Technical Note FL525
Standard File Package Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&As can be found on the Macintosh Technical Q&As web site.

[Oct 01 1990]

Standard File Package directory defaults

Date Written: 1/22/93

Last reviewed: 6/14/93

When I double-click a document that launches my application, the current directory for the Standard File package (at location $398 in memory) is set to the directory of my application and not my document. This seems to be a bug according to the text on page 3-31 of the new *Inside Macintosh: Files* manual. Is there anything special I have to do?

You're right. The behavior described in *Inside Macintosh: Files* isn't entirely correct. It should say that the first time your application calls one of the Standard File Package routines, the default current directory (that is, the directory whose contents are listed in the dialog box) is determined by the way your application was launched.

* If the user launched your application directly (perhaps by double-clicking its icon in the Finder), the default directory

is the directory in which your application is located.

* If the user launched your application indirectly (perhaps by double-clicking one of your application's document icons) and your application is passed Finder information, the default directory is the directory of the last document listed in the Finder information. The Finder information is the data referenced by `AppParmHandle` and accessed by the Segment Loader routines `CountAppFiles`, `GetAppFiles`, `ClrAppFiles`, and `GetAppParms`.

Note that applications that are high-level event aware are passed the list of documents to open or print in a `kAEOpenDocument` or `kAEPrintDocument` Apple event. There's no Finder information (`AppParmHandle` will be NIL) and the default directory is the directory in which your application is located.

Back to top

Dialog filter control with subdialog boxes

Date Written: 12/10/92

Last reviewed: 6/14/93

My routine uses a dialog hook to set and retrieve certain values in new items added to the default box. Previously, with `SFPPutFile`, I was able to use a hit on the Save item to retrieve and save the values. This also works with `CustomPutFile` unless the Replace/Cancel dialog box appears, because the dialog hook routines are also called for it! With the dialog pointer now pointing at the small alert, any reference to expected items leads to disaster, since they don't exist. Isn't calling the dialog hook routine to respond to hits in the alert box wrong and therefore a bug?

———

Both Standard File and the Edition Manager in System 7 allow you to have control in your filter when one of the subdialog boxes comes up. You can differentiate between the main dialog and the subdialogs by looking in the refCon field of the dialog record passed to you. In Standard File's case, if the dialog is the main dialog, the refCon will be:

```
/* From StandardFile.h */
/* The refCon field of the dialog record during a modalfilter
/* or dialoghook contains one of the following: */
#define sfMainDialogRefCon 'stdf'
#define sfNewFolderDialogRefCon 'nfdr'
#define sfReplaceDialogRefCon 'rplc'
#define sfStatWarnDialogRefCon 'stat'
#define sfLockWarnDialogRefCon 'lock'
#define sfErrorDialogRefCon 'err '
```

This is described in detail on page 26-18 of *Inside Macintosh* Volume VI, in the middle of the section that describes all the callbacks and pseudo items for Standard File under System 7. The main purpose of this is to allow your additional dialog items to react properly when another dialog box is brought up in front of them, not to allow you access to the subdialogs. Also, since Standard File has no idea what types of items you've added to the dialogs, giving you control during subdialogs allows you to change the look of your subitems, or to keep them active if they need periodic time for any reason.

Back to top

Working around Standard File quirk when system heap is full

Date Written: 12/4/91

Last reviewed: 6/14/93

Standard File can fail to function properly when the system heap is very full; it just returns false in the reply.good field. This is a serious problem for us because we are unable to detect this situation; to our application, it just looks like the user clicked the Cancel button. Do you have any suggestions for working around this?

———

This is a significant problem, but we can't guarantee that the software will perform in any imaginable set of circumstances you want to set up. You're going to have to check to see if it will be able to work (preflight it) and then see if it fails. In virtually all such failure cases, the low-memory globals `MemErr` or `ResErr` will be set with an error, as tested with the functions `MemError()` or `ResError()`. You might also attempt to allocate an amount of memory in the

system heap, and if that allocation fails, inform the user that "Save As..." might fail, indicating possible solutions (such as turning off balloon help, etc). To preflight for about 50-60K in the system heap would probably be adequate.

