage$ = strMessage$ & Chr$(KEY_RETURN)
        strMessage$ = strMessage$ & "Mode Is Not Accessible."
    End If

    ' Display the message
    lblFileMode.Caption = strMessage$
End Sub
```

Move/Copy File - Copy Command

Copying files from one place to another requires alot of checking and takes alot of code and time to accomplish. The same is necessary for moving files. AdvDisk.dll help to accomplish both of these necessary items but also offers the following checks:

File Revision Checking, by date comparison. You can perform a copy/move but instruct AdvDisk to check the file date and halt the copy/move if the To date is newer than the From date. In other cases, you may want the To File to be overwritten no matter what.

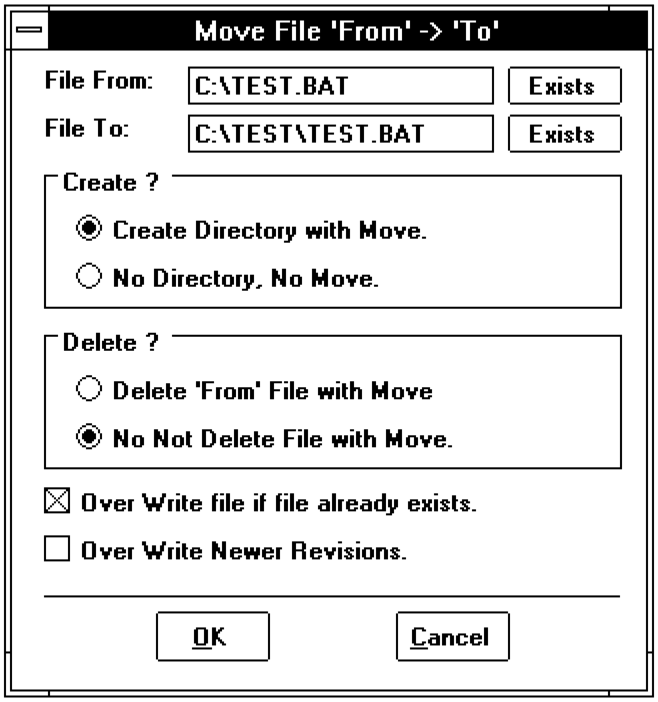
Decide if **Over Writing the existing file** is appropriate or not. You may want AdvDisk to overwrite the To file if it exists, no matter what. The only item that will override this is the File Revision Checking command.

Create the directory if they do not exist or if the directories do not exist, do not move/copy the file. This feature will create the requested directory, subdirectory, to whatever level is required if you instruct AdvDisk to do so. Or you may want AdvDisk to abort the move/copy if the To directory does not exist.

Check for available disk space of the destination. AdvDisk will not attempt to move/copy a file if there is not enough available disk space. AdvDisk will also return a code informing you if this condition exists.

Move/Copy File - Copy Command

The MOVE dialog box in Figure 4 is followed by the code to perform the AdvDisk command. In the below Figure, we will create the directories if they do not exist, rename AUTOEXEC.BAT to TEST.BAT. and we will delete the From file when the move is performed (in this case this is not a good idea), and we will over write the file if it exists, but if the file does in fact exist, we will not overwrite the file if the To file is a newer revision (has a later date).


Move File 'From' -> 'To'

File From: C:\TEST.BAT

File To: C:\TEST\TEST.BAT

Create ?

Create Directory with Move. [✓]

No Directory, No Move.

Delete ?

Delete 'From' File with Move

No Not Delete File with Move. [✓]

☒ Over Write file if file already exists.

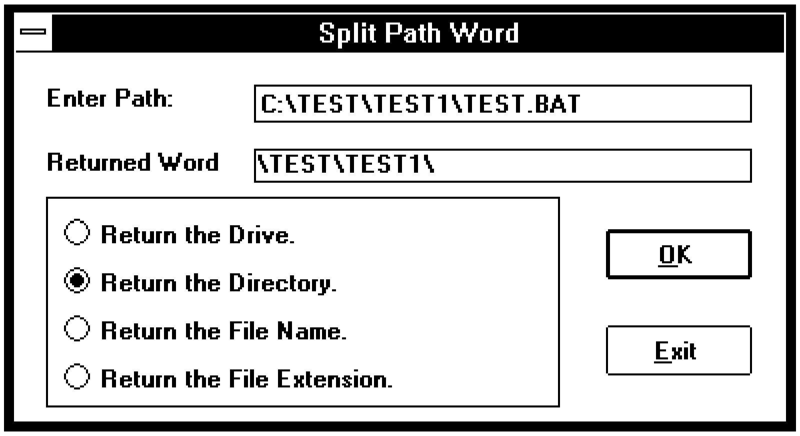
☐ Over Write Newer Revisions.

Figure 4 - The Move Dialog

