Setup Toolkit for Windows

Microsoft®

WINDOWS™

SOFTWARE DEVELOPMENT KIT

Microsoft® Windows™

Version 3.1

Setup Toolkit

For the Microsoft Windows Operating System

Microsoft Corporation

Document Number 30213

Contents

Introduction

The Microsoft® Windows™ operating system is a single-user personal computer operating system that employs a graphical user interface. Microsoft provides a variety of tools you'll find useful as you develop Windows applications. One of these tools, Setup, helps you create installation kits for your Windows applications. This manual, *Setup Toolkit for Windows,* explains how to use the Setup procedures and sample script files to create installation kits.

- Chapter 1, "Creating a setup Program," explains how to use the basic components of the toolkit to create installation files for your product.

- Chapter 2, "Designing Dialog Boxes," explains how to use the Windows Dialog Editor to modify the dialog box templates for Setup. The second part of the chapter describes the C functions you use to modify the associated dialog box procedures.

- Chapter 3, "Creating a setup Script," explains how to modify sample script files to meet the specific needs of your application's installation.

- Chapter 4, "Using the Disk Layout Utilities," explains how to use the Disk Layout Utilities to create and update disk images.

- Chapter 5, "Setup Script Procedures," describes the Basic procedures (functions and subroutines) that you use to create a setup script.

- Appendix A, "INF File Format," describes the structure of an .INF file and provides a list of software default values.

- Appendix B, "Command Option Flags," describes the command option flags that you can use as arguments for many of the Setup script procedures.

System Requirements for Setup

To use the Setup toolkit, you must have the Windows version 3.1 Software Development Kit (SDK), the Microsoft C compiler, and the C run-time libraries (including MDLLCEW.LIB) installed on your computer.

Document Conventions

The following document conventions are used throughout this manual:

Convention	Meaning
Bold text	Denotes a term or character to be typed literally, such as language keywords or function names (**AS INTEGER** or **BackupFile**); MS-DOS commands (**dir**); and MS-DOS command-line options (**/P**).
Italic text	Denotes a placeholder or variable: You must provide the actual value. For example, the statement **BackupFile** *szFullPath$ szBackup$* requires that you substitute values for the *szFullPath$* and *szBackup$* arguments.
`Monospaced text`	Represents code samples.
CAPITAL LETTERS	Represent filenames, directory names, drive letters, or symbolic constants.
Initial Capitals	Represent names of programs, menus, menu commands, dialog boxes and options, buttons, and named windows.
( )	Enclose one or more arguments that you pass to a function.
[ ]	Enclose optional parameters.

Chapter 1: Creating a Setup Program

When you install a Microsoft software program on your computer, you may be using Setup and its supporting functions to decompress and copy the program files onto your hard disk. Setup is a tool that you can use to create scripts that will install a Windows application on a user's computer. Microsoft is providing the Setup toolkit as part of the Windows version 3.1 SDK so that you can take advantage of its many automated procedures when you create an installation program for your own product.

The Basic Components of Setup

The Setup toolkit includes the following basic components:

- A bootstrapper program, SETUP.EXE, which copies the Setup driver (_MSTEST.EXE) and other supporting files to a temporary directory on the user's hard disk and then launches your Setup script. When your Setup script is complete, SETUP.EXE removes the temporary files and directory.

- A run-time version of Microsoft Test, _MSTEST.EXE, which Setup uses to interpret its scripting language. The Setup sample files also contain Test commands that define a DEBUG flag and include the files that define Setup procedures. Test commands give you greater flexibility for modifying the Setup sample files. In addition, the Test development environment includes a debugger and other useful tools. You do not have to modify the Test commands for your script, but you may want to purchase and use Test in conjunction with the Setup procedures.

- Sample script files (SAMPLE1.MST, SAMPLE2.MST, and SAMPLE3.MST), which you use as a starting point for creating your own Setup script.

- Sample dialog box templates and procedures (DIALOGS.DLG, DIALOGS.RES, and DLGPROCS.C), which you modify using the Windows

Dialog Editor to create the dialog and message boxes you need for your installation.

- Six .DLL files that contain useful routines for detecting the hardware and software environment, managing dialog and message boxes, copying files, modifying .INI files, and performing other program management functions. These procedures are described in Chapter 5, "Setup Script Procedures;" you will use them to create your Setup script.

- Disk Layout Utilities, DSKLAYT.EXE and DSKLAYT2.EXE, which you use to create the installation disks you will ship with your product.

- An MS-DOS utility, _MSSETUP.EXE, which you can use to update system files that are locked while in use by Windows.

Use these components as described in the next section to create your own Setup script. Using Setup for your product installations will ensure that the process is safe and efficient, and that the installation meets Windows programming standards.

Steps for Creating a Setup Script

Use the following procedure to create a setup script for your product. The remainder of this section discusses each step in more detail.

To create a setup installation script:

1. Identify the files that will be installed for your product.

2. Design the directory structure for those files.

3. Identify all user-defined parameters.

4. Design the dialog boxes you will need for the installation.

5. Modify the sample script files so that they will install your product's files.

6. Create the images for the installation disks using the Disk Layout Utilities.

7. Test your installation script.

Step 1: Identify the Files That Will Be Installed for Your Product

Before you start modifying sample files and dialog box templates, it's a good idea to make a list of the files you will need to install. For each file that you will install, answer the following questions:

- Is this file unique for your product, is it a shared file, or is it a system file? For example, a shared file could be a language dictionary used by more than one product for your company. A system file could be a newer version of COMMDLG.DLL or a TrueType font. If the file is a shared file or a system file, you will want the installation script to check to see if it already exists and whether it is currently in use before copying it onto the user's hard disk.

- Can the user decide whether to install this file? For example, is the installation of tutorial files optional? If so, you'll need to design a dialog box that asks the user to choose the files to install.

- If an older version of the file already exists, should you overwrite it or rename it? Or, if you want to delete it, is the older version of the file under a different filename? If so, you will want the installation script to remove it, as well as install its newer version.

Beside each filename on your list, make notations indicating the answers to these questions. These notations will help you later when you design dialog boxes or set the properties for each file with the Disk Layout Utilities.

Step 2: Design the Directory Structure for the Installable Files

Take the time now to sketch out the directory structure for your product by organizing the installable files into categories that make sense. For example, you might put all computer-based training files in one subdirectory and all font files in another. You may need to place some files in the Windows installation directory or in one of its subdirectories. On the other hand, one directory (with no subdirectories) may suffice for a product that has only a few files.

Step 3: Identify All User-Defined Parameters

Identify the dialog and message boxes you will need for your Setup program. What input does the user provide during installation? For example, will you store the user's name, the company name, and the product serial number in a file? Can the user decide which directory to use for installation? Can the user decide not to install some of the product files? You should also note whether

you will allow network installations and, if so, how the installation process will differ when installing to a network drive.

Are there any issues that you need to communicate to the user? For example, do you need a message box to notify the user that you will be updating or deleting existing files? If so, note these as well.

Step 4: Design Dialog Boxes

Make a rough sketch of each dialog box and identify the controls (buttons, check boxes, and list boxes) that will be needed. This will help you choose the most appropriate template to modify. Then use the Windows Dialog Editor to customize the templates. You may also need to modify the dialog box procedures (in DLGPROCS.C) to process the user's responses. For more information about this process, see Chapter 2, "Designing Dialog Boxes."

Step 5: Modify the Sample Script Files to Create Your Installation Script

The Setup toolkit contains three sample script files (SAMPLE1.MST, SAMPLE2.MST, and SAMPLE3.MST) and three associated sample .INF files (SAMPLE1.INF, SAMPLE2.INF, and SAMPLE3.INF). The sample .MST files contain variable declarations and calls to Setup procedures that you would typically use to install your product. The sample .INF files describe for Setup the installation media and installable files, and they show the entries that the Disk Layout Utilities would create based on your choices for these items.

Each sample installs a slightly different type of product: One installs a set of files with no special requirements; one uses more complicated dialog boxes and installs several sets of files based on the user's choices; and one installs files that are shareable resources. Use these samples as a starting point for creating your own installation script.

Once you've named and saved your version of a sample .MST file, you'll want to update SETUP.LST to include the new name. SETUP.EXE, the bootstrapper program mentioned earlier, reads this file to determine which files are needed to run the installation and copies them to a temporary directory on the user's hard disk.

For more information about each of these files, see Chapter 3, "Creating a Setup Script." For descriptions of the procedures used in these files, see Chapter 5, "Setup Script Procedures."

Step 6: Create the Images for the Installation Disks

Gather all of your product files and installation program files into the directory structure that you sketched out earlier. Then use the Disk Layout Utilities to define each file's properties (such as whether it can be put on a writable disk) and to create the images for the installation disks. The Disk Layout Utilities automatically create the .INF file as part of this process.

You can use the Disk Layout Utilities throughout your software development project, until you create your master disks. Each time you release another version of your product, use the Disk Layout Utilities to update the .INF file and disk images.

For more information about the Disk Layout Utilities, see Chapter 4, "Using the Disk Layout Utilities."

Step 7: Test Your Installation Script

Once you've created your own Setup installation script, test it thoroughly. Test the script under a variety of situations and computer configurations. Check the results by verifying that the files were copied to their appropriate directories and that system files were updated correctly. When you are satisfied that the installation is correct, create your master disks by copying the disk images onto floppy disks.

Chapter 2: Designing Dialog Boxes

The Setup toolkit includes dialog box templates that you can customize to meet your installation's specific needs. You can use the Microsoft Windows Dialog Editor (DLGEDIT.EXE) to edit the templates.

The Dialog Editor is a tool that lets you design and test a dialog box on the display screen instead of defining dialog statements in a resource script. Using the Dialog Editor, you can add, modify, and delete controls in a dialog box. The Dialog Editor saves the changes you make as resource script statements. You then compile these statements into a binary resource file that is linked to your Setup application's executable file of dialog procedures.

The Setup toolkit provides the following files that contain sample dialog box templates and procedures:

- DIALOGS.DLG, which contains dialog box templates. See Table 2.1 for descriptions of these templates. DIALOGS.DLG is updated automatically when you use the Dialog Editor to read its companion file, DIALOGS.RES.
- DIALOGS.RC, which contains the resource statements for the bitmaps and the icons that are used in the DIALOGS.DLG and DIALOGS.H sample files.
- DLGPROCS.C, which contains the C code for sample dialog box procedures associated with each template. You modify this file to update the procedures for the dialog box templates that you edited using the Dialog Editor. You can also add new dialog procedures to this file.
- DIALOGS.H, which contains the dialog control identification number definitions. This file is updated automatically when you use the Dialog Editor.
- MSCUISTF.DLL, which is the customized user interface library created from the preceding files.
- CUI.H, which is the header file for the Setup toolkit dialog box C functions.
- MAKEFILE, which you can use to compile the preceding files.

Use these files with the Dialog Editor, the C compiler, and the linker to create your own dialog boxes.

To customize the dialog box templates for your installation program, follow these steps:

1. Use the Dialog Editor to modify DIALOGS.RES. For each dialog box that you need, choose a template that closely resembles the design of the dialog box and modify it as necessary.

   When you save your changes, the Dialog Editor updates the script statements in DIALOGS.DLG and the constants in DIALOGS.H.

2. If necessary, edit DLGPROCS.C to update the dialog box procedures for the templates that you modified.

   The Setup toolkit provides a set of functions that you can use in the dialog procedures in addition to the standard Windows functions. These Setup functions are written in C and are described in detail in the following section.

   Note: DLGPROCS.C uses the Symbol Table, a temporary storage area in memory, to transfer information between dialog box procedures and Setup. The comments embedded in the code for each dialog box procedure identify the symbols the procedure uses. For more information about the Symbol Table, see Chapter 3, "Creating a setup Script."

   Note: Two constants are defined in CUI.H that can directly affect dialog box procedures: STF_REINITDIALOG and STF_ACTIVATEAPP. If your Setup script has called the **UIStartDlg** function for a dialog box that is already on top of the dialog box stack (that is, to update the contents of the dialog box), Windows returns the STF_REINITDIALOG constant to let you know. If the user has switched to another application during your installation, Windows returns the STF_ACTIVATEAPP constant to let you know when the user has switched back to Setup.

3. Compile the dialog box procedures with MAKEFILE to create the .DLL file with your changes.

   MAKEFILE compiles the dialog procedures and dialog resources into MSCUISTF.DLL. You then use the name of the .DLL file, the resource identification numbers of the dialog boxes, the help description resource identification numbers, and the names of the associated dialog box procedures as parameters for the **UIStartDlg** function, which is called from the .MST script file.

Note: All dialog boxes must have a style of WS_CHILD in order to run properly.

The following table provides a quick guide to the dialog box procedures provided in DLGPROCS.C and their associated templates provided in DIALOGS.DLG. Each procedure in DLGPROCS.C is preceded with comments indicating what the procedure does and what symbols it uses. You can also preview each template using the Dialog Editor to open DIALOGS.RES.

Procedure name	Used for	Associated template(s)
FCheckDlgProc	Dialog boxes with one to ten check boxes.	CHECK
FCustInstDlgProc	Dialog boxes with one to ten check boxes. Each check box can have an associated push button. This procedure also supports a push button that displays the current installation path and allows the user to change it.	CUSTINST
FEditDlgProc	Dialog boxes that contain one edit control.	DESTPATH
FHelpDlgProc	Dialog boxes that contain help messages.	APPHELP
FInfoDlgProc	Dialog boxes that present information to the user. The user must respond.	WELCOME
FInfo0DlgProc	This procedure is the same as FInfoDlgProc but does not support an Exit button.	BADPATH, EXITFAILURE, EXITQUIT, EXITSUCCESS, TOOBIG
FListDlgProc	Dialog boxes that contain one single-selection list box.	SINGLELIST
FModelessDlgPro	Dialog boxes that present information to the user during lengthy operations. The user does not have to respond.	MODELESS

FMultiDlgProc	Dialog boxes with one multiple-selection list box.	EXTENDEDLIST MULTILIST
FQuitDlgProc	Dialog boxes that let the user either quit or resume the installation process.	ASKQUIT
FRadioDlgProc	Dialog boxes with a single group of one to ten radio buttons.	OPTIONS

Table 2.1 Dialog Box Procedures

For information about the **UIStartDlg** function, see Chapter 5, "Setup Script Procedures." For information about using the Dialog Editor, refer to "Designing Dialog Boxes: The Dialog Editor" in *Microsoft Windows Programming Utilities.*

Dialog Box Functions

Assert

void Assert(*fValue***)**
BOOL *fValue* /* Boolean value to assert */

The **Assert** function asserts whether a boolean expression is true when the DEBUG compiler flag is defined. When DEBUG is not defined, the function is ignored.

Argument

fValue

Specifies the Boolean value that you want to assert.

Return Value

This function has no return value.