Back to top

Custom Standard File dialog edit fields under System 7

Date Written: 8/14/91

Last reviewed: 6/14/93

How do I change the active edit field inside a custom Standard File dialog under System 7? In a related problem I am finding that the selection range for all edit fields in the dialog equals the number of characters in the file name field when tabbing around.

The Standard File Package (SFP) routines don't behave exactly the same as they did under System 6. Therefore, doing something like trying to change the active item number doesn't work under System 7's version of the SFP routines. The problem is that System 7 Standard File has a whole set of interfaces dedicated to the active item list and which item is currently active, whereas System 6 SF routines just use the dialog data to store this information. The solution to the problem, then, is to use `CustomGetFile` to accomplish the same thing if you are running under System 7. I've included a sample program which uses the new routine to change the focus and check the bounds of several editText items.

Your second problem is brought on by a bug in Standard File. The workaround is to install an activate procedure for Standard File (it's a parameter to the CustomGetFile call) which calls SellText on the appropriate field to select the entire range. The included sample also does this.

```
/*    CustomGetFile example

    This sample uses CustomGetFile to add two edit text fields
    to the standard get file box, and checks the values the user
    enters into those fields. If the values are incorrect, the
    user is alerted to change them, and the focus of the dialog
    is changed to the proper field.

    The standard file bug causing selection ranges to be calculated
    improperly is also fixed in this sample by calling SelIText in
    the activate procedure for edit text items.
*/

/* prototypes */

void InitStuff(void);
void CustomGet(void);
pascal void MyActProc(DialogPtr theDlg,short item,
                      Boolean activating, Ptr data);
pascal short DlgHook(short item,DialogPtr theDlg,Ptr userData);
Boolean CheckField(DialogPtr theDlg,short item);

/* constants */

#define    kTextField1    10
#define    kTextField2    11
#define    kSFDlg        128
#define    kAlertDlg     129

void main(void)
{
    InitStuff();
    CustomGet();
}

/* initialize managers */
```

```
void InitStuff(void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(nil);
    FlushEvents(everyEvent,0);
    InitCursor();
}

/* do getfile */

void CustomGet(void)
{
    Point where = {-1,-1};
    SFReply reply;
    DialogPtr theDialog;
    short item;
    StandardFileReply sfReply;
    short activeList[5];

    /* set-up active items list */

    activeList[0] = 3;
    activeList[1] = 7;
    activeList[2] = 10;
    activeList[3] = 11;

    CustomGetFile(nil,-1,nil,&sfReply,kSFDlg,where,DlgHook,
                  nil,activeList,MyActProc,nil);
}

/*  activate procedure- this procedure handles the activate/deactivate of
    textedit items and corrects the selection bug in standard file which
    normally causes the selStart and selEnd fields of the texthandle to be
    incorrect
*/

pascal void MyActProc(DialogPtr theDlg,short item,Boolean activating,Ptr data)
{
    short iType;
    Handle iHndl;
    Rect iRect;
    TEHandle textH;

    GetDItem(theDlg,item,&iType,&iHndl,&iRect);
    if (iType != editText)
        return;

    if (activating) {
        SelIText(theDlg,item,0,32000);
        return;
    }
}

/*  this dialog hook checks the contents of the additional edit fields
    when the user selects a file. The focus of the dialog is changed if one
    of the fields is out of range.
```

```
*/

pascal short DlgHook(short item,DialogPtr theDlg,Ptr userData)
{
    if (item==ok) {
        if (!CheckField(theDlg,kTextField1))
            item = kTextField1 + sfHookSetActiveOffset;
        else if (!CheckField(theDlg,kTextField2))
            item = kTextField2 + sfHookSetActiveOffset;
    }

    return item;
}

/*  this procedure checks the range of a given edittext item to make sure it
    contains a number from 0 to 256. If not, it alerts the user that the
    field must be re-entered.
*/

Boolean CheckField(DialogPtr theDlg,short item)
{
    short iType;
    Handle iHndl;
    Rect iRect;
    Str255 iText;
    long num;

    GetDItem(theDlg,item,&iType,&iHndl,&iRect);
    GetIText(iHndl,iText);

    StringToNum(iText,&num);
    if (num<0 || num>256 || iText[0]==0) {
        StopAlert(kAlertDlg,nil);
        return false;
    }
    else
        return true;
}

The resource file follows:

/*
    Dialog and Alert templates for use with CustomGetFile example.

    Steve Falkenburg -- MacDTS
*/

#include "types.r"

/* CustomGetFile dialog */

resource 'DLOG' (128, purgeable) {
    {0, 0, 206, 344},
    dBoxProc,
    invisible,
    noGoAway,
    0x0,
    128,
    ""
};
```

```
resource 'DITL' (128, purgeable) {
    {    /* array DITLarray: 11 elements */
        /* [1] */
        {135, 252, 155, 332}, Button { enabled, "Open" },
        /* [2] */
        {104, 252, 124, 332}, Button { enabled, "Cancel" },
        /* [3] */
        {0, 0, 0, 0}, HelpItem { disabled,
            HMScanhdlg {
                -6042
            }
        },
        /* [4] */
        {8, 235, 24, 337}, UserItem { enabled },
        /* [5] */
        {32, 252, 52, 332}, Button { enabled, "Eject" },
        /* [6] */
        {60, 252, 80, 332}, Button { enabled, "Desktop" },
        /* [7] */
        {29, 12, 159, 230}, UserItem { enabled },
        /* [8] */
        {6, 12, 25, 230}, UserItem { enabled },
        /* [9] */
        {91, 251, 92, 333}, Picture { disabled, 11 },
        /* [10] */
        {175, 16, 191, 91}, EditText { enabled, "" },
        /* [11] */
        {175, 106, 191, 181}, EditText { enabled, "" }
    }
};
/* input check value alert */
resource 'ALRT' (129) {
    {110, 130, 208, 414},
    129,
    {    /* array: 4 elements */
        /* [1] */
        OK, visible, sound1,
        /* [2] */
        OK, visible, sound1,
        /* [3] */
        OK, visible, sound1,
        /* [4] */
        OK, visible, sound1
    }
};

resource 'DITL' (129) {
    {    /* array DITLarray: 2 elements */
        /* [1] */
        {68, 218, 88, 276},Button {enabled,"OK"},
        /* [2] */
        {10, 61, 62, 278},
        StaticText { disabled,
            "Your field entry is out of range.  Pleas"
            "e enter a number between 0 and 256."
        }
    }
};
```