```
Sub cmdOK_Click ()
    If txtFrom.Text = "" Or txtTo.Text = "" Then Exit Sub

    strFrom$ = txtFrom.Text
    strTo$ = txtTo.Text

    Screen.MousePointer = HOURGLASS
    lReturn& = MoveCopyFile(strFrom$, strTo$, nDirectoryCreate, nDelete, nOverWrite,
nNewerRevision)

Select Case lReturn&
        Case MOVE_DISK_UNFORMATTED
            lReturn& = DiskFormat()
        Case 0
            DisplayError lReturn&
    End Select

    Screen.MousePointer = DEFAULT
    Me.Hide
    DoEvents
End Sub
```

Split Path

One of the problems with Microsoft Visual Basic the inability to split the path word up into segments.. Presently, in order to parse off a path string, you must use some kind of a search looping method. This takes time to develop, test and debug each time it is necessary to use. But with the AdvDisk **SplitPath** command, you can now pass two variants and receive the string along with the string size. Figure 5 below displays the sample dialog box used to send and receive the path string.


Split Path Word

Enter Path: C:\TEST\TEST1\TEST.BAT

Returned Word \TEST\TEST1\

( ) Return the Drive.
(●) Return the Directory.
( ) Return the File Name.
( ) Return the File Extension.

Figure 5 - Split Path.

The sample code for this is as follows:

```
general
     Dim nValue As Integer

Sub cmdOK_Click ()
    szBuffer$ = String(255, 0)

    If txtPath.Text = "" Then Exit Sub
    strPath$ = txtPath.Text

    nLength% = PathSplit(strPath$, szBuffer$, nValue)

    If nLength% = 0 Then
        MsgBox "Cannot Split request", MB_ICONSTOP, "Split Return Error"
        Exit Sub
    End If
```

```
        strWord$ = Left$(szBuffer$, nLength%)
        lblSplitWord.Caption = strWord$
    End Sub
```

Create and Delete Directory

In Microsoft Visual Basic if you try and create a multi-levell directory, such as with the following command, you will receive a PATH NOT FOUND error if the directory \testtest\ does not already exist.

