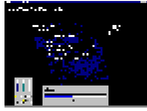
Get the latest versions of InstallShield help files

For the latest versions of InstallShield help files, visit our documentation Web page.

Visit the InstallShield Knowledge Base

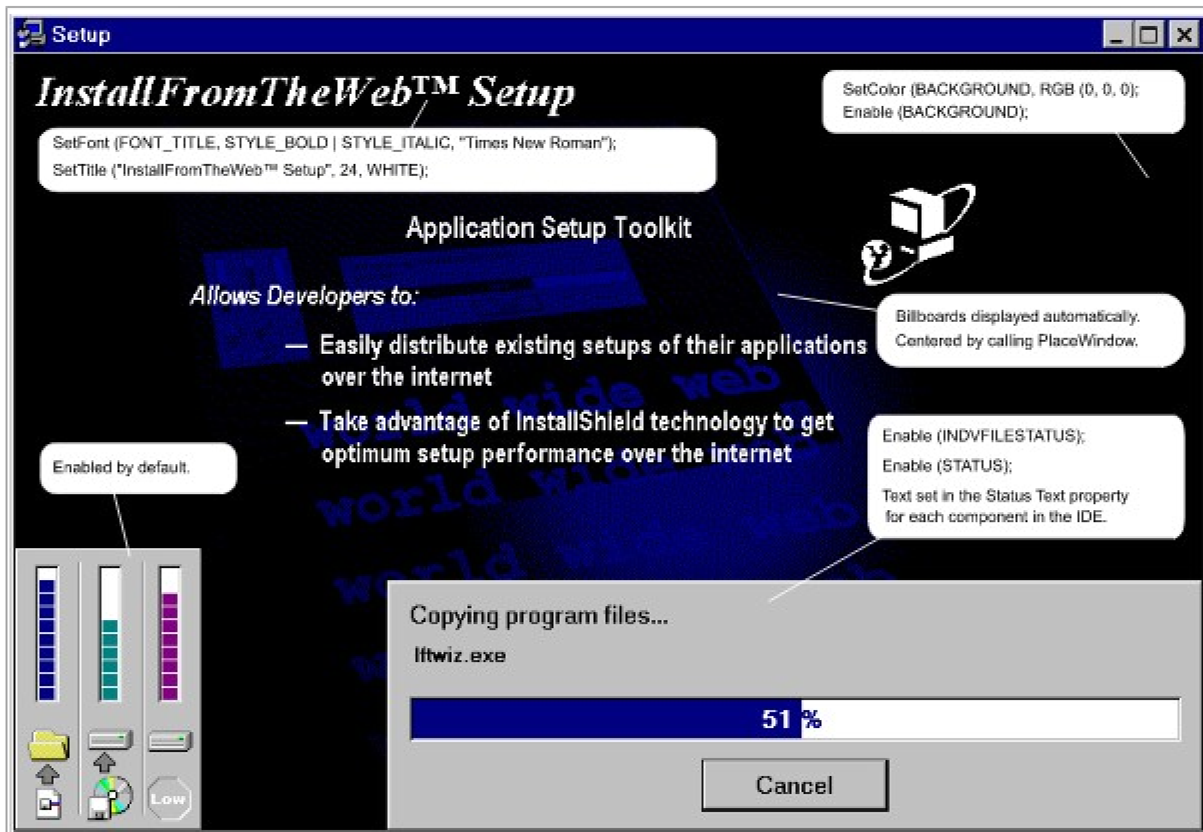
For answers to many commonly asked questions and new information about InstallShield that may not appear in the documentation, visit the InstallShield Knowledge Base.

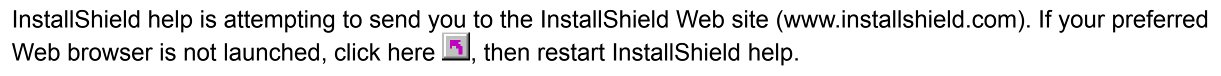
Overview: Main Installation Window



The main installation window is how your end user sees your setup. Thus, it creates your customer's first impression of your application.

The InstallShield graphical user interface provides an intuitive way for the end user to interact with the setup. It establishes a common, consistent communication channel between the setup and the user. The InstallShield graphical user interface elements include popup windows, messages, dialog boxes, icons, bitmaps, and sound and video clips.


Setup
InstallFromTheWeb™ Setup
SetFont (FONT_TITLE, STYLE_BOLD | STYLE_ITALIC, "Times New Roman");
SetTitle ("InstallFromTheWeb™ Setup", 24, WHITE);
SetColor (BACKGROUND, RGB (0, 0, 0);
Enable (BACKGROUND);
Application Setup Toolkit
Allows Developers to:
— Easily distribute existing setups of their applications over the internet
— Take advantage of InstallShield technology to get optimum setup performance over the internet
Billboards displayed automatically. Centered by calling PlaceWindow.
Enabled by default.
Enable (INDVFILESTATUS);
Enable (STATUS);
Text set in the Status Text property for each component in the IDE.
Copying program files...
Iftwiz.exe
51 %
Cancel
Low



InstallShield help is attempting to send you to the InstallShield Web site (www.installshield.com). If your preferred Web browser is not launched, click here, then restart InstallShield help.

InstallShield help found Setbrows.exe but could not run it. Double-click the <InstallShield location>\Program\Setbrows.exe icon.

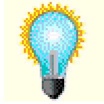
InstallShield help could not find Setbrows.exe. You may have deleted this file; check to see if Setbrows.exe is in your <InstallShield location>\Program folder.

Which graphics formats InstallShield supports

You can display bitmaps, billboards, and video files during your setup using InstallShield. InstallShield supports these image formats:

- 16- and 256-color and 24-bit .bmp or .dib files (that is, any bitmap you can create with MS Paint)
- 16-color transparent bitmaps
- placeable windows metafiles (.wmf files) if your setup is 32-bit
- AVI files

InstallShield allows you to display these images at various points throughout your setup and gives you control over various special effects.



Remember that your setup will run on machines with varying display settings. Always test the visual look of your setup on as many different video configurations as possible, especially if you're using 256-color images or video files.

Billboards are bitmaps or metafiles that you display during the file transfer process. Billboards can communicate, advertise, educate, promote, intrigue, entertain, and otherwise keep your customer's attention.

Place bitmaps according to the target screen resolution

Your users will invariably have screens of different sizes and resolutions. For example, VGA resolution is 640 pixels by 480 pixels, Super VGA is 800 by 600, and XGA is 1,024 by 768. A screen that has a resolution of 1,024 x 768 places a bitmap much closer to the edge than a screen that has a resolution of 640 x 480. Furthermore, a bitmap image appears larger and more "jagged" as the resolution decreases because pixels are larger at lower resolutions. Since the number of pixels in an image does not change, its size must change. The diagram below illustrates the general effect of different screen resolutions on relative placement and size of a bitmap.

Since screens come in different sizes, it is a good idea to get the resolution of the target system's screen before you place a bitmap. To get the size of the screen in pixels, call the GetExtents function as shown:

```
GetExtents(nDx, nDy);
```

The statement shown above returns the width of the screen (in pixels) in the nDx parameter, and the height of the screen (in pixels) in the nDy parameter. Using the nDx and nDy variables, you can place a bitmap in the same relative position, regardless of the screen resolution of the target system.

You specify where you want a bitmap to display with the PlaceBitmap function. PlaceBitmap asks for the number of pixels from the edge of the screen where you want to place the bitmap. You can then use the nDx and nDy results from GetExtents to achieve device-independent placement of the bitmap.

The following example illustrates this concept and assumes that you've already included the Mybitmap.bmp file in the InstallShield IDE Setup Files pane:

```
szName = "Mybitmap.bmp";
PlaceBitmap(szName, nID, nDx/5, nDy/4, UPPER_RIGHT);
```

InstallShield
www.installshield.com

The InstallScript functions for working with bitmaps

These functions allow you to customize the display of bitmaps and other graphic images:

Disable
: Disables user interface objects individually. A disabled object is not displayed.

Enable
: Enables user interface objects individually. When an object is enabled, it is displayed.

GetExtents
: Returns the resolution of the screen on the target system to help place the bitmap.

PlaceBitmap
: Places a bitmap or metafile image anywhere on the screen. Also deletes bitmaps and metafiles when used with the REMOVE argument in the last parameter. PlaceBitmap supports transparent bitmaps with 16 or fewer colors.

SdBitmap
: Displays a bitmap in a dialog box, which can be used at the beginning of the setup.

SetDisplayEffect
: Sets the display effect for bitmap or metafile images. Once the effect is set, it remains in place until changed by another call to SetDisplayEffect.

SizeWindow
: Changes the size of a specific user interface object.

InstallShield
www.installshield.com

Display a bitmap in a dialog box

Call the SdBitmap function to display a bitmap in its own dialog box. SdBitmap is particularly useful for creating splash screens in your setup.

If your bitmap is included in the Setup Files pane, then InstallShield searches the path contained in the SUPPORTDIR system variable at run time. For example, if you wanted to display the bitmap shown above, your script would look like this:

```
szBitmap = "DemoShield.bmp";
SdBitmap( "Title", "Message", szBitmap );
```

When using SdBitmap or any Sd dialog box, remember to `add #include "sddialog.h"` before the program block, and `#include "sddialog.rul"` after the endprogram statement. If you are using the setup script generated by the Project Wizard, these lines are already included for you.

InstallShield
www.installshield.com

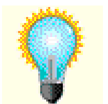
Placing resource files in the Setup Files pane

Place all setup resource files into the Setup Files pane under the folder for the appropriate target language and platform. Setup resource files include DLLs, bitmaps, metafiles, AVIs, and so on, that you need to access during setup.

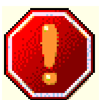
The Setup Files pane has the following structure. Which folder you place your resource files in depends on the types of files you'll be accessing during setup. For more information about which type of files belong in which folder, click on a folder name below.

'<project name>' Setup Files

- **Splash Screen**
  - <target language>
- **<target language>**
  - <target platform>
- **Advanced Files**
  - Disk 1
  - Last Disk
  - Other



When you place a file in the Setup Files pane, InstallShield actually makes a copy of the file and places the copy under the default location: My Installations\<project name>\Setup Files. Therefore, if you change the source file, those changes will not be automatically included in the copy of the file in your setup project. You must re-insert the file into the Setup Files pane in order for the new file to be included with your project.



InstallShield cannot handle project file paths with more than 260 characters (including the file name). The names of the following project elements are incorporated in project paths and file names:

- the project itself
- file groups

- media names
- files in the Scripts pane
- files inserted into the Setup Files pane

To help you keep within the 260-character limit, InstallShield warns you if you attempt to assign a name with more than 50 characters to a project, file group, or media name.

InstallShield
www.installshield.com

The Setup Files\Splash Screen folder

'<project name>' Setup Files

Splash Screen

<target language>

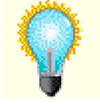
The only file that you should place under the Splash Screen folder is the startup graphic that is displayed when InstallShield initializes your setup.

To insert the startup graphic (which must have a file name of Setup.bmp or Setup16.bmp) into the Setup Files pane

1. Highlight the appropriate folder for your target language. The Setup Files property sheet opens.

   For example, if your startup graphic displayed a corporate logo that was international, highlight the Splash Screen\Language Independent folder. Place each language-specific version of your startup graphic under the corresponding <target language> folder. For more information about including multiple language support in your setup, see How do I choose which languages I want to include in my project.
2. Right-click in the Setup Files property sheet and select Insert Files. A dialog opens that allows you to explore your development system for the location of your startup graphic.
3. Highlight Setup.bmp and/or Setup16.bmp. Click Open.
4. Build your setup with the Media Build Wizard. At this point you can specify which languages you want to include in your built setup.

InstallShield copies Setup.bmp to the Disk1 folder under the default media build location. When you run your setup, you will see the startup graphic displayed during setup initialization.



The Setup Files\<target language>\<target platform> folders

'<project name>' Setup Files

<target language>

<target platform>

Most setup resource files should be placed in the <target language>\<target platform> folders. Place the DLLs and bitmaps that you want to access during setup into these folders, according to the language and platform for which the files are appropriate.

When you build your setup with the Media Build Wizard. InstallShield compresses files under the <target language>\<target platform> folders into _user1.cab. During setup initialization InstallShield creates a temporary support folder and decompresses _user1.cab into it. You can access any file in _user1.cab by appending its file name to the SUPPORTDIR system variable.

For example, suppose you wanted to display Image1.bmp during setup. Image1.bmp is a 256-color bitmap appropriate for all target languages and platforms. Follow these steps:

1. Highlight the appropriate folder for the file's target language and platform—in this case, '<project name>' Setup Files\Language Independent\Operating System Independent. The Setup Files property sheet opens.

   Place each language-specific resource file under the corresponding <target language> folder. For more information about including multiple language support in your setup, see How do I choose which languages I want to include in my project.

2. Right-click in the Setup Files property sheet and select Insert Files. A dialog opens that allows you to explore your development system for the location of Image1.bmp. Highlight Image1.bmp and click Open.

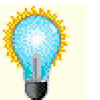
3. Go to your setup script in the IDE's Scripts pane. Insert these lines at the point where you would like to display Image1.bmp; remember to declare all variables before the program block of your script:

```
szName = SUPPORTDIR ^ "Image1.bmp";
PlaceBitmap (szName, nID_BITMAP, nDx, nDy, nDrawOp);
```

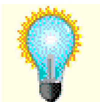
4. Build your setup with the Media Build Wizard. At this point you can specify which languages you want to include in your built setup.

Note that you cannot place files directly in the <target language> folder; you must place them under the <target platform>folder. When you build your setup with the Media Build Wizard, InstallShield also includes only those files in your setup that are appropriate for the target platform(s) you choose professional edition only

.



If you are accessing large files from a CD-ROM or files that are already tightly compressed, such as .avi files, you most likely will not want to store them in the Setup Files\<target language>\<target platform> folders. You might instead choose to leave these files uncompressed on the distribution media. For more information, see The Setup Files\Advanced Files folder.



The Setup Files\Advanced Files folder

'<project name>' Setup Files

Advanced Files

Disk 1

Last Disk

Other

Place files that you want to access during setup from the distribution media into these folders. InstallShield does not compress the resource files that you store in the Setup Files\Advanced Files folders.

Files that are stored on Disk 1 of the distribution media can be accessed at run time—that is, during setup—with the path in the SRCDIR system variable.

And files that are stored on the last disk of the distribution media can be accessed by appending the name of the folder to the SRCDISK system variable. For example, if LargFile.bmp is stored in the Setup Files\Advanced Files\ Last Disk folder, which is named Disk4 on your CD-ROM, you can display the bitmap with the following code:

```
szName = SRCDISK ^ "Disk4\\LargFile.bmp";
PlaceBitmap (szName, 1, nDx, nDy, nDrawOp);
```

When you place files in the Advanced Files\Other folder, you will be asked which folder on the distribution media you want to place those files in when you build your setup with the Media Build Wizard. You can also access those files by appending the name of the folder to the SRDISK system variable.

To place a file in the Advanced Files folders

1. Highlight the appropriate folder under '<project name>' Setup Files\Advanced. The Setup Files property sheet opens.
2. Right-click in the Setup Files property sheet and select Insert Files. A dialog opens that allows you to explore your development system for the location of the file. Highlight the desired file(s) and click Open.
3. Build your setup with the Media Build Wizard. The file is copied to the specified disk on the distribution media.

{button ,AL(`Play video while setup is running;Playing sound during setup;Prompt for the next distribution disk',0,`',`')} See also

Using metafiles (.wmf files) as images

Metafiles create graphical images you can display in your setup. InstallShield supports the use of "placeable windows metafiles" during 32-bit setups. You cannot use metafiles in 16-bit setups.

InstallShield only supports the use of placeable windows metafiles.
It does not support enhanced metafiles

You should be aware that metafiles differ from bitmaps in several ways:

- Metafiles occupy less space on the distribution media.
- A metafile is an instruction set that describes how to draw an image. Because the instructions must be executed one by one on the target system, a metafile image may not display as quickly as a bitmap.
- Metafiles are screen-independent—you can resize the image to accommodate different screen resolutions

InstallShield provides a default position for metafile images that is device-independent. However, you can use two InstallScript functions to manually resize a metafile image or background window: GetExtents and SizeWindow. (Note that you cannot use or resize bitmaps using the SizeWindow function). Call the GetExtents function to determine the resolution of the screen on the target system, as shown below:

```
GetExtents (nDx, nDy);
```

Apply the information obtained from the GetExtents function in the SizeWindow function to place the metafile image on the user interface of the target system in the same relative location, regardless of the screen resolution:

```
SizeWindow (METAFILE, nDx/5, nDy/4);
```

The SizeWindow function instructs InstallShield to make the metafile image or background window the same relative size on the user interface. For example, if a screen has a resolution of 640 x 480, then the SizeWindow example above will place the image in a 128 x 120 (640/5 x 480/4) window. If another screen has a resolution of 800 x 600, the SizeWindow function will place the image in a 160 x 150 (800/5 x 600/4) window.

After you have included image sizing instructions, call the PlaceBitmap function to place and display the image.

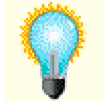
Remove a bitmap

Call the PlaceBitmap function when you want to delete a bitmap. Enter REMOVE in the nCorner parameter.

This example removes Welcome.bmp:

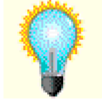
```
szName = SUPPORTDIR ^ "Welcome.bmp";
PlaceBitmap(szName, nID, 5, 5, REMOVE);
```

Always remove a bitmap with the REMOVE constant before placing another bitmap in the same location during the installation. If you place the new bitmap over the previous one, you may see both bitmaps being painted. Leaving layers of bitmaps on the user interface can also slow down the installation.



If you are placing transparent bitmaps, always remove the previous bitmap so that it doesn't show through the transparent areas.

Also, if the next bitmap you want to display is smaller in size than the current one, make sure you remove the current one so that its edges do not show through the next bitmap.



Using transparent bitmaps

Transparent bitmaps allow you to display corporate logos or similar bitmaps as an integral part of the window. The transparent feature is especially useful for placing a bitmap image such as a corporate logo against the main window background. You can "see through" the letters in the bitmap.

To display a transparent bitmap, call the PlaceBitmap function with the BITMAPICON option. By default, InstallShield selects RGB (128, 0, 128) (purple) as the transparent or "see-through" color. You can specify the "transparent" color in the first parameter (szName) of PlaceBitmap. Simply place a semicolon (;) after the name of the bitmap followed by the RGB values for the color you want to specify as transparent. Then set the portions of the bitmap you want to designate as transparent to that color.

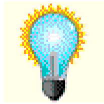
For example, you can enter the following:

```
szName = SUPPORTDIR ^ "TransImage.bmp;255, 128, 64";
```

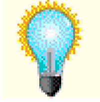
The string instructs InstallShield to set all instances of the color specified by the RGB designation (yellow) to be "transparent" in the bitmap when the bitmap appears in the installation window. The SUPPORTDIR path means that the bitmap is stored in the Setup Files pane.

This example sets the color yellow as transparent in the bitmap TransImage.bmp:

```
szName = SUPPORTDIR ^ "TransImage.bmp;255,128,64";
PlaceBitmap(szName, nID, nDx, nDy, BITMAPICON);
```



InstallShield sets as transparent all instances of the color you designate as transparent, regardless of where the color appears in the bitmap. In other words, double-check where that color appears in your bitmap or undesired areas may be displayed as transparent.



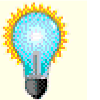
Access bitmaps during setup

When stored in the Setup Files pane...

You can always store a bitmap file directly in the InstallShield IDE's Setup Files pane under the appropriate target language(s) and platform(s). Since the PlaceBitmap function requires a fully qualified path and file name, append the name of the .bmp or .wmf file to the SUPPORTDIR system variable, as follows:

```
szName = SUPPORTDIR ^ "Welcome.bmp";
```

When displaying bitmaps from .bmp files with the PlaceBitmap function, you can use any number as an ID (the nID_BITMAP parameter); however, each bitmap used in the setup must have a unique ID.



Do not place bitmap files directly onto the distribution media. For example, if InstallShield needs to redraw a bitmap stored on Disk1, but Disk2 is currently in the drive, the effects will be undesirable. Instead, store all graphics in the Setup Files pane.

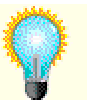
When stored as a resource in a DLL...

You can also store a bitmap file as a resource in a dynamic-link library (DLL). Be sure to place your DLL in the Setup Files pane under the appropriate target language(s) and platform(s), and then you can access it by appending the DLL's file name to the SUPPORTDIR system variable.

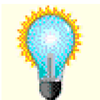
The ID number for the bitmap must match the resource ID for that bitmap in the DLL's .rc file. The resource ID for an existing bitmap in a DLL file can be determined by using MSVC++ 5.0.

Suppose that you placed Images.dll in the Setup Files pane and the bitmap's resource ID was 12004 and you wanted it to appear in the upper right corner of the main installation window, call PlaceBitmap as follows:

```
szName = SUPPORTDIR ^ "Images.dll";
nID_BITMAP = 12004;
PlaceBitmap (szName, nID_BITMAP, nDx/20, nDy/20, UPPER_RIGHT);
```



InstallShield provides you with _isuser.dll, an empty DLL that you can populate with resources for use in your setup. Since _isuser.dll is dynamically renamed to allow for multiple concurrent setups, you should use the ISUSER system variable to access resources stored in this file.



Tile a bitmap across the main window

Call the PlaceBitmap function with the TILED option as follows:

```
szName = SUPPORTDIR ^ "TileImg.bmp";
PlaceBitmap (szName, nID, nDx, nDy, TILED);
```

The above code assumes that TileImg.bmp was stored in the Setup Files pane and is therefore accessible from the path contained in the SUPPORTDIR system variable.

Depending on the screen resolution and the size of your main installation window, the bitmap may appear clipped at the right or the bottom of the window. You may instead choose to display a single tiled bitmap stretched to cover the entire main installation window. For more information, see How do I get a bitmap to cover the entire main window.

{button ,AL(`Using full-screen or windowed main windows',0,`',`')} See also

Get a bitmap to cover the entire main window

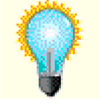
Call the PlaceBitmap function with the FULLSCREENSIZE option as follows:

```
szName = SUPPORTDIR ^ "FullScr.bmp";
PlaceBitmap (szName, 10, nDx, nDy, FULLSCREENSIZE);
```

The FULLSCREENSIZE options stretches the image to fill the entire main installation window. Use the FULLSCREEN option if you want to center a large bitmap in the main window and have the background color filled in on the edges when the image is smaller than the complete screen, depending on the target screen resolution.

The above code assumes that FullScr.bmp was stored in the Setup Files pane and is therefore accessible from the path contained in the SUPPORTDIR system variable.

{button ,AL(`Set the background color and pattern;Tile a bitmap across the main window;Using full-screen or windowed main windows',0,`',`')} See also



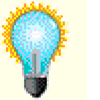
Displaying billboards

Billboards are bitmap or metafile images that you display to communicate, advertise, educate, promote, intrigue, entertain, and otherwise keep your users' attention during the setup.

Billboards are activated only during the file transfer phase of setup, which means that you can display billboards only when you are (decompressing and) copying files with ComponentMoveData. Billboards are enabled automatically when file transfer begins—you do not call a function to enable billboards during the file transfer process. All you have to do is include your billboard files in the Setup Files pane under the appropriate target language(s) and platform(s) and name the billboards correctly.

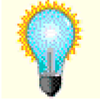
You cannot specify a length of time for a given billboard to display. InstallShield spaces them evenly throughout the file transfer process, basing each billboard's display time on the size of the files being transferred. To lengthen the time that a particular billboard is on screen, you must display the same billboard more than once but with a different name each time.

You can display as many billboards as you wish, but be aware that InstallShield displays each billboard for a minimum of two seconds. So, for example, if you list 25 billboards but your file transfer lasts only 20 seconds on a particular system, only the first ten billboards will display.



InstallShield does not remove any billboards until file transfer has completed—each billboard is drawn directly over the previous one. This method can cause problems displaying transparent bitmaps (the previous billboards' opaque areas can show through the transparent areas) or displaying bitmaps of different sizes (larger bitmaps will be visible behind smaller bitmaps).

{button ,AL(`Displaying bitmaps and billboards with special effects;Move my billboards to a different location;Preventing color distortion',0,`',`')} See also



Name my billboards

Billboards are displayed only during file transfer, which begins with your call to ComponentMoveData.

Since InstallShield automatically determines the target screen resolution, you can include billboards for both high and low resolutions. InstallShield will then display only those billboards appropriate for the target screen resolution. To display a billboard on low-resolution screens, simply insert an "l" (as in "low") into the name of the billboard, as described below.

The default naming convention for billboards is to use "Bbrd" followed by an "l" for low resolution (VGA) where appropriate, followed by a number and a .bmp or .wmf file extension. InstallShield automatically displays any billboards named according to this convention. For example, the first billboard displayed on a system with high screen resolution might be named Bbrd1.bmp.

Since billboards must be placed into the Setup Files pane, InstallShield searches the SUPPORTDIR folder first for .wmf files and then for .bmp files. If InstallShield does not find any, it assumes that you are not displaying billboards during file transfer.

{button ,AL(`Displaying billboards',0,`',`')} See also

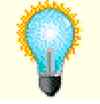
Move my billboards to a different location

By default, InstallShield displays billboards in the center of the main installation window. To specify a different location, call the PlaceWindow function with the BILLBOARD option. For example, to place your billboards 10 pixels from the upper left-hand corner of the screen, make the following call:

```
PlaceWindow (BILLBOARD, 10, 10, UPPER_LEFT);
```

Since billboards are only displayed during file transfer, make sure that you call PlaceWindow before you begin file transfer by calling ComponentMoveData.

{button ,AL(`Displaying billboards',0,`',`')} See also



Displaying bitmaps and billboards with special effects

professional edition only

The SetDisplayEffect function allows you to display your bitmaps with different special effects when they first appear on the main installation window. Choose one of SetDisplayEffect's options before you display a bitmap with PlaceBitmap or display billboards during file transfer.

EFF_FADE
When you use this option, the bitmap or billboard slowly fades in and out.

EFF_REVEAL
When you use this option, the bitmap or billboard gradually fills in from the center toward all sides.

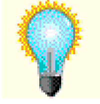
EFF_HORZREVEAL
When you use this option, the bitmap or billboard gradually scrolls out from its center horizontally.

EFF_HORZSTRIPE
When you use this option, the bitmap or billboard partially fills in horizontally from the outside in and then completely fills in from the center out.

EFF_VERTSTRIPE
When you use this option, the bitmap or billboard partially fills in vertically from the outside in and then completely fills in from the center out.

EFF_BOXSTRIPE
When you use this option, the bitmap or billboard partially fills in from all sides and then completely fills in toward all sides.

EFF_NONE
This option is the default setting. You can use it to clear the display effects after calling one of the other options.



Preventing color distortion

If you are experiencing problems with color distortion, it is most likely due to shift in the color palette. This section explains how colors are apportioned on Windows systems and offers troublehshooting tips to help you minimize any distortion in your setup's graphical interface.

On systems operating in 256-color mode, there is one 256-color palette for displaying all available colors, in 16-color mode a 16-color palette, regardless of the number of colors that the video driver supports. Systems operating in high-color or true-color mode do not use a color palette of any type; instead, the colors are displayed directly. hat is, you should never experience any color distortion on these systems. On systems using a color palette, entries are allocated and used as needed when an object, such as a bitmap, is displayed.

Once the current color palette has been allocated, to display a new color Windows will attempt to use a color similar to one that was already allocated. Therefore, distortion may result on a 256-color system if multiple objects that contain several different colors are displayed simultaneously.

Also, remember that MS Windows always reserves 20 palette entries for system usage. These colors are always allocated, even if another object requires these colors. For further details on color palettes, refer to the *Microsoft Development Reference Guides*.

Use the tips below to minimize any color distortion related to color palette shifts. Color palette tips apply to all images you display during setup, including the background color, AVIs, metafiles, and bitmaps.

Main installation window background

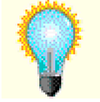
- Try using a solid color background, which requires only one palette entry. Call the SetColor function with a solid background constant to create a solid background.
- Try using a 16-color gradient background, which requires only 16 palette entries. Call the SetColor function with one of the gradient color constants to create this type of background.
- Try using a tiled or centered bitmap background. Call the PlaceBitmap function to create one of these backgrounds.
- Avoid using a 256-color gradient background, which requires about 80 palette entries.