Comments

If the asserted value is true, the **Assert** function simply returns. If the asserted value is false, the program displays a message box containing the source filename and the line number of the failed assertion. You must click OK to continue.

CbGetListItem

unsigned CbGetListItem(*szSym, n, szItem, cbItemMax*)
LPSTR *szSym* /* symbol name */
unsigned *n* /* index to the item in the list */
LPSTR *szItem* /* buffer */
unsigned *cbItemMax* /* buffer size */

The **CbGetListItem** function copies the specified list item into the provided buffer as a zero-terminated string.

Arguments

szSym

Specifies the name of the symbol whose associated value is the list you want.

n

Specifies the index number (one-based) for the list item you want to copy into the buffer.

szItem

Specifies the buffer for the copy of the list item.

cbItemMax

Specifies the size of the buffer.

Return Value

If the function is successful, the return value is the length (in bytes) of the full string of the specified list item. If *szSym* or *n* doesn't exist, the return value is zero and the empty string is placed in the buffer.

Comments

If you specify a buffer that is smaller than the length of the symbol value, the **CbGetListItem** function will copy in as many characters as will fit (including a trailing zero). However, the return value will be the full length of the string.

See Also

UsGetListLength, FAddListItem, FReplaceListItem. For information about setting symbol values in the Symbol Table, see **FSetSymbolValue.**

CbGetSymbolValue

unsigned CbGetSymbolValue(*szSymbol, szValue, cbMaxLen*)
LPSTR *szSymbol* /* symbol */
LPSTR *szValue* /* value */
unsigned *cbMaxLen* /* buffer size */

The **CbGetSymbolValue** function copies the specified value from the symbol-value pair in the Symbol Table into a buffer.

Arguments

szSymbol

Specifies the name of the symbol whose value you want to copy into the buffer.

szValue

Specifies a buffer for the value associated with the symbol.

cbMaxLen

Specifies the length of the buffer.

Return Value

If the function is successful in copying the value into the buffer, the return value is the length of the value string (excluding the terminating zero). If the symbol does not exist or is an empty string, the return value is zero.

Comments

If you specify a buffer length that is smaller than the length of the value, the function will copy in as many characters as will fit (including a trailing zero). However, the return value will be the full length of the specified value.

See Also

FSetSymbolValue, FRemoveSymbol

DoMsgBox

int DoMsgBox(*szText, szCaption, wType*)
LPSTR *szText* /* message text*/
LPSTR *szCaption* /* dialog box caption */
word *wType* /* message box type */

The **DoMsgBox** function launches a Windows message box of the style specified by *wType*. The return value is the identification for the user's response, such as IDOK.

Arguments

szText

Specifies the text you want to appear in the message box.

szCaption

Specifies the caption for the message box.

wType

Specifies the contents of the message box. *wType* can be a combination of values.

Return Value

The return value is the value of the button control that the user selected (such as IDOK). If there is not enough memory to create the message box, the return value is zero.

Comments

This function is similar to the Windows **MessageBox** function. The valid message box values and control values are the same as for the **MessageBox** function.

See Also

For more information on the **MessageBox** function, see the *Microsoft Windows Programmer's Reference.*

FAddListItem

BOOL FAddListItem(*szSym, szItem*)
LPSTR *szSym* /* symbol name */
LPSTR *szItem* /* item */

The **FAddListItem** function adds the specified item to the end of the list of items associated with the symbol in the Symbol Table.

Arguments

szSym

Points to a zero-terminated string that identifies the symbol.

szItem

Points to a zero-terminated string that identifies the item you want to add to the list associated with *szSym*.

Return Value

If the function is successful in adding the item, the return value is **fTrue** (one). Otherwise, the return value is **fFalse** (zero).

Comments

You can initialize an empty list by setting its associated symbol value to "" with the **FSetSymbolValue** function. You can then add values to the list using **FAddListItem**.

See Also

FReplaceListItem, CbGetListItem, UsGetListLength

FCloseHelp

BOOL FCloseHelp()

The **FCloseHelp** function closes the currently open help dialog box, if one exists.

Return Value

The return value is **fTrue** (one) if the help dialog box is successfully closed. Otherwise, it is **fFalse** (zero).

See Also

HdlgShowHelp

FHandleOOM

BOOL FHandleOOM()

The **FHandleOOM** function displays a message box when an "Out Of Memory" error occurs and waits for a user response. This function lets the user switch out of the current application and free up some memory by closing other applications.

Return Value

If the user presses the RETRY button, the return value is **fTrue** (one). If the user presses the ABORT button, the return value is **fFalse** (zero).

FRemoveSymbol

BOOL FRemoveSymbol(*szSym***)**
LPSTR *szSym* /* symbol name*/

The **FRemoveSymbol** function removes a symbol and its associated value from the Symbol Table.

Argument

szSym

Specifies the name of the symbol you want to remove.

Return Value

If the function is successful, the return value is **fTrue** (one). Otherwise, the return value is **fFalse** (zero).

See Also

FSetSymbolValue

FReplaceListItem

BOOL FReplaceListItem(*szSym, n, szItem*)
LPSTR *szSym* /* symbol name */
unsigned *n* /* index to list item */
LPSTR *szItem* /* item */

The **FReplaceListItem** function replaces the specified item in the list of items associated with the symbol in the Symbol Table.

Arguments

szSym

Specifies the name of the symbol. *szSym* must be a zero-terminated string.

n

Specifies the index number (one-based) of the item you want to replace.

szItem

Identifies the item you want to use to replace the existing item. *szItem* must be a zero-terminated string.

Return Value

If the function successfully replaces the item, the return value is **fTrue** (one). If the index is invalid or the application is out of memory, the return value is **fFalse** (zero).

See Also

FAddListItem, CbGetListItem, UsGetListLength

FSetSymbolValue

BOOL FSetSymbolValue(*szSymbol, szValue*)
LPSTR *szSymbol* /* symbol */
LPSTR *szValue* /* value */

The **FSetSymbolValue** function inserts a new symbol-value pair into the Symbol Table. If the symbol already exists, the function replaces the symbol's associated value.

Arguments

szSymbol

Specifies the name of the symbol you want to create or whose associated value you want to replace.

szValue

Specifies the value you want to add or replace. If *szValue* is NULL, an empty string is added or used to replace the existing value.

Return Value

If the function is successful, the return value is **fTrue** (one). If the application is out of memory, the return value is **fFalse** (zero).

See Also

CbGetSymbolValue, FRemoveSymbol

HdlgShowHelp

HWND HdlgShowHelp ()

The **HdlgShowHelp** function displays the help dialog box for the dialog box that is currently on the top of the dialog box stack.

Return Value

The return value is the handle to the help dialog. If the help dialog does not exist and cannot be created, the return value is NULL.

See Also

FCloseHelp

ReactivateSetupScript

void ReactivateSetupScript()

The **ReactivateSetupScript** function returns control to the Setup script.

Return Value

This function has no return value.

Comments

This function is the vehicle for returning from a dialog box procedure to the Setup script.

UsGetListLength

unsigned UsGetListLength(*szSym***)**
LPSTR *szSym* /* symbol name */

The **UsGetListLength** function determines the number of items in the list associated with the specified symbol.

Argument

szSym

Specifies the name of the symbol. *szSym* must be a zero-terminated string.

Return Value

The return value is the number of items in the list associated with the symbol.

See Also

CbGetListItem, FReplaceListItem

Chapter 3: Creating a Setup Script

Once you've identified the list of installable files and used the Windows Dialog Editor to design dialog boxes and message boxes, you're ready to create your installation script file. A script file contains the procedure calls that Setup uses to install your product on the user's system.

The Setup toolkit comes with three sets of sample files that you can modify to create your own installation script. You'll also want to modify the sample SETUP.LST file so that the Setup driver can find the appropriate .DLLs and other supporting files that are needed to run your installation. This chapter will walk you through the process of creating your own script and SETUP.LST file.

Start by reviewing the files that make up the Setup toolkit. The following table lists the name and purpose of each file in the toolkit.

Filename	**Description**
SETUP.EXE (required)	The bootstrapper program that copies the files identified in SETUP.LST into a temporary directory on the user's hard disk. These files run the installation and include _MSTEST.EXE, _MSSETUP.EXE, and any .DLL files. You should always include SETUP.EXE on the first installation disk. SETUP.EXE should not be compressed.
SETUP.LST (required)	A text file that contains the list of files SETUP.EXE copies into the temporary directory on the user's hard disk. You should always include SETUP.LST on the first installation disk. SETUP.LST should not be compressed.
_MSTEST.EXE (required)	A limited, run-time version of Microsoft Test that Setup uses as its driver. _MSTEST.EXE is the interpreter of your installation script. You should always include _MSTEST.EXE on the first installation disk.

_MSSETUP.EXE (optional)	An MS-DOS program that reads the MSSETUP.BAT file and updates system files (that is, files that have the SYSTEM attribute and may be in use when Windows is running). The Setup ExitExecRestart function shuts down Windows, runs _MSSETUP.EXE to update files listed in MSSETUP.BAT, and restarts Windows. If you plan to install or update any system files, you must include _MSSETUP.EXE on the first installation disk.
Setup .DLL files: (required)	These .DLL files contain the code for the Setup procedures that you call from your script file. You should list them in the SETUP.LST file and include them on the first installation disk.
MSCOMSTF.DLL	The common library, which contains supporting routines for the other .DLL files.
MSCUISTF.DLL	The customized user interface library. You modify this .DLL file when you create dialog boxes and message boxes. For more information about this process, see Chapter 2, "Designing Dialog Boxes."
MSDETSTF.DLL	The detection library, which contains procedures that return information about the user's system, such as the version of Windows.
MSINSSTF.DLL	The install library, which contains procedures that install files on the user's hard disk.
MSUILSTF.DLL	The user interface library, which contains procedures that manipulate the user interface, such as displaying a dialog box or deleting a dialog box from the dialog stack.
MSSHLSTF.DLL	The shell library, which contains the routines that manage the frame window.
VER.DLL	The version checking .DLL file that is required if the user can install the product on a Windows version 3.0 system.

Setup include files:	The .INC files define variables and declare the subroutines and functions that you call from your script. You include these files in your script (.MST) file as necessary. Any files that you have listed in the .MST file must also be included on the first installation disk. Note: If you include both MSSHARED.INC and MSREGDB.INC, you must place the '$INCLUDE statement for MSREGDB.INC before the statement for MSSHARED.INC.
SETUPAPI.INC (required)	The common API include file, which contains definitions of constants used by Setup and declarations for the most commonly used Setup procedures. You must include SETUPAPI.INC in your script and on the first installation disk.
MSDETECT.INC (optional)	The detection API include file, which contains declarations of functions that return information about the user's system, such as the number of disk drives and so on. The more commonly used detection routines, such as the ones that query for the Windows version, are declared in SETUPAPI.INC. Include MSDETECT.INC in your script if you intend to use any of the functions it declares.
MSREGDB.INC (optional)	The Registration Database API include file, which contains declarations for the Setup procedures that read and write to the Registration Database. Include MSREGDB.INC in your script if you intend to use any of the functions it declares.
MSSHARED.INC (optional)	The shared files API include file, which contains declarations for procedures that install or update shared files. Include MSSHARED.INC in your script if you intend to use any of the functions it declares.
Sample script files:	The script (.MST) files contain sample code that you can modify to create your own script. You must list both your .MST and .INF files in SETUP.LST, and you must include them on the first installation disk. (continued next page)

Sample script files: (continued)	To create your script, choose the .MST file that most closely resembles your installation and modify it. Or, combine sample code from more than one file. The corresponding .INF files are provided for information only; use the Disk Layout Utilities to create the .INF file for your installation script. For more information, see Chapter 4, "Using the Disk Layout Utilities."
SAMPLE1.MST	The sample script file for a simple, straightforward installation. This file provides sample Microsoft Test debug code and asks the user to choose which files will be installed.
SAMPLE2.MST	The sample script file for a moderately complex installation. The file asks the user to choose from two sets of optional files for the installation, to enter the name of the installation directory, and so on. The script also checks the available disk space on the user's hard disk for installing the files.
SAMPLE3.MST	The same script file for an installation that contains shared files. The script checks to see if the shared file is a newer version than the one on the user's system and then adds it, if necessary, to the global copy list. The script also updates the Registration Database. For more information about shared files, see "Installing Shared Files or System File," later in this chapter
TESTDRVR.HLP	A Windows online help file that contains information about Microsoft Test. Use this file to understand the function of the Test commands that are included in the sample script files. You should not include this file on your installation disks.
README.TXT	A text file that contains information about Setup that could not be included in this manual. You should read this file before creating an installation script. Do not inlude this file on your installation disks.

Table 3.1 Setup Toolkit Files

To run the sample files and see them from the user's perspective, choose the Run command from the File menu. Type the name of the Microsoft Test interpreter and the sample filename in the command line, and click OK. For example, the command line to run SAMPLE1.MST would be:

_mstest.exe sample1.mst

Note: Make sure that you have VER.DLL in your path. If not, you may want to put a copy of it in the Setup toolkit directory. If you allow your product to be run under Windows version 3.0, you must also include VER.DLL in SETUP.LST and on the first installation disk.

The rest of this chapter explains the process of creating an installation script and defining which files you need to include on the first installation disk.

Choosing and Modifying a Sample Script File

Up to this point, you have defined the list of files you will be installing and designed the dialog boxes you will need. Your focus has been on identifying the choices you will ask the user to make–for example, to determine the directory in which you will install your product.

To create a script file, your focus must now shift slightly. When you create the code that handles the dialog boxes and installs the product files, you must pay attention to designing a safe and efficient installation process. Therefore, before you start modifying one of the Setup sample script files, answer these questions:

- How much disk space will the installation use? Is it a significant amount? If so, you will need to ensure that the user's hard disk has enough disk space available. Also, if your product includes optional files that the user has chosen, you may want to inform the user how much disk space each optional file will use.

- Does your product have minimum hardware and system software requirements? If so, you may want to check the version of MS-DOS or Windows, or check for the existence of, for example, a math coprocessor. If the user's system falls below your minimum requirements, you will want to display a warning message.

- Will you be installing shareable files? For example, does your product require a spelling dictionary that the user already may have installed with another product? If so, you will need to take some special steps to handle these files.

- Will you be installing any system files? If so, you may need to use special Setup functions to exit Windows and update files that would be in use while Windows is running.

- Will your installation need to update WIN.INI or AUTOEXEC.BAT? If so, you will want to let the user decide whether to have these files updated automatically.

- What other parts of your installation have associated risks? Do you need to post warnings of any kind for the user? Do you need to check the validity of paths and filenames the user enters?

The answers to these questions provide the information you need to design the code for your script file. You should design the installation script just as you would any other program: Identify the types of routines you will need, determine the logical order in which those routines should occur, and then draw a flowchart of the installation process. The answers to the above questions will also help you choose which sample script file to modify.