Back to top

Displaying invisible files under Systems 6 & 7 without typeList

Date Written: 8/9/91

Last reviewed: 6/14/93

Under System 7 my filter procedure for displaying invisible data files no longer works. How can I use Standard File to display the names of invisible files of a specific type under System 7?

System 7 can show invisible files in the standard `SFGetFile` dialog box; however, not all System 6 Standard File package calls are handled the same in System 7.

When using invisible files under System 7, you should perform type filtering within a filter proc and not with the `typeList` field of the `SFGetFile` call. System 7 no longer allows a `typeList` for detecting invisible files. The actual check for invisible files of a particular type or types should be done within the file filter proc.

The `SFGetFile` call below displays only folders and invisible 'TEXT' files in the standard `SFGetFile` dialog box. With the `numTypes` parameter set to -1, all types of files will be passed to the filter proc.

```
SFGetFile( where, "", myFilterProc, -1, typeList, nil, &reply );
```

In this example, the filter proc's return value depends on the file's type and Finder flags.

```
pascal Boolean myFilterProc( fp )
FileParam *fp;
{
  if ((fp->ioFlFndrInfo.fdFlags & fInvisible) &&
      (fp->ioFlFndrInfo.fdType == 'TEXT'))
    return FALSE;
  else
    return TRUE;
}
```

Back to top

Standard File and nontrashable Macintosh folders

Date Written: 7/24/91

Last reviewed: 6/14/93

When we use Standard File to get a Macintosh file in a folder, it becomes impossible to throw that folder away and empty the trash without quitting first. Is this because the working directory is still open? It is my understanding that applications shouldn't close working directories that were opened by Standard File. Is there something I should be doing, or is this just a limitation in the system?

Pre-System 7 Standard File calls (SFGetFile/SFPutFile/etc...) call PBOpenWD to open a working directory to the folder where the selected file resides.

This working directory, along with all others created within any application, are closed by MultiFinder (or the Process Manager under System 7) when the application is quit. Before the application quits, you will not be able to throw away the folder. After quitting, however, the directory is closed and the folder can be trashed.