```
MkDir "c:\testtest\test1"
```

But with the AdvDisk **CreatePath** command, you can easily create up to nine multi-level directories with a single command. In order to accomplish this without AdvDisk, you will have to perform a loop which each time adding another level to the Visual Basic MkDir command. The **DeletePath** command functions the same as the create command but deletes multi-level directories. Figure 6 depicts the sample dialog used to either create or delete a multi-level directory.

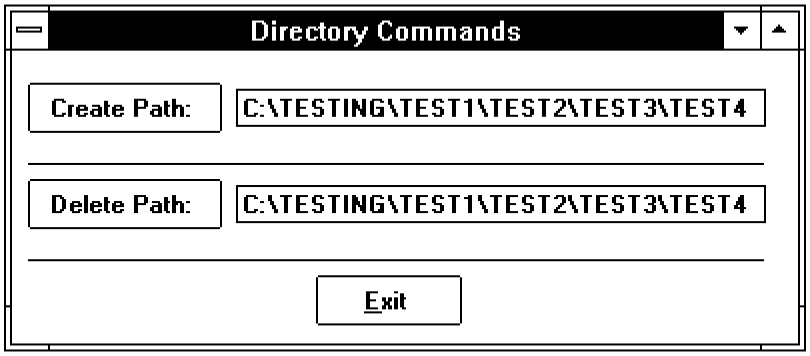

Directory Commands
Create Path:
C:\TESTING\TEST1\TEST2\TEST3\TEST4
Delete Path:
C:\TESTING\TEST1\TEST2\TEST3\TEST4
Exit

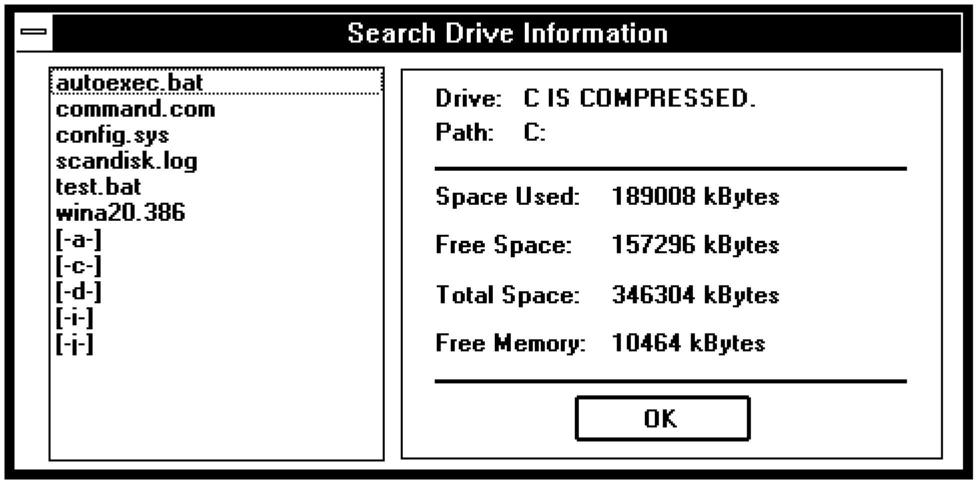
Figure 6 - Create and Delete Path.

The sample code for both the create and delete multi-level directories:

```
    Sub cmdOK_Click (Value As Integer)
        Screen.MousePointer = HOURGLASS
        If txtPath(Value).Text = "" Then Exit Sub

        strPath$ = txtPath(Value).Text

        Select Case Value
            Case 0
                nReturn% = CreatePath(strPath$)

                If nReturn% = FILE_EXISTS Then
                    MsgBox "Directory was created"
                Else
                    DisplayError (nReturn%)
                End If
            Case 1
                nReturn% = DeletePath(strPath$)

                If nReturn% = FILE_EXISTS Then
                    MsgBox "Directory was Deleted"
                Else
                    DisplayError (nReturn%)
                End If
        End Select

        Screen.MousePointer = DEFAULT
    End Sub
```

Search List

The Search List command provides your users with an selection interface which when picked, will return the file selection.


Search Drive Information

```
autoexec.bat
command.com
config.sys
scandisk.log
test.bat
wina20.386
[-a-]
[-c-]
[-d-]
[-i-]
[-j-]
```

```
Drive:  C IS COMPRESSED.
Path:   C:

Space Used:    189008 kBytes
Free Space:    157296 kBytes
Total Space:   346304 kBytes
Free Memory:   10464 kBytes
```

Figure 7 - Search List Dialog.

The sample code for the SearchList:

```
    Sub cmdSearchList_Click ()
        szBuffer$ = String(255, 0)

        strPath$ = "C:\*.*"
        iAttr% = SEARCH_NORMAL
        iDelimited% = SEARCH_SEMI_COLON

        nLength% = SearchList(strPath$, iAttr%, iDelimited%, szBuffer$)
        DoEvents

        If nLength% = 0 Then
            MsgBox "Search Item(s) not returned", , "Search List Return Error"
            Exit Sub
        End If