The Basic Components of a Sample Script File

Each sample file contains several basic components that, once understood, are easy to modify to match your installation's needs. This section describes those components, using SAMPLE1.MST for the code illustrations.

To execute a sample file, choose the Run command from the File menu, and type

_mtest.exe sample*n*.mst

where *n* is the number of the sample file. (You may have to copy VER.DLL into the Setup toolkit directory for the sample files to run successfully.)

Debug Code

At the beginning of each sample, you can define a DEBUG flag that Microsoft Test recognizes. You can then use the DEBUG flag to include debug code in your script for testing purposes. SAMPLE1.MST defines the DEBUG flag as follows:

```
'$DEFINE DEBUG
```

The following code appears later in the script:

```
'$IFDEF DEBUG
.
.
.
'$ENDIF 'DEBUG
```

These are Microsoft Test metacommands. You do not have to include them in your script, but they can be very useful. For more information about these commands and their uses, refer to the Windows online help file, TESTDRVR.HLP, that comes with the Setup toolkit.

Note: You should remove the **'$DEFINE DEBUG** line from your script (.MST) file before you ship your product.

Include Files

The appropriate .INC files are listed at the top of each sample file. SAMPLE1.MST uses the following commands to include SETUPAPI.INC and MSDETECT.INC:

```
'$INCLUDE 'setupapi.inc'
'$INCLUDE 'msdetect.inc'
```

These are Microsoft Test commands. Edit these lines to include the files that your installation will need, based on the types of Setup procedures that you call in your script.

For a description of each .INC file, see Table 3.1. For more information about the syntax of Test commands, refer to the Windows online help file, TESTDRVR.HLP, included in the Setup toolkit.

Note: If you have included the **'$DEFINE DEBUG** line in your script file, the .INC files will have argument checking enabled.

Dialog Box Constants

Each sample file declares constants for the dialog boxes it will use. The code for SAMPLE1.MST looks like this:

```
CONST WELCOME        = 100
CONST ASKQUIT        = 200
CONST DESTPATH       = 300
CONST EXITFAILURE    = 400
CONST EXITQUIT       = 600
CONST EXITSUCCESS    = 700
CONST OPTIONS        = 800
CONST APPHELP        = 900
```

These constants represent the resource identification numbers for the dialog boxes you will use in your installation. Edit this list to match the dialog boxes you designed using the Windows Dialog Editor.

Initialization

The initialization section of each sample sets the background bitmap and title for the Setup frame (or main) window. It also initializes variables, retrieves information from the Symbol Table, and reads the .INF file. Edit this section to include the following:

- The name of the bitmap logo file to appear in the background of the frame window
- The title to appear in the frame window
- The name of the .DLL file that contains the procedures for your dialog boxes
- The path and name of your .INF file

Welcome and Other Dialog Boxes

The next sections of the sample files provide code that display the dialog boxes that ask the user to make choices for the installation. For example, SAMPLE1.MST uses the following code to display the Welcome dialog box:

```
WELCOME:
  sz$ = UIStartDlg(CUIDLL$,WELCOME,"FInfoDlgProc",
   APPHELP, HELPPROC$)
  IF sz$ = "CONTINUE" THEN
     UIPop 1
  ELSE
     GOSUB ASKQUIT
     GOTO WELCOME
  ENDIF
```

Edit these sections of the script as necessary to include the code for your dialog boxes.

Install Subroutine

SAMPLE1.MST declares an **Install** subroutine at the beginning of the file. The purpose of this routine is to build the global copy list and perform the installation tasks. The **Install** subroutine for SAMPLE1.MST performs the following tasks:

- Opens and writes to the installation log file
- Builds the global copy list based on the files listed in the .INF file and the choices the user has made
- Installs the files on the user's hard disk
- Updates WIN.INI
- Creates a Program Manager group and item for the product

Edit this routine to include the code necessary to install your product. SAMPLE2.MST and SAMPLE3.MST handle these tasks differently because they represent more sophisticated installations. Depending on the complexity of your installation, the **Install** subroutine can be the bulk of your installation script. Therefore, you may want to break this subroutine into smaller, more manageable chunks. For an example of a more complicated installation, see the Install section of SAMPLE2.MST.

Modifying SETUP.LST to Match Your Script File

When you have modified a sample script file, you must create a version of SETUP.LST that matches your script. SETUP.LST must contain two sections: Params and Files. The Params section looks like this:

```
[Params]
  WndTitle    = Microsoft Setup
  WndMess     = Initializing Setup...
  TmpDirSize  = 500
  TmpDirName  = ms-setup.t
  CmdLine     = _mstest sample1.mst /C "/S %s %s"
  DrvModName  = DSHELL
```

To edit the Params section of SETUP.LST to match your script:

1. If you want, change the value of *WndTitle*.

   This text displays in the title bar of the Setup initialization window while SETUP.EXE is copying the files into the temporary directory on the user's hard disk.

2. If you want, change the value of *WndMess*.

   This message displays in the center of the client area of the Setup initialization window.

3. Set the value of *TmpDirSize* to an amount (in kilobytes) that will accommodate the files SETUP.EXE copies into the temporary directory.

   You can calculate this value by adding the sizes of the files listed in the Files section of SETUP.LST, dividing the result by 1024, and rounding it to a whole number.

4. If you want, change the value of *TmpDirName* to the desired temporary directory name.

   The name you choose must accept one character and still be a valid name.

5. Edit the value of *CmdLine* to include the name of your script (.MST) file.

Note: Do not change the last line that sets the value of *DrvModName*.

The Files section of SETUP.LST contains a list of the files that SETUP.EXE should copy into the temporary directory. At a minimum, this list must include the following files:

- Your script (.MST) file
- Your .INF file
- SETUPAPI.INC
- All .DLL files including MSCUISTF.DLL
- _MSTEST.EXE

The list can also include:

- Any additional .INC files that you included in your script
- Any additional custom .DLL files that include procedures you called in your script
- _MSSETUP.EXE, the MS-DOS program that you can use to update system files

The Files section of SETUP.LST looks like this:

```
[Files]
  sample1.mst   = sample1.mst
  sample1.inf   = sample1.inf
  setupapi.inc  = setupapi.inc
  msdetect.inc  = msdetect.inc
  mscomstf.dll  = mscomstf.dll
  msinsstf.dll  = msinsstf.dll
  msuilstf.dll  = msuilstf.dll
  msshlstf.dll  = msshlstf.dll
  mscuistf.dll  = mscuistf.dll
  msdetstf.dll  = msdetstf.dll
  _mstest.exe   = _mstest.exe
  _mssetup.exe  = _mssetup.exe
```

The filenames on the left side of the equal sign (=) are the names of the files that appear on the first installation disk. The filenames on the right side of the equal sign are the names to which the files will be copied in the temporary directory on the user's hard disk. (Typically, the filenames will differ if you have compressed the files for distribution. Compression is recommended for all but SETUP.EXE and SETUP.LST.)

Edit the filenames (potentially, on both sides) to match the files you need for your installation. For more information about each of the Setup .DLL and .INC files, see Table 3.1.

Using the Symbol Table

Whether your installation is simple or complex, you will probably use the Symbol Table to store values. The Symbol Table is a temporary storage area in memory that contains a table of text symbols and their associated text values. Setup uses the Symbol Table to store information such as directory names and data that is passed between the script and the .DLL files.

Setup automatically creates and sets three symbols that you can use:

- STF_SRCDIR, which is the source directory
- STF_CWDDIR, which is the current working directory or the temporary directory for Setup
- STF_SRCINFPATH, which is the path for the .INF file (usually empty, unless you or the user supplied it as part of the SETUP.EXE command line)

For example, before reading the .INF file, SAMPLE1.MST uses the STF_CWDDIR symbol to create the path and filename for the .INF file:

```
szInf$ = GetSymbolValue("STF_CWDDIR") +
"SAMPLE1.INF"
```

The procedures in MSCUISTF.DLL (the customized dialog box routines) also use the Symbol Table to store the user's responses. For example, SAMPLE1.MST uses the following line of code to retrieve the value of the button the user chose in a dialog box:

```
OPTCUR$ = GetSymbolValue("ButtonChecked")
```

You'll see other uses of the Symbol Table interspersed throughout the sample code. Use the Symbol Table to pass data between your script and the .DLL files.

Note: To conserve memory, clear all Symbol Table strings after you use them.

Creating Customized .DLL Files

Depending on the special needs of your installation, you may want to create your own customized routines in a special .DLL file. Setup can easily accommodate these routines. Simply create the .DLL file as you would any other .DLL file. Then include it in the list of .DLL files in the SETUP.LST file.

To access functions in your custom .DLL files, you must declare the functions in your script (.MST) file or in an include file. For example, to access the **MyFunc** function, you must declare it as follows:

```
DECLARE FUNCTION MyFunc LIB "My.dll" (arg1%,
arg2%) AS INTEGER
```

For more information about defining your own library functions, look at the declarations in the Setup .INC files or refer to the online help file, TESTDRVR.HLP, that comes with the Setup toolkit.

Installing Shared Files or System Files

If you are planning to install shared files or system files, you must handle that part of the installation with extra care. This section describes some of the issues involved with installing shared files or system files and how Setup handles them.

Shared Files

A shared file is a file that may be used by more than one application on the user's system. For example, your company may have two products that use the same spelling dictionary. If the user has already installed one of the products,

that dictionary file may already be installed. Furthermore, if the user has that product running during the installation process, the dictionary file may be in use and can't be updated.

To handle this problem, the **CopyFilesInCopyList** procedure checks each file listed in your .INF file with the SHARED attribute to see if the file is in use. If so, Setup displays an error message. The user can fix the problem by:

- Switching out of Setup, closing the other application, switching back into Setup, and then choosing the Retry button.
- Exiting Setup and rerunning the installation after the other application is closed.
- Ignoring the message. If the file is marked as vital, Setup will display another error message. If the file is not vital, Setup skips it but continues to copy other files in the list.

System Files

A system file is a file that may be in use by Windows when Windows is running. The **CopyFilesInCopyList** procedure checks each file listed in your .INF file with the SYSTEM attribute to see if the file is in use. If so, the procedure copies the file to a temporary location (the restart directory) and adds a command to the _MSSETUP.BAT file.

Toward the end of the installation, your script should call the **RestartListEmpty** function. If the function returns zero, there are system files that need to be updated. You should inform the user about this and then use the **ExitExecRestart** function to shut down Windows, update the files, and restart Windows. The **ExitExecRestart** function uses the MS-DOS-based program _MSSETUP.EXE to copy the files listed in _MSSETUP.BAT. If you use these procedures, you must list _MSSETUP.EXE in the SETUP.LST file and include it on the first installation disk.

Note: Windows version 3.0 does not support the **ExitExecRestart** function.

For more information about the **CopyFilesInCopyList**, **RestartListEmpty**, and **ExitExecRestart** procedures, see Chapter 5, "Setup Script Procedures."

Chapter 4: Using the Disk Layout Utilities

The Disk Layout Utilities automate tedious, error-prone tasks by taking your project files and creating efficiently organized disk images for your product installation. As part of this process, the Disk Layout Utilities also create the .INF file.

Typically, you will use the Disk Layout Utilities in the following manner:

- As soon as you have a distributable product release, even if it is planned for internal release within your company, use the Disk Layout Utilities to create the .INF file and the disk images.
- For each subsequent release, run the Disk Layout Utilities to update the directory of disk images for any files you may have added or changed.

After the first time you use the Disk Layout Utilities to create a software release, the programs remember which files have already been included and tell you if you have added any new files. The Disk Layout Utilities also update disk images and compressed files only when the files change.

This chapter describes the disk layout process and explains the use of the Dsklayt and Dsklayt2 programs to create a layout file, disk images, and the .INF file.

Understanding the Disk Layout Process

The Disk Layout Utilities consist of two parts:

- A Windows-based program (Dsklayt) that you use to specify the properties for all files that will go into your product release
- An MS-DOS-based program (Dsklayt2) that creates the disk images and the .INF file for the installation

You use Dsklayt to create a layout file containing file specifications. Dsklayt2 then uses the directives in the layout file to create the disk images and the .INF file. You can use Dsklayt2 as part of your product build process.

To run Dsklayt:

1. In File Manager, choose Run from the File menu.

2. In the Command Line box, type \DSKLAYT.TLS\DSKLAYT, and then click OK.

   The main window for Dsklayt appears with a dialog box open, as shown in Figure 4.1.

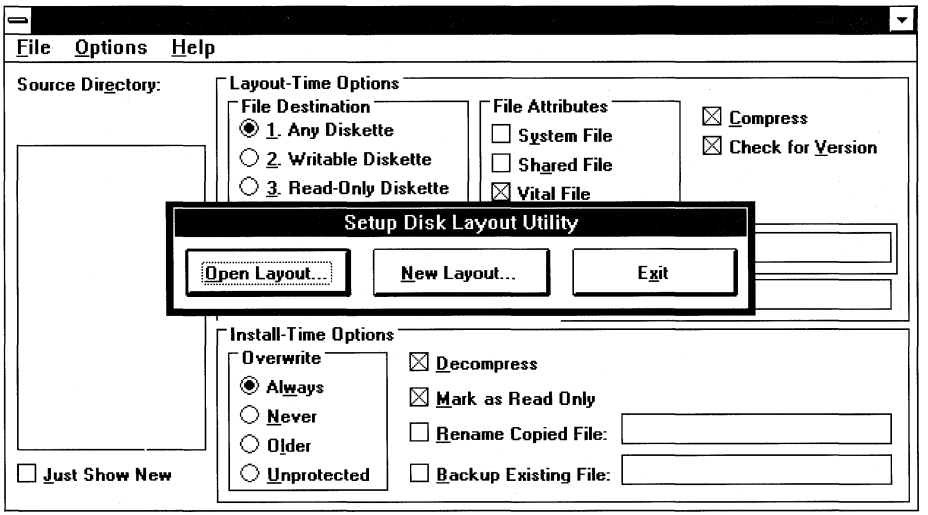

Figure 4.1

3. In the Microsoft Disk Layout Utilities dialog box, click either Open Layout to open an existing layout file or New Layout to create a new layout file.

4. If you are opening an existing layout file, specify the name of the file and click OK. If you are creating a new layout file, specify the directory where your product files are stored and click OK.

The files from the source directory appear in the list box on the left side of the main window. You can then select one or more files from the list box and specify their properties.

Using the Disk Layout Utilities Commands

Dsklayt has a main window and three menus–File, Options, and Help. This section describes the contents of the main window and each of the commands on the menus.

Dsklayt Main Window

You can use the options in the main window to set most of the file specifications for your installation. To set specifications for one file or for a group of files, you simply select the file(s) you want to affect from the list box and specify the options you want the file(s) to have.

Figure 4.2 illustrates the Dsklayt main window. The options are divided roughly into two types: *layout time* options, which affect how the files are stored on the installation disks, and *install time* options, which affect how the files are copied onto the user's hard disk. Each option is described below.