As described in the Macintosh Technical Note "Working Directories and MultiFinder," this is accomplished in the following way: When Standard File calls `PBOpenWD()`, the `ioWDProcID` field is ignored, and MultiFinder replaces its contents with a unique process identifier. When your application quits, MultiFinder indexes through all open working directories with your unique process ID and closes them.

Your understanding is correct that you don't have to close these Standard File working directories yourself. If, however, you want the user to be able to delete the directory *while* your application is still running, you will have to issue a PBCloseWD() call yourself, as in the following example:

```
WDPBRec theWD;
Point where = {100,100};

SFGetFile ( where, nil, nil, -1, nil, nil, &reply );
<do file stuff here>
theWD.ioVRefNum = reply.vRefNum;
err = PBCloseWD(&theWD,false);
```

If you're running under System 7, you are much better off using the new StandardGetFile() and StandardPutFile() routines. They do not use working directories at all, and instead return FSSpecs to refer to files.

If none of the above helps, your problem may be that you have left a file open in the directory the user is trying to delete. This would cause the same error as the one you described.

Back to top

How to override System 7.0 Standard File dialog centering

Date Written: 6/19/91

Last reviewed: 8/1/92

Any way to override the new default screen location (upper-middle) for Standard File calls under System 7.0? My Standard File dialog needs to be somewhere else on the screen.

You can use the CustomGetFile *(Inside Macintosh* Volume VI, page 26-22) and CustomPutFile *(Inside Macintosh* Volume VI, page 26-20) to place the related Standard File dialogs in the location you specify as a parameter. This should override the centering feature that System 7.0 uses on the StandardGetFile *(Inside Macintosh* Volume VI, page 26-22) and StandardPutFile *(Inside Macintosh* Volume VI, page 26-20) calls.

Back to top

Tabbing between SFPPutFile custom dialog text fields

Date Written: 6/7/91

Last reviewed: 8/1/92

How can I get the tab key to tab between text fields in my SFPPutFile custom dialog instead of switching drives?

Here is an event filter that beeps whenever the tab key is pressed (under System 6):

```
pascal Boolean MyDlgFilter(DialogPtr theDialog,EventRecord *theEvent,short
*itemHit)
{
    WindowPtr updateWindow;
    char theChar;

    switch (theEvent->what) {
        case keyDown:
        case autoKey:
            theChar = theEvent->message & charCodeMask;
            switch (theChar) {
                case 0x0d:    /* CR */
                case 0x03:  /* enter */
                    *itemHit = OK;
                    return true;
                case 0x1b:  /* ESC */
                    *itemHit = Cancel;
                    return true;
                case '\t':
                    SysBeep(1); //   <----- do your "tabbing" here
                    *itemHit = 0; //   <----- this is what you need to
add
                    return true;
            }
            break;
    }
    return false;
}
```

Within dialog event filters, when the filter decides to process the event, the filter not only must return true, but must also return the item number acted on by the filter. Under System 7, tab is handled by the system automatically and is not controllable from dialog event filters.

Back to top

Filtering out invisible folders from a Standard File dialog list

Date Written: 6/10/91

Last reviewed: 6/14/93

I want to display only visible files and folders in a Standard File dialog, but I can't find a way to filter out invisible folders--specifically the 000Move&Rename folder. The `FileFilter` routine filters only files, not folders. If I put in a nonzero `TypeList`, invisible folders seem to be removed, but I want to open all types of files, just not invisible files or folders. Any suggestions?

This is, in fact, impossible under System 6 using general methods. The problem is that passing -1 as numTypes means not only to display all items, but to display invisible items. A file filter can be used to remove the invisible files but cannot affect invisible folders. The only current way to do this is to use CustomGetFile under System 7, as described in the Standard File Package chapter of *Inside Macintosh* Volume VI. This provides a filter that allows you to filter both files and folders. This will give you the right functionality, but will work only under System 7. We recommend that you use this method under System 7, and a more standard SFGetFile when running under earlier systems.

Back to top

File handling within SFPGetFile & SFPPutFile DlgHook functions

Date Written: 4/2/91

Last reviewed: 8/1/92

How can I obtain the volume reference information in my DlgHook function for a file selected by the user before SFPPutFile or SFPGetFile has completed the reply record?