        strWord$ = Left$(szBuffer$, nLength%)
        MsgBox "Search Item Returned: " & strWord$, , "Search List Return Error"
        DoEvents
    End Sub
```

Disk Format and **Disk Copy**

Disk Formatting and the Disk Copy may not be accomplished easily from Visual Basic. The sample code below illustrates these two features.

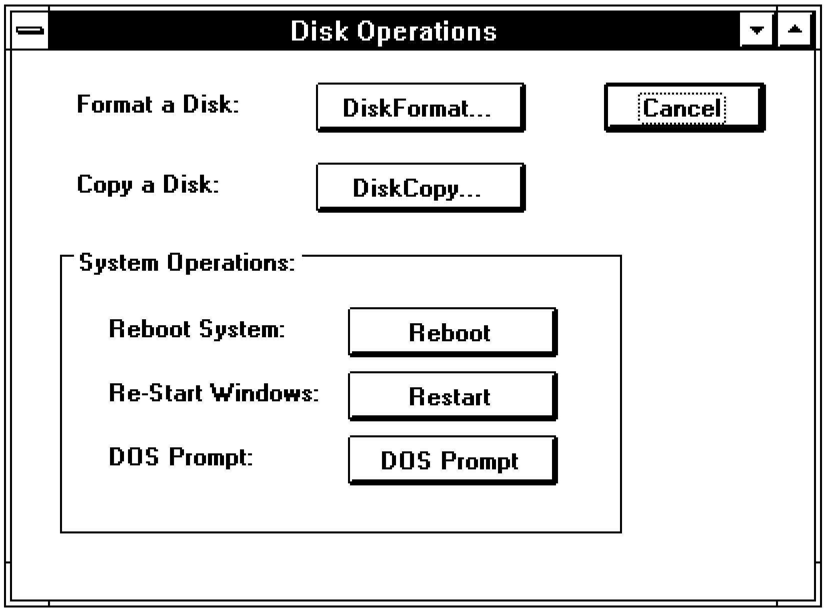


Figure 8 - Disk Format and Disk Copy.

```
Const DISK_FORMAT = 0
Const DISK_COPY = 1

Sub cmdManager_Click (Index As Integer)
    Screen.MousePointer = HOURGLASS

    Select Case Index
        Case DISK_FORMAT
            lReturn& = DiskFormat()
        Case DISK_COPY
            lReturn& = DiskCopy()
    End Select

    Screen.MousePointer = DEFAULT
End Sub
```

For either of the above functions, a return value of **True** denotes no errors while a return value of **False** or any other values denotes an error.

SystemWindow

SystemWindow provides the user with a way to Reboot the system, Restart Windows, or to goto a DOS Prompt from within your application.

Figure 9 - System Commands.

```
Const REBOOT = 0
Const RESTART = 1
Const PROMPT = 2

Sub cmdWindows_Click (Index As Integer)
    Screen.MousePointer = HOURGLASS

    lReturn& = DiskFormat(Index)
```

```
        Screen.MousePointer = DEFAULT
    End Sub
```

For either of the above functions, a return value of **True** denotes no errors while a return value of **False** or any other values denotes an error. But for Rebooting and Restarting return values will be invalid.

Visual Basic Declare Statement

The following list is provided to show all DLL calls from within Visual Basic:

Used to call the AdvDisk.dll Disk Free Space
Declare Function DiskFreeSpace Lib "advdisk.dll" (ByVal nDrive%) As Long

Used to call the AdvDisk.dll Total Space
Declare Function DiskTotalSpace Lib "advdisk.dll" (ByVal nDrive%) As Long

Used to call the AdvDisk.dll Drive Check
Declare Function DriveCheck Lib "advdisk.dll" (ByVal strDriveInfo$, ByVal nCheck%) As Integer

Used to call the AdvDisk.dll Drive Type
Declare Function DriveType Lib "advdisk.dll" (ByVal nDrive%) As Integer

Used to call the AdvDisk.dll Move/Copy File
Declare Function MoveCopyFile Lib "advdisk.dll" (ByVal strFrom$, ByVal strTo$, ByVal nCreate%, ByVal nDelete%, ByVal nOverwrite%, ByVal nRevision%) As Long

Used to call the AdvDisk.dll check if the file or directory exists
Declare Function FileExists Lib "advdisk.dll" (ByVal strFile$) As Integer

Used to call the AdvDisk.dll About Box
Declare Function AboutBox Lib "advdisk.dll" () As Integer

Used to call the AdvDisk.dll Split Path
Declare Function PathSplit Lib "advdisk.dll" (ByVal strPath$, ByVal strBuffer$, ByVal nValue%) As Integer

Used to call the AdvDisk.dll Search List
Declare Function SearchList Lib "advdisk.dll" (ByVal strPath$, ByVal iAttr%, ByVal iDelimited%, ByVal strBuffer$) As Integer

Used to call the AdvDisk.dll Create Directory
Declare Function CreatePath Lib "advdisk.dll" (ByVal strPath$) As Integer

Used to call the AdvDisk.dll Delete Directory
Declare Function DeletePath Lib "advdisk.dll" (ByVal strPath$) As Integer

Used to call the AdvDisk.dll Disk Format
Declare Function DiskFormat Lib "advdisk.dll" () As Long

Used to call the AdvDisk.dll Disk Copy
Declare Function DiskCopy Lib "advdisk.dll" () As Long

Used to call the AdvDisk.dll System Windows Commands
Declare Function SystemWindow Lib "advdisk.dll" (ByVal nType%) As Long

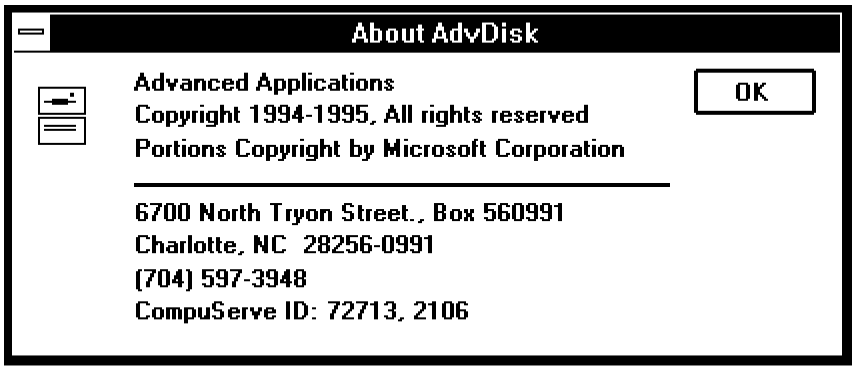
Constants

The supplied Global Constants are provided to help with common definitions and may be used within your application. Also supplied are the Microsoft Visual Basic CONSTANT.TXT constants which supplied with Visual Basic.