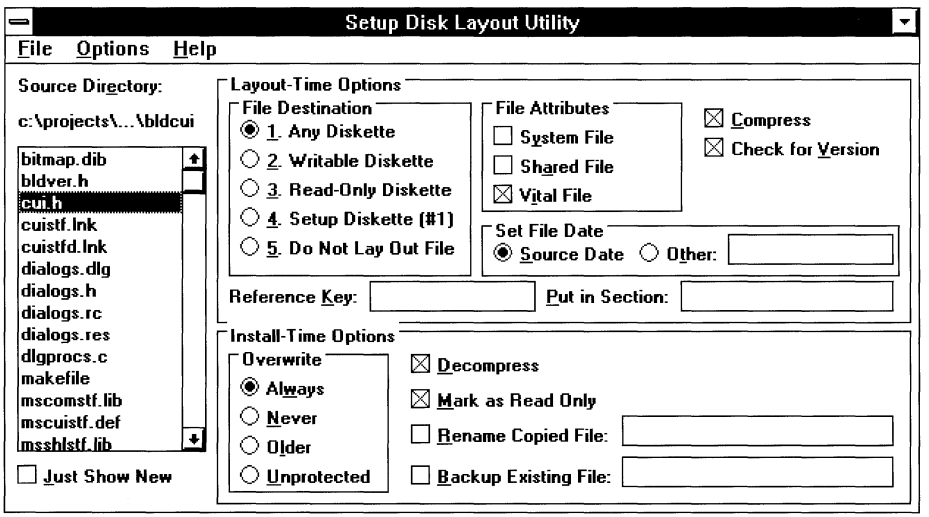

Figure 4.2

Source Directory: Displays the name of the top-level directory for your product files.

List box: Displays all the files in the source directory and its subdirectories. Choose files from this list to set their attributes. You can choose:

- A single file, by clicking on it.
- A contiguous range of files, by clicking on the first file, holding down the SHIFT key, and then clicking on the last file.
- A discontiguous range of files, by holding down the CTRL key and clicking on each file.

Just Show New: Checking this box displays only the files that are new or have been updated since you last created the layout file. Use this option when you are doing successive product releases and only need to add specifications for the new or changed source files.

Layout Time Options:

File Destination: Determines the type of disk on which the selected file can be placed. Choose one of five options:

1. Any Diskette: Indicates that the selected file can go on any diskette in the installation set. This option is the default.

2. Writable Diskette: Indicates that the selected file must go on a disk that Setup can write to.

3. Read-Only Diskette: Indicates that the selected file must go on a write-protected disk. For example, you may want to store uncompressed binary files (.EXE files) that might be targets of viruses on a read-only diskette.

4. Setup Diskette (#1): Indicates that the selected file must go on the first disk in the installation disk set. For example, you would choose this option for your Setup script file.

5. Do Not Lay Out File: Indicates that the selected file should not be placed on an installation disk. Use this option for files that reside in your project directories but are not part of the product installation, such as source code management files.

File Attributes: Marks a file in the layout file as having one or more attributes. Check one or more of the following:

- **System File:** The file is a system file, such as WINHELP.EXE or GDI.EXE.
- **Shared File:** The selected file may be shared by one or more applications, such as a common code library that ships with all of your company products.
- **Vital File:** The installation will fail unless the selected file is installed successfully. This option is the default.

Set File Date: Specifies the date stamp used for the file when it is copied into a disk image directory by Dsklayt2 and when Setup copies the file onto the user's hard disk. Choose one of the following options:

- **Source Date:** Uses the date of the installable file.
- **Other:** Uses the date you specify in the adjacent text box (in the format YYYY-MM-DD). Use this option when you want the date on all installed files to be a significant date, such as the product release date.

Compress: Determines whether the selected file should be compressed by Dsklayt2.

Check For Version: Tells Dsklayt2 to use VER.DLL to check the source file for the existence of a version resource. If the version resource exists, Dsklayt2 puts this information into the .INF file. Otherwise, Dsklayt2 issues a warning and leaves this portion of the file description blank in the .INF file.

Reference Key: Specifies a unique reference for the selected file. Use this option when you want the Setup script to determine whether to install the selected file based on information available at the time of the installation. For example, the type of monitor on the user's system could affect the files you

install for your product. You can also use this option when you want to display reference keys rather than filenames in the dialog boxes displayed by the installation, because the keys are more descriptive than the filenames.

Put In Section: Specifies a unique .INF section name for the selected file. Use this field when you want installation files organized by categories rather than all listed in the default "Files" section of the .INF file.

Install-Time Options:

Overwrite: Specifies what should happen if the selected file already exists on the user's hard disk. Choose one of the following options:

- **Always**: The installed file will always overwrite any existing version of the file.
- **Never:** The installed file will never overwrite an existing version of the file.
- **Older:** The installed file will overwrite an existing version of the file only if the existing version is older. Setup will look for version information; if none exists, it will use the file dates to determine which file is older.
- **Unprotected:** The installed file will overwrite an existing version of the file only if the existing version has an MS-DOS file attribute of "Write."

Decompress: Indicates that Setup should check to see if the source file is compressed and, if so, decompress it before copying it onto the user's hard disk. You should leave this option checked in most cases, even if the source file is not compressed.

Mark as Read Only: Indicates that you want the file to have a MS-DOS file attribute of "Read Only" when it is copied onto the user's hard disk.

Rename Copied File: Indicates that you want to rename the file to the filename you supply in the adjacent text box when it is copied onto the user's hard disk.

Backup Existing File: Indicates that you want to back up an existing version of the file to the filename you supply in the adjacent text box before copying the source file. If you type an asterisk (*), Setup will back up the file to the same name with a .BAK extension.

File menu

New
Creates a new, untitled layout file.

Open
Displays a dialog box that you can use to open an existing layout file so that you can update it. Dsklayt checks for new product files and notifies you if there are any.

Save
Saves any changes you have made to the currently open layout file.

Save As
Displays a dialog box that you can use to save the current layout file under a name you specify.

Exit
Exits Dsklayt and returns you to the most recently active window. If you have made changes but did not save them, Dsklayt prompts you to save.

Options menu

Disk Labels
Displays a dialog box (Figure 4.3) that lets you add, delete, or modify the disk labels for your installation disks.

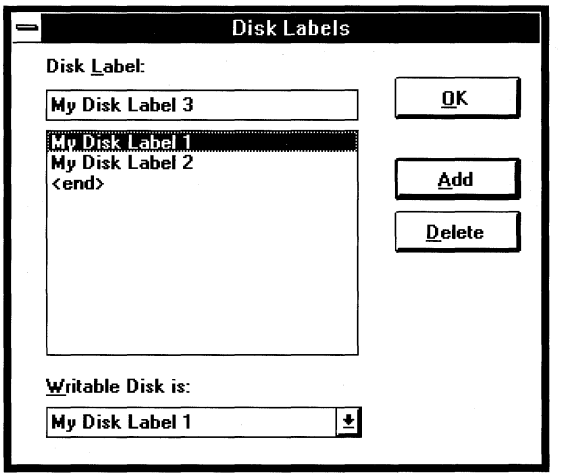

Figure 4.3

To insert a new label, select the label in the list box that you want to follow the new label, type the new label name in the text box, and then click Add. To delete a label, select it from the list box and then click Delete. To modify a label, select it from the list box, click Delete, type the correct text in the text box, and then click Add.

You can use generic labels, such as "Disk 1," until you see how the files are organized on the disks. You can then rename the labels to reflect the content of the disks.

You can specify which disk label goes on the writable disk by selecting the label in the drop-down list box at the bottom of the dialog box.

Note: The order in which you add disk labels is the order in which Dsklayt2 will apply them to the disk images.

Remove Files List

Displays a dialog box (Figure 4.4) in which you can create one or more lists of files that you want Setup to remove from the user's hard disk during product installation. Use this list to remove obsolete files or older versions of files whose names differ from the newer versions.

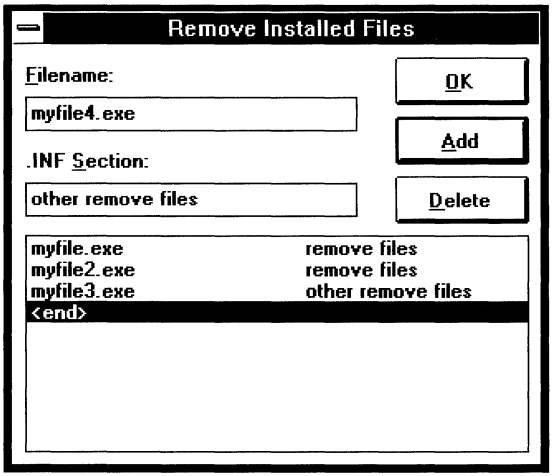

Figure 4.4

To add a file to a removal list, type its filename in the Filename box, type the name of the .INF section in the .INF Section box (if necessary), and then click Add. If you don't type a section name in the .INF Section box, Dsklayt2 adds the name of the file to the "Files" section in the .INF file.

To delete a file from a removal list, select it from the list box and click Delete. To modify a filename in a removal list, select it from the list box, click Delete, type the correct filename in the Filename box, and then click Add.

Help menu

About

Displays a dialog box that provides copyright and version information for Dsklayt.

Using the MS-DOS-Based Dsklayt2 Program

After saving your specifications in a layout file, use the MS-DOS-based Dsklayt2 program to generate disk images and the .INF file for your product

installation. Dsklayt2 reads the directives in the layout file and creates a directory of disk images that you can copy onto disks using the Diskcopy command.

The Dsklayt2 program has the following command line syntax:

Dsklayt2 *[drive:][path]layout_filename [[drive:][path]INF_filename] [options]*

Parameter	Description
drive:path for layout_filename	The drive letter and path of the layout file.
layout_filename	The name of the layout file that Dsklayt2 should read to create the disk images. This argument is required.
drive:path for INF_filename	The drive letter and path of the .INF file.
INF_filename	The name of the .INF file that you want Dsklayt2 to create. If you do not specify a filename, Dsklayt2 uses SETUP.INF.
Options:	
/k{*n*}	Specifies the type of disk Dsklayt2 should target when creating the disk images. For *n,* you can specify: ■ A standard size (360, 720, 12, or 144), N for Network ■ O *n/m* for Other, where *n/m* is the bytes per cluster and the cluster per disk, respectively. The default is 1.2 MB.
/f	Specifies that Dsklayt2 should overwrite the existing .INF file, if necessary.
/w{*n*}	Specifies the writable disk. For *n,* specify a disk number. This field overrides the specification in the layout file. If you omit this option, Dsklayt2 makes the last disk in the installation the writable disk. If you specify

the option without a disk number, Dsklayt2 assumes that all disks are read-only.

/d{*destdir*}

Specifies the destination directory for the disk images. Dsklayt2 creates a directory for each disk image, named DISK 1, DISK 2, and so on. If your product will be installed from a network drive (that is, if the **/k** option specifies a network device), all files are placed at the top-level destination directory. If you omit this option, Dsklayt2 will not create any files in the destination directory.

/c{*compdir*}

Specifies the directory where Dsklayt2 can put compressed versions of the files. If you specify the same directory each time you run Dsklayt2, the program adds or updates only those files that are new or have changed. If you omit this option, Dsklayt2 creates a COMP subdirectory in the parent of the source directory. If the source directory is the root directory, Dsklayt2 displays an error message and aborts.

/z{*compcmd*}

Specifies a compression utility that Dsklayt2 can use to compress files. For *compcmd,* specify the MS-DOS command that will execute the compression utility. Dsklayt2 will call this command with two arguments: the source directory and the destination directory. If this option is not specified, Dsklayt2 looks for COMPRESS.EXE in the path.

Chapter 5: Setup Script Procedures

This chapter describes the functions and subroutines that you can call in your installation script. They are listed in alphabetical order.

Note: All functions and subroutines are declared in the Setup .INC files. You simply call the functions and routines from your script (.MST) file. The descriptions in this chapter show the calling syntax for each procedure.

To_:	Use these procedures:
Manipulate what the user sees on the screen	**DoMsgBox** **RestoreCursor** **SetBeepingMode** **SetBitmap** **SetCopyGaugePosition** **SetSilentMode** **SetTitle** **ShowWaitCursor** **UIPop** **UIPopAll** **UIStartDlg**
Manipulate a list associated with a symbol in the Symbol Table	**AddListItem** **GetListItem** **GetListLength** **GetSymbolValue** **MakeListFromSectionKeys** **RemoveSymbol** **ReplaceListItem**
Modify the contents of the global list of installable files (the copy list)	**AddSectionFilesToCopyList** **AddSectionKeyFileToCopyList** **AddSpecialFileToCopyList** **ClearCopyList** **CopyFilesInCopyList** **DumpCopyList** **GetCopyListCost**

Control aspects of the copy list installation	**SetCopyMode** **SetDecompMode**
Manipulate billboard dialog boxes and the global billboard list	**AddBlankToBillboardList** **AddToBillboardList** **ClearBillboardList**
Manipulate a file on the user's system	**BackupFile** **CopyFile** **DoesFileExist** **FindFileInTree** **FindFileUsingFileOpen** **FindTargetOnEnvVar** **GetDateOfFile** **GetSizeOfFile** **GetVersionNthField** **GetVersionOfFile** **IsFileWritable** **RemoveFile** **RenameFile** **StampResource**
Manipulate a directory on the user's system	**CreateDir** **RemoveDir** **DoesDirExist** **GetWindowsDir**
Update an .INI file	**CreateIniKeyValue** **CreateSysIniKeyValue** **DoesIniKeyExist** **DoesIniKeyExist** **DoesIniKeyExist** **GetIniKeyString** **GetNthFieldFromIniString** **RemoveIniKey** **RemoveIniSection**
Create a Program Manager group and item for your product	**CreateProgmanGroup** **CreateProgmanItem** **ShowProgmanGroup**
Add information to or get information from the Registration Database	**CreateRegKey** **CreateRegKeyValue** **DeleteRegKey** **DoesRegKeyExist** **GetRegKeyValue** **SetRegKeyValue**

Install system resources (that may be in use while Windows is running)	**ExitExecRestart** **RestartListEmpty** **SearchForLocationForSharedFile** **SetRestartDir**
Work with MS-DOS help files	**AddDos5Help**
Create a record of what occurred during an installation	**CloseLogFile** **OpenLogFile** **SetAbout** **WriteToLogFile**
Query the user's environment	**GetConfigLastDrive** **GetConfigNumBuffers** **GetConfigNumFiles** **GetConfigRamdriveSize** **GetConfigSmartdrvSize** **GetDOSMajorVersion** **GetDOSMinorVersion** **GetEnvVariableValue** **GetFreeSpaceForDrive** **GetLocalHardDrivesList** **GetNetworkDrivesList** **GetNumWinApps** **GetParallelPortsList** **GetProcessorType** **GetRemovableDrivesList** **GetScreenHeight** **GetScreenWidth** **GetSerialPortsList** **GetTotalSpaceForDrive** **GetTypeFaceNameFromTTF** **GetValidDrivesList** **GetWindowsMajorVersion** **GetWindowsMinorVersion** **GetWindowsMode** **GetWindowsSysDir** **Has87MathChip** **HasMonochromeDisplay** **HasMouseInstalled** **IsDriveLocalHard** **IsDriveNetwork** **IsDriveRemovable** **IsDriverInConfig** **IsDriveValid** **IsWindowsShared**

Parse a date field	**GetDayFromDate** **GetHourFromDate** **GetMinuteFromDate** **GetMonthFromDate** **GetSecondFromDate**
Retrieve information about the topmost frame window	**HinstFrame** **HwndFrame**
Read or manipulate information from the .INF file	**GetSectionKeyDate** **GetSectionKeyFilename** **GetSectionKeySize** **GetSectionKeyVersion** **MakeListFromSectionKeys** **ReadInfFile** **RemoveSymbol** **ReplaceListItem**

AddBlankToBillboardList subroutine

AddBlankToBillboardList *lticks&*

The **AddBlankToBillboardList** subroutine adds a hidden dialog box to the global billboard list. The hidden dialog box destroys the previous billboard dialog box and delays the display of the next billboard dialog box.