On exit, SFPGetFile and SFPPutFile generate a working directory reference number in the vRefNum field of the reply record. This is not available to you from within the operation of a DlgHook function. WDRefNums are provided to allow compatibility with older, pre-HFS functions that took vRefNum values of integer size with the older flat file system.

We suggest that, unless you plan to support the flat file system of 64K ROM Macintosh systems, you move your file system interfaces to the HFS interfaces documented in the File Manager sections of *Inside Macintosh* Volumes IV and V (or to the equivalent high-level calls as documented in the Macintosh Technical Note "New High-Level File Manager Calls"). If you're using the HFS calls, low-memory globals SFSaveDisk and CurDirStore contain, respectively, the negative of the "real" volume reference number for the current volume and the HFS ID of the directory that Standard File is displaying. You then have all the information you need to create, open, rename, or delete files from within the SFPGetFile and SFPPutFile DlgHook functions. If a user is accessing an MFS volume on an HFS system, these calls are designed tohandle file access transparently.

Moving your file system interfaces to the HFS-level conventions has a side benefit of being closer to the System 7 file system specifications. If you look at the new high-level file system calls in *Inside Macintosh* Volume VI, you'll recognize much of the HFS information embedded in the new data structures.

If your file system interfaces depend on MFS-style vRefNums, or WDRefNums in the HFS nomenclature, you can use the HFS functions PBOpenWD, PBCloseWD, and PBGetWDInfo to open, close, and obtain volume reference numbers and directory IDs. This is particularly important if, for instance, you're using the THINK C ANSI file I/O functions, which rely on SetVol to operate correctly.

Complete information on the HFS-level calls that will be most useful in Standard File customization is contained in the File Manager chapters of *Inside Macintosh* Volumes IV and V, and in the Macintosh Technical Notes "Determining Which File System Is Active," "HFS Ruminations," "HFS Elucidations," "Why PBHSetVol is Dangerous," "Setting ioNamePtr in File Manager Calls," and "Working Directories and MultiFinder." For C users, the Macintosh Technical Note "Mixing HFS and C File I/O" summarizes a list of the difficulties with mixing C file I/O with Macintosh file I/O. Macintosh Technical Notes "Customizing Standard File" and "Standard File Tips" discuss a few points of Standard File customization from the point of view of HFS.

Back to top

What to do instead of nested SFPGetFile calls

Date Written: 1/14/90

Last reviewed: 2/6/91

I am nesting two Macintosh SFPGetFile dialogs through custom routines. Under some circumstances after I call up the second SFPGetFile dialog and return (usually via Cancel) to the first one, I lose all of the custom controls in the first SFPGetFile dialog.

The SF package is not re-entrant, so there isn't really a way to do what you want here. MOST of the information is kept around when you nest calls, but the main problem is in the resources SF uses for the items. When the nested SF dialog closes (on a cancel, for example) it releases the resources that Standard File is using. Unfortunately, this also releases the resources that are being used by the original dialog, so that's where your items are getting messed up. And while there is potentially a workaround by doing some kludgy stuff, I can guarantee that anything I tell you now will be completely wrong under System 7.0, so I can't do that. However, you can use sequential calls, instead of nested. This is a little more of a pain, but it'll work.

Call SFPGetFile. In your filter routine, when the user hits the control that you want to bring up the nested box, set a flag in your application saying "bringUpOther," and tell Standard File that you're done by passing item 1 or 2 back. You return, put up your second SFPGet, process that info, then bring the original SFPget back. I realize that this is not what you're looking for, since it'll be a little messy as the dialogs open and close, but it's the only way to do it with any chance of success on more than one system.

Back to top

Working directory not necessary for new Macintosh applications

Date Written: 12/12/90

Last reviewed: 8/1/92

Why does closing the working directory also close it for other users?

———

It is not necessary to use "Working Directories." When the Macintosh first came out there was no notion of directories. MFS was a flat file system; all files were stored in a single directory. Hence, all of the original applications specified a file by its name and the volume it was on. In other words, a vRefNum and a fileName.