Bitmaps and metafiles

- Make sure that you are using the intended display mode in your setup. The default display mode is 256-color mode, if you are attempting to display a 16-color gradient background, you must first call the Disable function with the BITMAP256COLORS option to enable 16-color mode before calling the SetColor function to create the background. Re-enable 256-color mode by calling Enable(BITMAP256COLORS).
- When a bitmap is removed, all color palette entries used by that bitmap that are not being used by any other currently displayed image are freed from the color palette when other colors are needed to display other objects. Therefore, always remove a bitmap when it is no longer being displayed by calling the PlaceBitmap function with the REMOVE option (instead of covering it with another bitmap).
- Try to use similar colors for bitmaps and billboards displayed during the setup to reduce the number of color palette entries needed.
- You can always determine the maximum colors available on the target system by calling the GetSystemInfo function with the COLORS option. Perhaps you will want to display bitmaps of different resolutions based on the return value. For information on displaying 16-color billboards, see Name my billboards.
- 24-bit bitmaps do not include a custom color palette in the bitmap file. When displaying a 24-bit bitmap on a 256-color system, only the currently available palette entries will be used, even if there are additional color palette entries available. If you are displaying 24-bit bitmaps in your setup and you expect it to run on 256-color systems, it is recommended that you also include 256-color versions of your bitmaps.
- Since metafiles are drawn instead of placed, additional color palette entries are not allocated when a metafile is drawn; only the existing color palette entries are used to display the metafile. If you expect your setup to run on 256 color syste,ms it is recommended that you use only the standard 16 system colors in your metafile, because these 16 colors will be available regardless of the current color palette's availability.

- Verify that the system's video driver supports 256 or more colors. If the video driver doesn't support 256 colors, the bitmap will not display properly, even if your video card and monitor support 256 colors.
- If a monitor is not capable of displaying 256-color bitmaps, images can appear blurry or "spotted." Blurry images occur when the target system attempts to closely approximate the requested color.

{button ,AL(`Place bitmaps according to the target screen resolution;Troubleshooting problems loading bitmaps',0,`',`')} See also

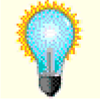


Troubleshooting problems loading bitmaps

Assuming that you're using the correct image formats and have made the function calls correctly, your bitmaps should load and display without a hitch. Use these tips as a checklist if you're running into problems getting bitmaps to load:

- Verify that you have specified the parameters to PlaceBitmap correctly. Try specifying the path to the bitmap explicitly instead of using variables, constants, or string identifiers.
- Verify that the bitmap is available at the location specified when PlaceBitmap is called. If necessary use the MessageBox function to temporarily stop the installation and then verify that the file does exist via Explorer. If you have placed the bitmap into the Setup Files pane, it should be located in the temporary support folder (SUPPORTDIR) during setup. You can display the value of SUPPORTDIR at run time or use the InstallShield Visual Debugger to display its contents.
- If the bitmap has been placed in a DLL, make sure that no RLE compression has been specified. InstallShield cannot display RLE compressed bitmaps.
- Open the bitmap in MS Paint. If the bitmap is not displayed properly in Paint, then it cannot be displayed properly by InstallShield. If the bitmap is displayed correctly in Paint, save it as a 16- or 256-color bitmap and attempt to display it in during setup.

{button ,AL(`Access bitmaps during setup;Displaying billboards;Preventing color distortion;The InstallScript functions for working with bitmaps;Which graphics formats InstallShield supports',0,`',`')} See also



The InstallScript functions for setting the main window's characteristics

You can use the following InstallScript functions to set the characteristics of the main installation window and place objects in it.

Disable
  Disables user interface objects individually. A disabled object is not displayed.

Enable
  Enables user interface objects individually. When an object is enabled, it is displayed.

FindWindow
  Retrieves a handle to a window.

PlaceBitmap
  Places a custom bitmap anywhere on the screen. Also identifies bitmaps with transparent areas.

PlaceWindow
  Sets the position of most user interface objects. If you don't call PlaceWindow, InstallShield uses the default positions.

SetColor
  Changes the color of various elements.

SetDialogTitle
  Creates custom dialog box titles.

SetErrorMsg
  Changes the text of error messages.

SetErrorTitle
  Changes the title of error message boxes.

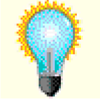
SetFont
  Sets the font type and style.

SetStatusWindow
  Sets the text for the status window and sets the initial percentage complete displayed progress indicator.

SetTitle
  Sets the text and color of the title for the main window.

SizeWindow
  Specifies the size of most user interface objects.

StatusUpdate
  Links the information gauges to the progress indicator and sets the percent complete to display in the progress indicator after the next file transfer operation.



Set the background color and pattern

The first task in your setup script is to establish the look and feel of the setup. If you like the default settings, you do not need to change anything. The default background color is solid teal—RGB (0, 128, 128).

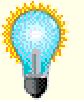
To change the color of the main installation window, call the SetColor function with the BACKGROUND option. In the second parameter you can specify either a predefined constant or an RGB expression. For example, the statement below sets the background to gradient yellow:

```
SetColor (BACKGROUND, RGB (255, 255, 0) );
```

Many of the options available for setting the background color already provide a gradient effect. If you want to add a gradient effect to any custom color or solid color provided in one of the constants, simply combine it with the BK_SMOOTH option using the bitwise OR operator. If you wanted, for example, to create a gradient purple background, you would type:

```
SetColor (BACKGROUND, RGB (128, 0, 255) | BK_SMOOTH);
```

If you are trying to display a background color as soon as setup starts, you must call SetColor before calling Enable to enable the background or set the main installation window to full-screen or windowed mode.



You can also fill the main installation window with bitmaps. For more information, refer to How do I get a bitmap to cover the entire main window? and How do I tile a bitmap across the main window?

{button ,AL(`Preventing color distortion;Using full-screen or windowed main windows',0,`',`')} See also

A gradient effect is achieved by gradually blending a color at the top of the main installation window with a black background at the bottom.

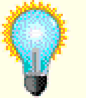
Set a title for the main window

The default font and style is Times New Roman, bold, italic, and shadowed. To customize the look, call the SetFont function to set the font style. If you wanted to display the title in MS Sans Serif, bold, and with a shadowed effect, you would type:

```
SetFont (FONT_TITLE, STYLE_BOLD | STYLE_SHADOW, "MS Sans Serif");
```

Then, call the SetTitle function to set the title, the font size, and its color as in this script fragment:

```
SetTitle ("Welcome to Setup!", 45, YELLOW);
```



Currently, InstallShield only supports placing the installation's title in the upper left corner. However, you can use spaces in the beginning of the title string to push it to the right, or break it across multiple lines using the newline escape sequence (\n).

Another method is to make a bitmap of your title and place it with the PlaceBitmap function instead of calling SetTitle.

Using full-screen or windowed main windows

You have the choice of creating the main installation window as a full-screen object or as a window. By default, InstallShield displays a full-screen main window.

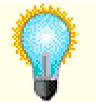
To create a windowed main window, make the following function calls:

```
Enable(DEFWINDOWMODE);
Enable(BACKGROUND);
```

You can control a windowed main window with these functions:

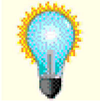
PlaceWindow
  Sets the position of the main window (when windowed). InstallShield uses default positions if you do not call this function.

SizeWindow
  Specifies the size of the main window (when windowed).



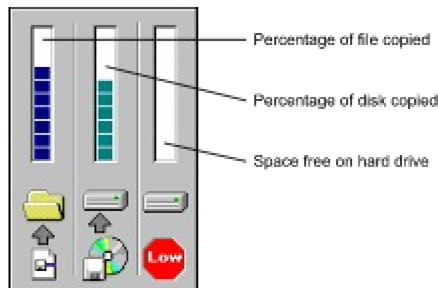
You can call the GetExtents function to get the dimensions of the target screen and then adjust the size of your main installation window proportionally.

{button ,AL(`Get a bitmap to cover the entire main window;Set the background color and pattern;Tile a bitmap across the main window',0,`',`')} See also



The information gauges

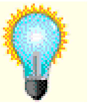
The three information gauges in the main installation window provide your end user with feedback during setup:



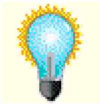
The left-hand gauge shows the progress of each file as it is being copied to the target system. It is the most active gauge in the main window, constantly moving from the bottom (0%) to the top (100%) as each file is copied.

The middle gauge displays the progress of each distribution disk as it is being copied to the target system.

The right-hand gauge indicates how much disk space is still available on the target drive. When the target drive has less than 5% space available, the Low light becomes bright red.



The information gauges display by default. To avoid displaying them, call Disable(FEEDBACK_FULL) before beginning file transfer and before enabling the progress indicator with Enable(STATUSDLG).



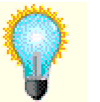
Enable and disable user interface objects

InstallShield provides many setup options that you can turn on and off by calling the Enable and Disable functions.

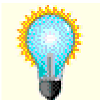
Call the Enable function when you want to allow an individual user interface object to be displayed in the main installation window. You can enable only one object each time you call the Enable function in the script. For example, to enable the caption bar and the progress indicator, type:

```
Enable(BACKGROUNDCAPTION);
Enable(STATUS);
```

Call the Disable function if you want an object in the main window to disappear. The Disable function removes an object from the user interface. You can disable only one object each time you call Disable.



Many events and objects are already either enabled or disabled by InstallShield by default.



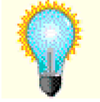
Displaying dialog boxes in the main window

Dialog boxes are windows that pop up and prompt you to perform some action. Dialog boxes allow you to check items in a window list, set push buttons for various choices, and directly enter strings and integers from the keyboard. When you select one or more of the various dialog box buttons, check boxes, and so on, InstallShield provides the means to process the message information by passing the return value from the dialog box function to a procedure in the setup script.

There are two types of dialog boxes you can create and use in the setup script:

- Built-in dialog boxes and Sd dialog boxes you create or modify using InstallScript functions
- Custom dialog boxes you create using a resource editor, such as MSVC++ 5.0, and then use in the setup script

For more information about dialog boxes, see the Dialog Boxes section of this help file.



Displaying the progress indicator

InstallShield can display and increment a progress indicator dialog box during file transfer—that is, once you call ComponentMoveData, CopyFile, or XCopyFile in your setup script.

You can also display a progress indicator to keep the user informed of the progress of other events in your setup. In addition, you can call various InstallScript functions to set certain characteristics of the progress indicator.

The progress indicator is enabled by default. Call the Enable function to display the progress indicator style of your choosing. These style options are available:

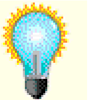
STATUSDLG
The standard progress indicator, similar to a dialog box with a Cancel button.

STATUS
The progress indicator with a Cancel button but neither a border nor a title bar. This is the progress indicator that is enabled if you use the setup script generated by the Project Wizard.

STATUSOLD
The progress indicator without a Cancel button.

Once you enable the progress indicator, you can further set its characteristics, as described below:

- You can specify a message that you want to appear in the progress indicator as each component is installed. (Components are installed when you call the ComponentMoveData function.) For more information, see How do I display a message for each component in the progress indicator.
- If you are transferring files with ComponentMoveData, CopyFile, or XCopyFile, you can display each file's name in the progress indicator. Call the Enable function with the INDVFILESTATUS option to have InstallShield automatically display each file's name.
- Call the SetDialogTitle function with the DLG_STATUS option to change the title of the progress indicator. You must then call Enable(STATUSDLG) for the change to take effect.
- Call the SetColor function with the STATUSBAR option to change the color of the STATUS-type progress bar.



You cannot use the following functions to change the progress indicator that displays once you call ComponentMoveData to begin file transfer.

- Call the SetStatusWindow function to set the progress bar's percentage and to display a line of general information in the progress indicator.
- Call the StatusUpdate function to control the percentage that the progress bar moves to during an event.

{button ,AL(`Display and increment a progress indicator for a setup event;Specify where I want the progress indicator to display;Display a message for each component in the progress indicator',0,`',`')} See also

Display and increment a progress indicator for a setup event

You can update the progress bar to reflect the performing of operations not involving file transfer (for example, the creation of a program folder and icons). First, you must enable the style of progress indicator you would like to display.

Then, call SetStatusWindow to display the progress indicator. Increment the progress bar by passing a new percentage as the first parameter; the progress bar will immediately "jump" to its new position.

```
SetStatusWindow(95, "Creating program folder and icons...");
// Progress bar is now at 95%.
...
// Create program folder and icons.
...
SetStatusWindow(100, "Installation complete.");
// Progress bar has now jumped to 100%.
```

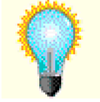
To increment SetStatusWindow smoothly, call it within a while loop, incrementing the nPercent parameter a percentage point with each iteration.

To change the text in the progress indicator without changing the progress bar, call SetStatusWindow with -1 as the first parameter and the new text as the second parameter.

{button ,AL(`Displaying the progress indicator',0,`',`')} See also

Specify where I want the progress indicator to display

Call the PlaceWindow function to move the progress indicator to another location on the user interface.



Display special characters (© ,® , and TM) in dialogs and titles

To produce the copyright and registered trademark characters, hold down the Alt key and type the indicated numbers on the numeric keypad. (Make sure that the Num Lock is on when typing the numbers.)

	Symbol	Keystrokes
Copyright symbol	©	Alt+0169
Trademark symbol	TM	Alt+0153 *
Registered trademark	®	Alt+0174

* The trademark character (TM) is not supported by all text editors, so you may need to use the letters TM.

Alternatively, you can create the string with the Character Map program in the Windows Accessories folder, and then paste it into your script. When you use the Character Map program, you can only copy characters from the same font group for the dialog box to which you are adding the string. If you use any special characters from other font types, they will be mapped to their ASCII equivalent in the appropriate font table when the character is displayed, and this character will probably not be similar to the intended character. It is not possible to mix font types when displaying a dialog box in InstallShield.

{button ,JI(`GETRES.HLP',`Which_fonts_InstallShield_dialog_boxes_use')} See also

Bring the main window to the front

professional edition only

Often setups launch other applications that place their windows on top of the main installation window. To bring the main window back on top and give it focus you can call the Windows API SetForegroundWindow. See the example script for more information.

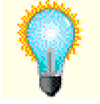
Script for bringing the main window to the front

```
  HWND hMain;
  NUMBER nReturn;

  // Prototype the Windows API SetForegroundWindow.
  // Note:  if you're running a 16-bit setup, prototype the 16-bit version
  // of the function (Use User.dll, instead of User32.dll.)
  prototype User32.SetForegroundWindow(HWND);

program

  // Specify windowed mode, as opposed to full-screen mode.
  Enable(DEFWINDOWMODE);
  Enable(BACKGROUND);

  // Launch an application that will put a window on top of the InstallShield window.
  MessageBox("Launch program.", INFORMATION);
  LaunchApp("Notepad.exe", "");

  // Get the handle of the main installation window.
  hMain = GetWindowHandle(HWND_INSTALL);

  // Bring the main installation window to the foreground.
  nReturn = SetForegroundWindow(hMain);

  // Wait five seconds for testing so we can see that it was the last function that
brought
  // the window to the foreground and not the MessageBox we're going to display.
  Delay (5);

  // Report whether or not the SetForegroundWindow function was successful.
  if (nReturn < 0) then
     MessageBox("Failure!", WARNING);
  else
     MessageBox("Success!", INFORMATION);
  endif;

endprogram

// Source file:  Is5gr001.rul
```

Get the handle for the main window

Call the GetWindowHandle function to get the handle for the main installation window as follows:

```
hMain = GetWindowHandle (HWND_INSTALL);
```



Display a startup graphic

You can display a custom startup graphic during setup initialization to advertise your company or product.

InstallShield will automatically display your startup graphic in the center of the screen if you include a file named Setup.bmp in the IDE's Setup Files pane under the Splash Screen\Language Independent folder. The maximum size for Setup.bmp is 520 x 316 pixels.

InstallShield assumes that Setup.bmp is a 256-color bitmap. Since 256-color bitmaps may not display the intended colors on 16-color systems, you can also include a 16-color version of the startup graphic and title it Setup16.bmp. InstallShield automatically determines the target screen's maximum color depth during setup initialization and will display Setup16.bmp if the target system supports only 16 colors. If you don't include a Setup16.bmp, you can always use a 16-color bitmap for Setup.bmp.

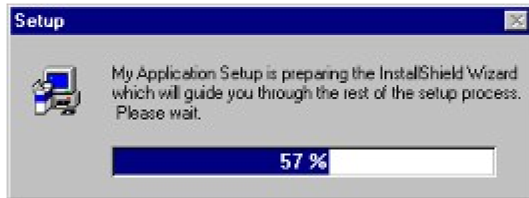
InstallShield also automatically relocates the startup message to the lower right-hand corner of the main installation window to allow Setup.bmp to display in the center.

{button ,AL(`Display my company name in the startup message',0,`',`')} See also

Display my company name in the startup message

professional edition only

The startup message is designed to be consistent with Microsoft's *User Interface Guidelines for Windows Setup Programs*. A consistent look and feel for all setup programs increases the end user's confidence in the installation process. An example startup message is shown below.



The name that appears in the startup message is the same value you enter for Application Name in the Project Wizard - Welcome panel.

You can later edit what is displayed in the beginning of the text string (where "My Application" is located in the above example). The text that is displayed is taken from the AppName value in Setup.ini.

Showing video during setup

professional edition only

You can now show video and animation clips on the main installation window. Use video and animation to advertise your company's products and give your setup an engaging and professional look!

InstallShield offers built-in support for AVI files. We recommend using 16-color AVI files to avoid problems with the color palette, but InstallShield also supports 256-color AVI files.

Make sure you place your video files in the InstallShield IDE's Setup Files pane.

Know your audience. Naturally, you will want to tailor your setup for your customers' needs. Before using AVI files, consider the following:

- What type of multimedia effects are appropriate for this product and its purchasers?
- Will they be likely to have faster systems with high resolution monitors?
- Can I assume that my customers will have the necessary hardware and/or drivers to play video and sound?

Also take your distribution media into account. Displaying video is recommended only for CD-ROM setups because of the size involved. AVIs are already tightly compressed and are often larger than 1.4 MB diskettes.

{button ,AL(`Place video on the main window;Play video while setup is running;The InstallScript functions for playing sound and video',0,`',`')} See also

InstallShield
www.installshield.com

Playing sound during setup

InstallShield
www.installshield.com

You can now play sound during your setup. InstallShield supports both WAV and MIDI formats. Of course, your customers must already have the necessary drivers and hardware on their systems to be able to hear the sound play.

Be sure to place your audio files in the InstallShield IDE's Setup Files pane.

Consider the following before deciding which sound format you want to use:

- MIDI files are more compact than WAV files, but have lower resolution or clarity. In addition, MIDI files require special drivers to run.
- WAV files, on the other hand, are less compact than MIDI files, but have higher resolution and clarity. In addition, all Windows systems have a WAV file driver present.

{button ,AL(`Play background music during setup;Stop setup to play audio;The InstallScript functions for playing sound and video',0,`',`')} See also

InstallShield
www.installshield.com

The InstallScript functions for playing sound and video

InstallShield
www.installshield.com

Use these functions for playing and controlling AVI, WAV, and MIDI files.

PlaceWindow
Specifies where on the main window you want an AVI file to run.

PlayMMedia
Plays a video or sound file. Also specifies how you want it played.

SizeWindow
Determines the size of the window in which the AVI file plays.

Place video on the main window

Before you call the PlayMMedia function to begin playing your AVI, you can specify where you want it to play and the size of the window you would like to play in.

Call the PlaceWindow function to tell InstallShield where to place your AVI file. The following script fragment places an AVI 50 pixels from the upper left corner of the main installation window:

```
PlaceWindow (MMEDIA_AVI, 50, 50, UPPER_LEFT);
```

Call the SizeWindow function to specify the size of the window in which your AVI will play. The following script fragment runs Promo.avi in a 60 by 100 pixel window:

```
SizeWindow (MMEDIA_AVI, 60, 100);
```

You can use the nDx and nDy values from the GetExtents function to help you size your window proportionally.

Play background music during setup

Call the PlayMMedia function with the MMEDIA_PLAYASYNCH option to play the WAV or MIDI file "asynchronously." Playing audio asynchronously means that InstallShield will continue with the setup while it plays the WAV or MIDI file in the background. For example, if you wanted to play Mozart.wav continuously in the background during your setup, you would type:

```
szFileName = SUPPORTDIR ^ "Mozart.wav";
PlayMMedia (MMEDIA_WAVE,
            szFileName,
            MMEDIA_PLAYASYNCH | MMEDIA_PLAYCONTINUOUS,
            // play both in the background and continuously
            0);
```

The above code assumes that Mozart.wav was stored in the Setup Files pane.

InstallShield
www.installshield.com

Stop setup to play audio or video

InstallShield
www.installshield.com

Call the PlayMMedia function with the MMEDIA_PLAYSYNCH option to play the AVI, WAV, or MIDI file "synchronously." Playing a multimedia file synchronously means that InstallShield will stop setup to run the file and then resume execution once the file has completed. For example, to make setup wait while Welcome.mid plays, type:

```
szFileName = SUPPORTDIR ^ "Welcome.mid";
PlayMMedia (MMEDIA_MIDI,
            szFileName,
            MMEDIA_PLAYSYNCH     // stop setup until done
            0);
```

The above code assumes that Welcome.mid was stored in the Setup Files pane and is therefore accessible from the path contained in the SUPPORTDIR system variable.

InstallShield
www.installshield.com

Play video while setup is running

InstallShield
www.installshield.com

Call the PlayMMedia function with the MMEDIA_PLAYASYNCH option to play the AVI file "asynchronously." Playing video asynchronously means that InstallShield will continue with the setup while it plays the file in the main window. For example, if you wanted to play Logo.avi during your setup, you would type:

```
szFileName = SUPPORTDIR ^ "Logo.avi";
PlayMMedia (MMEDIA_AVI,
            szFileName,
            MMEDIA_PLAYASYNCH // play while setup runs
            0);
```

The above code assumes that Logo.avi was stored in the Setup Files pane and is therefore accessible from the path contained in the SUPPORTDIR system variable.

Setup.ini

Setup.ini is an initialization file that InstallShield creates to control certain elements of the setup. InstallShield only fills in certain keynames and values in Setup.ini. After InstallShield creates Setup.ini, it is placed in the Disk folder in the default location: My Installations\<project name>\Media\<media name>\Disk Images.

If you want to customize Setup.ini further, you must modify it yourself with a text editor. But be careful about copying any files to the disk image folders or modifying any of the existing files. The next time that you build your setup with the Media Build Wizard using the same media name, it will delete the existing files.

Setup.ini contains two predefined sections:

{button ,JI(`GetRes.HLP',`The_Startup_Section')} [Startup]

{button ,JI(`GETRES.HLP',`The_Mif_Section')} [Mif]

You can add additional sections to Setup.ini to pass information to your setup script. You can then call the GetProfString and GetProfInt functions to transfer the information from the Setup.ini file to your setup.

Here is an example Setup.ini file:

```
[Startup]
AppName=My Application
FreeDiskSpace=1941
CmdLine=/fTest1.ins
EnableLangDlg=Y

[Mif]
Type=SMS
File name=Ishield
SerialNo=IS50-32XYZ-12345
```

If you need to access the file later (after Disk1 has been removed), you should copy Setup.ini to the support folder (SUPPORTDIR) at the beginning of the setup.

InstallShield
www.installshield.com

The [Startup] section

You can use the following keynames in the [Startup] section of Setup.ini:

- AppName
- FreeDiskSpace
- CmdLine
- EnableLangDlg

AppName

The AppName keyname identifies an application or product name to be displayed at the beginning of the text string in the startup message dialog box.

FreeDiskSpace

The FreeDiskSpace keyname tells InstallShield how much free disk space (in kilobytes) the files compressed in _user1.cab and _sys1.cab will require (both compressed and decompressed). InstallShield calculates this figure and enters it into Setup.ini for you.

CmdLine

InstallShield reads the command line parameters specified in Setup.ini first, and then appends any command line parameters passed to Setup.exe from the command prompt.

For more information about available command line parameters, see Setup.exe and command line parameters.

EnableLangDlg

The EnableLangDlg tells InstallShield whether to display the Language dialog during setup initialization. The Language dialog lets your end user decide which available language setup should run in.

For more information about the Language dialog, see How InstallShield determines which language the setup should run in.

InstallShield
www.installshield.com

The [Mif] section

If the [Mif] section is present InstallShield will automatically create a setup .mif file in the Windows folder. For more information, see How do I create a setup .mif file.

These are the keynames under the [Mif] section of Setup.ini:

- Type
- File name
- SerialNo

Type

Set this key to SMS.

File name

This key is optional. It provides the alternate name for the .mif file to be created. If this key is not included, InstallShield tries to use the AppName key under the [Startup] section of Setup.ini as the .mif file name. If the AppName key is also not present, InstallShield creates a file with the default name Status.mif.

The file name should not include an extension, since .mif files must have the .mif extension. The file name should not include a path—it is placed in the Windows folder by default.

SerialNo

This key is also optional. If provided, the information from this key is placed in the "Serial Number" section in the .mif file. If this key is not present, InstallShield instead places "".

The following is an example of a Setup.ini file for a setup that will automatically create an .mif file.

```
[Startup]
AppName=InstallShield

[Mif]
Type=SMS
File name=IShield
SerialNo=IS50-32XYZ-12345
```

InstallShield
www.installshield.com

Overview: Dialog Boxes

Dialog boxes contain controls for communicating with the user. Controls such as lists, edit fields, check boxes, radio buttons, and push buttons inform the user, allow the user to select options, and prompt the user for input.

When the user responds to a dialog box via its controls, the dialog box sends corresponding messages to the system. InstallShield provides these messages to your setup script, where they can be used to control script execution.

InstallScript contains many built-in functions that automatically display dialog boxes. You display these built-in dialogs with simple function calls. In addition, InstallShield provides many powerful and versatile Sd dialog boxes. You can also display an Sd dialog box with a simple function call. (To make the Sd dialog boxes accessible, you must put two additional statements in your script.)

Built-in and Sd dialog box functions are described below, according to the function they perform in your setup. Please glance through these topics to determine if InstallShield provides a function that will automatically display the dialog box you want. If a built-in or Sd dialog box function needs only minor modifications to meet your requirements, you may be able to call the DialogSetInfo function to achieve the desired result.

If your setup needs a dialog not supplied by InstallShield, then you must create your own custom dialog boxes using a resource editor. You then store the dialog in a DLL and use the InstallScript language to handle the dialog, as described in the Custom Dialog Boxes section.

Including Sd dialog functions

All Sd dialog box functions are prototyped in Sddialog.h and defined in Sddialog.rul (found in the <InstallShield location>\Include folder). To be able to call an Sd dialog function, you must include Sddialog.h before the program block and Sddialog.rul after the program block. For example, a simple script with a call to SdRegisterUser might look like this:

```
   STRING szTitle, szMsg, svName, svCompany;
   #include "Sddialog.h"

program

   szTitle = "User Information";
   szMsg = "Please type your name and the company you work for.";
   SdRegisterUser (szTitle, szMsg, svName, svCompany);

endprogram

   #include "Sddialog.rul"
```

If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Tips for using Sd dialog boxes

Here are some important things to keep in mind when using Sd dialog boxes:

- In order to call an Sd dialog box function, you must include two additional statements in your script.
- If you pass a null string ("") to any string parameter, the Sd dialog functions use the default text supplied by the dialog.
- You can change certain elements in some Sd dialog boxes by calling the DialogSetInfo function.
- All dialog boxes can return any of the standard button values, which means that even if the default dialog box does not contain an Ignore button, you can add the push button to the dialog and the script can handle the push button.
- %P (or an extra space, such as in SdWelcome's "Welcome to the Setup....") is used in static text fields throughout the dialog functions. Call the SdProductName function to replace %P with your product name wherever %P appears in an Sd dialog.
- Sd dialog functions use these constants: RETRY, IGNORE, YES, NO, HELP, and NEXT.
- To speed up compilation of your Setup.rul file and reduce the size of the compiled Setup.ins file, include only the Sd dialog files required by your script. To do this, define the constant SD_SINGLE_DIALOGS:

  ```
  #define SD_SINGLE_DIALOGS
  ```

  Then, to include a needed Sd dialog, define the constant in the format SD_<DIALOGNAME>. For example, to include the SdShowInfoList function, define SD_SHOWINFOLIST:

  ```
  #define SD_SHOWINFOLIST
  ```

  Place these lines after the endprogram statement and before `#include "Sddialog.rul"`.

InstallShield
www.installshield.com

Modifying built-in dialog boxes

In general you can do very little to modify built-in InstallShield dialogs. Many built-in dialogs are created directly by calling Windows APIs. Others have resources stored in _isres.dll, but InstallShield Corporation neither recommends nor supports modifying built-in dialogs. Instead, create a custom dialog box.