Argument

lTicks&

Defines the amount of time you want to delay the display of the next billboard dialog box. The unit is arbitrary, relative to a total number of units.

Comments

Use this subroutine prior to calling **CopyFilesInCopyList**.

AddDos5Help subroutine

AddDos5Help *szProgName$, szProgHelp$, cmo%*

The **AddDos5Help** subroutine adds the specified program name and help description to the DOSHELP.HLP file.

Arguments

szProgName$

Specifies the program name for the help file. This name cannot start with the @ character or contain spaces or tabs. Also, the length of the name must be greater than zero and less than nine characters.

szProgHelp$

Specifies the help text string. You must specify a non-empty string for this argument. If you want to specify multiple lines of text, embed CHR$(10) for each line. This will create a line end and nine spaces as an indent for the next line.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone* for the command option flag

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

AddListItem subroutine

AddListItem *szSymbol$, szItem$*

The **AddListItem** subroutine adds a new item to the end of the list associated with the specified symbol name.

Arguments

szSymbol$

Specifies the name of the symbol in the Symbol Table to which the list is associated.

szItem$

Specifies the item that you want to add to the list.

Comments

If *szSymbol$* is previously undefined, a new list with the specified item is created and associated with the symbol name. You can create a new, empty list by using the **SetSymbolValue** subroutine and specifying the value as "".

AddSectionFilesToCopyList subroutine

AddSectionFilesToCopyList *szSection$, szSrc$, szDest$*

The **AddSectionFilesToCopyList** subroutine adds all file descriptions from the specified section of the .INF file to the global list of installable files (the copy list).

Arguments

szSection$

Specifies the name of the section in the .INF file that contains the files you want to add to the copy list.

szSrc$

Specifies the full path of the directory where the files currently reside. Typically, you use the value associated with the symbol STF_SRCDIR for *szSrc$*.

szDest$

Specifies the full path of the directory to which the files will be copied.

Comments

You must call the **ReadInfFile** subroutine before using this subroutine.

AddSectionKeyFileToCopyList subroutine

AddSectionKeyFileToCopyList *szSection$, szKey$, szSrc$, szDest$*

The **AddSectionKeyFileToCopyList** subroutine adds a file description identified by the reference key from the .INF file to the global list of installable files (the copy list).

Arguments

szSection$

Specifies the name of the section in the .INF file that contains the file you want to add to the copy list.

szKey$

Specifies the reference key for the file you want to add to the copy list.

szSrc$

Specifies the full path of the directory where the file currently resides. Typically, you use the value associated with the symbol STF_SRCDIR for *szSrc$*.

szDest$

Specifies the full path of the directory to which the file will be copied.

Comments

You must call the **ReadInfFile** subroutine before using this subroutine.

AddSpecialFileToCopyList subroutine

AddSpecialFileToCopyList *szSection$, szKey$, szSrc$, szDest$*

The **AddSpecialFileToCopyList** subroutine adds the file description of a special file, such as a shared file, from

the .INF file to the global list of installable files (the copy list).

Arguments

szSection$

Specifies the name of the section in the .INF file that contains the file you want to add to the copy list.

szKey$

Specifies the reference key for the file you want to add to the copy list.

szSrc$

Specifies the full path of the directory where the file currently resides. Typically, you use the symbol STF_SRCDIR for *szSrc$*.

szDest$

Specifies the full path of the file to be copied.

Comments

You must call the **ReadInfFile** subroutine before using this subroutine.

AddToBillboardList subroutine

AddToBillboardList *szDll$, idDlg%, szProc$, lTicks&*

The **AddToBillboardList** subroutine adds a billboard dialog box to the end of the global billboard list. The dialog box will be displayed during the next **CopyFilesInCopyList** subroutine call.

Arguments

szDll$

Specifies the name of the .DLL file that contains the dialog box resource and procedure.

idDlg%

Specifies the dialog box resource identification number.

szProc$

Specifies the name of the dialog box procedure.

lTicks&

Defines the amount of time you want the billboard dialog box to display. The unit is arbitrary, relative to the total number of units specified at the time the files are copied onto the user's hard disk or network drive.

BackupFile subroutine

BackupFile *szFullPath$, szBackup$*

The **BackupFile** subroutine backs up the specified file by renaming it.

Arguments

szFullPath$

Specifies the full path and name of the file you want to create a copy of.

szBackup$

Specifies the filename of the copy.

Comments

The copy is placed in the same directory as the original file(as specified by *szFullPath$*). This subroutine is identical to **RenameFile**.

ClearBillboardList subroutine

ClearBillboardList

The **ClearBillboardList** subroutine deletes all dialog boxes from the global billboard list.

ClearCopyList subroutine

ClearCopyList

The **ClearCopyList** subroutine removes all file entries from the global list of installable files (or copy list).

CloseLogFile subroutine

CloseLogFile

The **CloseLogFile** subroutine closes the currently open log file.

CopyFile subroutine

CopyFile *szFullPathSrc$, szFullPathDst$, cmo%, fAppend%*

The **CopyFile** subroutine copies the specified file from its source directory to its destination directory.

Arguments

szFullPathSrc$

Specifies the full path of the file you want to copy.

szFullPathDst$

Specifies the full path of the destination directory for the file.

cmo%

Specifies one or more command option flags. You can use one or more of the following for *cmo%* (by adding them together): *cmoDecompress, cmoTimeStamp, cmoReadOnly, cmoOverwrite, cmoNone,* or *cmoAll* .

fAppend%

Specifies whether you want any existing file to be appended to. A value of one indicates that you want to append to an existing file; zero indicates that you want to remove the existing file before copying the new file.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

CopyFilesInCopyList subroutine

CopyFilesInCopyList

The **CopyFilesInCopyList** subroutine sorts the file descriptions in the global list of installable files (the copy list) and then copies them from their source directory to their destination directory.

Comments

The files are sorted by their source disk identification number to minimize the number of times the user has to insert disks during the installation. The source and destination directories are specified in the functions that add the files to the copy list.

CreateDir subroutine

CreateDir *szDir$, cmo%*

The **CreateDir** subroutine creates a directory with the specified path and name.

Arguments

szDir$

Specifies the complete path and name of the directory you want to create (starting with the disk drive letter and backslash).

cmo%

Specifies a command option flag. You can use *cmoVital* or *cmoNone*. If the directory already exists, the subroutine does nothing.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

CreateIniKeyValue subroutine

CreateIniKeyValue *szFile$, szSect$, szKey$, szValue$, cmo%*

The **CreateIniKeyValue** subroutine creates a symbol and an associated value in the designated section of the .INI file.

Arguments

szFile$

Specifies the name of the .INI file in which you want to create the symbol and value. If the file you specify is WIN.INI, you do not have to provide the full path. If the file you specify does not exist, it is created.

szSect$

Specifies the section name in which you want to create the symbol and value. *szSect$* must be a non-empty string.

szKey$

Defines the name of the symbol you want to create. If *szKey$* already exists, this subroutine will fail unless you specify *cmoOverwrite* for the command option flag.

szValue$

Defines the value that will be associated with the symbol.

cmo%

Specifies the command option flag. You can use *cmoVital*, *cmoNone*, *cmoAll*, or *cmoOverwrite*.

See Also

Appendix B, "Command Options Flags," for a list of command option flags and advice on their use.

CreateProgmanGroup subroutine

CreateProgmanGroup *szGroup$, szPath$, cmo%*

The **CreateProgmanGroup** subroutine creates a new Program Manager group by the specified name and a .GRP file with the specified file and path.

Arguments

szGroup$

Specifies the name of the Program Manager group you want to create. This name will be displayed in the group window title bar (or below the icon when the group window is minimized).

szPath$

Specifies the name and path for the .GRP file you want to create. If you provide an empty string for *szPath$* (the suggested method), a default file is created.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone*.

Comments Typically, you should use an empty string for *szPath$*.

See Also Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

CreateProgmanItem subroutine

CreateProgmanItem *szGroup$, szItem$, szCmd$, szOther$, cmo%*

The **CreateProgmanItem** subroutine creates a new item in the specified Program Manager group.

Arguments *szGroup$*

Specifies the name of the Program Manager group in which you want to create the item. If *szGroup$* does not exist, this subroutine does nothing.

szItem$

Specifies the description that will be displayed below the item.

szCmd$

Specifies the path and executable filename for the new item.

szOther$

Specifies an optional icon file, icon resource index, *x* and *y* icon positions for the new item, and the working directory, separated by commas. If you provide an empty string for *szOther$*, defaults are used. However, if you need to specify one of the latter options in the string, you must specify the preceding ones.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoOverwrite*. If you use *cmoOverwrite* and the user is running Windows version 3.1, Setup will replace an existing Program Manager item.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use. For more information about creating the optional icon file, resource index, and position, see the *Guide to Programming*.

CreateRegKey subroutine

CreateRegKey *szKey$*

The **CreateRegKey** subroutine creates a Registration Database key that is a subkey of HKEYS_CLASSES_ROOT.

Argument

szKey$

Specifies the name of the key you want to create. This key will have no associated value.

Comments

To use this subroutine, you must include the MSREGDB.INC file in your Setup script (.MST) file.

CreateRegKeyValue subroutine

CreateRegKeyValue *szKey$, szValue$*

The **CreateRegKeyValue** subroutine creates a Registration Database key that is a subkey of HKEYS_CLASSES_ROOT and associates a value with the key.

Arguments

szKey$

Specifies the name of the key you want to create.

szValue$

Specifies the value you want to associate with the key.

Comments

To use this subroutine, you must include the MSREGDB.INC file in your Setup script (.MST) file.

CreateSysIniKeyValue subroutine

CreateSysIniKeyValue *szFile$, szSect$, szKey$, szValue$, cmo%*

The **CreateSysIniKeyValue** subroutine adds the specified symbol and its associated value to the .INI file.

Arguments

szFile$

Specifies the full path of the .INI file. This subroutine should not be used to modify WIN.INI.

szSect$

Specifies the name of the section in which you want to add the symbol-value pair. *szSect$* must be a non-empty string.

szKey$

Specifies the name of the symbol you want to add. This string does not have to be unique.

szValue$

Defines the value you want to associate with the symbol.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone*.

Comments

Use this subroutine, rather than **CreateIniKeyValue**, to add symbol-value pairs with non-unique keys to the .INI file. For example, you could add a line such as

DEV = *.VGA

where DEV = is likely to occur several times in the file.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

DeleteRegKey subroutine

DeleteRegKey *szKey$*

The **DeleteRegKey** subroutine removes the specified Registration Database key, its associated values, and subkeys.

Argument

szKey$

Specifies the name of the key you want to remove.

Comments

To use this subroutine, you must include the MSREGDB.INC file in your Setup script (.MST) file.

DoesDirExist function

Integer% = DoesDirExist (*szDir$*)

The **DoesDirExist** function determines if the specified directory exists.

Argument

szDir$

Specifies the name of the directory.

Return Value

If the directory exists, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

DoesFileExist function

Integer% = DoesFileExist (*szFile$, mode%*)

The **DoesFileExist** function determines if the specified file exists, can be read from, can be written to, or all of these states.

Arguments

szFile$

Specifies the name of the file you want to inquire about.

mode%

Specifies the file exist mode (femExists, femRead, femWrite, femReadWrite) you want to inquire about.

Return Value

If the answer is yes, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

DoesIniKeyExist function

Integer% = DoesIniKeyExist (*szFile$, szSect$, szKey$*)

The **DoesIniKeyExist** function determines if the specified key exists in the specified section of the .INI file.

Arguments

szFile$

Specifies the name of the .INI file. If you specify WIN.INI, you do not have to provide the full path.

szSect$

Specifies the section of the file. *szSect$* must be a non-empty string.

szKey$

Specifies the key (or symbol) you are looking for.

Return Value

If the key exists, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

DoesIniSectionExist function

Integer% = DoesIniSectionExist (*szfile$, szSect$*)

The **DoesIniSectionExist** function determines if the specified section exists in the .INI file.

Arguments

szFile$

Specifies the name of the .INI file. If you specify WIN.INI, you do not have to provide the full path.

szSect$

Specifies the section of the file. *szSect$* must be a non-empty string.

Return Value If the section exists, the return value is one. Otherwise, the return value is zero.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

DoesRegKeyExist function

Integer% = DoesRegKeyExist (*szKey$*)

The **DoesRegKeyExist** function checks for the existence of the specified key in the Registration Database.

Argument *szKey$*

Specifies the name of the key. This key is assumed to be a subkey of HKEYS_CLASSES_ROOT.

Return Value If the key exists, the return value is one. Otherwise, the return value is zero.

Comments To use this subroutine, you must include the MSREGDB.INC file in your Setup script (.MST) file.

DoMsgBox function

Integer% = DoMsgBox (*szText$, szCaption$, wType%*)

The **DoMsgBox** function launches a Windows message box containing the specified caption and text.

Arguments

szText$

Specifies the text you want to appear in the message box.

szCaption$

Specifies the caption for the message box.

wType%

Specifies the contents of the message box. *wType%* can be a combination of values.

Return Value

The return value is the value of the button control that the user selected (such as IDOK). If there is not enough memory to create the message box, the return value is zero.

Comments

This function is similar to the Windows **MessageBox** function. The valid message box values and control values are the same as for the **MessageBox** function.

See Also

For more information on the **MessageBox** function, message box values, and control values, see the *Programmer's Reference.*

DumpCopyList subroutine

DumpCopyList *szFile$*

The **DumpCopyList** subroutine prints the contents of the global list of installable files (the copy list) to the specified file.