The Hierarchical File System (HFS) introduced the concept of directories to the Macintosh. This meant that applications now had to specify a directory ID along with the `vRefNum` and fileName. The problem then was that old (pre- HFS) applications had to be able to work with directories other than the root. That was accomplished by having the File System create fake `vRefNums` that represent both a real `vRefNum` and a dirID. These fake `vRefNums` are called working directories and span a special number range (less than -32000). When the file system notices a `vRefNum` in that range, it interprets it as a working directory. Using a look-up table it matches it to a real vRefNum and dirID. This allows old applications to work with sub-directories. Essentially, older applications treat each directory as a distinct volume.

In other words, working directories are for providing compatibility within the file system for old (pre-HFS) applications. New applications and XCMDs shouldn't be creating or using them. Standard File still returns working directories, and you can use those, but it is recommended to convert them into real `vRefNum/DirID` pairs as soon as `SFGet/PutFile` returns.

But what if you do use working directories? Under MultiFinder, the ioWDProcID field is filled in with the process ID that MultiFinder creates when it launches a new application. When you create a working directory, an entry is created for each `vRefNum/dirID/procID` triplet. In other words, if two applications create a working directory for the same folder, they will get two different working directory values. Closing one of them should not effect the other.

XRef: DTS Macintosh Technote "Getting a Full Pathname"

Back to top

How to control path used by SFGetFile

Date Written: 11/1/90

Last reviewed: 8/1/92

I would like to be able to control which path `SFGetFile` uses to display the initial list of files for the user to choose from. I need to create the equivalent of `SFGetDisplayFolder` and `SFSetDisplayFolder` functions.

———

To set the directory for standard file dialogs, set the low memory global `SFSaveDisk` ($214) to the negative of the `vRefNum` for the volume, and set `CurDirStore` ($398) to the directory ID. In Pascal it might look something like

```
PROCEDURE SetupStandardFile(newVRefNum: Integer; newDirID: LongInt);
TYPE
  LongIntPtr = ^LongInt;
  IntegerPtr = ^Integer;
CONST
  SFSaveDisk = $214;   { address of two-byte vRefNum }
  CurDirStore = $398;  { address of four-byte dirID  }
VAR
  SFSaveDiskPtr: IntegerPtr;
  CurDirStorePtr: LongIntPtr;
BEGIN
  SFSaveDiskPtr := IntegerPtr(SFSaveDisk);
  CurDirStorePtr := LongIntPtr(CurDirStore);

  SFSaveDiskPtr^ := -1 * newVRefNum;
  CurDirStorePtr^ := newDirID;  { ignored under MFS }
END;
```

or in C:

```
void SetupStandardFile(short newVRefNum, long newDirID)
{
  enum { SFSaveDisk = 0x214, CurDirStore = 0x398 };

  *(short *) SFSaveDisk = -1 * newVRefNum;
  *(long *) CurDirStore = newDirID;
}
```

This is documented in the Macintosh Technical Note "Standard File Tips." Note that the vRefNum should be the true vRefNum for the desired volume, not a working directory refNum. Standard File dialogs are also unrelated to and unaffected by the default directory (GetVol/SetVol) and Macintosh programs should almost never have a need to get or set the default directory.

Back to top

Saving correct Macintosh "user file last used" information

Date Written: 11/17/89

Last reviewed: 8/1/92

I save the vRefNum returned by Standard File so I can easily get back to the file the user last used, but why do I sometimes get "file not found" errors when I try to open the file?

What you have to remember is that under HFS, vRefNums are almost always working directory reference numbers, containing both volume and directory information. wdRefNums have always been transient and not guaranteed to remain valid between system boots. Under MultiFinder, wdRefNums are even more restrictive and are not valid after applications exit. (See the Macintosh Technote "Working Directories and MultiFinder".)

What you should do is translate the wdRefNum into something more permanent. Try a volume name, a working directory DirID, and a filename. Use PBGetWDInfo to determine the real vRefNum (in ioWDVRefNum) and the DirID in ioWDDirID. Then use PBGetVInfo to determine the volume name from the real vRefNum (keeping the vRefNum is insufficient since the vRefNum is likely to change depending on the order of mounting volumes). Also store the volume creation date to distinguish between volumes with the same name.

This is the best you can do to save file information for later use under System 6. Under System 7, create and store an alias for the FSSpec returned by Standard File and use that to later locate the file.

Back to top

Downloadables

Acrobat version of this Note (K) Download

Back to top