```
''''''''''''''''''''''''''''''''''''''''''''''''
' Copyright by Advanced Applications 1994 - 1995
' All rights reserved
'
''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''
' Variable
'
Global lDriveInfo As Long

''''''''''''''''''''''''''''''
' AdvDisk.dll Constants
'
Global Const NO_ERRORS = 0

Global Const FREE_SPACE = 0
Global Const TOTAL_SPACE = 1

Global Const FILE_EXISTS = 0
Global Const FILE_NOT_EXIST = 1

Global Const DEFAULT_DRIVE = 0
Global Const DRIVE_NOT_VALID = -1
Global Const DRIVE_UNDETERMINED = 0
Global Const DRIVE_REMOVABLE = 2
Global Const DRIVE_FIXED = 3
Global Const DRIVE_REMOTE = 4
Global Const DRIVE_CDROM = 5
Global Const DRIVE_RAM = 6

Global Const EXISTANCE_CHECK = 0
Global Const WRITE_CHECK = 2
Global Const READ_CHECK = 4
Global Const READ_WRITE_CHECK = 6

''''''''''''''''''''''''''''''
' Error Value Constants
'
Global Const ARGUMENT_LIST_TOO_LONG = 7
Global Const PERMISSION_DENIED = 13
Global Const BAD_FILE_NUMBER = 9
Global Const RESOURCE_DEADLOCK_WOULD_OCCUR = 36
Global Const MATH_ARGUMENT = 33
Global Const DRIVE_EXISTS = 17
Global Const INVALID_ARGUMENT = 22
Global Const TOO_MANY_OPEN_FILES = 24
Global Const FILE_NOT_FOUND = 2
Global Const DIRECTORY_NOT_FOUND = 3
Global Const EXEC_FORMAT_ERROR = 8
Global Const NOT_ENOUGH_MEMORY = 12
Global Const NO_SPACE_LEFT_ON_DEVICE = 28
Global Const RESULT_TOO_LARGE = 34
Global Const CROSS_DEVICE_LINK = 18
Global Const DISK_FULL = 39
Global Const FILE_ERROR = -1