Argument

szFile$

Specifies the path and name of the file to which you want the list to be copied.

Comments

Use this subroutine when debugging your installation files.

ExitExecRestart function

Integer% = ExitExecRestart ()

The **ExitExecRestart** function installs system files that may be in use by Windows (and therefore can't be overwritten).

Return Value

If the function succeeds, it doesn't return. The return value is true (one) if the restart list is empty or the user is running Windows version 3.0 (see Comments below). The return value is false (zero) if there are write errors, the restart fails, Windows can't exit, or the function can't find _MSSETUP.EXE.

Comments

This function returns a value but can fail for one of three reasons: if the restart list is empty, if the user is running Windows version 3.0 (which does not support the function), or if an error occurs (such as Windows not exiting because of open MS-DOS boxes or applications). Therefore, you should use the **RestartListEmpty** function to determine the contents of the restart list before calling this function. You should also check the current version of Windows. If the user is running Windows version 3.0, you will need to provide a message box explaining that the user must exit Windows and run the batch file to update the shared resources. In addition, you should warn the user to close all MS-DOS boxes and applications.

ExitExecRestart exits Windows and executes _MSSETUP.EXE to read any commands that have been placed in _MSSETUP.BAT. (Commands are placed in _MSSETUP.BAT when a file in the copy list is specified as a system file, Setup determines that it is a newer version, and it is currently in use.) **ExitExecRestart** will then delete _MSSETUP.EXE and _MSSETUP.BAT before restarting Windows. You must call the **SetRestartDir** subroutine before using this function.

FindFileInTree function

String$ = FindFileInTree (*szFile$, szDir$*)

The **FindFileInTree** function searches for the specified file in a directory and its subdirectories.

Arguments

szFile$

Specifies the name of the file you are trying to locate.

szDir$

Specifies the top-level directory of the tree structure you want to search.

Return Value

The return value is the full path of the first instance of the file. If the file isn't found, the return value is an empty string.

Comments

To use this function, you must include the file MSDETECT.INC in your Setup script (.MST) file.

FindFileUsingFileOpen function

String$ = FindFileUsingFileOpen (*szFile$*)

The **FindFileUsingFileOpen** function locates a file by using the Windows **FileOpen** function.

Argument

szFile$

Specifies the name of the file you want to find.

Return Value

The return value is the full path of the file. If the file isn't found, the return value is an empty string.

FindTargetOnEnvVar function

String$ = FindTargetOnEnvVar (*szFile$, szEnvVar$*)

The **FindTargetOnEnvVar** function searches for the specified file in directories based on the designated environment variable (such as PATH).

Arguments

szFile$

Specifies the name or partial path of the file you are trying to locate.

szEnvVar$

Specifies the environment variable for the search.

Return Value

The return value is the full path of the specified file. If the file isn't found, the return value is an empty string.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetConfigLastDrive function

String$ = GetConfigLastDrive ()

The **GetConfigLastDrive** function determines the LASTDRIVE set in the CONFIG.SYS file.

Return Value

The return value is the string for LASTDRIVE.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetConfigNumBuffers function

Integer% = GetConfigNumBuffers ()

The **GetConfigNumBuffers** function determines the number of BUFFERS set in the CONFIG.SYS file.

Return Value

The return value is the number of BUFFERS.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetConfigNumFiles function

Integer% = GetConfigNumFiles ()

The **GetConfigNumFiles** function determines the number of FILES set in the CONFIG.SYS file.

Return Value

The return value is the number of FILES.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetConfigRamdriveSize function

Integer% = GetConfigRamdriveSize ()

The **GetConfigRamdriveSize** function determines the installed size of RAMDRIVE.SYS set in the CONFIG.SYS file.

Return Value

The return value is the size of RAMDRIVE.SYS.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetConfigSmartdrvSize function

Integer% = GetConfigSmartdrvSize ()

The **GetConfigSmartdrvSize** function determines the installed size of SMARTDRV.SYS set in the CONFIG.SYS file.

Return Value

The return value is the size of SMARTDRV.SYS.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetCopyListCost function

Long& = GetCopyListCost (*szExtras$, szCosts$, szNeededs$*)

The **GetCopyListCost** function examines the files listed in the global list of installable files (the copy list) and determines the amount of disk space needed to copy, back up, and overwrite files. The values retrieved are associated with symbols in the Symbol Table.

Arguments

szExtras$

Symbol whose associated value identifies the extra, or incidental, disk space needed on each disk drive to update files such as .INI files. The symbol value is a list of as many as 26 integers. Missing values are assumed to be zero.

szCosts$

Symbol whose associated value is set to a list of 26 numbers, each of which identifies the cost per disk drive to copy and update files. A positive number is the amount of free space that will be used by new or larger files. A negative number is the amount of space that will be freed by removing files or replacing existing files with smaller versions.

szNeededs$

Symbol whose associated value is set to a list of 26 numbers, each of which identifies the additional space needed per disk drive. Each entry in *szCosts$* that is not zero has a corresponding entry in this list that is the value in *szCosts$* minus the current free space on that disk drive. A positive number indicates that the new files will not fit (and by how much). A negative number indicates how much free space will be left after the new files are copied.

Return Value

The return value is the total additional free disk space needed. This value is the sum of the positive numbers in *szNeededs$*. If there is enough free disk space on the appropriate disk drives for the **CopyFilesInCopyList** subroutine to succeed, the return value is zero.

GetDateOfFile function

String$ = GetDateOfFile (*szFile$*)

The **GetDateOfFile** function determines the file date of the specified file.

Argument

szFile$

Specifies the full path of the file.

Return Value

The return value is the date in YYYY-MM-DD-HH-MM-SS format. If *szFile$* does not exist, or if it has an

invalid date, the return value will be 1980-01-01-00-00-00.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetDayFromDate function

Integer% = GetDayFromDate (*szDate$*)

The **GetDayFromDate** function retrieves the day field from the return value for the **GetDateOfFile** function.

Argument

szDate$

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value

The return value is an integer with a valid range from 1 through 31.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetDOSMajorVersion function

Integer% = GetDOSMajorVersion ()

The **GetDOSMajorVersion** function determines the major version number of the currently installed MS-DOS.

Return Value

The return value is the major version number of MS-DOS.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetDOSMinorVersion function

Integer% = GetDOSMinorVersion ()

The **GetDOSMinorVersion** function determines the minor version number of the currently installed MS-DOS.

Return Value The return value is the minor version number of MS-DOS.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetEnvVariableValue function

String$ = GetEnvVariableValue (*szEnvVar$*)

The **GetEnvVariableValue** function determines the associated value for the specified environment variable.

Argument *szEnvVar$*

Specifies the environment variable name.

Return Value The return value is the string associated with the environment variable. If there is no associated value, the string is empty.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetFreeSpaceForDrive function

Long& = GetFreeSpaceForDrive (*szDrive$*)

The **GetFreeSpaceForDrive** function determines the amount of free space available on the specified disk drive.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

The return value is the amount of free disk space. If *szDrive$* is not a valid disk drive, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetHourFromDate function

Integer% = GetHourFromDate (*szDate$*)

The **GetHourFromDate** function retrieves the hour field from the return value for the **GetDateOfFile** function.

Argument

szDate$

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value

The return value is an integer with a valid range from 0 through 23.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetIniKeyString function

String$ = GetIniKeyString (*szFile$, szSect$, szKey$*)

The **GetIniKeyString** function searches the .INI file for the specified key string.

Arguments

szFile$

Specifies the .INI file that you want to search. If you specify WIN.INI, you do not have to provide the full path.

szSect$

Specifies the section of the file to be searched. *szSect$* must be a non-empty string.

szKey$

Specifies the key or symbol you want to search for.

Return Value

If the key is found, the return value is the full string associated with the key. Otherwise, the return value is an empty string.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetListItem function

String$ = GetListItem (*szSymbol$, n%*)

The **GetListItem** function retrieves a single item from the list associated with the specified symbol name.

Arguments

szSymbol$

Specifies the name of the symbol in the Symbol Table with which the list is associated.

n%

Specifies the index (one-based) of the item you want to retrieve.

Return Value The return value is the item from the list.

Comments If the string is empty, the symbol or item does not exist.

GetListLength function

Integer% = GetListLength (*szSymbol$*)

The **GetListLength** function determines the number of items in the list associated with the specified symbol name.

Argument *szSymbol$*

Specifies the name of the symbol in the Symbol Table with which the list is associated.

Return Value The return value is the number of items in the list.

GetLocalHardDrivesList subroutine

GetLocalHardDrivesList *szSymbol$*

The **GetLocalHardDrivesList** subroutine sets the specified symbol to a list of all local hard drives (that is, "A", "B", and so on).

Argument *szSymbol$*

Specifies the name of the symbol to associate with the list.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetMinuteFromDate function

Integer% = GetMinuteFromDate (*szDate$*)

The **GetMinuteFromDate** function retrieves the minute field from the return value for the **GetDateOfFile** function.

Argument *szDate$*

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value The return value is an integer with a valid range from 0 through 59.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetMonthFromDate function

Integer% = GetMonthFromDate (*szDate$*)

The **GetMonthFromDate** function retrieves the month field from the return value for the **GetDateOfFile** function.

Argument *szDate$*

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value The return value is an integer with a valid range from 1 through 12.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetNetworkDrivesList subroutine

GetNetworkDrivesList *szSymbol$*

The **GetNetworkDrivesList** subroutine sets the specified symbol to a list of all network drives (that is, "A", "B", and so on).

Argument *szSymbol$*

Specifies the name of the symbol to associate with the list.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetNthFieldFromIniString function

String$ = GetNthFieldFromIniString (*szLine$, iField%*)

The **GetNthFieldFromIniString** function extracts the specified comma-separated field from the given string.

Arguments

szLine$

Specifies the string from which you want to extract the field.

iField%

Specifies the index (one-based) of the field that you want to extract.

Return Value

The return value is the field. If the requested field doesn't exist or is invalid, the return value is an empty string.

GetNumWinApps function

Integer% = GetNumWinApps ()

The **GetNumWinApps** function determines the number of unique instances of Windows applications currently running in the system.

Return Value

The return value is the number of applications.

Comments

To use this subroutine, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetParallelPortsList subroutine

GetParallelPortsList *szSymbol$*

The **GetParallelPortsList** subroutine sets the specified symbol to a list of all parallel ports (that is, "LPT1","LPT2", and so on).

Argument

szSymbol$

Specifies the name of the symbol to associate with the list.

Comments

To use this subroutine, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetProcessorType function

Integer% = GetProcessorType ()

The **GetProcessorType** function determines the type of processor being run on the user's system.

Return Value

The return value is either zero (for 8086), one (for 80186), two (for 80286), three (for 80386), or four (for 80486).

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetRegKeyValue function

String$ = GetRegKeyValue (*szKey$*)

The **GetRegKeyValue** function determines the value associated with the specified Registration Database key.

Argument

szKey$

Specifies the name of the key whose value you want to retrieve. Because all keys are assumed to be subkeys of HKEYS_CLASSES_ROOT, you only have to specify the name of the key.

Return Value

The return value is the value associated with *szKey$* in the Registration Database.

Comments

To use this function, you must include the MSREGDB.INC file in your Setup script (.MST) file.

GetRemovableDrivesList subroutine

GetRemovableDrivesList *szSymbol$*

The **GetRemovableDrivesList** subroutine sets the specified symbol to a list of all removable drives (that is, "A", "B", and so on).

Argument

szSymbol$
Specifies the name of the symbol to associate with the list.

Comments

To use this subroutine, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetScreenHeight function

Integer% = GetScreenHeight ()

The **GetScreenHeight** function determines the height of the screen.

Return Value

The return value is the height of the screen (in pixels).

GetScreenWidth function

Integer% = GetScreenWidth ()

The **GetScreenWidth** function determines the width of the screen.

Return Value

The return value is the width of the screen (in pixels).

GetSecondFromDate function

Integer% = GetSecondFromDate (*szDate$*)

The **GetSecondFromDate** function retrieves the seconds field from the return value for the **GetDateOfFile** function.

Argument

szDate$

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value

The return value is an integer with a valid range from 0 through 59.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetSectionKeyDate function

String$ = GetSectionKeyDate (*szSection$, szKey$*)

The **GetSectionKeyDate** function retrieves the date from a file description in the .INF file.

Arguments

szSection$

Specifies the name of the section where the file description resides.

szKey$

Specifies the name of the key associated with the file description.

Return Value

The return value is the date. This string can be parsed by the **GetDayFromDate**, **GetMonthFromDate,** and **GetYearFromDate** functions.

GetSectionKeyFilename function

String$ = GetSectionKeyFilename (*szSection$, szKey$*)

The **GetSectionKeyFilename** function retrieves the filename from a file description in the .INF file.

Arguments

szSection$

Specifies the name of the section where the file description resides.

szKey$

Specifies the name of the key associated with the file description.

Return Value

The return value is the filename.

GetSectionKeySize function

Long& = GetSectionKeySize (*szSection$, szKey$*)

The **GetSectionKeySize** function retrieves the file size from a file description in the .INF file.

Arguments

szSection$

Specifies the name of the section where the file description resides.

szKey$

Specifies the name of the key associated with the file description.

Return Value

The return value is the file size in bytes.

GetSectionKeyVersion function

String$ = GetSectionKeyVersion (*szSection$, szKey$*)

The **GetSectionKeyVersion** function retrieves the version number from a file description in the .INF file.

Arguments

szSection$

Specifies the name of the section where the file description resides.

szKey$

Specifies the name of the key associated with the file description.

Return Value

The return value is the version number. This string can be parsed by the **GetVersionNthField** function.

GetSerialPortsList subroutine

GetSerialPortsList *szSymbol$*

The **GetSerialPortsList** subroutine sets the specified symbol to a list of all serial ports (that is, "COM1","COM2", and so on).

Argument

szSymbol$
Specifies the name of the symbol to associate with the list.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetSizeOfFile function

Long& = GetSizeOfFile (*szFile$*)

The **GetSizeOfFile** function determines the size of the specified file.

Argument

szFile$

Specifies the path and name of the file.

Return Value

The return value is the size of the file in bytes. If *szFile$* is invalid or does not exist, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetSymbolValue function

String$ = GetSymbolValue (*szSymbol$*)

The **GetSymbolValue** function finds the value associated with a symbol in the Symbol Table.

Argument

szSymbol$

Specifies the name of the symbol to be found in the Symbol Table.

Return Value

The return value is the value associated with *szSymbol$* in the Symbol Table. If there is no value associated with *szSymbol$*, the return value is an empty string.

Comments

Use this function to retrieve information stored in the Symbol Table by other .DLL or .MST file procedures.