However, you can call the SetDialogTitle function to change the default title of the AskOptions, AskText, AskYesNo, EnterDisk, or MessageBox, dialog. Once you change the default title for a dialog box, the new title is in effect for all instances of that type of dialog box until the title is changed to something else, again by calling SetDialogTitle.

For example, call these lines of code to change the title of the AskText dialog box.

```
SetDialogTitle (DLG_ASK_TEXT, "This Title Has Been Changed");
AskText ("Please enter your name.", "John Smith", svResult);
```

InstallShield
www.installshield.com

Welcome the user

Typically, the first dialog that appears in a setup welcomes the user. You can call a built-in or Sd dialog box for this purpose:

Welcome

SdWelcome

InstallShield
www.installshield.com

Register users and confirm registration

Call one of these Sd dialog functions to display a registration dialog box. Default Name and Company values are supplied from the registry (32-bit Windows) or User.dll (16-bit Windows).

SdRegisterUser

SdRegisterUserEx

Call this Sd dialog function to display a dialog box that asks the user to confirm the registration information entered in the above functions:

SdConfirmRegistration

Let the user select a destination folder

Most setups allow the user to select the folder where setup will install the application's main files. Call one of these functions to display a destination folder dialog for the end user and return that path to your setup script:

AskDestPath

AskPath

SdAskDestPath

SelectDir

This Sd dialog function displays a dialog asking the user to confirm the selected destination folder:

SdConfirmNewDir

Let the end user select setup types and components

You can give your end users a great deal of flexibility to choose setup options. Your users can select which setup types or components and subcomponents to install when you display one of the dialog boxes listed below.

Call one of these functions to display a list of setup types available in your file media library and return that information to your script:

SetupType

SdSetupType

SdSetupTypeEx

You must always define at least one setup type for your project in the InstallShield IDE.

It is recommended that you offer your end user a choice of setup types. In order for the component dialogs (below) to correctly display the component choices that you defined in the IDE, there must already be a default setup type selected. If you do not display one of the setup type dialogs (above) to let your end user select the setup type, then you must call the ComponentSetupTypeSet function to set a default setup type *before* displaying a component dialog.

Call one of these functions to display a list of components available (either components defined in your file media library or script-created components) and return that information to your script:

ComponentDialog

SdAskOptions

SdAskOptionsList

SdComponentDialog

SdComponentDialog2

SdComponentDialogAdv

SdComponentMult

Call one of these functions to display options to your end user and have the selections returned to the script. Note that the functions below do not automatically handle file media library selections; you would have to populate the dialogs yourself with setup type and component options and then handle the return values accordingly.

AskOptions

SdOptionsButtons

Call this function to display the component or setup type choices that the end user has made:

SdDisplayTopics

{button ,AL(`Displaying icons in component dialogs;Getting setup type selection from the end user',0,`',`')} See also

InstallShield
www.installshield.com

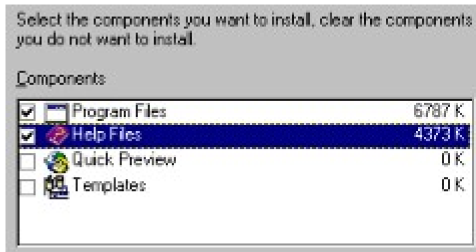
Displaying icons in component dialogs

By following the instructions below you can display icons for component selections in these dialog boxes: ComponentDialog, SdComponentDialog, SdComponentDialog2, SdComponentDialogAdv, and SdComponentMult.

1. Create a "bitmap array." (A bitmap array is a series of 16 by 16 pixel bitmap images. InstallShield automatically displays the 16 by 16 pixel images next to their component choices in left-to-right then top-to-bottom order. Make any desired transparent areas purple—RGB value 255, 0, 255.) The image below is an example bitmap array for component choices:

2. Save the bitmap array as View.bmp.
3. In the Setup Files pane, highlight the folder <project name> Setup Files\Language Independent\Operating System Independent. Its properties sheet opens to the right.
4. Right-click in the properties sheet and select Insert Files. A browse-type dialog opens.
5. Go to the folder where you saved View.bmp and highlight this file.
6. Click OK.
7. Build your setup with the Media Build Wizard so that View.bmp will be available the next time you run your setup.

Suppose you had called the ComponentDialog function in your script, and that the example View.bmp was included in your setup project. Your components would display as in the following image:



{button ,AL(`Let the end user select setup types and components',0,`',`')} See also

InstallShield
www.installshield.com

Let the user select a program folder

These functions automatically display existing folders and subfolders (or program groups under the Program Manager shell) in a dialog box. The user can select either an existing folder, the default name for your application, or type in a new name, and this information is returned to your setup script.

SelectFolder

SdSelectFolder

InstallShield
www.installshield.com

Display file modifications

You might want to inform the end user before you make any changes to system files. You can call this function to display proposed modifications, inform the end user of the consequences of not making the recommended changes, and return the end user's selections:

SdShowFileMods

InstallShield
www.installshield.com

Get text and yes/no input

As you write your setup script, you may want to ask your user for information or ask a yes/no question before continuing. InstallShield provides the following functions for customizing your setup in this manner:

AskText

SdShowDlgEdit1

SdShowDlgEdit2

SdShowDlgEdit3

AskYesNo

{button ,AL(`The InstallScript functions for manipulating strings in text files',0,`',`')} See also

InstallShield
www.installshield.com

Display messages

You can call any of the functions below to display messages during setup. These functions are useful for:

- informing the user of your setup's progress
- displaying the numeric value of error codes
- involving the user in your setup
- displaying proprietary information

MessageBox

SprintfBox

SdBitmap

SdLicense

SdShowInfoList

SdShowMsg

SdShowDlgEdit1

SdStartCopy

{button ,AL(`Using message boxes as debugging tools',0,`',`')} See also

InstallShield
www.installshield.com

Using message boxes as debugging tools

The message box functions MessageBox and SprintfBox are very useful for debugging scripts. (For complete information on debugging a setup, refer to the Visual Debugger Help.

You can use MessageBox to indicate that the script executed as far as a given call to MessageBox. The example below contains several calls to MessageBox, each declaring, in effect, "The script executed this far."

```
   STRING svPath, szMsg, szTitle;

program

   svPath = "C:\\Program Files\\Company\\SampleApp";

   MessageBox ("svPath assigned.", INFORMATION);

   szMsg  = "Setup will install SampleApp in the following folder.\n\n" +
            "To install in this folder, click Next.\n\n" +
            "To install in a different folder, click Browse " +
            "and select another one.\n\n" +
            "Click Cancel to exit Setup.";

   MessageBox ("szMsg assigned.", INFORMATION);

   szTitle = "Choose Destination Location";

   MessageBox ("szTitle assigned.", INFORMATION);

   AskDestPath (szTitle, szMsg, svPath, 0);

   MessageBox ("AskDestPath called.", INFORMATION);

endprogram

   // Source file:  Is5gs002.rul
```

You must be certain that your call to MessageBox is correct, or you will not be able to count on it as a debugging tool. You can ensure that you are using it correctly by copying an instance that is known to work, or by quickly creating and testing a script containing a single MessageBox call.

Be aware that the display of a message box as a marker in the script does not indicate the success or failure of operations up to that point. It only shows that the script executed to that point.

You can also call SprintfBox as a flag to indicate the extent of script execution. SprintfBox is doubly useful because it can readily display the values of variable or constants at different points in the script. The value of a variable can indicate the success or failure of operations in the script.

For example, you can test the return value of a function or the result of an operation. The example below shows how to test the return value from a call to the RegDBSetDefaultRoot function.

```
   LONG lResult;

program

   lResult = RegDBSetDefaultRoot (HKEY_USERS);

   SprintfBox (INFORMATION, "Function Test", "Result: %ld", lResult);

endprogram

   // Source file:  Is5gr003.RUL
```

When using SprintfBox, be careful to use the correct data type for the variable and format specifier.

Prompt for the next distribution disk

Call the EnterDisk function to prompt the user to insert another disk into the drive.

The ComponentMoveData function handles prompting for the next disk automatically—you do not need to call EnterDisk during file transfer.

InstallShield
www.installshield.com

Reboot the system

Call one of these functions to display a dialog offering your user options for finishing the setup:

RebootDialog

System

SdFinish

SdFinishReboot

Displaying extended ASCII characters (Æ, á, or ñ)

InstallShield dialog boxes display some extended characters, but not all. We do not maintain a comprehensive list, so you must test the individual ASCII character by trying to display it in a dialog box.

InstallShield
www.installshield.com

How InstallShield handles an exit condition

There are four ways to exit a setup:

- Pressing the F3 key anytime during setup
- Pressing Alt+F4 or selecting Close from the main installation window's system menu (windowed mode installations only)
- Pressing the Esc key anytime during setup
- Pressing the Cancel button in a dialog box

The results of the various exit-related events depend on which user interface objects are displayed at the time the user chooses to exit:

{button ,JI(`GETRES.HLP>(w95sec)',`Exiting_when_no_dialog_box_is_displayed')} When no dialog box is displayed

{button ,JI(`GETRES.HLP>(w95sec)',`Exiting_when_a_built_in_dialog_box_is_displayed')} When a built-in dialog box is displayed

{button ,JI(`GETRES.HLP>(w95sec)',`Exiting_when_an_Sd_dialog_box_is_displayed')} When an Sd dialog box is displayed

{button ,JI(`GETRES.HLP>(w95sec)',`Exiting_when_a_custom_dialog_box_is_displayed')} When a custom dialog box is displayed

Exiting when no dialog box is displayed

When the user presses the F3 key or Alt+F4 or selects Close from the main installation window's system menu anytime during setup, InstallShield calls the currently defined exit handler. The default exit handler asks the user whether to exit setup or continue. You cannot modify the default exit handler.

If you have defined a customer exit handler with the Handler function, InstallShield calls it. Otherwise, InstallShield calls the default exit handler.

Exiting when a built-in dialog box is displayed

If the user presses the F3 key or the Cancel button while a built-in dialog box is being displayed, InstallShield calls the currently defined exit handler.

Since the Cancel button causes InstallShield to call the currently defined exit handler, the setup script never receives CANCEL as a return value from a built-in dialog box function call.

The Cancel button in a built-in dialog cannot be disabled during without modifying the actual dialog resource stored in _isres.dll. InstallShield Corporation does not recommend modifying _isres.dll for any reason.

Exiting when an Sd dialog box is displayed

If the user presses the Esc key while an Sd dialog is being displayed, the WaitOnDialog function call in the Sd dialog's script file (found in the <InstallShield location>\Include folder) returns an ID of 2, which corresponds to the SD_PBUT_CANCEL constant defined in Src.h. The Sd dialog script responds by calling the Do function to carry out the exit handler.

If the user presses the Cancel button in an Sd dialog, the WaitOnDialog function call in the Sd dialog's script file returns an ID of 9, which corresponds to the SD_PBUT_EXITSETUP constant. The Sd dialog script responds by calling the Do function to carry out the exit handler.

This functionality can be modified by modifying the Sd dialog scripts. However, for compatibility with built-in dialog boxes, it is recommended that you instead call the Handler function with the EXIT option to define a custom exit handler, which will be called when the Cancel button is pressed.

InstallShield
www.installshield.com

Exiting when a custom dialog box is displayed

If the user presses the Esc key while a custom dialog box is being displayed, the WaitOnDialog function returns a value of 2. The custom dialog box should handle this value by calling the Do function with the EXIT option to carry out the exit handler.

If the user presses the Cancel button while a custom dialog box is being displayed, the WaitOnDialog function returns the control ID for the Cancel button of the custom dialog. You should therefore set the ID for the Cancel button in your custom dialog to 9 to ensure compatibility with built-in and Sd dialog boxes. Design your custom dialog box to respond to this return value by calling Do(EXIT) to carry out the exit handler.

The following constants are defined in Sddialog.h and can be used to test for the return values listed above:

Constant	Value
SD_PBUT_EXITSETUP	9
SD_PBUT_CANCEL	2

The CANCEL constant is also defined as 2, but it should not be used by any external dialogs.

Here is a fragment that shows how a custom dialog box can handle these return values:

```
// nId is returned from WaitOnDialog
switch(nId)

   case SD_PBUT_CANCEL: // generated from Esc key
      Do( EXIT );

   case SD_PBUT_EXITSETUP:
      Do( EXIT );

   // Handle other cases here.
```

InstallShield
www.installshield.com

Caching dialogs

Dialogs are cached by default, giving the display of dialogs a "wizard-like" effect, but caching dialogs can cause an undesirable effect. When you exit an InstallShield dialog box using the Next button, the dialog may not immediately disappear if it was larger than the current dialog. Instead, the next dialog to be displayed appears over the first.

To disable dialog caching, simply call the Disable function with the option. Once this is done, each dialog disappears immediately when the Next button is pressed.

Dialog caching is only a problem for dialogs that are not the same size. As a rule of thumb, all InstallShield dialog boxes with a Back button are the same size.

If you don't want your custom dialog to be cached, simply create them from the InstallShield dialog template so that they are the same size as InstallShield dialog boxes, and there should be no problems.

Which fonts InstallShield dialog boxes use

All InstallShield dialog box fonts use a point size of 8. The font used by dialog boxes in InstallShield is determined as follows:

Dialog boxes created internally with the Windows MessageBox API

The MessageBox, SprintfBox, and AskYesNo functions all call the Windows API MessageBox to create an appropriate dialog box. The font used for the text in these dialog boxes is system dependent, and is set in the Control Panel under Message Box Font in the Appearance folder in the Display Properties dialog box. Most systems use MS Sans Serif, the default font.

All other built-in dialog boxes

The default exit handler, Read-Only-File Warning, and Not Enough Disk Space warning dialog boxes use Helv. All other built-in dialog boxes use MS Sans Serif.

Sd dialog boxes

The SdShowFileMods, SdShowDlgEdit1, SdShowDlgEdit2, SdShowDlgEdit3, SdAskOptions, and SdComponentDialog dialog boxes use Helv. All other Sd dialog boxes use MS Sans Serif.

Font information is stored in the dialog box definitions in <InstallShield location>\Redistributable\Compressed Files\<target language>\<target platform>_isres.dll. To change these fonts, you should copy the dialog definitions to _isuser.dll and modify them there; never modify _isres.dll directly. For more information, see Creating custom dialog boxes from Sd dialog boxes.

If the specified font is not available...

Any dialog box that attempts to use a font that is not available on the user's system will be displayed with the system font instead. Therefore, you should always use fonts which either you have installed or are common to most Windows setups, such as MS Sans Serif or Helv.

The system font

The system font is system dependent and cannot be changed by the user. However, the size can be changed in the standard Control Panel dialog for screen appearance.

To ensure uniformity with most users' systems, your custom dialog boxes should use the Ms Sans Serif font with a point size of 8.

Modifying Sd dialog box source files

(InstallShield Corporation does not recommend modifying Sd dialog functionality.)

The Sd dialogs are InstallScript-based functions. The source code for individual Sd dialog boxes are included by the Sddialog.h and Sddialog.rul script files. All Sd dialog script files can be found in the <InstallShield location>\Include folder.

Follow these steps to modify the Sd dialog script files:

1. First, back up all the files in the Include folder to a convenient location. The back-up files will serve as a safeguard if you need to refer to the original script files.
2. Determine which script file defines the Sd dialog you wish to modify. Click the Dialog Sampler icon in the InstallShield program folder, then click the Sd dialog in question. The Dialog Sampler will display the name of that script file.

3. Open the appropriate Sd dialog script file.
4. Make any desired modifications to the script.
5. Save the modified Sd dialog script file. Remember that the InstallShield compiler searches for include files in the Include folder. Therefore, any modified scripts must be saved into this folder.
6. Recompile your setup script. The new Sd dialog files will be included automatically.

Be careful when modifying these scripts. It is acceptable to add or delete (comment out) certain lines in the script, but try not to change the general logic of the script, otherwise the Sd dialog may not function correctly when called. Also, be aware that some Sd dialog scripts include undocumented functions used to control internal handling of Sd dialogs. These functions should never be removed, or unexpected results may occur.

Here are some further things to be aware of when making modifications to Sd dialog scripts:

{button ,JI(`GETRES.HLP>(w95sec)',`The_DLG_INIT_routine')} The DLG_INIT routine

{button ,JI(`GETRES.HLP>(w95sec)',`The_function_header_block')} The function header block

{button ,JI(`GETRES.HLP>(w95sec)',`The_name_of_the_Sd_dialog_box')} The name of the Sd dialog box

{button ,JI(`GETRES.HLP>(w95sec)',`The_handle_of_the_Sd_dialog_box')} The handle of the Sd dialog box

---

{button ,AL(`Creating custom dialog boxes from Sd dialog boxes;Making advanced modifications to Sd dialog boxes',0,`',`')} See also

The DLG_INIT routine

All Sd dialogs have a DLG_INIT routine. This routine is executed when the Sd dialog is first defined. This routine can be found under the case DLG_INIT: statement located under the call to EzDefineDialog. Typically, all initialization of the dialog is carried out in this routine, so many Sd dialog modifications need to take place here.

The function header block

The header block of an Sd dialog function (or any function) is the area in the function where variables are declared. This section begins after the function declaration but before the begin keyword. This function header block is similar to the declare block in the main setup script, except that additional function prototypes cannot be declared in this header block

The name of the Sd dialog box

The name of an Sd dialog box is defined in the script function for the Sd dialog. You can use either one of these methods to get this information.

- Start the InstallShield Dialog Sampler and single click an Sd dialog box function name in the "Dialog functions" list. A dialog ID will be displayed in the Dialog ID field below the Dialog functions list. For example, the dialog ID displayed for the SdAskDestPath dialog box is SD_NDLG_ASKDESTPATH (12005). Use the SD_N... expression (without the N) as your dialog ID. Thus, the SdAskDestPath dialog box ID you would pass to CmdGetHwndDlg would be SD_DLG_ASKDESTPATH. (The Sd dialog box whose handle you are getting must be included in your script.)
- Open the .rul file for the Sd dialog box whose handle you wish to get. The Sd dialog box .rul files are located in the Include folder. Look just below the begin keyword in the beginning of the Sd dialog's function definition for the value assigned to the szDlg variable (or whatever variable is passed to WaitOnDialog later in the function definition). The value assigned to the variable is the Sd dialog box ID you can pass to CmdGetHwndDlg.

The handle of the Sd dialog box

The handle of an Sd Dialog box is available in an Sd dialog script any time after the DLG_INIT routine. It is obtained in the DLG_INIT from the CmdGetHwndDlg function and is typically stored in the hwndDlg variable. If you need to pass the handle of an Sd dialog to another function, you can use the hwndDlg variable to do so without having to call CmdGetHwndDlg a second time.

InstallShield
www.installshield.com

Modifying Sd dialog boxes with DialogSetInfo

InstallShield
www.installshield.com

Call the DialogSetInfo function to modify these attributes of certain Sd dialog boxes:

- Check box style
- File size display units (KB or MB)
- File size display precision (250 KB vs. 250.2 KB)
- Dialog image (the bitmap displayed)

Check box style

```
DialogSetInfo ( DLG_INFO_CHECKSELECTION, "",  nCheckBoxStyle );
```

The following constants are available for the nCheckBoxStyle parameter:

CHECKMARK	for check mark (default)
CHECKBOX	for Windows check box
CHECKLINE	for highlighted line (standard listbox)
CHECKBOX95	for Windows 95 check box

File size display units in KB or MB

```
DialogSetInfo( DLG_INFO_KUNITS, "", bSizeStyle );
```

The following values are available for the bSizeStyle parameter:

TRUE	for displaying size in kilobytes
FALSE	for displaying size in megabytes

File size display precision (250 KB vs. 250.2 KB)

```
DialogSetInfo( DLG_INFO_USEDECIMAL, "", bUseDecimal );
```

The following values are available for the bUseDecimal parameter:

TRUE	to display size with a decimal
FALSE	to display size without a decimal

Dialog image

```
DialogSetInfo( DLG_INFO_ALTIMAGE, szImage, nSet );
```

szImage specifies the bitmap path and file name when setting the alternate bitmap. For example, if Altbmp.bmp were the alternate bitmap stored in the Setup Files pane, use the following syntax:

```
szImage = SUPPORTDIR ^ "Altbmp.bmp";
```

When resetting the bitmap to the default, szImage should be a NULL string ("").

The following values are available for the nSet parameter:

TRUE	when setting the bitmap
-1	when resetting the bitmap to the default one

{button ,AL(`Displaying an alternate bitmap in Sd dialog boxes',0,`',`')} See also

InstallShield
www.installshield.com

Displaying an alternate bitmap in Sd dialog boxes

InstallShield
www.installshield.com

To change the bitmap that is displayed in Sd dialog boxes, call the DialogSetInfo function with the DLG_INFO_ALTIMAGE option to specify an alternate bitmap to be displayed in Sd dialogs. All Sd dialog boxes that appear after the call to DialogSetInfo display the new bitmap.

The example below uses the result from calling the GetSystemInfo function to conditionally display a bitmap in the SdFinish dialog box. If the target system supports 256 or more colors, ImgHiRes.bmp, a 256-color bitmap, displays; if the target system supports less than 256 colors, ImgLoRes.bmp, a 16-color bitmap, displays. Both bitmaps are stored in the Setup Files pane and are therefore accessible through the SUPPORTDIR path.

```
GetSystemInfo (COLORS, nvResult, svResult);

if nvResult >= 256 then
   DialogSetInfo (DLG_INFO_ALTIMAGE,
                  SUPPORTDIR ^ "ImgHiRes.bmp", TRUE);
else
   DialogSetInfo (DLG_INFO_ALTIMAGE,
                  SUPPORTDIR ^ "ImgLoRes.bmp", TRUE);
endif;

SdFinish (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);
```

By default, InstallShield places all setup project files in the following folder:

<Windows drive>\My Installations\<project name>

You can change the default location, if desired. On the Tools menu select Options. Open the Project Location tab in the Options dialog and type in or browse to the name of a new default folder.

InstallShield always copies the disk images of the completed setup to the Media\<media name>\Disk Images folder under the project files folder. You can also change the default location for your disk images. For more information, see Specify a location for my media files.

Overview: Custom Dialog Boxes

If InstallShield does not provide a dialog box that meets your needs, then you must create your own custom dialog box using a resource editor. InstallShield provides you with a dialog template to make this process even easier. (Of course, advanced Windows programmers can create dialog boxes manually, without the aid of a visual resource editor.)

You then store the dialog in a DLL (placed in the Setup Files pane) and use the InstallScript language to handle the dialog.

Alternatively, you can customize an Sd dialog.

This section also includes information on troubleshooting custom dialog box problems.

You must have 16-bit resources for 16-bit setups, and 32-bit resources for 32-bit setups. If you use _isuser.dll or another DLL to store your custom dialog box resources, you must create a 16-bit version for use with 16-bit setups, and a 32-bit version for use with 32-bit setups. If you're targeting both 16- and 32-bit operating systems in a cross-platform setup, then you must place the appropriate version of the DLL in the correct target platform folder in the Setup Files pane.

Create a custom dialog box from a template

The easiest way for you to create a custom dialog box is to modify the InstallShield dialog template using a resource editor.

In these four steps we'll be taking you through a tutorial example using Microsoft Visual C++ 4 to create a 32-bit DLL containing a custom dialog box:

{button 1,JI(`GETRES.HLP>(w95sec)',`Make_a_copy_of_the_dialog_template')} Make a copy of the dialog template.

{button 2,JI(`GETRES.HLP>(w95sec)',`Give_the_new_dialog_an_ID')} Give the new dialog an ID.

{button 3,JI(`GETRES.HLP>(w95sec)',`Create_control_IDs')} Create IDs for the controls.

{button 4,JI(`GETRES.HLP>(w95sec)',`Populate_the_dialog_with_controls')} Populate the dialog with controls and build.

To create a 16-bit custom DLL, follow a similar procedure using the files in <InstallShield location>\Custom Dialog\16bit rather than <InstallShield location>\Custom Dialog\VC++ 4 Project.

InstallShield
www.installshield.com

Make a copy of the dialog template

InstallShield
www.installshield.com

The dialog template project files are located in _isuser.dll The 16-bit version of the _isuser.dll project is located in <InstallShield location>\Custom Dialog\16bit, and the 32-bit version is located in <InstallShield location>\Custom Dialog\VC++ 4 Project. You must make a copy of the dialog template inside _isuser.rc. You can then modify the copy of the dialog template to create your own dialog.

To make a copy of the dialog template, follow these steps:

1. Copy the VC++ 4 Project folder and all its contents and subfolders to a working folder. This will keep your InstallShield folders "clean" and ready for a complete uninstallation should you ever want to uninstall InstallShield, and will preserve the original files for future custom dialog projects.
2. Start Microsoft Visual C++ Developer Studio. From the File menu, select Open Workspace. Open the _isuser.mdp file in your working VC++ 4 Project folder. Set the build type to Win32 Release in the Select Default Project Configuration list box on the toolbar. You can compile in Win32 Debug if you wish, but this example uses Win32 Release. If the Project Workspace is not in view, select it from the View menu.
3. In the Project Workspace, click the ResourceView tab.
4. Double-click the “_isuser resources” folder.
5. Double-click the Dialog folder when it appears.
6. Select the DLG_TEMPLATE icon.
7. Select Copy from the Edit menu.
8. Select Paste from the Edit menu. Developer Studio adds a copy of DLG_TEMPLATE named DLG_TEMPLATE1 to the Dialog folder. This is your working copy of the template.

Give the new dialog an ID

Once you have made a copy of the dialog template, you must give it a unique ID. To give your dialog an ID, follow these steps:

1. Open the DLG_TEMPLATE1 dialog resource by double-clicking the icon.
2. Double-click in the title bar of the dialog template to open its properties sheet.
3. For this example, enter the number 30001 in the ID: field. Enter Modem Installation in the Caption: field. Press Enter to close the properties sheet.

You can enter either a string or a number in the dialog's ID field. The DefineDialog and EzDefineDialog functions, which you will use in your script to make the dialog available, allow you to specify either a string or a number as a dialog's ID. The Windows programming standard is to use a number; using a number such as 30xxx will keep your custom dialog easily distinguishable from InstallShield's built-in dialogs (which have 10xxx IDs) and Sd dialogs (12xxx).

Create control IDs

Before you populate your custom dialog with controls, you must identify the controls and their ID numbers. You use the Microsoft Visual C++ Symbol Browser to do this. When you compile the project, Microsoft Visual C++ will create #define statements for the control names and ID numbers in a file named Resource.h in your working directory. Here is an example of creating control names and IDs:

1. Open the Symbol Browser by selecting Resource Symbols from the View menu.
2. Click the New button. The New Symbol dialog opens.
3. Enter ID_PULSE in the Name: field, and 100 in the value field. Click OK.
4. Repeat the above two steps to create these control name and value pairs:

   ID_TONE 110
   ID_DIAL9 220
   ID_COMPORT 330

   Close the Symbol Browser when you are done.

InstallShield
www.installshield.com

Populate the dialog with controls

InstallShield
www.installshield.com

Once you create a copy of the dialog template and enter your dialog control IDs in the Symbol Browser, you must populate your dialog with controls and enter control ID information into the controls' properties sheets.

1. Populate the dialog with controls (as shown below) that match the control names and values you entered in step 3, Create control IDs. The controls in this example are a list box, two radio buttons (Pulse and Tone), and a check box (Dial "9" First).
2. Double-click each control to open its properties sheet.
3. Select the ID for each control from the ID drop-down list box on the General tab of the properties sheet. Recall that the IDs were established in step 3, Create control IDs, using the Symbols Browser.
4. After selecting the ID for each control, close the dialog editor window.
5. From the Build menu, select Rebuild All. The project will rebuild itself with the new dialog you just created. _isuser.dll, containing your new custom dialog, will be created in the working VC++ 4 Project\Release folder.
6. When the project rebuild is complete, close Microsoft Developer Studio.
7. From a DOS prompt or Windows Explorer, check the date and time information for VC++ 4 Project\Release_isuser.dll to verify that it is the one you just built.

After you verify that your new custom dialog is correctly constructed, you are ready to use the dialog in your setup. (Click here to see a script that displays the example dialog box and processes its controls.) After you have tested the dialog, you can remove the template dialog from the resource file and do a final rebuild of the project.

Place the modified _isuser.dll into the Setup Files pane under the appropriate target language(s) and operating system(s). InstallShield will include _isuser.dll in your setup the next time you build it with the Media Build Wizard.