Global Const MOVE_TO_NOT_OPENED = 100
Global Const MOVE_TO_NOT_CLOSED = 101
Global Const MOVE_FROM_NOT_CLOSED = 102
Global Const MOVE_FROM_NOT_REMOVED = 103
Global Const MOVE_FROM_NOT_EXIST = 104
Global Const MOVE_TO_DISK_FULL = 105
Global Const MOVE_TO_DIR_CREATE_ERROR = 106
Global Const MOVE_TO_DO_NOT_CREATE_DIR = 107
Global Const MOVE_DO_NOT_OVER_WRITE = 108
Global Const MOVE_NEWER_VERSION = 109
Global Const MOVE_DISK_UNFORMATTED = 22

Global Const DRIVE_COMPONENT = 0
Global Const DIRECTORY_COMPONENT = 1
Global Const FILENAME_COMPONENT = 2
Global Const EXTENSION_COMPONENT = 3

Global Const COMMAND_OK = -1       ' No Errors.
Global Const COMMAND_ERR00 = 0   ' System was out of memory, executable file was corrupt, or relocations
were invalid.
Global Const COMMAND_ERR01 = 1   ' System command not available
Global Const COMMAND_ERR02 = 2   ' File was not found.
Global Const COMMAND_ERR03 = 3   ' Path was not found.
Global Const COMMAND_ERR05 = 5   ' Attempt was made to dynamically link to a task, or there was a sharing or
network-protection error.
Global Const COMMAND_ERR06 = 6   ' Library required separate data segments for each task.
Global Const COMMAND_ERR08 = 8   ' There was insufficient memory to start the application.
Global Const COMMAND_ERR10 = 10 ' Windows version was incorrect.
Global Const COMMAND_ERR11 = 11 ' Executable file was invalid. Either it was not a Windows application or
there was an error in the .EXE image.
Global Const COMMAND_ERR12 = 12 ' Application was designed for a different operating system.
Global Const COMMAND_ERR13 = 13 ' Application was designed for MS-DOS 4.0.
Global Const COMMAND_ERR14 = 14 ' Type of executable file was unknown.
Global Const COMMAND_ERR15 = 15 ' Attempt was made to load a real-mode application (developed for an
earlier version of Windows).
Global Const COMMAND_ERR16 = 16 ' Attempt was made to load a second instance of an executable file
containing multiple data segments that were not marked read-only.
Global Const COMMAND_ERR19 = 19 ' Attempt was made to load a compressed executable file. The file must be
decompressed before it can be loaded.
Global Const COMMAND_ERR20 = 20 ' Dynamic-link library (DLL) file was invalid. One of the DLLs required to
run this application was corrupt.
Global Const COMMAND_ERR21 = 21 ' Application requires Microsoft Windows 32-bit extensions.

Global Const SEARCH_NORMAL = 0
Global Const SEARCH_READONLY = 1
Global Const SEARCH_HIDDEN = 2
Global Const SEARCH_SYSTEM = 3
Global Const SEARCH_DIRECTORY = 4
Global Const SEARCH_ARCHIVED = 5
Global Const SEARCH_INCLUDE_DRIVES = 6
Global Const SEARCH_HIDDEN_SYSTEM = 7

Global Const SEARCH_ONLY_READONLY = 8
Global Const SEARCH_ONLY_HIDDEN = 9
Global Const SEARCH_ONLY_SYSTEM = 10
Global Const SEARCH_ONLY_DIRECTORY = 11
Global Const SEARCH_ONLY_ARCHIVED = 12
Global Const SEARCH_ONLY_INCLUDE_DRIVES = 14
Global Const SEARCH_ONLY_HIDDEN_SYSTEM = 13

Global Const SEARCH_SPACE = 0
Global Const SEARCH_COMMA = 1
Global Const SEARCH_SEMI_COLON = 2
Global Const SEARCH_TAB = 3
Global Const SEARCH_CR = 4
```

AdvDisk.dll About Box

AdvDisk.dll is supplied as a development tool providing disk utilities not normally found with Microsoft Visual Basic.



Advanced Applications
Copyright 1994-1995, All rights reserved
Portions Copyright by Microsoft Corporation

6700 North Tryon Street., Box 560991
Charlotte, NC 28256-0991
(704) 597-3948
CompuServe ID: 72713, 2106

Thank you for choosing AdvDisk as one of your development tools. If you have any questions, comments, suggestions or otherwise opinions, please feel free to contact us here at Advanced Applications.

Thank you.