GetTotalSpaceForDrive function

Long& = GetTotalSpaceForDrive (*szDrive$*)

The **GetTotalSpaceForDrive** function determines the total amount of disk space for the specified disk drive.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

The return value is the total capacity in bytes of the disk drive. If *szDrive$* is not a valid disk drive, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetTypeFaceNameFromTTF function

Integer% = GetTypeFaceNameFromTTF (*szFile$, szBuff$, cbBuff%*)

The **GetTypeFaceNameFromTTF** function extracts the typeface name from a TrueType font file.

Arguments

szFile$

Specifies the name of the TrueType font file.

szBuff$

Specifies the buffer you are providing for the storage of the typeface name.

cbBuff%

Specifies the size of the buffer you are providing.

Return Value

If the file you specified is not a TrueType font file, the return value is zero. Otherwise, the return value is the actual length of the typeface name.

Comments

Check to ensure that the return value is not greater than the size of *szBuff%*. If the return value is greater, the typeface name has been truncated.

GetValidDrivesList subroutine

GetValidDrives *szSymbol$*

The **GetValidDrivesList** subroutine sets the specified symbol to a list of all valid disk drives (that is, "A", "B", and so on).

Argument

szSymbol$
Specifies the name of the symbol to associate with the list.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetVersionNthField function

Long& = GetVersionNthField (*szVersion$, nField%*)

The **GetVersionNthField** function extracts the specified field from the return value for the **GetVersionOfFile** function.

Arguments

szVersion$

Specifies the string returned from the **GetVersionOfFile** function.

nField%

Specifies the number of the field you want to extract (1 through 4 from the version string).

Return Value

The return value is the integer extracted from *szVersion$*.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

GetVersionOfFile function

String$ = GetVersionOfFile (*szFile$*)

The **GetVersionOfFile** function determines the version of the specified file.

Argument

szFile$

Specifies the path and name of the file.

Return Value

The return value is a string in the format N.N.N.N, where each N is an integer with as many as digits.

GetWindowsDir function

String$ = GetWindowsDir ()

The **GetWindowsDir** function determines the name and path of the Windows directory.

Return Value

The return value is the path name terminated with a backslash; for example, "C:\WINDOWS\".

GetWindowsMajorVersion function

Integer% = GetWindowsMajorVersion ()

The **GetWindowsMajorVersion** function determines the major version number for the currently installed Windows software.

Return Value

The return value is the Windows major version number.

GetWindowsMinorVersion function

Integer% = GetWindowsMinorVersion ()

The **GetWindowsMinorVersion** function determines the minor version number for the currently installed Windows software.

Return Value

The return value is the Windows minor version number.

GetWindowsMode function

Integer% = GetWindowsMode ()

The **GetWindowsMode** function determines the current mode of Windows.

Return Value

The return value is zero for Real mode, one for Standard mode, or two for Enhanced mode.

GetWindowsSysDir function

String$ = GetWindowsSysDir ()

The **GetWindowsSysDir** function determines the name and path of the Windows system directory.

Return Value

The return value is the path which ends with a backslash.

GetYearFromDate function

Integer% = GetYearFromDate (*szDate$*)

The **GetYearFromDate** function retrieves the year field from the return value for the **GetDateOfFile** function.

Argument

szDate$

Specifies the date in YYYY-MM-DD-HH-MM-SS format. This value is obtained from the **GetDateOfFile** function.

Return Value

The return value is an integer with a valid range from 1980 through 2099.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

Has87MathChip function

Integer% = Has87MathChip ()

The **Has87MathChip** function checks for an 87 math coprocessor on the user's system.

Return Value If an 87 math coprocessor exists, the return value is one. Otherwise, the return value is zero.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

HasMonochromeDisplay function

Integer% = HasMonochromeDisplay ()

The **HasMonochromeDisplay** function checks for a monochrome display on the user's system.

Return Value If the monochrome display exists, the return value is one. Otherwise, the return value is zero.

Comments To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

HasMouseInstalled function

Integer% = HasMouseInstalled ()

The **HasMouseInstalled** function determines if a mouse is installed on the user's system.

Return Value

If a mouse exists, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

HinstFrame function

Integer% = HinstFrame ()

The **HinstFrame** function retrieves the instance handle for the Setup program.

Return Value

The return value is the instance handle.

HwndFrame function

Integer% = HwndFrame ()

The **HwndFrame** function provides the handle for the Setup application frame window (the main window).

Return Value

The return value is the handle of the frame window (the main window).

Comments

Use the return value from this function for the *hwnd%* argument in procedures that require it.

IsDirWritable function

Integer% = IsDirWritable (*szDir$*)

The **IsDirWritable** function determines if the specified directory is writable (so that Setup can create a file in it).

Argument

szDir$

Specifies the name of the directory.

Return Value

If the directory is writable, the return value is one. Otherwise, the return value is zero.

IsDriveLocalHard function

Integer% = IsDriveLocalHard (*szDrive$*)

The **IsDriveLocalHard** function determines whether the specified disk drive is a local hard disk.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

If the disk drive is a local hard disk, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

IsDriveNetwork function

Integer% = IsDriveNetwork (*szDrive$*)

The **IsDriveNetwork** function determines if the specified disk drive is a network drive.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

If the disk drive is a network drive, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

IsDriveRemovable function

Integer% = IsDriveRemovable (*szDrive$*)

The **IsDriveRemovable** function determines if the specified disk drive is a removable disk drive.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

If the disk drive is removable, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

IsDriverInConfig function

Integer% = IsDriverInConfig (*szDevice$*)

The **IsDriverInConfig** function determines if the specified device driver is in the CONFIG.SYS file.

Argument

szDevice$

Specifies the name of the device driver you want to find.

Return Value

If the device driver statement is found, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

IsDriveValid function

Integer% = IsDriveValid (*szDrive$*)

The **IsDriveValid** function determines if the specified disk drive exists.

Argument

szDrive$

Specifies a string identifying the disk drive letter (A through Z).

Return Value

If the specified disk drive is valid, the return value is one. Otherwise, the return value is zero.

Comments

To use this function, you must include the MSDETECT.INC file in your Setup script (.MST) file.

IsFileWritable function

Integer% = IsFileWritable (*szFile$*)

The **IsFileWritable** function determines whether the specified file is writable.

Argument

szFile$

Specifies the full path and name of the file you want to write to.

Return Value

If the file is writable, the return value is one. Otherwise, the return value is zero.

IsWindowsShared function

Integer% = IsWindowsShared ()

The **IsWindowsShared** function determines if Windows is shared by comparing the Windows and system directories.

Return Value

If Windows is shared, the return value is one. Otherwise, the return value is zero.

MakeListFromSectionKeys subroutine

MakeListFromSectionKeys *szSymbol$, szSection$*

The **MakeListFromSectionKeys** subroutine creates a list of the reference key values found in a section of the currently open .INF file and associates the list with the specified symbol name.

Arguments

szSymbol$

Specifies the symbol name of the list in the Symbol Table.

szSection$

Specifies the name for the section in the .INF file that contains the reference key values.

Comments

Use this function to create a list that can be used in dialog list boxes.

OpenLogFile subroutine

OpenLogFile *szFile$, fAppend%*

The **OpenLogFile** subroutine opens a log file that the Setup program or your script can use for writing status information.

Argument

szFile$

Specifies the path and name of the log file. If this file does not exist, it is created.

fAppend%

Specifies whether to add information to the log file or overwrite the existing contents of the file. Specify one to add information and zero to overwrite information.

ReadInfFile subroutine

ReadInfFile *szFile$*

The **ReadInfFile** subroutine opens and reads the contents of the specified .INF file.

Arguments

szFile$

Specifies the path and name of the .INF file.

Comments

The script (.MST) file should call this subroutine to read an .INF file before attempting to call any of the procedures that access .INF file information.

RemoveDir subroutine

RemoveDir *szDir$, cmo%*

The **RemoveDir** subroutine deletes the specified directory.

Arguments

szDir$

Specifies the path and the name of the directory to be deleted.

cmo%

Specifies the command option flag. You can use either *cmoVital* or *cmoNone* for the command option flag.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

RemoveFile subroutine

RemoveFile *szFile$, cmo%*

The **RemoveFile** subroutine deletes the specified file.

Arguments

szFile$

Specifies the full path of the file you want to delete.

cmo%

Specifies the command option flag. You can use *cmoVital, cmoForce,* or *cmoNone.*

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

RemoveIniKey subroutine

RemoveIniKey *szFile$, szSect$, szKey$, cmo%*

The **RemoveIniKey** subroutine deletes a symbol from the specified .INI file.

Arguments

szFile$

Specifies the name of the .INI file. If the .INI file is WIN.INI, you do not have to provide the full path.

szSect$

Specifies the name of the section where the symbol to be deleted exists. *szSect$* must be a non-empty string.

szKey$

Specifies the name of the symbol you want to delete.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone*.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

RemoveIniSection subroutine

RemoveIniSection *szFile$, szSect$, cmo%*

The **RemoveIniSection** subroutine deletes the specified section from the designated .INI file.

Arguments

szFile$

Specifies the name of the .INI file. If the .INI file is WIN.INI, you do not have to provide the full path.

szSect$

Specifies the name of the section you want to delete. *szSect$* must be a non-empty string.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone*.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

RemoveSymbol subroutine

RemoveSymbol *szSymbol$*

The **RemoveSymbol** subroutine deletes a symbol from the Symbol Table.

Argument

szSymbol$

Specifies the name of the symbol in the Symbol Table.

Comments

Use this subroutine to free up space occupied by unused symbols and their associated values in the Symbol Table.

RenameFile subroutine

RenameFile *szFullPath$, szRename$*

The **RenameFile** subroutine renames the specified file.

Arguments

szFullPath$

Specifies the full path and name of the file you want to rename.

szRename$

Specifies the new name for the file.

Comments

The renamed file is placed in the same directory as the original file (as specified by *szFullPath$*). This subroutine is the same as the **BackupFile** subroutine.

ReplaceListItem subroutine

ReplaceListItem *szSymbol$, n%, szItem%*

The **ReplaceListItem** subroutine replaces an item in the list associated with the specified symbol.

Arguments

szSymbol$
Specifies the name of the symbol whose associated value is the list.

n%
Specifies the index (one-based) of the item to be replaced.

szItem%

Specifies the new item.

RestartListEmpty function

Integer% = RestartListEmpty ()

The **RestartListEmpty** function determines if any files have been added to _MSSETUP.BAT that need to be copied or deleted by the **ExitExecRestart** function when Windows is restarted.

Return Value

If the restart list is empty, the return value is one. Otherwise, the return value is zero.

Comments

Setup adds files to _MSSETUP.BAT as a result of reading the .INF file and finding files identified as a system resource (that is, the file description contains a SYSTEM flag). If the file is newer than the file on the user's system, Setup adds its name to _MSSETUP.BAT.

See Also

Appendix A, "INF File Format," for more information about the SYSTEM flag.

RestoreCursor subroutine

RestoreCursor *hPrev%*

The **RestoreCursor** subroutine restores the previous cursor state.

Argument

hPrev%

Specifies the identification number of the previous cursor state. Use the return value from the **ShowWaitCursor** function for this parameter.

SearchForLocationForSharedFile function

String$ = SearchForLocationForSharedFile (*szRegDbKey$, szWinIniSect$, szWinIniKey$, iWinIniField%, szDefault$, szVersion$*)

The **SearchForLocationForSharedFile** function uses information from the Registration Database and WIN.INI to determine where the specified shared file should be installed and whether the file will actually be copied.

Arguments

szRegDbKey$

Specifies the Registration Database key that might have an associated value that is the path of an existing copy of the file. You can tell the function to ignore the Registration Database in its search by specifying an empty string for this argument. If you provide a non-empty string for this argument, you must include the MSREGDB.INC file. Otherwise, the argument will be ignored.

szWinIniSect$

Specifies the section in the WIN.INI file that might have an entry that is the path of an existing copy of the file. You can tell the function to ignore WIN.INI in its search by specifying an empty string for this argument.

szWinIniKey$

Specifies the key for the WIN.INI file that will contain a path to an existing copy of this shared file.

iWinIniField%

Specifies the index (one-based) of the field in the WIN.INI line entry that contains a path to an existing copy of this shared file.

szDefault$

Specifies the default path of the file if no existing copy can be found.

szVersion$

Specifies the version for the new copy of the file as a string of 1 to 4 integers separated by periods; for example, 3.1.0.16.

Return Value

The return value is the full path for installing the shared file. This function also sets the global variable **SharedFileNeedsCopying** to one (if the file should be copied) or zero (if the file shouldn't be copied).

Comments

To determine a location for the shared file, the function first uses the Registration Database path. If this file doesn't exist, the function will retrieve only the filename from the Registration Database and perform a **FileOpen** function to try to determine the path. If either of these methods works and the file found is writable or newer, the function returns that path. If these methods fail, the function uses the path from the WIN.INI field. If that file doesn't exist in the WIN.INI field, the function uses the path from WIN.INI and searches by using a **FileOpen** function. If these two methods don't succeed, *szDefault$* will be used.

After determining where the file should be installed, the function then determines if the file will be copied later when the **CopyFilesInCopyList** subroutine is called. **SearchForLocationForSharedFile** then sets the value of the global variable **SharedFileNeedsCopying** accordingly.

To use this function, you must include the MSDETECT.INC and MSSHARED.INC files in the Setup script (.MST) file.

SetAbout subroutine

SetAbout *szString1$, szString2$*

The **SetAbout** subroutine adds the specified strings to Setup's About dialog box.

Arguments

szString1$

Specifies the first string you want to add to the dialog box, typically the product name.

szString2$

Specifies the second string you want to add to the dialog boxtypically the product version, the date, or the copyright notice.

SetBeepingMode function

Integer% = SetBeepingMode (*mode%*)

The **SetBeepingMode** function allows the Setup script to specify whether error messages, requests for diskettes, and similar messages will be accompanied by a beep.

Argument

mode%

Specifies the beeping mode value. If *mode%* is one, beeping mode is on and the message will generate beeps. If *mode%* is zero, beeping mode is off.

Return Value

The return value is the value of the previous beeping mode.

SetBitmap subroutine

SetBitmap *szDll$, Bitmap%*

The **SetBitmap** function defines the logo bitmap used in the background of the frame (or main) window.

Arguments

szDll$

Specifies the name of the .DLL file that contains the bitmap resource.

Bitmap%

Specifies the identification number of the bitmap resource.

Comments

It is best to use a plain, white bitmap. This subroutine automatically adds a shadow down the right side to give the impression that the bitmap is above the plane of the background.

SetCopyGaugePosition subroutine

SetCopyGaugePosition *x%,y%*

The **SetCopyGaugePosition** subroutine specifies the display position of the Copy Gauge dialog box during subsequent **CopyFilesInCopyList** subroutine calls.

Arguments

x%

Specifies the *x* coordinate in dialog units relative to the upper-left corner of the client window frame. A value of 1 tells the subroutine to precisely center the dialog box horizontally.

y%

Specifies the *y* coordinate in dialog units relative to the upper-left corner of the client window frame. A value of 1 tells the subroutine to center the dialog box vertically one-third of the distance from the top margin.