The InstallShield dialog template is a Windows 95-style dialog containing a bitmap image and Back, Next, and Cancel buttons. This dialog template design helps your setup maintain the Explorer shell look and feel, lending it consistency and ease of use.

You can find the InstallShield dialog template in <InstallShield location>\Custom Dialog\VC++ 4 Project_isuser.rc.

The characters @10550,10551;1;1;0,128,128;0 in the upper left-hand corner of the dialog template are part of a special format string in a static text field. The format string instructs InstallShield to place an image in the static text field, which helps give the dialog the three-dimensional Explorer shell look.

{button ,} See also

Use the dialog box in a script

After you create the custom dialog box, you must customize your setup script to handle the dialog box. Whenever you use a custom dialog box, you must register the dialog box with InstallShield, display the dialog box, process controls, and end the dialog box.

This section continues taking you through the example that began when you modified a copy of the InstallShield dialog template to create a custom dialog box resource in How do I create a custom dialog box from a template.

First, take a few minutes to read through the example script which demonstrates displaying this custom dialog box.

Next, read the sections that explain this script step-by-step:

{button ,JI(`GETRES.HLP>procedur',`Include_the_header_file_and_define_variables')} Include the header file and define variables.

{button ,JI(`GETRES.HLP>procedur',`Create_a_list_to_display_in_the_list_box')} Create a list to display in the list box.

{button ,JI(`GETRES.HLP>procedur',`Register_the_dialog_in_the_script')} Register the dialog in the script.

{button ,JI(`GETRES.HLP>procedur',`Display_the_dialog')} Display the dialog.

{button ,JI(`GETRES.HLP>procedur',`Process_the_Next_button')} Process the Next button.

{button ,JI(`GETRES.HLP>procedur',`Process_the_Cancel_button')} Process the Cancel button.

{button ,JI(`GETRES.HLP>procedur',`Process_the_close_dialog_box_button')} Process the close dialog box button.

{button ,JI(`GETRES.HLP>procedur',`Process_radio_buttons_in_the_group_box')} Process radio buttons in the group box.

{button ,JI(`GETRES.HLP>procedur',`Process_dialog_box_errors')} Process dialog box errors.

{button ,JI(`GETRES.HLP>procedur',`Identify_the_end_of_dialog_box_processing')} Identify the end of dialog box processing.

{button ,JI(`GETRES.HLP>procedur',`Free_the_dialog_box_and_list_from_memory')} Free the dialog box and list from memory.

Custom dialog box example script

```
   // Include the header file and define variables.
   #include "Resource.h"
   #define SD_PBUT_BACK          12
   #define SD_PBUT_CONTINUE       1
   #define SD_PBUT_EXITSETUP      9

   BOOL    bDone;
   STRING  svComPort;
   NUMBER  nCmdValue, nPulseState, nToneState, nDial9State;
   LIST    listID;

program

   // Create a list to display in the list box.
   listID = ListCreate(STRINGLIST);
   ListAddString(listID, "COMM1:", AFTER);
   ListAddString(listID, "COMM2:", AFTER);
   ListAddString(listID, "COMM3:", AFTER);
   ListAddString(listID, "COMM4:", AFTER);

   // Register the dialog in the script.
   EzDefineDialog( "MYCOMDIALOG", ISUSER, "", 30001);

   // Display the dialog.
   bDone = FALSE;
   while (bDone=FALSE)
      nCmdValue = WaitOnDialog("MYCOMDIALOG");

      switch (nCmdValue)

         case DLG_INIT:
            CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            CtrlSetList("MYCOMDIALOG", ID_COMPORT, listID);
            CtrlSetState("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);

         // Process the Next button.
         case SD_PBUT_CONTINUE:
            CtrlGetCurSel("MYCOMDIALOG", ID_COMPORT, svComPort);
            nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
            nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);
            nDial9State = CtrlGetState("MYCOMDIALOG", ID_DIAL9);
            bDone = TRUE;

         // Process the Cancel button.
         case SD_PBUT_EXITSETUP:
            bDone = TRUE;

         // Process the close dialog box button.
         case DLG_CLOSE:
            bDone = TRUE;

         // Process radio buttons in the group box.
         case ID_PULSE:
            nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);

            if (nPulseState = BUTTON_CHECKED) then
               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
               CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
```

```
            else
               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
               CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
            endif;

         case ID_TONE:
            nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);

            if (nToneState = BUTTON_CHECKED) then
               CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            else
               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
               CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
            endif;

         // Process dialog box errors.
         case DLG_ERR:
            MessageBox("Internal dialog box error", SEVERE);
            bDone = TRUE;

      endswitch;

   endwhile;

   // Identify the end of dialog box processing.
   EndDialog("MYCOMDIALOG");

   // Free the dialog box and list from memory.
   ReleaseDialog("MYCOMDIALOG");
   ListDestroy(listID);

endprogram

   // Source file:  Is5gr004.RUL
```

InstallShield
www.installshield.com

Define control IDs and variables

InstallShield
www.installshield.com

The custom dialog example script begins by defining the necessary control IDs. Then, the variables are declared before the program block:

```
#define ID_PULSE              100
#define ID_TONE               110
#define ID_DIAL9              220
#define ID_COMPORT            330

#define SD_PBUT_BACK           12
#define SD_PBUT_CONTINUE        1
#define SD_PBUT_EXITSETUP       9

BOOL    bDone;
STRING  svComPort;
NUMBER  nCmdValue, nPulseState, nToneState, nDial9State;
LIST    listID;
```

```
program
```

The #define statements provide the control IDs for the Back, Next, and Cancel buttons and the controls in the dialog box. You can cut and paste the define statements for the dialog box control IDs from the Resource.h file generated by your resource editor.

InstallShield
www.installshield.com

Create a list to display in the list box

InstallShield
www.installshield.com

To create a list and place it into the Comm Ports list box, enter the following lines of code in your script before calling the EzDefineDialog function:

```
listID = ListCreate(STRINGLIST);
ListAddString(listID, "COMM1", AFTER);
ListAddString(listID, "COMM2", AFTER);
ListAddString(listID, "COMM3", AFTER);
ListAddString(listID, "COMM4", AFTER);
```

The ListCreate function call instructs InstallShield to create a string list and assigns the ID given to the list to the listID variable. Then the ListAddString function calls add the lines "COMM1", "COMM2", and so on to the list. The AFTER constant instructs the program to place each new element after the previous element in the list.

InstallShield
www.installshield.com

Register the dialog in the script

InstallShield
www.installshield.com

Call the EzDefineDialog function to register the dialog box resource with InstallShield. The EzDefineDialog function does not create or display the dialog box, it only defines it in the script.

For example, the following statement from the custom dialog box example script defines the dialog box you created in the InstallShield help example Create a custom dialog box from a template:

```
EzDefineDialog( "MYCOMDIALOG", ISUSER, "", 30001);
```

MYCOMDIALOG is the unique name used to register the dialog box with InstallShield. The ISUSER system variable contains the fully qualified path and file name of the _isuser.dll file loaded for the current instance of InstallShield. 30001 is the ID of the dialog box as it resides in the DLL. The "" in the third parameter indicates the resource ID is not a string (that is, it is a number). If the ID of the dialog box is a string, enter 0 (zero) in the final parameter, and the string ID in the third parameter.

InstallShield also provides the DefineDialog function. The DefineDialog function may be used to define or register the dialog box to InstallShield. The DefineDialog function has eight parameters instead of four, including parameters for parent window handle and owner window handle. Under most circumstances, it is easier to use the EzDefineDialog function.

InstallShield
www.installshield.com

Display the dialog

InstallShield
www.installshield.com

Call the WaitOnDialog function to display the dialog box. The WaitOnDialog function displays the dialog box you defined with the EzDefineDialog or DefineDialog function.

While the EzDefineDialog and DefineDialog functions do register the dialog box with InstallShield, they do not display the resource during setup. Call the WaitOnDialog function in a while loop as shown below to format and display the dialog box and to wait for messages and receive information from the dialog box:

```
bDone = FALSE;
while (bDone=FALSE)
   nCmdValue = WaitOnDialog("MYCOMDIALOG");
```

You assign the return value from the WaitOnDialog function to nCmdValue for use in the switch statement.

The first test in the switch statement is for the DLG_INIT message, which is returned when the dialog first opens and initializes. After receiving the DLG_INIT message, you can fill combo boxes, list boxes, and edit fields with desired text and set default check boxes or radio buttons. This section of code from the custom dialog example script sets initial states for several controls in the custom dialog box:

```
      switch (nCmdValue)

         case DLG_INIT:
            CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
            CtrlSetList("MYCOMDIALOG", ID_COMPORT, listID);
            CtrlSetState("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
```

The first CtrlSetState call sets the initial state of the Tone radio button to checked. The second CtrlSetState call loads the list box with the contents of the list defined by listID. And the third CtrlSetState call sets the initial state of the Dial "9" First check box to checked.

InstallShield
www.installshield.com

Process the Next button

InstallShield
www.installshield.com

The case statement after DLG_INIT in the custom dialog example script handles the Next button:

```
case SD_PBUT_CONTINUE:
   CtrlGetCurSel("MYCOMDIALOG", ID_COMPORT, svComPort);
   nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
   nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);
   nDial9State = CtrlGetState("MYCOMDIALOG", ID_DIAL9);
   bDone = TRUE;
```

The first CtrlGetCurSel call instructs InstallShield to see which item in the list box you have selected and stores the selection in the svComPort variable. The next three lines instruct InstallShield to retrieve the state of the Pulse radio button, the Tone radio button, and the Dial "9" First check box and store these states in the nPulseState, nToneState, and nDial9State variables, respectively. The final line of this section sets the bDone flag to TRUE to end switch processing and perform the next instruction in the script.

InstallShield
www.installshield.com

Process the Cancel button

InstallShield
www.installshield.com

The following case statement, just after the case SD_PBUT_CONTINUE statement in the custom dialog example script, handles the Cancel button:

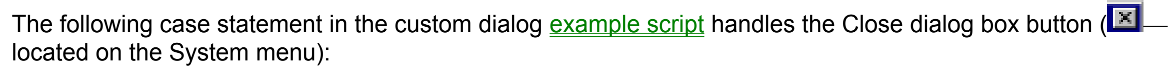
```
case SD_PBUT_EXITSETUP:
   bDone = TRUE;
```

The code in this section sets the bDone flag to TRUE to end switch processing and perform the next instruction in the script.

InstallShield
www.installshield.com

Process the close dialog box button

InstallShield
www.installshield.com

The following case statement in the custom dialog example script handles the Close dialog box button (— located on the System menu):

```
case DLG_CLOSE:
   bDone = TRUE;
```

The code in this section also sets the bDone flag to TRUE to end switch processing and perform the next instruction in the script.

InstallShield
www.installshield.com

Process radio buttons in the group box

InstallShield
www.installshield.com

The following case statement from the custom dialog example script is required to handle the Tone and Pulse radio buttons because you did not specify Auto radio buttons when you defined the radio buttons in the example dialog. Auto radio buttons in the same group automatically enforce exclusive selection. If you had specified Auto radio buttons, Windows would enforce exclusive selection and you would not need to use the section of script shown below:

```
case ID_PULSE:
   nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);

   if (nPulseState = BUTTON_CHECKED) then
      CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
      CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
   else
      CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
      CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
   endif;

case ID_TONE:
   nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);

   if (nToneState = BUTTON_CHECKED) then
      CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
      CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
   else
      CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
      CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
   endif;
```

The example above instructs InstallShield to retrieve the state of the Pulse radio button and store the state in the nPulseState variable. If nPulseState contains BUTTON_CHECKED, then the script instructs InstallShield to set the Tone button to unchecked and the Pulse button to checked. The code manually sets the state of the radio buttons to be mutually exclusive—if one radio button is checked, the other is unchecked.

The else option sets the Tone button to checked and the Pulse button to unchecked. The same logic handles the ID_TONE message.

Process dialog box errors

This case statement handles dialog box errors in the custom dialog example script:

```
         case DLG_ERR:
            MessageBox("Internal dialog box error", SEVERE);
            bDone = TRUE;

      endswitch;

   endwhile;
```

InstallShield uses the portion of script shown above to display a message box if it detects a dialog box error. After informing you of the error, the example sets the bDone flag to TRUE to end switch processing and perform the next instruction in the script. The endswitch and endwhile statements close the while loop and switch statements.

Identify the end of dialog box processing

After handling all possible conditions in the dialog box, the custom dialog example script ends dialog box functionality by calling the EndDialog function as shown below:

```
EndDialog("MYCOMDIALOG");
```

This section informs InstallShield that processing for the MYCOMDIALOG dialog box is complete.

InstallShield
www.installshield.com

Free the dialog box and list from memory

InstallShield
www.installshield.com

After you are finished using the dialog box and list, remove them from memory to conserve system resources. The custom dialog example script uses the ReleaseDialog and ListDestroy functions to release the dialog as shown below:

```
ReleaseDialog("MYCOMDIALOG");
```

InstallShield
www.installshield.com

Handling messages from custom dialog boxes and controls

InstallShield
www.installshield.com

You will need to handle the following messages that are returned from custom dialog boxes and controls. The easiest way is to create case statements of each of the messages and include appropriate code to handle the condition.

InstallShield also provides you with a number of functions that you can use to send information to and retrieve information from custom dialog boxes. For more information, see The InstallScript functions for processing controls .

Custom dialog box messages

These messages are mapped to Windows messages, so that you can access them in your script using these InstallScript constants:

DLG_INIT

The DLG_INIT message maps to the Windows WM_INITDIALOG message. The DLG_INIT message is the first message sent to the WaitOnDialog function when the dialog box is first created, before the dialog box is displayed. After receiving the DLG_INIT message, you now can fill combo boxes, list boxes, and edit fields with desired text, as well as set default check boxes or radio buttons.

DLG_CLOSE

The DLG_CLOSE message maps to the Windows WM_CLOSE message. It is sent to the WaitOnDialog function when the end user closes the dialog box by choosing Close from the system menu. You may want to handle the close condition in the following way:

```
case DLG_CLOSE:
   bDone = TRUE;
```

DLG_ERR

The DLG_ERR message is sent to the WaitOnDialog function whenever an error occurs concerning the dialog box you are displaying. You may want to handle the error condition in the following way:

```
      case DLG_ERR:
          MessageBox ("An error has occurred in setup",
                      SEVERE);
          bDone = TRUE;
      endswitch;
   endwhile;
```

Messages for controls

Dialog Control IDs map to the Windows WM_COMMAND message, as well as to any controls you have included in the dialog box. For example, if the dialog box contains a list box, a Next button, a Cancel button, and a single-line edit field, include the following case statements (with supporting code) in the script:

```
case DLG_ERR:
case SD_PBUT_CONTINUE:
case SD_PBUT_EXITSETUP:
case LISTBOXID:
case SINGLE_EDIT_FIELD:
```

Remember to define each of the constants in the script. For example, if the single line edit field has a resource ID of 250, include the following line in the script:

```
#define SINGLE_EDIT_FIELD  250
```

InstallShield
www.installshield.com

The InstallScript functions for processing controls

InstallShield
www.installshield.com

Call these InstallScript functions to send information to and retrieve information from custom dialog boxes, according to the type of control you want to process:

Push buttons, radio buttons, and check boxes

CtrlGetState
Retrieves the state of a radio button, check box, or push button control in a dialog box.

CtrlSetState
Sets the current state of a check box, radio button, or push button control in a dialog box.

Combo boxes

CtrlClear
Deletes the contents of an edit, static, list box, or combo box control.

CtrlGetCurSel
Returns the currently selected item from a list box or combo box.

CtrlPGroups
Retrieves a list of all of the program group names available on the target system.

CtrlSetList
Places the contents of a list into the specified list box or combo box.

CtrlDir
Fills a list box or combo box with either a folder listing or a file listing.

CtrlGetText
Retrieves the text from an edit or static field. This function can also be used to retrieve the contents of an edit box of a combo box.

CtrlSetCurSel
Finds and sets the current selection in a list box or combo box.

CtrlSetText
Sets the text in an edit field, static text field, or the edit field of a combo box.

Edit boxes

CtrlClear
Deletes the contents of an edit, static, list box, or combo box control.

CtrlGetText
Retrieves the text from an edit or static field. This function can also be used to retrieve the contents of an edit box of a combo box.

CtrlSetFont
Allows you to specify a font for any control in the dialog box.

CtrlSetMLEText
Sets the text in a multi-line edit field.

CtrlGetMLEText
Retrieves the text in a multi-line edit or static field.

CtrlSelectText
Selects the text displayed in an edit field.

CtrlSetText
Sets the text in an edit field, static text field, or the edit field of a combo box.

ListAddString
  Adds a string to the list.

List boxes

CtrlClear
  Deletes the contents of an edit, static, list box, or combo box control.

CtrlGetCurSel
  Returns the currently selected item from a list box or combo box.

CtrlSetCurSel
  Finds and sets the current selection in a list box or combo box.

CtrlSetMultCurSel
  Sets the current selection in a multi-selection list box.

CtrlDir
  Fills a list box or combo box with either a folder listing or a file listing.

CtrlPGroups
  Retrieves a list of all of the program group names available on the target system.

CtrlSetList
  Places the contents of a list into the specified list box or combo box.

Advanced control processing

CmdGetHwndDlg
  Retrieves the handle of the dialog.

CtrlGetSubCommand
  Retrieves the operation performed on the control after a WaitOnDialog function call.

GetFont
  Retrieves the handle of the font.

HIWORD
  Retrieves the high-order word from a 32-bit integer.

LOWORD
  Retrieves the low-order word from a 32-bit integer.

InstallShield
www.installshield.com

Get the current state of a button

InstallShield
www.installshield.com

Call the CtrlGetState function to retrieve the current state of a check box or radio control.

The CtrlGetState function returns the following values:

- BUTTON_CHECKED indicates the button's state is checked
- BUTTON_UNCHECKED indicates the button's state is unchecked
- DLG_ERR indicates the CtrlGetState function was unable to retrieve the state of the specified control.

InstallShield
www.installshield.com

Set the current state of a button

InstallShield
www.installshield.com

Call the CtrlSetState function to set the current state of a check box or radio control.

Call the CtrlSetState function in the `case DLG_INIT:` portion of the dialog box routine.

InstallShield
www.installshield.com

Clear text from a combo box

InstallShield
www.installshield.com

Call the CtrlClear function to remove all the text from a combo box control.

InstallShield
www.installshield.com

Fill a combo box control

InstallShield
www.installshield.com

Call the CtrlDir function to fill a combo box control with either a folder or file listing.

Call the CtrlPGroups function to place a list of existing program groups in a combo box control. It displays all the program groups that are displayed in the Program Manager.

InstallShield
www.installshield.com

Get text from a combo box

InstallShield
www.installshield.com

Call the CtrlGetCurSel function to retrieve the currently selected items in a combo box control.

Call the CtrlGetText function to retrieve the contents of a combo box control.

Select text in a combo box

InstallShield
www.installshield.com

Call the CtrlSelectText function to select all the text from a single- or multi-line edit box control. (The CtrlPGroups function is useful for highlighting default text in a single-line edit box.) Call CtrlSelectText in the `case DLG_INIT:` portion of the dialog box routine.

Call the CtrlSetCurSel function to see if a string exists in the combo box. If the string is found, the CtrlSetCurSel function highlights it.

Set text in a combo box

Call the CtrlSetCurSel function to place the contents of a string list into a combo box control.

Call the CtrlSetText function to place text in a combo box control.

InstallShield
www.installshield.com

Clear text from an edit box

InstallShield
www.installshield.com

Call the CtrlClear function to remove all the text from a single-line edit box control.

Get text from an edit box

Call the CtrlGetMLEText function to retrieve the contents of a multi-line edit box control. You must create a string list before calling CtrlGetMLEText by calling the ListCreate function with the STRINGLIST option.

Call the CtrlGetText function to retrieve the contents of a single-line edit box control.

InstallShield
www.installshield.com

Select text in an edit box

InstallShield
www.installshield.com

Call the CtrlSelectText function to select all the text from a single-line or multi-line edit box control. The CtrlSelectText function is useful for highlighting default text in a single-line edit box. Call CtrlSelectText in the `case DLG_INIT:` portion of the dialog box routine.

InstallShield
www.installshield.com

Format text in an edit box

InstallShield
www.installshield.com

Call the CtrlSetFont function to specify the type of font you want to use in a single-line or multi-line edit box control. Call CtrlSetFont in the `case DLG_INIT:` portion of the dialog box routine to set the font for the controls.

Set text in an edit box

Call the CtrlSetText function to place text in a single-line edit control.

Call the CtrlSetMLEText function to place text in a multi-line edit box control. Lines of text are retrieved from a string list and placed in the edit box control. Each element of the string list is placed as a line in the multi-line edit box control.

You must create a string list before calling CtrlSetMLEText by calling the ListCreate function with the STRINGLIST option. You then fill the string list with the strings you want displayed in the multi-line edit box.

The ListAddString function adds a single line to the list. The order you enter the strings in the list determines the order in which they are displayed in the edit box. The first string in the list will be displayed at the top of the edit box.

Call the CtrlGetMLEText function to fill the list with the contents of the edit box. Each line enters the list as a string. The edit box's top line is the first line in the list.

InstallShield
www.installshield.com

Clear a list box

InstallShield
www.installshield.com

Call the CtrlClear function to remove all the text from a list box control.

InstallShield
www.installshield.com

Fill a list box

InstallShield
www.installshield.com

Call the CtrlDir function to fill a list box control with either a folder or file listing.

Call the CtrlPGroups function to place a list of existing program groups in a list box control. CtrlPGroups displays all the program groups that are located by the Program Manager.

InstallShield
www.installshield.com

Get selected items from a list box

InstallShield
www.installshield.com

Call the CtrlGetCurSel function to retrieve the currently selected items in a multi-selection list box control.

Call the CtrlSetMultCurSel function to retrieve a range of lines selected in a list box control. You create an empty list and pass it to the CtrlGetMultCurSel function. CtrlGetMultCurSel places each selection into its own string in the list. The list box's first selection is the first string in the list.

InstallShield
www.installshield.com

Select an option in a list box

InstallShield
www.installshield.com

Call the CtrlSetCurSel function to see if a string exists in the list box. If the string is found, the CtrlSetCurSel function highlights the string.

InstallShield
www.installshield.com

Set the contents of a list box

InstallShield
www.installshield.com

Call the CtrlSetList function to place the contents of a string list into a list box control.

Call the CtrlSetMultCurSel function to set the contents of a list box from a list. CtrlSetMultCurSel takes a previously created list and searches the list box for the strings contained in the list. If the string is found in the list, the string is selected as an item.

Using bitmap buttons in dialog boxes

Custom dialog boxes can contain bitmap buttons. Follow these steps to create dialogs that contain bitmap buttons:

1. Identify a bitmap (.bmp file) or icon (.ico file) you want to display on or as a button. The bitmap or icon image must be small enough to fit in the button and should have the same general proportions as the button.
2. Create a button in a dialog to display the bitmap or icon. As an example, use the Modem Installation dialog created in the custom dialog example, explained in How do I create a custom dialog box from a template.
   a. In MSVC++, open the _isuser.rc file for that dialog project.
   b. Open the dialog with ID 30001.
   c. Place a button just below the Comm Ports list box. Size the button so that it is square.
   d. Open the button's properties sheet and note the ID (IDC_BUTTON1 in the example).
   e. If you are displaying an icon, enter the letter I (as in "Ike") in the Caption: field. This tells InstallShield to display the object as an icon.
   f. Check the Owner Drawn check box in the Styles tab, and close the properties sheet.
3. From the Insert Resource dialog select Insert... Resource, select Bitmap or Icon in, and click Import to select a file, or OK to create a bitmap or icon. In either case, the bitmap or icon editor will open.
4. With a bitmap or icon open in the bitmap or icon editor, or, with the bitmap or icon selected in the resource editor window, select Properties from the Edit menu to open the properties sheet.
5. Select the same ID as noted earlier (IDC_BUTTON1) for the newly created button. Then close the resource editor window.
6. Open _isuser.mak, and select Rebuild All from the Build menu.
7. Place the modified _isuser.dll into the Setup Files pane under the appropriate target language(s) and operating system(s). InstallShield will include _isuser.dll in your setup the next time you build it with the Media Build Wizard.

To make the bitmap button do anything meaningful, you must handle it in your script as you would any other button control.

Creating custom dialog boxes from Sd dialog boxes

If an Sd dialog box function needs only minor modifications to meet your requirements, you may be able to use the DialogSetInfo function to achieve the desired result.

You can use a resource editor such as Microsoft Visual C++ to change the text in static fields, to change the text in push buttons, to change the locations of controls, and even to change the bitmap images in Sd dialogs. These types of changes do not add controls that require handling, nor do they alter the IDs of existing controls. Therefore, these types of changes require no alterations to the internal Sd dialog source scripts that run the Sd dialogs.

Changes to Sd dialogs that would require alterations to the Sd dialog source scripts, such as adding controls or changing control IDs, are not recommended. Making such changes can damage your Sd dialogs.

If you wish to attempt to make changes to Sd dialogs that require you to alter the Sd dialog source scripts, then refer to Making advanced modifications to Sd dialog boxes.

First, locate _isres.dll, which contains the resources for all Sd dialog boxes. _isres.dll is found in the <InstallShield location>\Redistributable\Compressed Files\<target language>\<target platform> folder. Note that your copy of InstallShield may contain more than one _isres.dll file, depending on the languages and platforms that your setup will target. Locate all versions of _isres.dll that are appropriate for your target platform(s). (Remember that you must use 16-bit resources for a 16-bit setup and 32-bit resources for a 32-bit setup.)

Use the Dialog Sampler to determine the resource IDs of the dialog box that you intend to modify. Run the Dialog Sampler from the Tools menu in the InstallShield IDE. Select the Sd dialog that you want to modify. Note the numeric dialog ID (listed in parenthesis) located below the list box after the DialogID: label.

Copy the necessary resources to _isuser.dll. _isuser.dll is a DLL template in which you can place custom setup resources. The resources in _isuser.dll override those in _isres.dll; never modify _isres.dll directly. The 16-bit version of the _isuser.dll project is located in <InstallShield location>\Custom Dialog\16bit, and the 32-bit version is located in <InstallShield location>\Custom Dialog\VC++ 4 Project.

Make the desired changes to the dialog box resources in _isuser.dll. There are a number of things you can change about Sd dialog resources, including:

{button ,JI(`GETRES.HLP>(w95sec)',`Changing_static_text')} Changing static text.

{button ,JI(`GETRES.HLP>(w95sec)',`Adding_static_text')} Adding static text.

{button ,JI(`GETRES.HLP>(w95sec)',`Adding_a_field_that_contains_a_product_name')} Adding a field that contains a product name.

{button ,JI(`GETRES.HLP>(w95sec)',`Modifying_accelerators')} Modifying accelerators.

{button ,JI(`GETRES.HLP>(w95sec)',`Relocating_controls')} Relocating controls.

Save your changes. Finally, place the modified _isuser.dll into the Setup Files pane under the appropriate target language(s) and operating system(s). InstallShield will include _isuser.dll in your setup the next time you build it with the Media Build Wizard.

Using copies of SdAskOptions to create unique dialogs

You can create multiple instances of the SdAskOptions dialog box, modify their controls, and use them as new dialogs without making changes to the Sd dialog's source script. The SdAskOptions dialog box is the only dialog box that InstallShield allows you to duplicate without modifying the Sd dialog's source script.

Do not confuse the process of duplicating and modifying the SdAskOptions dialog with the process of creating a completely customized dialog from scratch, using an Sd dialog resource as a template.

To copy and use versions of the SdAskOptions dialog:

1. Open _isres.dll in a resource editor (open only the appropriate copy for the target language and platform), copy the dialog to _isuser.dll as described above, and give it a new, unique resource ID number.
2. Use a resource editor to customize the second instance of the SdAskOptions dialog box in the ways described above.   Save the file and place it in the Setup Files pane.
3. Call the SdAskOptions function in your script and use the ID number of the modified copy of the SdAskOptions dialog. The modified dialog will be displayed in your setup.

Changing static text

InstallShield
www.installshield.com

You can change the text in a dialog box control by double-clicking the control in the Visual C++ dialog editor and editing the text in the appropriate text field. Simply enter the new text you want to use and save the changes to _isuser.dll.

Adding static text

You can add static fields as long as you give them the default ID of -1. You can have more than one static text field with an ID of -1. The static text fields require no handling in the Sd dialog source script.

InstallShield
www.installshield.com

Adding a field that contains a product name

InstallShield
www.installshield.com

InstallShield allows you to place your product name globally in Sd dialog static text fields containing the %P place holder (sometimes appearing as an extra space). When you call the SdProductName function in your setup script, your product name will be displayed in place of %P in the static text fields.