Comments

By default, the Copy Gauge dialog box is centered over the client window frame. However, the default position can interfere with the display of billboard dialog boxes, depending on how the user has resized the window, the size of the monitor, and so on. Use this subroutine to control the position of the Copy Gauge dialog box.

SetCopyMode function

Integer% = SetCopyMode (*wMode%*)

The **SetCopyMode** function specifies whether the files from the copy list will actually be copied.

Argument

wMode%

Specifies whether copy mode is on (one) or off (zero). If the copy mode is off, the files will not be copied when the script calls the **CopyFilesInCopyList** subroutine.

Return Value

The return value is the prior value for the copy mode.

Comments

Use this function to streamline testing of your Setup script.

SetDecompMode function

Integer% = SetDecompMode (*wMode%*)

The **SetDecompMode** function specifies whether compressed files will be decompressed when they are copied.

Argument

wMode%

Specifies whether decompression is on (one) or off (zero). When decompression is off, the **CopyFilesInCopyList** subroutine copies compressed files byte for byte without decompressing them.

Return Value

The return value is the prior value for decompression.

Comments

You can use this function to install compressed files on a network drive.

SetRegKeyValue subroutine

SetRegKeyValue *szKey$, szValue$*

The **SetRegKeyValue** subroutine replaces the value associated with the specified Registration Database key with the specified value.

Arguments

szKey$

Specifies the name of the key. All keys are assumed to be subkeys of HKEYS_CLASSES_ROOT; therefore, you only have to specify the name of the key.

szValue$

Specifies the new value you want to associate with the key.

Comments

To use this subroutine, you must include the MSREGDB.INC file in the Setup script (.MST) file.

SetRestartDir subroutine

SetRestartDir *szDir$*

The **SetRestartDir** subroutine establishes the restart directory where _MSSETUP.EXE and _MSSETUP.BAT (used to restart the system) will be located.

Argument *szDir$*

Specifies the name of the directory. If *szDir$* does not exist, it is created.

SetSilentMode function

Integer% = SetSilentMode (*mode%*)

The **SetSilentMode** function allows the Setup script to determine whether or not the copy gauge, error messages, billboards, and so on will be displayed.

Argument *mode%*

Specifies the silent mode value. If *mode%* is one, silent mode is on and the Setup procedures cannot display message-related dialog boxes. If *mode%* is zero, silent mode is off.

Return Value The return value is the value of the previous silent mode.

SetSymbolValue subroutine

SetSymbolValue *szSymbol$, szValue$*

The **SetSymbolValue** subroutine associates a value with a symbol in the Symbol Table.

Arguments

szSymbol$

Specifies the name of the symbol in the Symbol Table.

szValue$

Specifies the value you want to associate with the symbol.

Comments

Use this subroutine to store information that can be shared among .DLL and .MST file procedures in the Symbol Table.

SetTitle subroutine

SetTitle *szTitle$*

The **SetTitle** subroutine defines the title used in the frame (or main) window.

Argument

szTitle$

Defines the title used in the frame window.

ShowProgmanGroup subroutine

ShowProgmanGroup *szGroup$, Cmd%, cmo%*

The **ShowProgmanGroup** subroutine minimizes, maximizes, or restores the window of the specified Program Manager group.

Arguments

szGroup$

Specifies the name of the Program Manager group.

Cmd%

Specifies the operation you want to perform on the group window. To activate and display the group window, use one; to activate and display the group window icon, use two; and to activate and display the group window maximized, use three.

cmo%

Specifies the command option flag. You can use *cmoVital* or *cmoNone*.

Comments

Creating a new item in a Program Manager group causes the Program Manager to become active and its window to come to the front. You can use the **ShowProgmanGroup** subroutine to determine what state the group window will be in when it comes to the front.

See Also

Appendix B, "Command Option Flags," for a list of command option flags and advice on their use.

ShowWaitCursor function

Integer% = ShowWaitCursor ()

The **ShowWaitCursor** function loads and displays the wait cursor.

Return Value

The return value is the previous cursor state.

Comments

Use the return value of this function for the *hPrev%* parameter in the **RestoreCursor** subroutine.

StampResource subroutine

StampResource *szSection$, szKey$, szDst$, wResType%, wResId%, szData$, cbData%*

The **StampResource** subroutine modifies the first *cbData%* bytes of a file resource with the specified data.

Arguments

szSection$

Specifies the section of the .INF file that contains the description line of the file to be modified.

szKey$

Specifies the reference key to the description line of the file to be modified.

szDst$

Specifies the destination directory where the file to be modified resides.

wResType%

Specifies the resource identification type.

wResId%

Specifies the resource identification number.

szData$

Defines the data that will be used to modify the resource.

cbData%

Specifies how many bytes of data will be replaced with *szData$*.

UIPop subroutine

UIPop n%

The **UIPop** subroutine destroys dialog boxes and removes them from the top of the dialog stack in memory.

Argument

n%

Identifies the number of dialog boxes you want to destroy and remove from the dialog stack.

Comments

If *n%* is greater than the number of dialog boxes in the dialog stack, the subroutine will destroy all the dialog boxes in the dialog stack.

UIPopAll subroutine

UIPopAll

The **UIPopAll** subroutine destroys all dialog boxes and removes them from the dialog box stack in memory.

UIStartDlg function

String$ = UIStartDlg (*szDll$, idDlg%, szDlgProc$, idHelpDlg%, szHelpProc$*)

The **UIStartDlg** function launches a dialog box and adds it to the top of the dialog stack in memory.

Arguments

szDll$

Specifies the name of the .DLL file that contains the dialog box template resources and procedures.

idDlg%

Specifies the identification number of the dialog box template resource in the .DLL file.

szDlgProc$

Specifies the name of the dialog box procedure exported in the .DLL file.

idHelpDlg%

Specifies the identification number of the associated help dialog box template resource in the .DLL file.

szHelpProc$

Specifies the name of the associated help dialog box procedure exported in the .DLL file.

Return Value

The function returns a string that is the value associated with the DLGEVENT symbol at the time the dialog ends.

Comments

If the dialog box is modeless, the function returns immediately; if the dialog box is modal, the function returns after a user action.

WriteToLogFile subroutine

WriteToLogFile *szStr$*

The **WriteToLogFile** subroutine writes the specified string to the log file.

Argument

szStr$

Defines the information you want to write to the log file. The string is written and terminated with a new-line character.

Comments

If the log file is not open, this subroutine does nothing.

Appendix A: INF File Format

This appendix provides information about the format of the .INF file. You do not need to create the .INF file yourself; the Disk Layout Utilities will do it for you. Refer to the information below when you are using the Disk Layout Utilities to define the properties of installable files.

The Setup toolkit contains three sample .INF files: SAMPLE1.INF, SAMPLE2.INF, and SAMPLE3.INF. Look at these files for examples of typical .INF files.

An .INF file will have at least three sections:

- Source Media Descriptions, which describes each of the disks in the installation set
- Files, which lists each of the files that will be installed by Setup
- Default File Settings, which describes the defaults Setup uses to install a file

The Files section can be split into as many sections as you want; you specify these section names using the Disk Layout Utilities. You can also specify one or more sections for files that you want to remove during the installation. Throughout the .INF file, you will see lines that begin with a semicolon at the left margin. These are comment lines.

Source Media Descriptions

This section of the .INF file contains one line for each of the disks you use to install your product. Source Media Description lines must be indented and contain four quoted strings, separated by commas:

- The disk identification number, which is a unique integer between 1 and 999

- The disk label, which you create using the Disk Layout Utilities
- The tag filename, which is the name of a file that resides on the disk
- The relative path for SETUP.EXE. This fourth string exists only if the installable files reside on a network disk drive

The following is the Source Media Descriptions line for the first installation disk. Its tag file is SETUP.EXE:

```
"1","My Disk Label 1","SETUP.EXE,""
```

Files

Each line in this section of the .INF file has one of two formats:

- The first format begins at the left margin with the identification number for the disk on which the file resides, followed by the filename and a list of nineteen file properties, separated by commas (see the table below). These entries are not enclosed in quotation marks. The following line, from SAMPLE1.INF, is a typical example of this format:

```
1, bldcui\dialogs.res,,,,1992-01 30,,,,,,,ROOT,,,
13839,,6,,,
```

- The second format begins at the left margin with a reference key enclosed in quotation marks, followed by an equal sign and an unquoted disk identification number. (A disk with that identification number must have a description line in the Source Media Descriptions section.) The reference key and disk identification number are followed by the filename and the list of file properties. The following line, from SAMPLE3.INF, describes a shared file with the reference key CustDict (for customer dictionary) :

```
"CustDict" = 1, custom.dic, ,,,1992-01-13,,,
OLDER,,,,,, SHARED, 69632, ,,, 0.2.0.2,
```

The following table lists the file properties that can be included in a file description line:

Property	Possible values	Meaning
Append	<empty> valid filename	Don't append to an existing file, overwrite it instead. Append to the specified file. Note: You cannot append the installable file to an existing file if you have specified Rename, Root, or Backup.
Backup	<empty> * valid filename	Use the default setting. Back up the file to the same filename with a .BAK extension. The filename you want to use for the backup copy.
Copy	COPY !COPY	Copy the file onto the user's hard disk or network server. Don't copy the file.
Date	<empty> YYYY-MM-DD	Use the default setting. Specify a date with a range from 1980-01-01 through 2099-12-31.
Decompress	<empty> DECOMPRESS !DECOMPRESS	Use the default setting. Decompress the file. Don't decompress the file.
Destination	<empty> full path of a valid directory	Use the destination directory specified in the script (for example, in the AddSectionFilesToCopyList subroutine). Override the specified destination directory with this one.
Overwrite	<empty> ALWAYS NEVER OLDER UNPROTECTED	Use the default setting. For information about these values, see Chapter 4, "Using the Disk Layout Utilities."

ReadOnly	<empty> READONLY !READONLY	Use the default setting. Set read-only attributes on the file after it has been installed. Don't set read-only attributes.
Remove	<empty> REMOVE !REMOVE	Copy the file using other properties. Remove the file from the user's hard disk or network server. Copy the file using other properties.
Rename	<empty> valid filename	Use the specified filename when the file is copied. Use the source filename when the file is copied. Note: You cannot specify a filename for this property if you have also specified Root or Append.
Root	<empty> ROOT !ROOT	Use the default setting. Strip any subdirectories from the filename when the file is copied. Don't strip subdirectories from the filename when the file is copied. Note: You cannot specify Root or Append if you have specified that the file be renamed when it is copied.
SetTimeStamp	<empty> SETTIME !SETTIME	Use the default setting. Use the DATE property. Use the current system time.
Shared	SHARED <empty> or !SHARED	Treat the file as if it is a shared file -- that is, an existing version of it may be in use during installation. Do not treat the file as if it is a shared file.
Size	integer	Size of the file in bytes (uncompressed).
System	SYSTEM <empty> or !SYSTEM	This is a system file; therefore, it will be replaced during system restart. Treat the file as if it is not a system file.

TimeToCopy	<empty> interger	Use the default setting. Increment the progress indicator by integers. This is an arbitrary number of time units relative to the other file description lines.
Reserved	<empty>	This is a reserved field and must be empty.
Version	<empty> 1 to 4 integers separated by periods	There is no version resource in source file. The version of the file. For information about this format, see the Microsoft Windows SDK documentation.
Vital	<empty> VITAL !VITAL	Use the default setting. The installation will fail if this file cannot be successfully installed. The installation will not fail if this file cannot be successfully installed.

Table A.1 File Properties

Default File Settings

This section contains default values that Setup uses if a file description line in the .INF file has no entry for the field. These lines always begin at the left margin with a symbol name enclosed in quotation marks, followed by an equal sign and another string enclosed in quotation marks that is the value associated with the symbol. The value is required but may be an empty string.

The following line is an example of a default setting from SAMPLE1.INF:

```
"STF_COPY" = "YES"
```

The following table lists the default settings.

Attribute/Symbol	Possible values or format	Default
STF_BACKUP	"*" = create a .BAK file or " " = don't create one	" "
STF_COPY	"YES" = copy or " " = don't copy	"YES"
STF_DECOMPRESS	"YES" = decompress or " " = don't copy	" "
STF_OVERWRITE	"ALWAYS" = always overwrite an existing version of the file, "NEVER" = never overwrite an existing version of the file, "OLDER" = overwrite the existing version of the file if it is older, or "UNPROTECTED" = overwrite the existing version of the file if it is not write-protected	"ALWAYS"
STF_READONLY	"YES" = when the file is copied, set the read-only file attribute or " " = do not set the read-only file attribute	" "
STF_ROOT	"YES" = when copying the file, strip any subdirectories from the path of the file in the .INF file description or " " = use the entire path for the file in the .INF file decription	" "

STF_SETTIME	"YES" = use the contents of the Date file property in the .INF file description to set the creation date of the copied file or " " = leave the creation date as is (the time at which the file was copied onto the users hard disk)	"YES"
STF_TIME	non-negative integer = use this number to calculate the display of the copy gauge or "" = use the value entered for the Size file property in the .INF file decription	value of Size
STF_VITAL	"YES" = this is a vital file that must be successfuly copied to the users hard disk or the installation will fail; or " " = the file is not vital to the success of the installation	"YES"

Table A.2 Default File Settings

Appendix B: Command Option Flags

This appendix provides a list of the command option flags that you can specify as arguments for many of the Setup script procedures. Use the list below to determine which command option flag to use for a function or subroutine. You can use combinations of command option flags with logical operators (such as AND or OR). For more information about Setup procedures and which ones use command option flags, see Chapter 5, "Setup Script Procedures."

Name	Meaning
cmoVital	This function must be successfully completed or the installation will fail and Setup will be terminated.
cmoDecompress	The file should be decompressed as it is copied. If this option is not specified, the file will be copied byte by byte, whether or not it is compressed.
cmoTimeStamp	Set the timestamp on the file after it has been copied.
cmoReadOnly	Set a read-only attribute on the file after it has been copied.
cmoBackup	Back up any existing version of the file (using the same filename with a .BAK extension) before the source file is copied onto the user's hard disk or network server.
cmoForce	Force the removal of a file from the user's hard disk or network server, even if it has a file attribute of read-only.

cmoOverwrite	Overwrite any existing version of the file on the user's hard disk or network server. Or, overwrite the entry in the .INI file.
cmoAppend	Append the file to the existing file on the user's hard disk or network server rather than replacing the existing file. Or, append the value to the existing value in the .INI file.
cmoPrepend	Prepend the value to the existing .INI value rather than replacing the value.
cmoNone	No command option flag is specified.
cmoAll	All command option flags are specified.

Table B.1 Command Option Flags

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Microsoft®

0392 Part No. 30213