If you are adding a static text field containing %P for use with the SdProductName function, give the static text field a unique (to that dialog) ID in the range of 701 through 799. IDs in the 701 through 799 range instruct InstallShield to scan the static text field for the existence of the %P place holder. If the %P place holder is found, InstallShield replaces %P with the string you entered in the SdProductName function.

InstallShield
www.installshield.com

Modifying accelerators

InstallShield
www.installshield.com

You can modify the static text associated with check boxes, radio buttons, push buttons, etc., which is useful for changing the accelerator keys you press to quickly access your application. To change the accelerator key for a control, double-click the control to open its style dialog. Move the "&" character from its current location to in front of the letter you wish to serve as the accelerator key. For example, **&Next** causes the control to display **Next**, where **N** is the accelerator key. Moving the "&" to precede the letter "x" (**Ne&xt**) causes "x" to become the accelerator key: **Next**.

InstallShield
www.installshield.com

Relocating controls

InstallShield
www.installshield.com

You can change the location of any push button control (push button, radio box, check box) in the dialog editor by simply moving the control to the desired location.

InstallShield
www.installshield.com

Making advanced modifications to Sd dialog boxes

InstallShield
www.installshield.com

All the parts of InstallShield have been tested for compatibility with a wide range of hardware and operating system configurations. Modification of the Sd dialog scripts should be avoided as it may create bugs. InstallShield does not recommend modifying an Sd dialog in any way that requires modification of the dialog's source script.

If you absolutely must modify one of these scripts, follow the directions given in this section. Be sure to make backup copies first. Test your modifications extensively.

You can add additional active controls to any Sd dialog box. For example, you can add an additional setup choice button to the SdSetupType dialog. In this case, the third button must have an ID of 103 (since the first button has ID 101 and the second 102). You could then add a text field with an ID in the range of 901 to 999 next to the button.

You can also add additional fields to Sd dialog boxes. For example, you can add an additional field to the SdAskDestPath function to handle multiple folders (for example, one folder for server files and another folder for client files).

Follow these steps to add an additional active control to an Sd dialog box:

1. Modify the dialog box by adding the appropriate controls. Use the table below to determine the resource ID of the control. Remember that controls must be sequentially numbered.

Control	Resource ID
Push Button	101 - 199
Edit Field	301 - 399
List Box	401 - 499
Check Box	501 - 599
Radio Button	501 - 599
Combo Box	601 - 699

2. Access the appropriate source script file for the desired dialog box. This file name is displayed ing the Script Path: field in the Dialog Sampler when the dialog's function name is highlighted.
3. Add the proper handling routine(s) in the Sd dialog's source file. Use switch statements inside the while loop following the WaitOnDialog function call to test for messages and respond accordingly. If you do not perform this step, the script will not handle the new control. The Sd dialog source scripts (.rul files) are found in the <InstallShield location>\Include folder.

{button ,AL(`Modifying Sd dialog box source files',0,`',`')} See also

Troubleshooting custom dialog box errors

Try each of the following troubleshooting techniques if you are having difficulties using custom dialog boxes in your setup.

Make sure the DLL is in the specified path

If the DLL is not found in the specified path, the DefineDialog and EzDefineDialog functions will not return an error. They put all the items in a structure and return 0 (zero). However, the WaitOnDialog function will return DLG_ERR if the DLL is not found in the specified path. Check the return values from the WaitOnDialog function. Then, do the following to make sure that the DLL is being loaded and referenced correctly:

- Verify that the DLL you are using is 32-bit if you are running a 32-bit setup or 16-bit if you are running a 16-bit setup. If you try to load a 16-bit DLL in a 32-bit setup or a 32-bit DLL in a 16-bit setup, the UseDLL call will fail and no dialog box will be displayed.
- Verify that the DLL is found in the InstallShield IDE's Setup Files pane under the correct folder.

  For example, if Mydll.dll (which contains installation dialog boxes separate from other resources) is placed in the Setup Files pane, call the DefineDialog function as shown below:

  ```
  //Build the path to the DLL.
  szDLL = SUPPORTDIR ^ "Mydll.dll";

  // Define the dialog in your script.
  DefineDialog("MYDIALOG",  //string that identifies dialog
     0,
     szDLL,     // full path for the DLL
     20,        // dialog ID in the Resource DLL
     "",
     HWND_DESKTOP,
     HWND_INSTALL,
     DLG_MSG_STANDARD | DLG_CENTERED);
  ```

- Debug your setup and verify that the renamed version of _isuser.dll is located in the folder (contained in the SUPPORTDIR system variable). Use the InstallShield Visual Debugger to check the ISUSER system variable to see what _isuser.dll has been renamed to during the current installation and verify that this file exists in the SUPPORTDIR folder. If the ISUSER system variable is blank, it means that InstallShield did not find _isuser.dll in the support folder.
- Make sure you have loaded your DLL using the UseDLL function. Be sure that the DLL file name is correct. (If your custom dialog is stored in _isuser.dll, you don't need to call UseDLL. _isuser.dll and _isres.dll are loaded automatically when setup initializes.) For example, if your dialog's resource is stored in IsCDlg.dll, which you had placed in the Setup Files pane, use the following syntax:

  ```
  UseDLL (SUPPORTDIR ^ "IsCDlg.dll");
  ```

- Verify that the DLL file has not been renamed. The actual file name of the DLL and the module name given in the .def file must match. You can use the Exehdr.exe (16-bit) or Dumpbin.exe utility (32-bit) to check the library name or module name and see if it matches the name of the DLL. The Exehdr.exe and/or Dumpbin.exe utility programs are included with MSVC++ 1.52 and MSVC++ 2.x - 5.x, respectively.
- Make sure that your .def file includes an accurate export section. In order for InstallShield to be able to access your DLL correctly, all functions in the DLL should be listed in the export section.

Make sure the resource works correctly

- Load the DLL into a resource editor and see if you can display the dialog box in the resource editor. If you cannot display the dialog box in a resource editor, the dialog box cannot be displayed in the installation. Verify the dialog box is operating correctly, reinsert it into the DLL, and try the dialog box in the script again.
- Try to display the dialog from a C++ program. If unable to do so, refer to a Windows programming guide to

verify that you have created the dialog correctly.

Make sure your call to EzDefineDialog is correct

Remember that the EzDefineDialog function simply stores information about the dialog to be used by the WaitOnDialog function and rarely returns an unsuccessful error code, regardless of whether the dialog can be successfully loaded. Verify that all parameters are correct:

szDialogName

This parameter is used to reference the dialog. Verify that any other function that references the dialog box uses this value. The string you enter in the first parameter of the EzDefineDialog function must match the string used in the WaitOnDialog function.

szDLL

This parameter requires the full path of the DLL, not just the file name. The system variable ISUSER includes the full path of _isuser.dll.

Make sure that, if you referred to _isuser.dll in the EzDefineDialog function call, you used the system variable ISUSER, instead of giving the path and file name explicitly. This parameter can also be set to a null string ("") when the dialog is located in _isuser.dll. InstallShield automatically looks in _isuser.dll and _isres.dll when this parameter is a null string.

szID

This parameter is normally set to a null string, because it is standard practice to use numeric IDs for all resources. But if you did decide to name your dialog box with a string ID, you need to specify it here.

nID

This parameter must match the ID of the dialog box in the DLL. If this parameter is incorrect, the dialog box will not display at all. Verify that this parameter matches the dialog's resource ID in the DLL.

A typical example of the correct syntax for EzDefineDialog when the dialog's resource is placed in _isuser.dll is:

```
EzDefineDialog(DIALOG_NAME, ISUSER, "", DIALOG_ID);
```

Overview: unInstallShield

InstallShield comes with a built-in uninstallation program called unInstallShield. During setup, you can record events for uninstallation—the creation of files, folders, program items, and registry entries. When launched, unInstallShield removes any properly created InstallShield setup.

unInstallShield is not only powerful, it is smart. unInstallShield protects you from accidentally deleting shared files, including Windows 95 Core Components files that your system requires to run properly.

As unInstallShield performs the uninstallation, a dialog box shows progress by checking off each category of uninstallable items. When uninstallation is complete, a message displays at the bottom of the uninstallation dialog box. If any items could not be removed, the message informs the user that the uninstallation was incomplete and that some items must be removed manually. Click the Details button to see a list of uninstallable items. You can easily copy this list to a text file.

Uninstallation involves many complex topics, often overlapping with Windows logo certification requirements. You needn't concern yourself further with unInstallShield if the uninstallation functionality provided by the Project Wizard completely removes your setup. Test your setup and then launch unInstallShield. unInstallShield tells you if it could not remove any items.

If you use the Setup.rul created by the Project Wizard, you will find that your setup is already uninstallable under Windows NT 4.0 and Windows 95. InstallShield takes care of setting up minimum uninstallation functionality by automatically:

- copying the unInstallShield executable to the Windows folder
- creating an uninstallation log file
- setting up the uninstallation string in the registry

What unInstallShield removes

unInstallShield removes files, folders, program items, and registry entries that were logged for uninstallation. unInstallShield will remove read-only files, but it cannot remove files with a hidden or system attribute, except .gid, .ftg, and .fts files created by Windows help engines that unInstallShield always removes.

On 32-bit Windows systems, unInstallShield can handle shared files automatically. On 16-bit systems InstallShield does not log shared files for uninstallation since the 16-bit registration database has no reference counters, so shared files are never removed..

Any program folders, menu items, and program groups that are created and logged during setup and are empty after unInstallShield has removed their contents, are also removed. But unInstallShield cannot remove items added after InstallShield runs or folders that contain new items. Be aware that registry entries added after the installation is complete will be removed by unInstallShield if they occur under a key that was logged for removal.

unInstallShield can reverse changes made to .ini files; modifications to .bat and .sys files cannot be undone.

{button ,AL(`What gets logged for uninstallation;Launch unInstallShield;Prepare my setup for uninstallation;Installing self-registering files;Installing shared files',0,`',`')} See also

InstallShield
www.installshield.com

Launch unInstallShield

Launch unInstallShield according to the operating system:

{button ,JI(`GETRES.HLP>(w95sec)',`Launching_unInstallShield_under_the_Explorer_shell')} On systems running the Explorer shell, unInstallShield is launched via the Add/Remove Programs icon in Control Panel.

{button ,JI(`GETRES.HLP>(w95sec)',`Launching_unInstallShield_under_the_Program_Manager_shell')} On systems running the Program Manager shell, unInstallShield is launched from an uninstall icon in the application's program folder.

{button ,JI(`GETRES.HLP>(w95sec)',`Launching_unInstallShield_from_a_custom_uninstaller')} Alternatively, you can launch unInstallShield from a custom uninstaller.

Also, unInstallShield is launched in silent mode to clean up an unfinished setup. When a setup is aborted in any of these ways (all of which call the default exit handler), unInstallShield removes all traces of the aborted setup:

- The user presses F3 or Cancel, and then clicks the Exit button
- The script calls the Do function with the EXIT option while the default exit handler is in effect
- The script calls the abort keyword

If a setup terminates abnormally, such as from a General Protection Fault or other serious error condition, unInstallShield may not be launched, and therefore the incomplete setup may not be removed.

{button ,AL(`The unInstallShield executable files;The command line parameters available for unInstallShield',0,`',`')} See also

Launching unInstallShield under the Explorer shell

Under the Explorer shell, the uninstallation process begins when the user clicks the Add/Remove Programs icon from the Control Panel. (The Control Panel is located on the Settings menu under the Start menu.)

When the user double-clicks the Add/Remove Programs icon, the Add/Remove Programs Properties sheet opens. The Install/Uninstall tab of the Add/Remove Programs Properties sheet lists all applications that can be automatically uninstalled. The user selects the application to be uninstalled and clicks Remove to launch unInstallShield.

A Confirm File Deletion dialog box appears after the user has chosen to remove the application. If the user chooses Yes to continue the uninstallation, Explorer executes the expression assigned to the `[UninstallString]` value name in the application uninstallation registry key. The user cannot abnormally terminate the uninstallation process once it starts.

Your application will be listed as an application that can be uninstalled in the Add/Remove Programs properties sheet Install/Uninstall tab as long as a valid application uninstallation key and entries exist in the registry. The following is an example of a valid application uninstallation key and entries:

```
[HKEY_LOCAL_MACHINE]\Software\Microsoft\Windows\Current Version\Uninstall\
InstallShield5 Free Edition
   [DisplayName] = "InstallShield5 Free Edition "
   [UninstallString] = D:\WIN95\IsUninst.exe -f"D:\Program Files\InstallShield\
InstallShield5 Free Edition\Uninst.isu""
```

InstallShield
www.installshield.com

Launching unInstallShield under the Program Manager shell

Windows 3.1 and Windows NT 3.51 run the Program Manager shell. Since Program Manager has no Add/Remove Programs functionality, unInstallShield must be launched from an icon in the application's program folder. The icon executes an uninstallation command expression like the following:

C:\Windows\IsUn16.exe -fC:\Windows\Uninst.isu

The uninstallation expression is the same expression that is assigned to the `[UninstallString]` value under the application uninstallation key in the 32-bit registry.

To create the uninstallation expression, use the UNINST system variable, which contains the full path and file name of the unInstallShield executable file. Concatenate this with the -f switch and the full path and file name of the uninstall log file (returned in the second parameter to DeinstallStart). Next, pass the concatenated expression in a call to AddFolderIcon to add the uninstaller icon to the application's program folder:

```
szProgram = UNINST + " -f" + svUninstLogFile;
AddFolderIcon(svFolder, "unInstallShield", szProgram, WINDIR,
              "", 0, "", REPLACE);
```

Launching unInstallShield from a custom uninstaller

If your needs include uninstallation functionality not supported by unInstallShield, you can write a custom uninstallation program to carry out the special functionality. At the end of your custom uninstallation program, you can launch unInstallShield to carry out the uninstallation prepared by InstallShield.

Use the uninstallation expression information described in Launching unInstallShield under the Explorer shell and Launching unInstallShield under the Program Manager shell to launch unInstallShield from your custom uninstallation program. Have your installation write the values of UNINST and svUninstLogFile (the second parameter to DeinstallStart) to a text file that your custom uninstallation program can access to obtain the uninstallation expression information needed to launch unInstallShield.

{button ,AL(`Call a custom DLL function from unInstallShield',0,`',`')} See also

unInstallShield's executable files

unInstallShield requires an executable file and an uninstallation log file in order to run. There is a 32-bit executable file for 32-bit platforms, and a 16-bit executable for 16-bit platforms.

IsUninst.exe (32-bit) and IsUn16.exe (16-bit) are the unInstallShield executable files. IsUninst.exe and IsUn16.exe are redistributable files that InstallShield compresses by default into _sys1.cab (placed into your main setup folder by the Media Build Wizard).

The IsUninst.exe and IsUn16.exe files are automatically installed in the Windows folder as long as they are properly placed on the distribution media. Use the UNINST system variable to obtain the full path and file name of the unInstallShield executable file for the current installation.

Do not modify the resources contained in IsUninst.exe and IsUn16.exe. IsUninst.exe and IsUn16.exe are shared by all applications uninstallable by unInstallShield, so modifying their resources may affect the uninstallation of applications other than your own.

IsUninst.exe and IsUn16.exe can be run from the command line like any other executable file, but they are usually run from the Add/Remove Programs Properties sheet on systems running Explorer, or from an uninstaller icon in the program folder on systems running Program Manager.

{button ,AL(`The uninstallation log file;The command line parameters available for unInstallShield',0,`',`')} See also

InstallShield
www.installshield.com

The uninstallation log file

InstallShield creates an uninstallation log file for each setup. The uninstallation log file contains a record of all the uninstallation-related events that occur during the installation.

The log file is actually initialized when you call the DeinstallStart function in your script. InstallShield places the log file in the location specified in the first parameter of DeinstallStart.

By default, the uninstallation log file is named Deisl*xx*.isu (where *xx* is a number one greater than the existing Deisl*xx*.isu in that folder). If there are no pre-existing uninstallation log files, InstallShield creates Deisl1.isu.

You can also specify a name for the uninstallation log file in the second parameter to DeinstallStart. If you provide a file name in the second parameter and that file already exists at the location specified in the first parameter and is a valid uninstallation log file, InstallShield will append the new uninstallation log file to the existing log file.

{button ,AL(`Appending to an existing uninstallation log file;What gets logged for uninstallation',0,`',`')} See also

Appending to an existing uninstallation log file

InstallShield lets you append a setup's uninstallation log file to an existing log file. This way, InstallShield does not create separate log files, and all uninstallation information is contained in a central location.

Specify the uninstallation log in the second parameter to the DeinstallStart function. This file must exist and be a valid log file.

A setup appending to a log file must use the same values in its calls to InstallationInfo and DeinstallStart as were used in the setup that created or previously appended to the existing log file.

InstallShield
www.installshield.com

What gets logged for uninstallation

All files, folders, registry keys, .ini file entries, program folders and program items created or replaced while logging is enabled are logged for uninstallation. Logging is enabled by default, but if you have disabled it for any reason, simply call Enable(LOGGING) to reverse it.

If you want to perfom some action in your setup that you do not want unInstallShield to remove, you can call Disable(LOGGING) beforehand.

InstallShield records setup events created by these functions for uninstallation in the uninstallation log file:

Functions that transfer files

ComponentMoveData

CopyFile

VerSearchAndUpdateFile

VerUpdateFile

XCopyFile

Functions that work with folders

AskDestPath

AskPath

CreateDir

SdAskDestPath

SelectDir

Functions that create program folders and icons

AddFolderIcon

CreateProgramFolder

Functions that create registry keys or write to the registry

The following registry entries will be logged for uninstallation:

- Registry entries created automatically by InstallShield.
- Registry entries and associated keys, and all subkeys located below such keys, that were automatically created by InstallShield. (unInstallShield does not automatically remove the company name key under HKEY_LOCAL_MACHINE\Software, since that might inadvertently remove subkeys for other applications.)

DeinstallStart

InstallationInfo

RegDBCreateKeyEx

RegDBSetAppInfo

RegDBSetItem

RegDBSetKeyValueEx

Functions that modify initialization (.ini) files

InstallShield can remove .ini file entries that were created with the functions listed below. For more information about uninstalling .ini file entries and the limitations involved, see Uninstalling initialization (.ini) file entries.

AddProfString

ReplaceProfString

WriteProfString

{button ,AL(`Uninstalling initialization (.ini) file entries;Installing self-registering files;Installing shared files;The uninstallation log file',0,`',`')} See also

Prepare my setup for uninstallation

Setting up uninstallation functionality is simple, considering the complexity of the operations that result. You must call certain functions to set up uninstallation functionality.

Required function calls

To set up uninstallation functionality, you must call InstallationInfo and DeinstallStart (as close to each other in your script as possible), even when the target system is 16-bit Windows. When the target platform is 32-bit Windows, you must also call RegDBSetItem so that the application uninstallation key is created. Calling RegDBSetItem on 16-bit Windows has no effect, but is in fact harmless. Therefore, including the call to RegDBSetItem in all your scripts helps make them platform independent.

If the possibility exists that the target platform will be running the Program Manager shell (Progman.exe), you must be prepared to call CreateProgramFolder and AddFolderIcon to create an uninstall icon to launch unInstallShield, since the Program Manager has no Add/Remove Programs icon.

An example script is included which demonstrates what you will need to write in your script to meet the minimum requirements to set up uninstallation functionality. To determine whether the target system is running Program Manager or Explorer, the script begins by setting bExplorer to FALSE, meaning the script assumes the shell is Program Manager unless otherwise indicated.

Next, GetSystemInfo is called with the OS option. If the target system is Windows NT or Windows 95, GetSystemInfo is called again, this time with the OSMAJOR option. If OSMAJOR yields a value greater than 3, bExplorer is set to TRUE to indicate that Explorer is the shell.

At the end of the example script, bExplorer controls whether or not a program group and uninstaller icon are created.

There are other functions that play important roles in setting up uninstallation functionality. For example, all keys created using the RegDBCreateKeyEx function are recorded by unInstallShield for uninstallation, unless logging is disabled. Also, if you call the Disable function to disable logging, remember to call Enable to enable the logging again for subsequent actions that must be recorded for uninstallation.

If you created your setup using the InstallShield Project Wizard, InstallShield will have already enabled minimum uninstallation functionality for you. If you need to modify the calls to InstallationInfo, DeinstallStart and RegDBSetItem, go to the ProcessBeforeDataMove function definition in Setup.rul (found in the IDE's Scripts pane).

{button ,AL(`The primary registry functions',0,`',`')} See also

InstallShield
www.installshield.com

Sample script for setting up uninstallation

```
   // Define constants for values required in function calls.
   #define APPBASE_PATH            "test_dir"
   #define COMPANY_NAME            "Test_Company"
   #define PRODUCT_NAME            "Test_App"
   #define PRODUCT_VERSION         "5.0"
   #define PRODUCT_KEY             "test.exe"
   #define DEINSTALL_KEY           "Test_DeinstKey"
   #define UNINSTALL_NAME          "Test_App 5.0"
   #define PROGRAM_FOLDER_NAME     "Test_App Folder"

   // Declare global variables.
   STRING szMsg, svTarget, svResult, svUninstLogFile, svFolder, szProgram;
   BOOL   bExplorer;
   NUMBER nvResult;

program

start:

   // Get a destination location for the installation.
   szMsg = "Select a destination location.";
   svTarget = TARGETDISK ^ APPBASE_PATH;
   if (AskDestPath("", szMsg, svTarget, 0) = BACK) then
      goto start;
   endif;

/*--------------------------------------------------------------------------*\
 * Determine whether Explorer or Program Manager is running based on OS type
 * and, when required, OS major version number. By default, we assume that
 * Explorer is NOT running.
\*--------------------------------------------------------------------------*/
   // Important! Must initialize bExplorer to FALSE.
   bExplorer = FALSE;
   GetSystemInfo(OS, nvResult, svResult);

   if (nvResult = IS_WINDOWSNT) || (nvResult = IS_WINDOWS95) then
      GetSystemInfo(OSMAJOR, nvResult, svResult);
      if (nvResult > 3) then
         bExplorer = TRUE;
      endif;
   endif;

   // Call InstallationInfo, which is required before calling DeinstallStart.
   InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
   // Call DeinstallStart, which is required to enable creation of the uninstall
   // log file and the application uninstallation registry key.
   DeinstallStart(svTarget, svUninstLogFile, DEINSTALL_KEY, 0);

/*--------------------------------------------------------------------------*\
 * Call RegDBSetItem (required) to create the application uninstallation key
 * and set the uninstall name in the Control Panel Add/Remove Programs dialog
 * (Explorer only). This call has no effect on 16-bit Windows, but it is
 * required on 32-bit Windows, regardless of whether the shell is Explorer
 * or Program Manager.
\*--------------------------------------------------------------------------*/

   RegDBSetItem(REGDB_UNINSTALL_NAME, UNINSTALL_NAME);
```

```
/*--------------------------------------------------------------------------*\
 * If the shell is Program Manager, install a program folder and then place an
 * uninstall icon in the folder. This is not required on Windows 95 or
 * Windows NT 4.0 or greater because they use Explorer, which provides access
 * to uninstallation via the Add/Remove Programs dialog in Control Panel.
\*--------------------------------------------------------------------------*/
   if (bExplorer = FALSE) then
      AppCommand(PROGMAN, CMD_RESTORE);
      svFolder = PROGRAM_FOLDER_NAME;
      CreateProgramFolder(svFolder);

      szProgram = UNINST;
      szProgram = szProgram + " -f" + svUninstLogFile;
      AddFolderIcon(svFolder, "unInstallShield", szProgram, WINDIR,
                    "", 0, "", REPLACE);
   endif;

endprogram

// Source file:  Is5gr005.rul
```

Call a custom DLL function from unInstallShield

unInstallShield allows you to call DLL functions to supplement its uninstallation capability. This extensibility feature lets you access your own DLL at two different points during the uninstallation process. For example, you can write custom DLL functions that supplement unInstallShield by removing changes made to batch files or reversing changes made to a file's attributes.

The DLL must contain at least two functions—an initialization function and an uninitialization function. The name of the initialization function must be UninstInitialize (called before unInstallShield begins uninstalling the application), and the name of the uninitialization function must be UninstUnInitialize (called after unInstallShield finishes uninstalling the application).

In order to call the uninstallation extension DLL, you have to modify the uninstallation string so that unInstallShield knows which DLL to load. A typical unInstallShield command line expression contains the path and file name of the uninstallation log file:

```
D:\Win95\IsUninst.exe -fC:\Program Files\MyApp\Uninst.isu
```

Modifying this command line with the -c<file name> parameter allows unInstallShield to call the uninstallation extension DLL:

```
D:\Win95\IsUninst.exe -fC:\Program Files\MyApp\Uninst.isu -cC:\Program Files\
MyApp\MyUninst.dll
```

Under 32-bit platforms this command line is used as the `[UninstallString]` value under the appropriate registry key; under 16-bit platforms this command line is associated with the uninstaller icon.

So that the uninstallation extension DLL can be removed during the uninstallation process, it is copied to a temporary location and renamed before it is actually loaded. unInstallShield only removes the original DLL if it was already logged for uninstallation. The renamed DLL is removed during the file closing phase of the uninstallation.

unInstallShield unloads and reloads the uninstallation extension DLL between calling UninstInitialize and UninstUnInitialize. Therefore, any static data created by the UninstInitialize function will be lost when the DLL is unloaded. UninstUnInitialize should only use data that it initializes explicitly during execution.

You will probably need to create your own log file to keep track of setup events that you want to uninstall with your custom DLL. For some hints on creating a custom uninstallation log file, see Creating and accessing a custom uninstallation log file.

UninstInitialize and UninstUnInitialize

When unInstallShield calls your custom DLL functions during uninstallation, the functions receive these parameters:

hwndDlg
Window handle of the main unInstallShield dialog.

hInstance
Instance handle for IsUninst.exe or IsUn16.exe.

lReserved
Reserved. This parameter can only contain 0 (zero).

UninstInitialize returns these values:

0
Continue with the setup.

< 0
Abort unInstallShield.

The return value for UninstUnInitialize is ignored by unInstallShield.

InstallShield
www.installshield.com

Creating and accessing a custom uninstallation log file

You will probably want to create an additional log file that your custom uninstaller can access to determine what items need to be removed during uninstallation.

You could install this file into a common folder, such as the Windows\System folder and then access it from there. But since this method could cause conflicts with other programs, we recommend placing the custom uninstallation log file in the same folder as unInstallShield's log file—typically the same folder as the application's executable, specified as the first parameter to DeinstallStart.

You can easily write code that finds the path of the unInstallShield log file. Then, use this location to locate your custom uninstallation log file. You can then append the name of your custom log file to this value and access it by calling Windows APIs.

Another suggestion is to store the window name and window class of the application. Then, call a function to check to see if there is a valid window handle for this window's class and name. If found, the function prompts the user to close down the application before continuing. Checking for the window class and window name prevents users from attempting to uninstall a running application.

If you include this feature in your uninstallation, remember to use a window class and window name for a window that is always open while your application is running. You should try and make this window class and window name as unique as possible to avoid having UninstInitialize or UninstUnInitialize mistake another running application for yours.

InstallShield
www.installshield.com

The command line parameters available for unInstallShield

These command line parameters can be used with IsUninst.exe or IsUn16.exe (the unInstallShield executable).

When passing a path and file name in a command line parameter, as you do with -f, -c, and -d, you must enclose a name containing one or more spaces in double quotation marks. If you don't, the command line will misinterpret the command.

Parameter	Result
-y	Suppresses the message box that asks the user to confirm that uninstallation should proceed. The feedback dialog box is still displayed, as is the shared file dialog box (which is displayed when the reference count of a shared DLL is decremented to zero).
-x	Deletes all files, including those core components that normally do not get removed. (All user interface elements are displayed.)
-f<log file name>	Specifies the location and name of the uninstallation log file. For example: `-f"C:\Program Files\Company Name\Uninst.isu"`
-c<DLL file name>	Specifies the location and name of the external DLL that is to be used at the time of uninstallation. For example: `-c"C:\Program Files\Company Name\Custom.dll"`
-a	When running in silent mode, if unInstallShield encounters a shared file for which it would normally display the dialog box asking the user whether to remove the shared file, it will automatically reduce the reference count to zero and not remove the file. Therefore, running unInstallShield in silent mode is functionally equivalent to an uninstallation in which the user selects the "No to all" option when this dialog box first appears.
-d	Identifies a single file that is to be deleted. The display of user interface elements is the same as when the -a switch is used. For example: `-dC:\Temp\Filename.ext`

InstallShield
www.installshield.com

Uninstalling initialization (.ini) file entries

unInstallShield will remove modifications that were made with the InstallScript initialization file functions while logging is enabled.

Click the individual function below to see what conditions apply when uninstalling its changes. Also, refer to general limitations.

{button ,JI(`GETRES.HLP>(w95sec)',`Uninstalling_AddProfString_entries')} AddProfString

{button ,JI(`GETRES.HLP>(w95sec)',`Uninstalling_ReplaceProfString_entries')} ReplaceProfString

{button ,JI(`GETRES.HLP>(w95sec)',`Uninstalling_WriteProfString_entries')} WriteProfString

---

{button ,AL(`What gets logged for uninstallation',0,`',`')} See also

Uninstalling AddProfString entries

unInstallShield will remove the keyname and value pair completely if the following conditions are true:

- If the key was successfully created by AddProfString.
- If the keyname and value pair that exist when unInstallShield is run exactly match the installed keyname and value pair. That is, if another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value which was created by the original installation will be removed—the key itself and any additional values will not be removed by the uninstaller.

For example, if the System.ini file originally read:

```
[386Enh]
device=votc.386
device=*vmpcd
```

and after your call to AddProfString, it read:

```
[386Enh]
device=*vmp32
device=votc.386
device=*vmpcd
```

unInstallShield would remove `device=*vmp32`.

If another installation had added a value to the existing line, only the value from the installation that is being uninstalled will be removed. The original keyname and the new value will be left in the file. For example, if you had added the line `Test Values=votc.386` in Test.ini under the `[Test]` section:

```
[Test]
Test Values=votc.386
Continuous Test=No
```

and another setup had added a new value to `Test Values` after your installation had run:

```
[Test]
Test Values=votc.386,pctcp.386
Continuous Test=No
```

when your application is uninstalled, `votc.386` will be removed and `TestValues=pctcp.386` will be left in the file:

```
[Test]
Test Values=pctcp.386
Continuous Test=No
```

{button ,AL(`General limitations',0,`',`')} See also

Uninstalling ReplaceProfString entries

unInstallShield will remove the keyname and value pair completely if the following conditions are true:

- The key was created successfully by ReplaceProfString.
- The key did not previously exist.
- If the keyname and value pair that exist when unInstallShield is run exactly match the installed keyname and value pair . That is, if another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value which was created by the original installation will be removed—the key itself and any additional values will not be removed by the uninstaller.

For example, if the System.ini file originally read:

```
[386Enh]
device=votc.386
device=*vmpcd
```

and after your call to ReplaceProfString, it read:

```
[386Enh]
device=*vmp32
device=votc.386
device=*vmpcd
```

unInstallShield would remove `device=*vmp32`.

If the key already existed when you called ReplaceProfString, unInstallShield will remove the newly added value if it meets all of the following conditions:

- If szOrigValue is a subset of szReplaceValue. That is, if the string passed to ReplaceProfString in szReplaceValue equals the value of szOrigValue plus the values being added.
- The new value must be appended as the first or last value of the existing value string. If the new value is added in the middle of the string, unInstallShield will not be able to remove it.
- If only commas and semicolons are used as delimiters.

If multiple keynames exist in a section, the first one found is considered to be the one with the appended string.

For example, if the existing .ini file read:

```
[386Enh]
network=*netbios,*vwc,*vnetsup.386,vredir.386
```

and it looked like this after your call to ReplaceProfString:

```
[386Enh]
network=*netbios,*vwc,*vnetsup.386,vredir.386,vserver.386
```

unInstallShield will remove `vserver.386` from the `network` key.

{button ,AL(`General limitations',0,`',`')} See also

InstallShield
www.installshield.com

Uninstalling WriteProfString entries

unInstallShield will remove the keyname and value pair completely if the following conditions are true:

- If the key was created successfully by WriteProfString.
- If the keyname and value pair that exist when unInstallShield is run exactly match the installed keyname and value pair . That is, if another installation or program modifies the installed key between the installation and uninstallation, the keyname and value pair will not be removed completely. Only the value which was created by the original installation will be removed—the key itself and any additional values will not be removed by the uninstaller.

{button ,AL(`General limitations',0,`',`')} See also

General limitations

The following limitations apply to removing .ini file entries and modifications automatically with unInstallShield:

- Logging must be enabled when the above functions are called.
- If an .ini file was created by any of the above functions, it is not removed.
- If a section name was created by any of the above functions, it is not removed, even if no more keys are present.
- If a key value is completely replaced by an InstallScript function (not appended to, or prepended to) the key values which existed before the installation are not restored. In this case InstallShield considers the replaced key a newly created key and will uninstall it as if it is a new key which was created by the installation.
- unInstallShield does not restore keys or values that were deleted using WriteProfString.
- In order to append a value to existing key (for example, `network` under `[386Enh]`), the new entry must either be appended at the end or prepended in the very beginning of the existing string but never inserted in the middle.
- Only the comma (,) and semicolon(;) are considered valid delimiters. If a string value to be uninstalled is found as a part of longer string, the value and the delimiter before or after it is also removed appropriately (based on the string's position)—only if the delimiter is a comma or a semicolon. For example, if unInstallShield removes `pqr` from the string `Key=pqr,rst,uvw`, it also removes the comma after `pqr`. A character other than a comma or a semicolon in its place will not be removed. As a rule, avoid adding spaces around delimiters.

InstallShield
www.installshield.com

Prevent unInstallShield from removing a running application

Call the DeinstallSetReference function to give unInstallShield the name of a reference file to check before removing the application. If the file is in use, unInstallShield will not continue with the uninstallation. If you want to specify more than one reference file, you can call DeinstallSetReference multiple times with different file names in the szReferenceFile parameter.

If you wanted unInstallShield to check if MyApp.exe and MyApp.dll, for instance, were in use before uninstalling the application, call DeinstallSetReference as follows:

```
// After calls to InstallationInfo and DeinstallStart
// to set up uninstallation functionality.

// svDir is the value that the end user returned
// for the application's destination folder.
szReferenceFile = svDir ^ "MyApp.exe";
DeinstallSetReference (szReferenceFile);

szReferenceFile = svDir ^ "MyApp.dll";
DeinstallSetReference (szReferenceFile);
```

If you created your setup using the Project Wizard, InstallShield will have already placed a call to DeinstallSetReference for you. Simply assign the name of your application's reference file(s) to szReferenceFile to activate this feature in your setup.

InstallShield
www.installshield.com

How Windows NT security permissions affect uninstallation

You may not be fully able to uninstall an application under Windows NT if you are not logged on with the same privileges that you had when you installed the application. There is nothing that unInstallShield can do to override security permissions.

For example, you may not be able to delete many registry keys if you are not logged on as an administrator. For more information about how default Windows NT security permissions affect creation and deletion of registry keys, folders, and icons, click here.

If the application was installed by someone with administrator privileges, unInstallShield has a built-in feature that checks whether you are uninstalling with the necessary privileges. If you are logged on as a user but the application was installed under an administrator account, unInstallShield displays an error message asking you to log on as an administrator and the uninstallation terminates.

This feature is available only in the InstallShield Professional Edition. For a list of differences between the Free and Professional editions, click here.

InstallShield
www.installshield.com

Overview: Installing Files

The core process of a setup involves transferring files from your distribution media to your end user's hard drive. Using InstallShield, you handle installing your application files in two distinct steps:

- In the InstallShield IDE, organize your application files into file groups, components, and setup types and set installation properties for each level.
- In your setup script, display setup type and/or component selections for your end user, then transfer selected components as appropriate.

These areas do overlap to some degree. Since the setup script is such a flexible tool, you can override many file group and component properties at run time. You also have complete control over how you present component and setup type choices to your end user. And most important, you can control which components are selected and, therefore, which files are transferred when you call the ComponentMoveData function in your script.

Organize your application files

To organize file transfer from the developer's perspective, InstallShield lets you group your files into file groups according to their type and purpose. (In no case do your end users concern themselves with files or file groups.) In the InstallShield IDE's File Groups pane you can set many installation properties that relate to the properties of the files themselves, such as the files' language, whether they are self-registering, should be compressed, and so on.

To make file transfer more understandable for your end user, InstallShield then allows you to group those file groups into components and to organize those components into one or more setup types. In the InstallShield IDE's Components pane you can set many installation properties that relate to how the component should be installed, such as where to install the component, whether to overwrite existing files, and so on.

You can offer your end user a choice of setup types (common options are Typical, Compact, and Custom). A Custom setup type means that your end user will have the option of selecting which visible components to install. You must always define at least one setup type for your project in the InstallShield IDE.

When you build your setup with the Media Build Wizard, InstallShield creates a file media library that contains all your data files and all the information that you entered into the InstallShield IDE about their file group, component, and setup type properties.

Transferring files

You present setup type and component selections to your end user using certain Sd and built-in dialog box functions. When your end user selects a predefined setup type, all components associate with that setup type are selected.

It is recommended that you offer your end user a choice of setup types. In order for the component dialogs to correctly display the component choices that you defined in the IDE, there must already be a default setup type selected. If you do not display a setup type dialog to let your end user select the setup type, then you must call the ComponentSetupTypeSet function to set a default setup type *before* displaying a component dialog.

You do the work of actually installing files in your setup script with the InstallScript component functions. The component functions are very flexible and have complete knowledge of file group, component, and setup type properties. The component functions let you

- prompt your end user for setup type and component selections
- transfer files to your end user's hard drive
- get and set component properties at run time
- create components at run time—that is, during setup
- check for necessary disk space
- enumerate file transfer errors
- verify a password, and much more

InstallShield
www.installshield.com

Script-created component set vs. file media library

You can create components at run time—that is, during setup—by calling the ComponentAddItem function in your setup script. These script-created components reside only in memory and have no direct connection to a file media library. And unlike the information stored in a file media library, script-created components are not (and cannot be) directly associated with file groups or a setup type.

However, script-created components do provide a convenient way of displaying component-like options for the end user. After the end user makes component selections in component dialogs, you can test the script-created components' selection status and use the result as the basis for carrying out some action. For example, you might want to install files with XCopyFile or VerUpdateFile, select components in the file media library, or create or edit a file.

To create script-created components, call the ComponentAddItem function. Then, set and access properties for the script-created component using the InstallScript component functions much as you would for components in a file media library (exceptions noted below).

Script-created components are often referred to collectively as a "script-created component set." The reason is that they are often handled just like file media library components by the component functions. That is, you must treat all components as a set when you pass their "media name" to the component functions. (You created the media name—in the szMedia parameter—when you first called ComponentAddItem. Use this same value when you create components and subcomponents as part of the same "script-created component set" and when you refer to existing script-created components in your script.)

Since the two types of components are so different, you call some of the component functions differently depending on whether you are working with a script-created component or a component in a file media library.

Some options in these functions work exclusively on either a component in a file media library or on a script-created component:

ComponentGetData

ComponentSetData

These functions work only with components in a file media library, but not with script-created components:

ComponentFileEnum

ComponentFileInfo

ComponentFilterLanguage

ComponentFilterOS

ComponentSetTarget

ComponentSetupTypeEnum

ComponentSetupTypeGetData

ComponentSetupTypeSet

ComponentValidate

And this function works exclusively with script-created components:

ComponentAddItem

{button ,AL(`The InstallScript component functions',0,`',`')} See also

InstallShield
www.installshield.com

The InstallScript component functions

InstallShield provides the following functions for handling components in the setup script:

Most of the InstallScript component functions require you to pass a media library name. You can use the MEDIA system variable to specify the default file media library name. If you are calling these functions to handle a script-created component set, use the same value that you used for the szMedia parameter to ComponentAddItem when you first created the component set.

ComponentAddItem
Adds a component to a set of script-created components.

ComponentCompareSizeRequired
Determines if enough disk space exists for all components.

ComponentError
Enumerates errors returned from component functions

ComponentFileEnum
Enumerates component details.

ComponentFileInfo
Retrieves all available information about a file.

ComponentFilterLanguage
Enables and disables filtering based on language—that is, whether or not language-dependent files should be installed.

ComponentFilterOS
Enables and disables filtering based on operating system (OS)—that is, whether or not OS-dependent files should be installed.

ComponentGetData
Retrieves all available information from the file media library or component.

ComponentGetItemSize
Determines the size in bytes of a component.

ComponentIsItemSelected
Determines whether a specified component is selected.

ComponentListItems
Enumerates all top-level components if a null string is entered in the szComponent parameter, or all subcomponents of the component specified in szComponent.

ComponentMoveData
Transfers and decompresses files in selected components.

ComponentSelectItem
Sets a component's selections status to either selected or deselected.

ComponentSetData
Sets properties for the specified component.

ComponentSetTarget
Specifies an alternate target folder for the component files.

ComponentSetupTypeEnum
Enumerates all setup types associated with the specified file media library.

ComponentSetupTypeGetData
Retrieves data associated with a specified setup type.

ComponentSetupTypeSet
Selects all components associated with the specified setup type.

ComponentTotalSize

Calculates the total size, in bytes, of selected components and subcomponents.

ComponentValidate
Validates the password of the entire file media library or a specified component.

{button ,AL(`Let the end user select setup types and components;Specifying components and subcomponents in function calls',0,`',`')} See also

InstallShield
www.installshield.com

The minimum function calls necessary to install component selections

The recommended way to install files is to use the InstallScript component functions on the file media library.

If you use the setup script generated by the Project Wizard and chose the Setup Type and Custom Setup options in the Project Wizard - Choose Dialogs panel, these function calls are already made for you.

Get the user's setup type selections

Call a setup type dialog box function, such as SdSetupTypeEx, to display the setup type selections you defined for your file media library. After the end user presses the next key, the SdSetupTypeEx function returns the name of the chosen setup type in the svSetupType parameter.

Assuming the end user selected a Typical or Compact setup type, you can proceed to call the ComponentMoveData function to install the files associated with that setup type.

If the end user selects a Custom setup type, you should display a component dialog that allows individual component selection.

Handle a Custom setup type selection

Call one of the InstallScript component selection dialog boxes, such as SdComponentDialog2. The SdComponentDialog2 function displays a dialog box that lets the end user select which components to install.

When you call the ComponentMoveData function, the selected components are installed.

Transferring files

Call the ComponentMoveData function to install the files that the user selected.

{button ,AL(`The InstallScript component functions;Getting setup type selection from the end user',0,`',`')} See also

The file media library contains your application's files and all the information that you entered into the InstallShield IDE about their file group, component, and setup type properties.

The file media library is created when you run the Media Build Wizard on a project. It is named Data1.cab, by default. It also has a "media name," the default value of which is contained in the MEDIA system variable.

You can set and access information in your file media library using the InstallScript component functions.

Some InstallScript component functions are specifically reserved for use with script-created components.

InstallShield
www.installshield.com

Specifying components and subcomponents in function calls

Component is a general term that refers to a set of file groups and/or subcomponents. A subcomponent is nothing more than a component that is located below another component, similar to the relationship between a folder and a subfolder.

"Top-level components" are the highest components in the hierarchy. Top-level components are never referred to as subcomponents.

Some InstallScript component functions ask you to refer to a single component, while others ask you to refer to multiple components.

To refer to a single component, use the component's name. To refer to a subcomponent, use a path-like expression where the name of each component in the hierarchy leading to that component is separated by double backslashes. For example, to specify the subcomponent Tutorials under the top-level component Help Files, use the following expression in your setup script:

```
szComponent = "Help Files\\Tutorials";
```

To refer to the subcomponent CBT under Tutorials, use the following:

```
szComponent = "Help Files\\Tutorials\\CBT";
```

Some component and setup type dialog box functions, such as SdComponentMult, display multiple components and their subcomponents. In these cases, you refer to multiple components by specifying the component just above them in the hierarchy. If the components are top-level components, use a null string ("") to refer to them.

For example, if you passed a null string to the SdComponentMult function, in the Components window it would display all the top-level components in your file media library or all the top-level components in your script-created component set, depending on the value of the MEDIA system variable. All subcomponents would display in the Subcomponents window.

On the other hand, if you passed a single top-level component (Help Files, in the example above) to the SdComponentMult function, it would display its subcomponents (at the level of Tutorials, from above) in the Components window and the next level lower of subcomponents (at the level of CBT, from above) in the Subcomponents window.

InstallShield
www.installshield.com

Organize my files into file groups

Inserting individual file links

1. In the File Groups pane, double-click the desired file group folder. Its Links subfolder opens.
2. Highlight the Links folder. The File Group Links window opens to the right.
3. On the Insert menu select Links into File Groups. The Insert File Links to File Group dialog opens.
4. Select the files you want to include in the file group.
5. Click the Open button. The selected files appear in the Files Group Links window.

Inserting folders and subfolders as file links

1. In the File Groups pane, double-click the desired file group folder. Its Links subfolder open.
2. Highlight the Links folder. The File Group Links window opens to the right.
3. From the Tools menu select Launch Explorer.   Windows Explorer opens.
4. Drag and drop the desired folders and files into the File Group Links window.

When you insert a folder into the File Group Links window, all of its subfolders are automatically inserted as well.

Remember that you can also create your own folder structure under the File Group Links icon by right-clicking and selecting New Folder. When you install the file group's components by calling ComponentMoveData, the folder structure is preserved.   That is, the folders and files are installed in the Destination folder exactly as they appear in the IDE's File Groups pane.

InstallShield
www.installshield.com

Mark application files as language dependent

1. In the File Groups pane highlight the desired file group. Its properties window opens.
2. Double-click the Language(s) property. The Language page opens.
3. Select the specific language of the file group. If the files belong to more than one language, press the Ctrl key and highlight each language.
4. Click OK.

When you build your setup with the Media Build Wizard, you can specify which languages you want to include in your built setup. For any given language that you choose, its language-dependent file groups are included in the completed setup's file media library. But if you have not purchased support for the language that your file groups are associated with, those file groups will not be included in your file media library. (For more information, see the note in Overview: Localization.)

Call the ComponentFilterLanguage function in your setup script to exclude any files that you do not want installed.

{button ,AL(`Filter language-dependent files based on the target systems language',0,`',`')} See also

Associate file groups with a component

1. In the Components pane highlight the desired component. Its properties window opens.
3. Double-click the Included File Groups property. The Included File Groups page opens. All file groups associated with the component are listed in the File Group Name list box.
4. To add a file group to this list, click the Add button. The Add File Group dialog opens.
5. In the Add File Group dialog, highlight the name of an existing file group, or enter the name of a new file group.

   Entering the name of an undefined file group in the File Groups edit box does not automatically create the file group. You are responsible for creating the named file group.

6. Click OK. The file group you chose is added to the Included File Groups page.
7. Click OK.

You can place a file group or a subcomponent under any component. If you have subcomponents under a component, you should not place a file group directly under that component. If all of a component's subcomponents are deselected, then the component is automatically deselected. Therefore, no files associated with a deselected component will be installed.

Installing self-registering files

InstallShield provides special functionality for installing self-registering files. The preferred method for installing self-registering files is to attach the Self Registered property to all the files in a file group, as described below. You can also install self-registering files by calling XCopyFile or VerUpdateFile with the SELFREGISTER option.

Organize all your self-registering files into unique file groups. Since a file group's properties apply to all its files, InstallShield will attempt to register every file in a file group that you specify as "Self Registered." Therefore, InstallShield will return an error if you include any non-self-registering files in a self-registering file group.

InstallShield offers two methods for installing self-registering files:

{button ,JI(`GETRES.HLP>(w95sec)',`The_batch_method')} The batch method (recommended)

{button ,JI(`GETRES.HLP>(w95sec)',`The_non_batch_method')} The non-batch method

For a comparison of differences, see The batch method vs. the non-batch method.

InstallShield takes care of registering the files in this file group during setup (or when the system reboots if the files are in use—be sure to handle locked files appropriately). unInstallShield unregisters the files if they are to be removed.

Tips for installing self-registering files

Following are some further guidelines to keep in mind when installing self-registering files:

- Make your function calls accurately with respect to calling ComponentMoveData to transfer the files. That is, you must call Enable before transferring the files, and you must call Do after the files are transferred.
- When you call the Do function to carry out batch self-registration, SELFREGISTERBATCH must be enabled. Otherwise, Do will return FALSE.
- Once Do is called, the internal list of queued, self-registering files gets reset, regardless of the success or failure of calling Do.
- If Do is called to perform self-registration but no files are queued, the function will return TRUE.
- If a self-registering DLL needs other DLLs to operate, the dependent DLLs must be in the path of the self-registering DLL before you call the Do function, or the self-registration will fail. InstallShield will change the current folder to the location where the DLL resides, thus making sure that the folder is in the path. By putting the dependent DLLs in the same location as the self-registering DLL, you can make sure that self registration succeeds.

  Also note that if you have DLLs that are needed by your self-registering DLLs but are not in the same path, the self-registration will fail.
- If you want to install files using the non-batch method after you have installed files using the batch method, you must first call Disable(SELFREGISTERBATCH).

In order for OLE applications to communicate, each OLE server, control, or container must place certain information in the Windows registry (32-bit Windows) or registration database (16-bit). Entering this information in the registry is referred to as registering the file or application.

Self-registering DLLs must contain two functions, DllRegisterServer and DllUnregisterServer, which are included and exported in each file that supports self-registration. In order for an .exe file to be an "automation server," it must support the command line argument /REGSERVER for registration purposes and /UNREGSERVER for unregistration.

When a file's DllRegisterServer function is called, it adds or updates registry information for all the classes implemented by the DLL, allowing OLE applications to see it. When the DLL's DllUnregisterServer function is called at uninstallation time, the DLL's information is removed from the registry.

InstallShield calls the necessary functions or passes the necessary command line arguments to self-registering files when you select the Self-Registered option for a file group, or when you call XCopyFile or VerUpdateFile with the SELFREGISTER option.

InstallShield
www.installshield.com

The batch method

1. In the File Groups pane, highlight the file group folder for your self-registering files. Its properties sheet opens to the right.
2. Double-click the Self-Registered field. The Self-Registered page opens.
3. Select the "Yes, all the files in this file group are self-registering" radio button.
4. Double click the Shared field. The Shared page opens.
5. Select the "Yes, the files are shared system files" radio button. InstallShield will register the files only if you select *both* the Self-Registered and Shared options.
6. Open up your setup script (Setup.rul found in the Scripts pane).
7. Before you transfer files by calling ComponentMoveData, call the Enable function with the SELFREGISTERBATCH option. All of the self-registering files are placed in an internal queue—that is, they are not immediately registered when they are installed.
8. After calling ComponentMoveData, call the Do function with the SELFREGISTRATIONPROCESS option. InstallShield registers all self-registering files when you call Do(SELFREGISTRATIONPROCESS).

After you call Do(SELFREGISTRATIONPROCESS), you can check for the success of self-registration. If Do fails for any reason, it will return -1. The names of the files that failed to self-register are stored in the ERRORFILENAME system variable, separated by a semicolon (;).

For example, the following code would register any self-registering files in the file media library.

```
// Enable batch method to queue self-registering files.
Enable ( SELFREGISTERBATCH );

// Install files.
nResult = ComponentMoveData( MEDIA, nDisk, 0 );

// Register the files, check for errors.
if Do ( SELFREGISTRATIONPROCESS ) < 0 then
   szMsg = "File(s) failed to self-register:  \n" + ERRORFILENAME;
   MessageBox (szMsg, WARNING);
endif;
```

{button ,AL(`The batch method vs. the non-batch method;The non-batch method',0,`',`')}   See also

The non-batch method

1. In the File Groups pane, highlight the file group folder for your self-registering files. Its properties sheet opens to the right.
2. Double-click the Self-Registered field. The Self-Registered page opens.
3. Select the "Yes, all the files in this file group are self-registering" radio button.
4. Double click the Shared field. The Shared page opens.
5. Select the "Yes, the files are shared system files" radio button. InstallShield will register the files only if you select *both* the Self-Registered and Shared options.

The non-batch method of installing self-registering files carries out the self-registration process as soon as the files are transferred when you call the ComponentMoveData function in your setup script.

{button ,AL(`The batch method;The batch method vs. the non-batch method',0,`',`')} See also

The batch method vs. the non-batch method

A limitation of the non-batch method is that if you are transferring DLLs that have "dependent" DLLs (that is, that require other DLLs to be installed first), you must install them in the correct order, or they will not self-register. When the files are uninstalled, they will be removed in the opposite order in which they were installed, which means that dependent DLLs will not unregister properly.

If you use the batch method, all files that are transferred with the Self-Registered file group option or the SELFREGISTER constant will install and uninstall correctly, regardless of the order in which they were installed on the user's system.

The other limitation of the non-batch method involves transferring all the files in the file media library at once with the ComponentMoveData function. If the self-registration process fails, then ComponentMoveData will also fail and return an error. However, if you use the batch method, the files are not registered until you call Do(SELFREGISTRATIONPROCESS). Therefore, file transfer will perform regardless of the success of the self-registration process, and you can check for the success of self-registration more easily.

InstallShield
www.installshield.com

Installing self-registering files with XCopyFile and VerUpdateFile

InstallShield provides special functionality for installing self-registering files. The preferred method for installing self-registering files is to attach the Self Registered property to all of the files in a file group. For more information, see Installing self-registering files.

You can also install self-registering files by calling XCopyFile or VerUpdateFile with the SELFREGISTER option. Always use SELFREGISTER together with the SHAREDFILE option, combining them with the bitwise OR operator (|). Consider the following call to XCopyFile:

```
TARGETDIR = WINSYSDIR;
XCopyFile ("App32res.dll", "App32res.dll", SELFREGISTER | SHAREDFILE);
```

The above code transfers App32res.dll from the installation source folder (SRCDIR) to the Windows\System folder. InstallShield immediately registers the file. unInstallShield will unregister the file if it is to be removed.

You can specify which folder you would like to reside App32res.dll in so that you can access it during setup. (Remember that XCopyFile and VerUpdateFile use the SRCDIR system variable as the source path.) Simply go to the Setup Files pane in the InstallShield IDE, highlight the desired folder under Advanced Files, and select File into Setup Files from the Insert menu.

The batch method

If you first call Enable(SELFREGISTERBATCH) in your setup script, then the SELFREGISTER option causes XCopyFile and VerUpdateFile to transfer the files and queue them up for later registration. These files (along with any self-registering files in the file media library that were transferred with the ComponentMoveData function) are registered once you call Do (SELFREGISTRATIONPROCESS).

This process is known as the "batch method" of installing self-registering files. The batch method is recommended for these reasons.

Tips for installing self-registering files

Following are some further guidelines to keep in mind when installing self-registering files:

- Make your function calls accurately with respect to calling XCopyFile, VerUpdateFile, and ComponentMoveData to transfer files. That is, you must call Enable before transferring the files, and you must call Do after the files are transferred.
- When you call the Do function to carry out batch self-registration, SELFREGISTERBATCH must be enabled. Otherwise, Do will return FALSE.
- Once Do is called, the internal list of queued, self-registering files gets reset, regardless of the success or failure of calling Do.
- You can call Do(SELFREGISTRATIONPROCESS) again to register another batch of self-registering files, after further calls to XCopyFile or VerUpdateFile. (You can only call ComponentMoveData once in a setup script.)
- If Do is called to perform self-registration but no files are queued, the function will return TRUE.
- If a self-registering DLL needs other DLLs to operate, the dependent DLLs must be in the path of the self-registering DLL before you call the Do function, or the self-registration will fail. InstallShield will change the current folder to the location where the DLL resides, thus making sure that the folder is in the path. By putting the dependent DLLs in the same location as the self-registering DLL, you can make sure that self registration succeeds.

  Also note that if you have DLLs that are needed by your self-registering DLLs but are not in the same path, the self-registration will fail.
- If you want to install files using the non-batch method after you have installed files using the batch method, you must first call Disable(SELFREGISTERBATCH).

{button ,AL(`Installing locked files with XCopyFile and VerUpdateFile;Installing shared files with XCopyFile and VerUpdateFile',0,`',`')} See also

InstallShield
www.installshield.com

Installing shared files

InstallShield provides special functionality for installing shared files. The preferred way to install shared files is to attach the Shared property to all of the files in a file group, as described below. However, you can also install shared files by calling XCopyFile or VerUpdateFile with the SHAREDFILE option.

A shared file is a file that can be used by more than one application. Many applications install shared .dll and .exe files in the Windows or Windows\System folder. These files are often considered Windows 95 or Windows NT core components. InstallShield also has special functionality for installing core component files. It is recommended that you always install shared files as core components (see below).

No specific guidelines exist for installing shared files on 16-bit platforms. In fact, there is no way to determine how many applications use a given shared file, since multiple copies of shared files are often installed in several places on a system and no registry reference count exists for shared files. This means that if you remove shared files when you uninstall your application, you run the risk of removing files needed by another applications. InstallShield has a straightforward solution: unInstallShield never removes shared files from 16-bit platforms.

Under 32-bit Windows, shared files can be handled more effectively. When a shared file is installed on a system, its registry reference count must be updated. InstallShield automates the handling of shared files, protecting you from the complexities involved.

Organize all your shared files into unique file groups. Since a file group's properties apply to all its files, InstallShield will treat all files in a "Shared" file group as shared files.

InstallShield offers two methods for installing shared files:

{button ,JI(`GETRES.HLP>(w95sec)',`Install_shared_files_as_core_components')} Install shared files as core components (recommended)

{button ,JI(`GETRES.HLP>(w95sec)',`Install_files_as_shared_files')} Install files as shared files

The reference count is an entry under the SharedDLLs registry key (on 32-bit platforms only) that is used to record the number of applications that use a shared file. For example, the registry entry for Ole32pro.dll might look like this:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current Version\SharedDLLs\
   C:\Win95\System\Ole32pro.dll=3
```

When you install a file as a shared file, InstallShield handles the reference count based on this algorithm (assuming you haven't enabled core component handling):

- If a shared file exists and has a reference count that is *not* set to zero, the reference count is incremented by one.
- If a shared file exists and has a reference count that *is* set to zero, the reference count is set to two.
- If a shared file already exists in the target folder but it has no reference count, one is created and set to two.
- If a shared file does not exist in the target folder and there is no reference count in the registry, one is created and set to one.

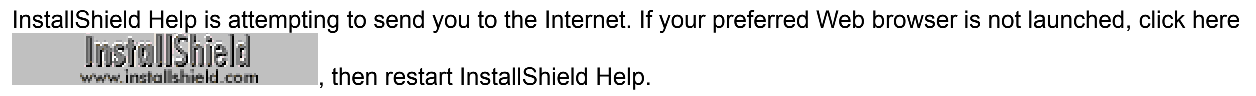
If logging was enabled during file transfer (as it is by default), unInstallShield decrements the reference counts before it removes shared files. If unInstallShield decrements a reference count to zero, a dialog will display asking the user if the shared file or all shared files should be removed.

"Core components" are considered to be those files that ship with Windows 95 or Windows NT 4.0, in addition to files that are added by installations such as ODBC, DAO, DirectX.

Currently, in order to obtain the "Designed for Microsoft Windows NT and Windows 95" logo certification, your application must adhere to the following installation and uninstallation requirements:

- Core components must not be refcounted in the registry.
- Core components must never be uninstalled.
- All files copied to the hard disk that are not shared must be removed when the application is uninstalled.
- All registry entries, except keys that may be shared, must be removed when the application is uninstalled.
- All files that are installed as shared files must be accurately refcounted during installation and uninstallation.
- If a component that will not be removed has a reference count of 0 (zero), the reference count must be left at 0.
- ODBC and DAO components must be registered and refcounted, but must never be removed.

For more information, visit http://www.microsoft.com/windows/thirdparty/winlogo.htm.

InstallShield Help is attempting to send you to the Internet. If your preferred Web browser is not launched, click here 

, then restart InstallShield Help.

Install files as shared files

1. In the File Groups pane, highlight the file group folder that contains only shared files. Its properties sheet opens to the right.
2. Double-click the Shared field. The Shared page opens.
3. Select the "Yes, the files are shared system files" radio button.

InstallShield treats all shared files as potentially locked files. To find out more about handling locked files, refer to Installing locked (in-use) files.

When you transfer these files by calling the ComponentMoveData function, InstallShield will install all files in this file group as shared files. However, if you have enabled core component handling, InstallShield sets the reference count as prescribed in CoreComp.ini.

Install shared files as core components

It is highly recommended that you enable core component handling in your script. This will prevent you from uninstalling core component files such as Mfc30.dll and Mfc40.dll. InstallShield handles core components differently than shared files, but you must follow the steps below to ensure successful installation and uninstallation. If core component handling is not enabled and if the files are not installed in the Windows\System folder, then these files will be installed as shared files. But it is recommended that you always install shared files as core components.

1. In the File Groups pane, highlight the file group folder that contains only shared files. Its properties sheet opens to the right.
2. Double-click the Shared field. The Shared page opens.
3. Select the "Yes, the files are shared system files" radio button.

InstallShield treats all shared files as potentially locked files. To find out more about handling locked files, refer to Installing locked (in-use) files.

4. In the Components pane, highlight the folder that contains the file group with the shared files. Its properties sheet opens to the right.
5. Double-click the Destination field. The Destination page opens.
6. Select Folders on Destination System\Windows Operating System\Windows System Folder. Click OK. <WINSYSDIR> is inserted as the value for the Destination field.

InstallShield will handle as core components only those shared files that are installed in the Windows\System folder.

7. Open up your setup script (Setup.rul found in the Scripts pane).
8. Before you transfer files by calling ComponentMoveData, call the Enable function with the CORECOMPONENTHANDLING option. (Core component handling is disabled by default.) It is very important that you check the return value from the Enable function to see if core component initialization succeeded. You may not want to proceed with the installation if Enable failed.

During file transfer, InstallShield will install all shared files in the Windows\System folder. As the files are installed, InstallShield logs them for uninstallation and makes reference count changes as prescribed in CoreComp.ini.

CoreComp.ini contains instructions for installing almost 900 core component files. The values in CoreComp.ini are based upon Microsoft core component requirements. InstallShield does not recommend modifying CoreComp.ini. But if you must make changes specific to your setup, refer to CoreComp.ini.

Note that this method for installing core components only applies to Windows 95 and Windows NT 4.0. It will have no effect under Windows NT 3.51, Win32s, or 16-bit Windows.

InstallShield www.installshield.com

Installing shared files with XCopyFile and VerUpdateFile

InstallShield provides special functionality for installing shared files. The preferred way to install shared files is to attach the Shared property to all of the files in a file group. For more information about this method and about how InstallShield handles installing shared files in general, see Installing shared files.

You can also install shared files by calling XCopyFile or VerUpdateFile with the SHAREDFILE option. (You cannot combine the SHAREDFILE and LOCKEDFILE options.) Assuming that you haven't enable core component handling (see below), InstallShield updates the reference counts for these shared files.

Note that InstallShield always treats shared files as potentially locked files. If you are transferring files with the SHAREDFILE option, then you must also handle these files as potentially locked. For more information, see Installing locked files with XCopyFile and VerUpdateFile.

Consider the following call to VerUpdateFile:

```
VerUpdateFile(WINSYSDIR ^ "Kernel32.dll",
              SHAREDFILE | VER_UPDATE_COND,
              svInstalledFilePath);
```

The above code transfers Kernel32.dll from the installation source folder (SRCDIR) to the Windows\System folder. InstallShield automatically increments the reference count as appropriate. unInstallShield will decrement the reference count as appropriate.

You can specify which folder you would like Kernel32.dll to reside in so that you can access it during setup. (Remember that XCopyFile and VerUpdateFile use the SRCDIR system variable as the source path.) Simply go to the Setup Files pane in the InstallShield IDE, highlight the desired folder under Advanced Files, and select File into Setup Files from the Insert menu.

Installing core components

InstallShield has further safeguards for installing shared files as core components. It is highly recommended that you enable core component handling in your script. This process will prevent you from uninstalling core component files such as Mfc30.dll and Mfc40.dll. It is recommended that you always install shared files as core components.

Before you transfer files by calling XCopyFile, VerUpdateFile, or ComponentMoveData, call the Enable function with the CORECOMPONENTHANDLING option. (Core component handling is disabled by default.) It is very important that you check the return value from the Enable function to see if core component initialization succeeded. You may not want to proceed with the installation if Enable failed.

Then, call XCopyFile or VerUpdateFile with the SHAREDFILE option to install your core component files.

Make sure that TARGETDIR is pointing to the Windows\System folder (WINSYSDIR) before you call XCopyFile. InstallShield will handle as core components only those shared files that are installed in the Windows\System folder.

As the files are installed, InstallShield logs them for uninstallation and makes reference count changes as prescribed in CoreComp.ini.

CoreComp.ini contains instructions for installing almost 900 core component files. The values in CoreComp.ini are based upon Microsoft core component requirements. InstallShield does not recommend modifying CoreComp.ini. But if you must make changes specific to your setup, refer to CoreComp.ini.

Note that this method for installing core components only applies to Windows 95 and Windows NT 4.0. It will have no effect under Windows NT 3.51, Win32s, or 16-bit Windows.

CoreComp.ini

CoreComp.ini is an initialization file that InstallShield uses to determine how to handle installing and uninstalling core component files. By default, CoreComp.ini is included with your setup in _sys1.cab. InstallShield does not recommend modifying CoreComp.ini. It will be updated regularly, available with newer product releases and at the InstallShield Web site.

CoreComp.ini is found in this folder: <InstallShield location>\Redistributable\Language Independent\Compressed Files\OS Independent. You should only modify a copy of CoreComp.ini, not the original file. Be sure to save the original in a convenient location. After completing changes to CoreComp.ini, you must copy it back to the original folder so that InstallShield can include the modified file with your new setup. Build your setup again with the Media Build Wizard so that InstallShield can include the newly revised CoreComp.ini file in _sys1.cab.

CoreComp.ini has a typical initialization file format. It has three sections: [Win40] contains only Windows NT 4.0 core components, [Win95] contains only Windows 95 core components, and [Win32] contains core components common to both operating systems.

The entries have the following format, *without* a space before or after the equal sign.

```
<file name>=<properties>
```

The <properties> value is in hexadecimal format. Each value has the meaning described in the table below.

Notice that CoreComp.ini already includes all of the files that you will need to install as core components, and each existing file already has a <properties> value associated with it. If you choose to add any other files to CoreComp.ini, you can select values from among those listed below. These values are mutually exclusive; you cannot combine ("OR") them. It is not recommended to modify the <properties> value of core components—inappropriate modification can prevent a setup from meeting Windows logo requirements.

Hex value	Installation/uninstallation properties
0x00000000	No registry entry is created for this file. This file is not logged for uninstallation, and is therefore never removed.
	Most of the core components fall into this category. ODBC and DAO DLLs fall into this category.
0x00000001	A registry entry is created, if one does not already exist, and the file is given a reference count of 0 (zero). If the entry already exists, it is not refcounted. The file is not logged for uninstallation, and is therefore never removed.
0x00000002	A registry entry for this file is created and refcounted. When the product is uninstalled, the reference count is decremented, but the file is not removed even if the reference count becomes 0 (zero).
0x0000004	A registry entry is created and refcounted. When the reference count falls to 0 (zero), this file is removed.
	Shared files typically fall into this category.

Installing locked (in-use) files

InstallShield provides special functionality for installing potentially locked files. The preferred way to install locked files is to attach the Potentially Locked property to all of the files in a file group, as described below. However, you can also install locked files by calling XCopyFile or VerUpdateFile with the LOCKEDFILE option.

A file that is currently in use by the operating system is known as a locked file. Since a setup cannot update locked files until the system reboots, you have to tell InstallShield which files you want to handle as potentially locked files and make certain function calls to ensure that those files that were locked can be updated.

Note that InstallShield always treats shared files as potentially locked files. If the files in a file group have the Shared option, then you must also handle these files as potentially locked. For more information, see Installing shared files.

Identify those files that may be locked during setup

1. In the File Groups pane, highlight the file group folder that contains only potentially locked files. (Remember that a file group's properties apply to each of its files.) The file group's properties sheet opens to the right.
2. Double-click the Potentially Locked field. The Potentially Locked page opens.
3. Select the "Yes, the files in this group may be locked" radio button.

Commit the locked files for updating and/or reboot the system

If you use the setup script generated by the Project Wizard, your script already has the necessary call to CommitSharedFiles to ensure that any locked files are updated the next time the system is restarted.

If it cannot update any files that are transferred as shared or potentially locked files because they are currently in use, InstallShield records these files and sets the BATCH_INSTALL system variable to TRUE. You can conditionally commit the locked files for updating and/or reboot the system based on the value of BATCH_INSTALL (see the script fragment, below).

You have the following options for committing locked files and/or rebooting the system:

- Call the CommitSharedFiles function to commit the locked files for updating the next time the system restarts. CommitSharedFiles should be executed only once in a setup script. However, if you are launching multiple scripts using DoInstall, then each individual script can call CommitSharedFiles once.
- Call the SdFinishReboot function to restart Windows or the system. This function automatically commits locked files for later updating.
- Call the RebootDialog function to restart Windows or the system. If the end user chooses to restart Windows, this function automatically commits locked files for later updating, so there is no need to call CommitSharedFiles. However, if the end user chooses to start Windows later, you must call CommitSharedFiles before restarting Windows or the system.

For example, after you have transferred files with the ComponentMoveData function and just before you wish to end the setup, you can use the following code to commit the locked files and reboot the system, if the end user so chooses:

```
if (BATCH_INSTALL) then
   SdFinishReboot(szTitle, szMsg1, nDefOption, szMsg2,
                  nReserved);
endif;
```

Installing locked files with XCopyFile and VerUpdateFile

InstallShield provides special functionality for installing locked files. The preferred way to install locked files is to attach the Potentially Locked property to all of the files in a file group. For more information about this method, see Installing locked (in-use) files.

A file that is currently in use by the operating system is known as a locked file. Since a setup cannot update locked files until the system reboots, you have to tell InstallShield which files you want to handle as potentially locked files and make certain function calls to ensure that those files that were locked can be updated.

You can check if a file is locked by calling the Is function with the FILE_LOCKED option.

Identify those files that may be locked during setup

If you suspect that a file may be locked during setup, use the LOCKEDFILE option in your call to XCopyFile or VerUpdateFile. Do not use the LOCKEDFILE and SHAREDFILE file options together.

Note that InstallShield always treats shared files as potentially locked files. If you are transferring files with the SHAREDFILE option, then you must also handle these files as potentially locked. For more information, see Installing shared files with XCopyFile and VerUpdateFile.

Commit the locked files for updating and/or reboot the system

If you use the setup script generated by the Project Wizard, your script already has the necessary call to CommitSharedFiles to ensure that any locked files are updated the next time the system is restarted.

If it cannot update any files that are transferred as shared or potentially locked files because they are currently in use, InstallShield records these files and sets the BATCH_INSTALL system variable to TRUE. You can conditionally commit the locked files for updating and/or reboot the system based on the value of BATCH_INSTALL (see the script fragment, below).

You have the following options for committing locked files and/or rebooting the system:

- Call the CommitSharedFiles function to commit the locked files for updating the next time the system restarts. CommitSharedFiles should be executed only once in a setup script. However, if you are launching multiple scripts using DoInstall, then each individual script can call CommitSharedFiles once.
- Call the SdFinishReboot function to restart Windows or the system. This function automatically commits locked files for later updating.
- Call the RebootDialog function to restart Windows or the system. If the end user chooses to restart Windows, this function automatically commits locked files for later updating, so there is no need to call CommitSharedFiles. However, if the end user chooses to start Windows later, you must call CommitSharedFiles before restarting Windows or the system.

For example, after you have transferred files with XCopyFile, VerUpdateFile, or ComponentMoveData and just before you wish to end the setup, you can use the following code to commit the locked files and reboot the system, if the end user so chooses:

```
if (BATCH_INSTALL) then
   SdFinishReboot(szTitle, szMsg1, nDefOption, szMsg2,
                  nReserved);
endif;
```

InstallShield
www.installshield.com

Leave files in a file group uncompressed

By default, InstallShield compresses all files in a file group. There may be certain files, such as .avi files, which are already tightly compressed, or files residing in a folder on a CD-ROM, that you might not want compressed among your application files. To leave all the files in a given file group uncompressed, follow these steps:

1. In the File Groups pane highlight the desired file group. Its properties window opens to the right.
2. Double-click the Compressed property. The Compressed page opens.
3. Select "No, do not compress the files in this file group."

{button ,AL(`Leave all files uncompressed on my CD-ROM;Modify an existing file group',0,`',`')}   See also

Associate components with a setup type

1. In the project workspace, click the tab of the Setup Types pane.
2. Double-click a setup type icon. The Setup Types page opens to the right.
3. In the Setup Types page check marks appear in the folder icons of components associated with the setup type. To associate a previously unassociated component (or vice versa), click its folder icon.

Create a file group

1. In the project workspace, click the tab of the File Groups pane.
2. Right-click in the File Groups pane. A popup menu appears.
3. From the popup menu, select New File Group.

InstallShield cannot handle project file paths with more than 260 characters (including the file name). The names of the following project elements are incorporated in project paths and file names:

- the project itself
- file groups
- media names
- files in the Scripts pane
- files inserted into the Setup Files pane

To help you keep within the 260-character limit, InstallShield warns you if you attempt to assign a name with more than 50 characters to a project, file group, or media name.

InstallShield
www.installshield.com

Delete a file from a file group

1. In the File Groups pane, click the plus sign (+) next to the file group containing the file you want to delete. The file group's Links icon is displayed.
2. Click the Links icon. The File Groups - <file group name>\Links dialog box opens.
3. In the File Groups - <file group name>\Links dialog box, right-click the file you want to delete. A popup menu appears.
4. From the popup menu, select Delete.

InstallShield
www.installshield.com

Delete a file group

1. In the File Groups pane, select the file group you want to delete.
2. Click the right mouse button. A popup menu appears.
3. From the popup menu, select Delete.

InstallShield
www.installshield.com

Disassociate a file group from a component

1. In the Components pane, click the component from which you want to disassociate a file group. The Components - <component name> window opens.
2. In the Components - <component name> window, double-click Included File Groups. The Included File Groups property page opens.
3. In the Included File Groups property page, select from the File Group Name list box the name of the file group you want to delete.
4. Click the Remove button.
5. Click OK.

Mark a component as critical, highly recommended, or standard

1. In the Components pane, select the desired component. Its properties sheet opens to the right.
2. Double-click the File Need field. The File Need page opens.
3. You have a choice of three options in the radio buttons on the File Need Page. Select one of the following options:

Value	File Need page text
CRITICAL	This component contains critical files and is crucial to the operation of the application.
HIGHLYRECOMMENDED	This component Is highly recommended. It provides useful services that enhance the usability of the application.
STANDARD	This component may be included or not included during setup.

Attaching the File Need property to a component does not cause a component to be selected or deselected. It merely attaches a property that you can access at run time—that is, during setup. Call the ComponentGetData function with the COMPONENT_FIELD_FILENEED option. You can then display a warning to your end user about deselecting "critical" components or a similar action based on the component's File Need property.

Components have other properties that determine whether a component is required or visible. For example, If a component is deselected by default, do not set its File Need property to CRITICAL or HIGHLYRECOMMENDED. Set it to STANDARD instead. Conversely, all CRITICAL and HIGHLYRECOMMENDED components should be selected by default.

In addition, if a component's Visible property is set to No, do not set its File Need property to CRITICAL or HIGHLYRECOMMENDED. Remember that you, are responsible for setting the selection status of invisible components.

InstallShield
www.installshield.com

Select the destination folder for a component

1. In the Components pane, select the desired component. Its properties sheet opens to the right.
2. Double-click the Destination field. The Destination page opens.
3. Select the folder on the target system in which you would like the files in this component to be installed.

If you select Windows Operating System, Program Files Folder, or one of their subfolders, the files will always be installed in one of those folders.

If you select General Application Destination, the files will be installed in the folder contained in the TARGETDIR system variable before your call to the ComponentMoveData function.

If you select Script-defined Folders, you must add a subfolder with the name of a variable in angle brackets (such as <szNewDir>). To assign a value to this variable in your setup script, you must call the ComponentSetTarget function.

Delete a component

1. In the Components pane, select the component you want to delete.
2. Click the right mouse button. A popup menu appears.
3. From the popup menu, select Delete.

InstallShield
www.installshield.com

Specify which folder I want uncompressed files to go in

1. In the Components pane, highlight the desired component. Its properties window opens to the right.
2. Double-click the CD-ROM Folder property. The CD-ROM Folder page opens.
3. Enter the name of a folder that you want InstallShield to create on the distribution media and place this component's files in.

{button ,AL(`Leave files in a file group uncompressed;Leave all files uncompressed on my CD-ROM',0,`',`')} See also

Create a component or subcomponent

First, open the Components pane for your project by clicking the Components tab.

Next, decide the relative level of placement for the new component. If you're creating a top-level component, highlight the '<project name>' Components icon. If you're creating a subcomponent, highlight the component under which you'd like it to appear.

Right-click anywhere in the Components pane and choose Component Wizard. Enter the name of your new component or subcomponent and click Next.
In the next panel you can set some of the properties for your new component. Decide whether this component will contain files that are shared system files. Then, decide whether you want the files transferred conditionally based on date or version. Finally, enter any (further) subcomponents of your new component, and click Finish.

The new component appears in the desired location. Notice its properties in the window to the right; you can now modify any of the component's properties and associate it with a setup type.

You can also create components and subcomponents at run time—that is, during setup—by calling the ComponentAddItem function.

InstallShield
www.installshield.com

Password-protect a component

1. In the Components pane, highlight the component you want to password-protect. Its properties window opens to the right.
2. Double-click the Password property. The Password dialog opens.
3. Enter the password with which you want to protect this component.

These instructions protect a single component from being accessed without the password you specified. To get the password from your end user and verify it, you have to display a dialog box and call the ComponentValidate function.

{button ,AL(`Get and validate a component or file media library password',0,`',`')}   See also

InstallShield
www.installshield.com

Get and validate a component or file media library's password

To get a password, call any InstallScript text input dialog box function, such as AskText. Then, pass the returned string to the ComponentValidate function. Halt script execution until the correct password is entered—that is, until ComponentValidate succeeds—so that all files can be transferred when you call ComponentMoveData later in the script.

The example below prompts for a password for the entire file media library, hence the null string passed to the second parameter of ComponentValidate. (Pass the component's name if you want to validate a password for an individual component.)   The end user has three opportunities to enter the correct password before setup terminates.

```
szQuestion = "Please enter the 8-digit code found " +
             "on the inside cover of this product's CD case.";
lResult = 1;
lCount  = 0;

// Prompt for password and attempt to validate it up to three times.
while (lCount < 3) && (lResult != 0)
   AskText (szQuestion, "", svResult);
   lResult = ComponentValidate (MEDIA, "", svResult);
   lCount = lCount + 1;
endwhile;

// Exit if the incorrect password is provided after 3 attempts.
// Otherwise, continue with the setup.
if lCount = 3 then
   MessageBox ("Sorry, incorrect password supplied.", SEVERE);
   abort;
else
   MessageBox ("Proceed with Setup.", INFORMATION);
endif;
```

{button ,AL(`Password-protect a component;Password-protect the entire media library',0,`',`')}   See also

InstallShield
www.installshield.com

Display a message for each component in the progress indicator

1. In the Components pane, highlight the folder for the desired component. Its Properties sheet opens.
2. Double-click the Status Text field to open the Status Text dialog.
3. Enter the desired message in the New Text field. (You can also enter a string identifier. To see a list of string identifiers or to add a new one, click the >> button.)
4. Click OK.

When that component is being installed during file transfer, the desired message will appear in the progress indicator.

{button ,AL(`Displaying the progress indicator',0,`',`')} See also

String identifiers are defined in the string tables stored in the InstallShield IDE's Resources pane. You can associate a string value and a comment with each string identifier. For each language-specific string table you have, you can associate a different string value and comment with a single string identifier.

When you include a language in your setup project, InstallShield creates a string table for that language and includes in it all existing string identifiers.

Many dialog boxes in the InstallShield IDE let you enter a string identifier for text that will be displayed during setup.

You can also use string identifiers in your setup script. They must be preceded by the at sign (@).

When you build your setup with the Media Build Wizard, all of the language-specific string tables are included in your setup.

InstallShield
www.installshield.com

Display a component's description in a component dialog

1. In the Components pane, highlight the folder for the desired component. Its properties window opens.
2. Double-click the Description field to open the Description page.
3. Enter the desired description in the New Text field. (You can also enter a string identifier. To see a list of string identifiers or to add a new one, click the >> button.)
4. Click OK.

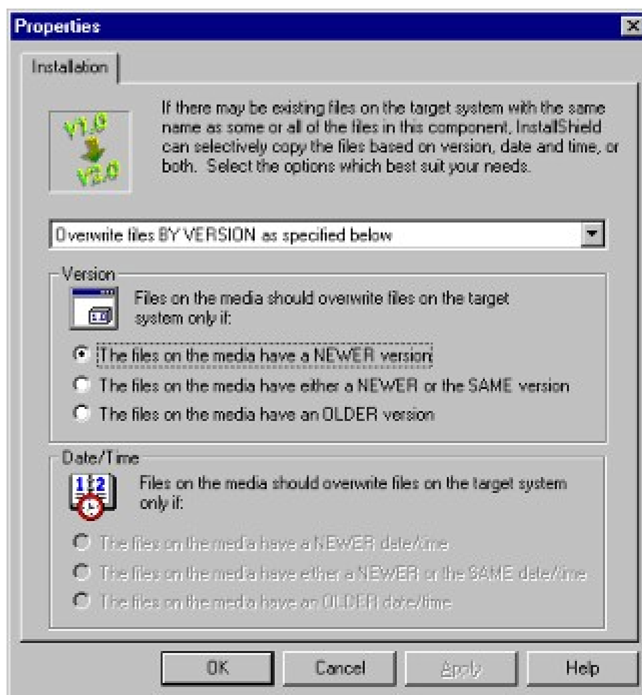
When that component's name is displayed in a component dialog box, the desired description will also appear.

{button ,AL(`Let the end user select setup types and components',0,`',`')} See also

Overwrite a file conditionally, based on version or date

1. In the Components pane, highlight the folder for the component or subcomponent you wish to modify. Its properties sheet opens to the right.
2. Double-click the Overwrite field. The Overwrite page opens.
3. Select one of the choices from the pull down list. If you select one of the three choices that specify conditional overwriting based on version and/or date/time, you must also select the appropriate radio button in the Version and/or Date/Time list box to further specify overwrite options.
4. Click OK.

For example, let's suppose that you have a subcomponent called System DLLs consisting of files that you only want installed if they have a later version than the files that are currently on the target system. Fill out the Overwrite page as shown below:



Create a setup type

1. Right-click anywhere in the Setup Types pane. A popup menu appears.
2. From the popup menu, select New Setup Type.
3. Type the name of the new setup type and press Enter.

You can now associate components with your new setup type.

You can display a setup type other than Typical, Compact, or Custom only in the SdSetupTypeEx dialog box. If you add a new setup type or rename an existing one, then you must call this function in your setup script to display setup type choices for your end user.

Getting setup type selection from the end user

InstallShield provides a number of dialog boxes you can use to display setup types for your end user and then return those selections to the setup script.

You must always define at least one setup type for your project in the InstallShield IDE.

It is recommended that you offer your end user a choice of setup types. In order for the component dialogs to correctly display the component choices that you defined in the IDE, there must already be a default setup type selected. If you do not display one of the setup type dialogs to let your end user select the setup type, then you must call the ComponentSetupTypeSet function to set a default setup type *before* displaying a component dialog.

If you defined only the standard Typical, Compact, and Custom setup types for your file media library, the call the SetupType or SdSetupType function. These dialogs automatically display these setup types and return the end user's selection to your setup script. They also automate component selection—if the end user selects a setup type, then all the components under that setup type are selected.

However, if you defined setup types beyond Typical, Compact, and Custom, you must call the SdSetupTypeEx function to display all your setup types and return the end user's selection.

If you use the setup script generated by the Project Wizard and chose the Setup Type option in the Project Wizard - Choose Dialogs panel, setup type display and handling is already done for you.

Let's suppose you had created these setup types in the InstallShield IDE: Typical, Compact, and Custom. Call the SdSetupTypeEx function at the point in your script when you want to offer your end user a choice of setup types to install. The following code displays a dialog box with Typical as the default setup type selection:

```
szTitle     = "Choose Setup Type";
szMsg       = "Highlight the setup type you wish to install," +
              " then click Next.";
svSetupType = "Typical";
SdSetupTypeEx (szTitle, szMsg, "", svSetupType, 0);
```

Assuming that the end user selected a Typical or Compact setup type, you can proceed to call the ComponentMoveData function to install the files associated with that setup type. If the end user selected a Custom setup type, display a component dialog box at this point to allow component selection.

{button ,AL(`Let the end user select setup types and components;The minimum function calls necessary to install component selections',0,`',`')} See also

InstallShield
www.installshield.com

Display text in the Description field of SdSetupTypeEx

1. In the Setup Types pane, right-click on the name of any setup type. A popup menu appears.
2. Select Properties from the popup menu. The Setup Type dialog appears.
3. Enter the description you would like to see in the Description field.
4. Click OK.
5. Repeat for each component.
6. Build your setup with the Media Build Wizard.

When you display setup type options in the SdSetupTypeEx dialog box, these descriptions will appear in the Descriptions static text field when each setup types is highlighted.

InstallShield
www.installshield.com

Display a different name for a setup type in SdSetupTypeEx

1. In the Setup Types pane, right-click on the name of any setup type. A popup menu appears.
2. Select Properties from the popup menu. The Setup Type dialog appears.
3. Enter a new name for the setup type in the Display Name field.
4. Click OK.
5. Build your setup with the Media Build Wizard.

When you display setup type options in the SdSetupTypeEx dialog box, the Display Name will appear as the name for that particular component.

Keep in mind, though, that changing the Display Name does not change the actual name of the setup type. When referring to your setup type in your script, you must still use the name you assign in the InstallShield IDE.

Overview: Setup Script

InstallShield5 was designed so that you could do most of the work of creating your setup in a point-and-click environment. Notice that you can run a compiled setup script as soon as you've run the Project Wizard.

The setup script is the driving force of a setup. The script determines the order of setup events and controls most of its actions. If you need to modify your setup in a way that the InstallShield IDE does not provide, you will need to customize your setup script.

The setup script for your project is found in the InstallShield IDE's Scripts pane. Click on Setup.rul, and your setup script opens in the InstallShield Script Editor. The setup script is written in InstallScript, a simple but powerful programming language for creating setups.

Write the setup script

InstallScript is similar to the C programming language in many aspects. But regardless of your programming background, you can quickly learn to build or modify a setup script with InstallScript. If you're unfamiliar with InstallScript, take a few moments to read through the script generated by the Project Wizard or one of the scripts provided with the InstallShield template projects. For a comprehensive guide to InstallScript, see the InstallScript Language Reference.

You can create a setup script in a variety of ways:

{button ,JI(`GETRES.HLP>(w95sec)',`Modify_the_script_that_the_Project_Wizard_generates')} Modify the setup script that the Project Wizard generates for you.

{button ,JI(`GETRES.HLP>(w95sec)',`Create_a_script_from_scratch')} Create a setup script from scratch for a new project.

{button ,JI(`GETRES.HLP',`Modify_an_existing_script')} Modify an existing setup script—for example, from an InstallShield template or from a setup that your company already uses.

Compile the setup script

The Project Wizard automatically compiles the script it generates for you. If you change it or produce your own setup script from scratch, you can compile the script by selecting Compile from the Build menu.

You can debug your script at this point by selecting Debug Setup from the Build menu.

Test the setup

Once you have compiled your script, you will need to test it. The InstallShield engine executes your compiled setup script once you launch Setup.exe.

Since testing is always an ongoing process, every successful setup should pass several testing phases before distribution, including:

- Testing the setup directly from the IDE. You can run and debug your setup from the InstallShield IDE before you've even built your distribution media. Select Run Setup and/or Debug Setup from the Build menu.
- Testing the setup on your development platform by running Setup.exe in the Disk1 folder in the default location: My Installations\<project name>\Media\<media name>\Disk Images.
- Testing the setup from the distribution media on various systems running the target platform(s).

{button ,AL(`Compiling the setup script;Debugging the setup script',0,`',`')} See also

Modify the script that the Project Wizard generates

The Project Wizard generates a setup script based on your choices in the wizard's panels. Select Run Setup from the Build menu to see the script in action. Assuming that you've selected the correct options and that this basic script meets your setup's needs, you should not have to make many changes to the wizard-generated script. Take note of ways you would like to change the setup.

You cannot run the Project Wizard again to select items you wish to modify in your setup script. You must either close the current project and run the Project Wizard on a new project or modify the setup script.

Open the Setup.rul file found in the InstallShield IDE's Scripts pane. Take some time to familiarize yourself with the wizard-generated script. The script is well commented and suggests exactly where you can place modifications such as prototypes, registry-related function calls, variable declarations, and so forth.

Pay particular attention to the sequence of events in the script. This script relies heavily on labels and goto and if statements to control the execution. If you want to add your own function calls, follow the logic to decide where to place your additions within the wizard-generated script.

Notice the frequent use of user-defined functions. The wizard-generated script encapsulates discrete setup tasks by placing most of the script logic and built-in InstallScript functions outside of the program block within user-defined functions. Most likely, you'll need to modify lines in these user-defined functions to customize your script.

You'll also notice that most of the events in the setup script center around dialog boxes. You generally will want to keep your user as involved as possible in the setup. In one sense, your end user actually controls the setup script. InstallShield is flexible enough to let you prompt your user for most information at run time, and then let you handle those responses and use those values to perform the setup.

Make all desired changes to the wizard-generated script. There are also a couple tasks that you must accomplish to customize the setup script:

{button ,JI(`GETRES.HLP>(w95sec)',`Associate_string_identifiers_with_values')} Associate string identifiers with values

{button ,JI(`GETRES.HLP>(w95sec)',`Add_program_folders_and_items')} Add program folders and items

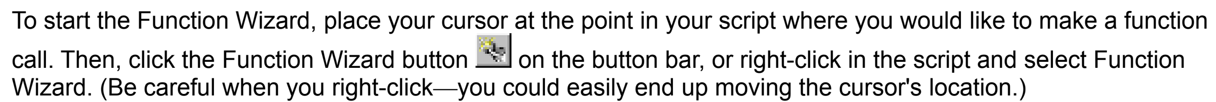
For more information on handling specific tasks in your setup script, refer to the appropriate subject headings under the Getting Results help file.

You can display a setup type other than Typical, Compact, or Custom only in the SdSetupTypeEx dialog box. If you add a new setup type or rename an existing one, then you must call this function in your setup script to display setup type choices for your end user. The Project Wizard-generated script instead displays the SetupType dialog box.

InstallShield
www.installshield.com

Using the Function Wizard

The Function Wizard automates the process of editing your setup script. The Function Wizard places a function call wherever you tell it to in your script. Note that it cannot edit an existing function call.

To start the Function Wizard, place your cursor at the point in your script where you would like to make a function call. Then, click the Function Wizard button  on the button bar, or right-click in the script and select Function Wizard. (Be careful when you right-click—you could easily end up moving the cursor's location.)

Step 1

The Function Wizard lets you select from an alphabetic listing of all functions or from listings of function categories. You can see the description for each function you highlight. And you can press the Help button to see the complete function description for any highlighted function.

Step 2

Fill in the arguments for each function. When you place the cursor in an edit field, the parameter's description appears at the bottom of the wizard's panel.

If there are predefined options available for a parameter, the edit field contains a pull down list box. Note that you can only select one option from a pull down list box, even when the options are not exclusive.

Often you can enter strings (or string identifiers) and numbers directly in to the edit fields. But pay careful attention to the purpose of the parameters. One clue is the InstallScript Hungarian notation in the variable name. For example, the nv- prefix denotes a value that is set or returned by a function.

You must declare all variable names that you use in all function calls. The Function Wizard does not declare them for you.

Click Finish and the function call is inserted into your setup script in the desired location. If you need further help, place the cursor in the function name and press the F1 key.

InstallShield
www.installshield.com

Associate string identifiers with values

You'll notice that many of the function calls in the Project Wizard-generated script have arguments that look like this, with the at sign (@) before a constant: @PRODUCT_NAME. For example, examine this function call:

```
InstallationInfo( @COMPANY_NAME, @PRODUCT_NAME,
                  @PRODUCT_VERSION, @PRODUCT_KEY );
```

These "string identifiers" take the place of string variables or literals. InstallShield replaces each string identifier with its string value at run time (that is, during setup).

For every string identifier used in the wizard-generated script, InstallShield places a string identifier in the string table for each supported language. Some of the values are filled in based on your entries in the Project Wizard's panels; others are default values that you must customize. You can easily modify the string values. Follow these steps:

1. Open up the InstallShield IDE's Resources pane.
2. Highlight String Table under '<project name>' Resources.
3. Highlight the desired language to open its language-specific string table.
4. Double-click a string identifier to edit its name, value, or comment. Right-click and select New to add a new string identifier.

You can add string identifiers directly from the script editor. Place your cursor anywhere in the script editor and right-click. Select String Entry. You can see a list of existing string identifiers and add or modify string table entries.

Since string identifiers are simply substituted with their values at run time, they behave slightly differently than string literals. A major difference is that escape sequences cannot be used in string values—"\t" would not appear as a tab in a dialog box, but as "\t". Also, when entering a path as the value of a string table entry, use single backslashes, not double backslashes (as you do when directly entering a path string in the script). For example, enter C:\MyApp\Program, not C:\\MyApp\\Program.

Also keep in mind how string identifiers are used in the script. For example, the Project Wizard-generated script uses the value of COMPANY_NAME as part of the default value of the destination folder. That folder could not be created if COMPANY_NAME contained an invalid character such as ? or *.

{button ,AL(`Create a string table for each supported language',0,`',`')} See also

Add program folders and items

The Project Wizard automatically creates the SetupFolders user-defined function to encapsulate all the steps involved in creating program folders and items for your application, associated applications, help files, and so on.

As you will notice, the SetupFolders function is just a framework for you to place function calls. You are responsible for making all the function calls to add the necessary program groups, folders, or items for your application. For more information, see The InstallScript functions for creating and modifying program folders and items.

Also, be aware that if you want to provide uninstallation functionality under the Program Manager shell, you must place an uninstaller icon in the application's program group. For more information, see How do I prepare my setup for uninstallation.

For example, if you wanted to add a shortcut for your application to the Windows 95 Start Programs menu, call the AddFolderIcon function as follows:

```
AddFolderIcon ("", // adds the icon to the Start Programs menu
               "My Application", // the name of the icon
               szCommandLine, // command line expression
               szDestPath, // working folder
               szCommandLine, // icon file
               0, // icon ordinal
               "", // shortcut key
               NULL); // run option
```

InstallShield
www.installshield.com

Create a script from scratch

You already have a completed application you want to install. Now you have to decide how you want to present your product to your customers and how you want to handle installing folders and files on an end user's system.

The best way to get ideas for creating a script is to look at other setups. Even if you choose to create your script from scratch, the template projects and Project Wizard-generated script may still prove quite helpful.

There is no single checklist or algorithm for determining your setup's unique needs. The steps below describe features that you may want to include in your setup program. This sequence is patterned after the logic of typical software setups. To learn more about how to accomplish each task in the setup script, refer to the individual topics in the Getting Results help file.

Sequence of operations in a typical setup script

1. Call the InstallationInfo function to specify product and company information.
2. Call DeinstallStart and RegDBSetItem to enable unInstallShield functionality.
3. Set up the per application paths key in the registry using RegDBSetItem.
4. Set up the screen and main window.
5. Welcome the user and get registration information.
6. Check whether the user's system meets the requirements for installing the application.
7. Get the destination location. Check if enough disk space is available.
8. Get setup type and component selection. Show settings/selections, allowing user to backtrack and change as desired.
9. Perform file transfer.
10. Make further registry changes as required.
11. Set up shortcuts and/or program folders and items (including the uninstallation icon when the shell is Program Manager).
12. Commit shared files and call reboot dialog. If necessary, terminate setup and reboot.
13. Offer user the opportunity to see Readme.htm or similar file.
14. Indicate that setup is complete and exit.

{button ,AL(`Modify the script that the Project Wizard generates;Compiling the setup script',0,`',`')} See also

Modify an existing script

If you're using a script created for InstallShield3, there are a number of changes to make to prepare the script for the enhanced InstallShield5 features. Please refer to Migrating from InstallShield3 in the Getting Started help file.

If you're using a project created with InstallShield5, open up the existing project by selecting File | Open. Select Run Setup from the Build menu to see the script in action. Take note of ways you would like to change the setup.

You cannot run the Project Wizard again to select items you wish to modify in the setup script. You must either close the current project and run the Project Wizard on a new project or modify the setup script.

Open the Setup.rul file found in the InstallShield IDE's Scripts pane. Take some time to familiarize yourself with the script.

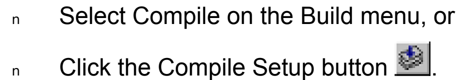
Pay particular attention to the sequence of events in the script. If you want to add your own function calls, follow the logic to decide where to place your additions within the setup script.

You'll probably notice that most of the events in the setup script center around dialog boxes. You generally will want to keep your user as involved as possible in the setup. Your end user ultimately controls the setup script. InstallShield is flexible enough to let you interact with your user, and then let you handle those responses and use those values to perform a very customized setup.

Make all desired changes to the script, compile it, and run the setup.

For more information on handling specific tasks in your setup script, refer to the appropriate subject headings under the Getting Results help file.

{button ,AL(`Create a script from scratch;Modify the script that the Project Wizard generates;Associate string identifiers with values;Compiling the setup script;Insert an existing script into my project',0,`',`')}   See also



Compiling the setup script

You must compile your setup script before you can run your setup. Your setup script must be named Setup.rul and reside in the default location: My Installations\<project name>\Script Files.

Remember that InstallShield searches only for a file named Setup.rul when compiling the setup script. You can include files with different names, but they must be included in Setup.rul with the #include preprocessor statement. Include files must also reside in the same folder as Setup.rul.

To compile your script

- Select Compile on the Build menu, or
- Click the Compile Setup button .

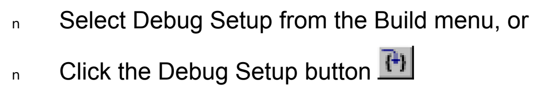
The compiler's status, including any error or warning messages, is displayed in the Compile output window. Double-click a compiler message to go to the line in your setup script where the error was found. For information about individual messages, see "Compiler messages" in the InstallScript Language Reference.

If your setup script compiled successfully, InstallShield creates Setup.ins (the object code that the InstallShield engine executes) and copies it to the Disk1 folder in the default location: My Installations\<project name>\Media\<media name>\Disk Images. Look in the Media pane to see which <media name> is the current default—the media name with a red flag on its icon—so you can be sure that InstallShield is copying Setup.ins to the correct disk images. Click on another media name to change the default and recompile the script, if running the setup does not reflect the most recent changes.

InstallShield automatically compiles your setup script once you've run the Project Wizard and every time you run your setup within the InstallShield IDE. However, InstallShield does not compile your setup script for you when you run the Media Build Wizard. Make sure that you compile your setup script beforehand.

You can set compiler options in the Compile tab under Settings on the Build menu.

{button ,AL(`Debugging the setup script',0,`',`')} See also



Debugging the setup script

You can easily solve problems with your setup using the InstallShield Visual Debugger. To launch your setup in debug mode:

- Select Debug Setup from the Build menu, or
- Click the Debug Setup button

For complete information, refer to Visual Debugger Help.

InstallShield
www.installshield.com

Create an include file

Right-click anywhere in the Scripts pane. Select New File from the popup menu. A new file with the default name of Extra*n*.rul (where *n* is a successive number) is added under <project name> Script Files. A blank script editor window opens to the right.

You can rename Extra*n*.rul by right-clicking its icon and selecting Rename from the popup menu. Include files typically have the .h file extension when they contain variable declarations, function prototypes, preprocessor statements, and so on. They typically have the .rul file extension when they contain function definitions.

Use the #include preprocessor statement to include .h files before the program block of your main setup script (Setup.rul) and to include .rul files after the program block.

{button ,AL(`Insert an existing script into my project',0,`',`')}   See also

Insert an existing script into my project

InstallShield searches only for a file named Setup.rul when compiling the setup script. You can include files with different names, but they must be included in Setup.rul with the #include preprocessor statement. To insert an existing script into your setup project, follow the instructions below:

1. Right-click anywhere in the Scripts pane. Select Insert Files from the popup menu.
2. Browse through the Insert Files into Script Files dialog box until you have highlighted the desired setup script and/or include file(s).
3. Click Open.

An icon for your existing script appears in the Scripts pane. Highlight the icon, and the script will appear in the script editor.

When you select a file for insertion, InstallShield copies the file and moves the copy to the default location: My Installations\<project name>\Script Files. After that, editing the file in an InstallShield script editor window makes changes to the copy, not to the original file. Changes to the original file will have no effect on your setup unless you re-insert the file.

InstallShield cannot handle project file paths with more than 260 characters (including the file name). The names of the following project elements are incorporated in project paths and file names:

- the project itself
- file groups
- media names
- files in the Scripts pane
- files inserted into the Setup Files pane

To help you keep within the 260-character limit, InstallShield warns you if you attempt to assign a name with more than 50 characters to a project, file group, or media name.

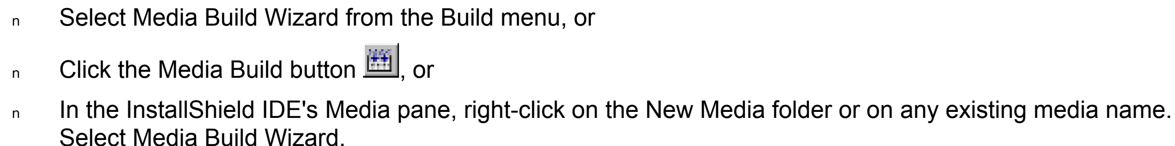
Overview: Building Distribution Media

After all of your setup types have been created and you have successfully compiled your setup script, you're ready to build your setup. InstallShield completely automates the building process with the Media Build Wizard. The build process is automated by the Media Build Wizard, which packages the content of your setup, creating disk images that you copy to your distribution media.

Before you run the Media Build Wizard, make sure that you have successfully compiled your setup script. For more information, see Compiling the setup script.

The Media Build Wizard is a very flexible tool, allowing you to customize almost any aspect of setup creation. It allows you to create different builds using different media names, select which Windows platforms you want to target, specify varying sizes for physical (CD-ROM, diskette) or virtual (LAN, Internet) distribution media, and much more.

To start the Media Build Wizard:

- Select Media Build Wizard from the Build menu, or
- Click the Media Build button , or
- In the InstallShield IDE's Media pane, right-click on the New Media folder or on any existing media name. Select Media Build Wizard.

Enter the desired settings in the Media Build Wizard's panels. For more information at any point, press the F1 key or click the Help button. Further details are available in this section of the Getting Results help file. Click the Finish button to build your setup. (If you chose Review Report Before Build in the Build Type panel, you will see the Media Library Report dialog box. Click OK to accept the report or Cancel to exit.)

Once your disk images are built, you can see them in the Media pane under the media name. These disk images are located in the following default location: My Installations\<project name>\Media\<media name>\Disk Images.

Be careful about copying any files to the disk image folders or modifying any of the existing files. The next time that you build your setup with the Media Build Wizard using the same media name, it will delete the existing files.

You can now copy your disk images to the actual distribution media.

Specify a location for my media files

By default, the Media Build Wizard creates a folder on your Windows drive to place the disk image folders: My Installations\<project name>\Media\<media name>\Disk Images. To change this location::

1. Go to the Media Build Wizard - Build Type panel. Click the Advanced button to open the Property Sheet.
2. Click the Build Location tab and enter a fully qualified path to the location where you would like your media files stored, or click Browse to navigate through your disk drive.
3. Click OK.

InstallShield cannot handle project file paths with more than 260 characters (including the file name). The names of the following project elements are incorporated in project paths and file names:

- the project itself
- file groups
- media names
- files in the Scripts pane
- files inserted into the Setup Files pane

To help you keep within the 260-character limit, InstallShield warns you if you attempt to assign a name with more than 50 characters to a project, file group, or media name.

{button ,AL(`Copy my media files to the actual media;Specify a folder for new setup projects',0,`',`')}   See also

Place product version and company information on my distribution media

InstallShield creates a tag file called Data.tag, a text file that contains information about your company and application. This file will be included on the distribution media with the other setup files in the Disk1 folder.

Enter the information for the tag file in the Media Build Wizard - Tag File panel.

The tag file is composed in this format:

```
[TagInfo]
Company=Your Company Name
Application=Your Application Name
Version=1.00.000
Category=Development Tool
Misc=Any other information you want to add here.
```

Leave extra space on one of my disks so I can add a file later

1. In the Media Build Wizard - Build Type panel, click the Advanced button to open the Property Sheet.
2. Click the Disk Information tab and then click the Add button. The Disk Size Reserved dialog opens.
3. Enter the disk number in the Disk ID field, and enter the amount of space in kilobytes you want to reserve in the Size Reserved field.

InstallShield
www.installshield.com

Leave all files uncompressed on my CD-ROM

1. In the Media Build Wizard - Disk Type panel, highlight either "CD-ROM, Default Size 650 MBytes" or "Custom Size."
2. Select the Data as files check box.

Recall that you already selected the properties for each file group in its properties sheet. (Open the File Group pane and double-click a file group folder to see its properties.) Regardless of whether you selected Yes or No for the Compressed property, by selecting Data as files all of your application's files will be uncompressed on the distribution media.

Also recall that you selected a folder where you wanted each component's files placed on a CD-ROM. If you select Data as files in the Media Build Wizard, you should have a CD-ROM folder specified for *each* component in your project. Otherwise, the uncompressed files are stored in the root folder for your setup.

Specify a custom size for my distribution media

In the Media Build Wizard - Disk Type panel, highlight Custom Size. Enter the capacity of your distribution media in kilobytes in the Custom Size edit field.

InstallShield
www.installshield.com

Set the time stamp on my setup or data files

Ordinarily, your setup files (Setup.exe, _user1.cab, Data.tag, and so on) reflect the date and time when you built your setup through the Media Build Wizard. You can specify a time stamp for your setup files in this manner:

1. In the Media Build Wizard - Build Type panel click the Advanced button to open the Property Sheet. Click the Date/Time tab.
2. Under Setup Files select the User Defined radio button. Enter a new date and time in the appropriate edit field.

When you compress your application's data files, they reflect their original dates. But when you leave files uncompressed on a CD-ROM, you can specify a time stamp that you want InstallShield to place on all of your data files:

1. In the Media Build Wizard - Build Type panel click the Advanced button to open the Property Sheet. Click the Date/Time tab.
2. Under Data Files (CD-ROM only) select the User Defined radio button. Enter a new date and time in the appropriate edit field.

InstallShield
www.installshield.com

Password-protect the entire file media library

1. In the Media Build Wizard - Build Type panel, click the Advanced button to open the Property Sheet.
2. Click the Password tab and select the Use password checkbox. Enter your password in the Password edit field.

These instructions protect your media library from being accessed without the password you entered. To get the password from your end user and verify it, you have to display a dialog box and call the ComponentValidate function in your setup script.

{button ,AL(`Get and validate a component or file media library password',0,`',`')} See also

InstallShield
www.installshield.com

Copy my media files to the distribution media

After you have built your setup in your distribution media folders with the Media Build Wizard, you can send your media files to the actual distribution media. Launch the Send Media Wizard with one of the following methods:

- In the Media pane highlight the name of the media name. Right-click and select Send Media To.
- From the Build menu, choose Send Media To.

Specify a folder for new setup projects

By default, setup project files are placed in the <Windows drive>\My Installations folder. To specify a different location for new projects, do the following:

1. From the Tools menu select Options. The Options dialog box opens.
2. In the Options dialog box, click the Project Location tab.
3. Type a path in the edit box, or click the Browse... button to select a folder.
4. Click OK.

Changing the Project Location setting does not affect the location of existing projects.

InstallShield cannot handle project file paths with more than 260 characters (including the file name). The names of the following project elements are incorporated in project paths and file names:

- the project itself
- file groups
- media names
- files in the Scripts pane
- files inserted into the Setup Files pane

To help you keep within the 260-character limit, InstallShield warns you if you attempt to assign a name with more than 50 characters to a project, file group, or media name.

{button ,AL(`Specify a location for my media files',0,`',`')} See also

Copy a project or template as a new project

1. Close any open projects. From the File menu, select close.
2. From the File menu, select New. The New dialog box opens.
3. In the New dialog box, click the appropriate tab.
   - To copy an existing project, click the Projects tab.
   - To copy a template, click the Template tab.
4. Select the project or template you want to copy.
5. Click the OK button. The new project opens.

The new project's name is based on the name of the project or template that was copied, and on the names of any already-existing copies of that same project or template. For example, the first time you copy Template One the new project is named "Template One-1". If one or more projects with names of the form "Template One-<n>" exist, the new project will be named "Template One-<m>" where m is one greater than the highest existing n.

{button ,AL(`Rename a project',0,`',`')} See also

InstallShield
www.installshield.com

Rename a project

1. If the project is open, close it: from the File menu, select close.
2. In the Projects window, right click on the project's icon. A popup menu appears.
3. From the popup menu, select Rename. The name text at the bottom of the project's icon becomes editable.
4. Click in the box surrounding the name text at the bottom of the project's icon.
5. Edit the name text.
6. Click in the white space in the Projects window.

Create a single executable file for distribution

Simply run PackageForTheWeb, which is available on your InstallShield CD-ROM or at the InstallShield Web site.

Create a time-locked evaluation copy of my distribution

Use TimeLOCK, which is available on your InstallShield CD-ROM or at the InstallShield Web site.

InstallShield
www.installshield.com

Redistributable files

The following files are redistributable with your software distribution, as discussed in your End-User License Agreement. The Media Build Wizard automatically includes them as needed in your disk image folders.

instXXX.ex
_isdel.exe
_isres.dll
_setup.dll
IsunXXXX.exe
Setup.exe
Corecomp.ini
Lang.dat
Os.dat

InstallShield
www.installshield.com

Overview: The Registry

Creating Windows setup programs is becoming increasingly complex. Invariably, you will have to create, modify, or read registry entries to integrate your application with others, associate file extensions, map file dependencies, modify environment variables, and so on. InstallShield makes it easy for you, providing almost 20 functions that can write, read, modify, and delete registry entries, even functions for connecting to a remote registry on a network system.

This section gives an overview of the Windows registry and explains some of the differences between the 32-bit registry and the 16-bit registration database. Even if you are already familiar with these issues, this material will help you find out more about how the InstallScript registry-related functions handle different platforms, work under different root keys, and get logged for uninstallation. You will also find important information on how InstallShield handles reference counts for shared files.

{button ,JI(`GETRES.HLP>(w95sec)',`The_32_bit_registry')} The 32-bit registry

{button ,JI(`GETRES.HLP>(w95sec)',`The_16_bit_registration_database')} The 16-bit registration database

{button ,JI(`GETRES.HLP>(w95sec)',`The_registry_16_vs_32_bit')} The registry: 16- vs. 32-bit

In addition, you need to be aware of some unique characteristics of the InstallScript registry-related functions. InstallShield provides three types of registry-related functions in order to make it easier for you to perform certain setup tasks.

Special registry-related functions
: Work only with certain predefined 32-bit registry keys. (Each is designed for a special purpose under that particular key.)

General registry-related functions
: Work with all 32-bit registry keys, including those handled by the special registry-related functions. The general registry-related functions also work with the 16-bit registration database. (There are some restrictions on how some general registry-related functions work with keys handled by the special registry-related functions. Refer to the individual function descriptions for details.)

Remote registry functions
: Open and close a connection to a remote registry. (Once you have connected to the remote registry, you can only edit keys and values under either HKEY_LOCAL_MACHINE or HKEY_USERS.)

The special registry-related functions

The special registry-related functions are designed to make it easier for script writers to set up the minimum required 32-bit registry keys and values. The special registry-related functions work only with the per application paths key, the application uninstallation key, or the application information key, as shown below. Refer to the individual function descriptions for more details.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\App Paths\ <per application paths key>

The per application paths key, or App Paths key, stores path information enabling 32-bit Windows to find your application's executable files.

InstallationInfo
: Provides the name of the application executable file, which is used to create the per application paths key. The key is not created until RegDBSetItem is called (see below).

RegDBGetItem
: Retrieves the value of [Path] or of [DefaultPath] under the per applications path key.

RegDBSetItem
: Results in the creation of the per application paths key, and sets the value of [Path] or of [DefaultPath] under that key.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\ <application uninstallation key>

The application uninstallation key stores information enabling uninstallation functionality.

DeinstallStart
: Creates the application uninstallation key and sets the [UninstallString] value under that key.

RegDBGetItem
: Retrieves the value of [DisplayName] under the application uninstallation key.

RegDBSetItem
: Sets the value of [DisplayName] (the name displayed in the Add/Remove Programs window) under the application uninstallation key.

HKEY_LOCAL_MACHINE\Software\<company key>\<product key>\<version key>

Your setup program should create an application information key for each application it installs. The application information key stores information about the application. Windows 3.1 uses .ini files to store such information. Windows 95 and Windows NT use the application information key.

HKEY_LOCAL_MACHINE\Software\<company key>\<product key>\<version key> is collectively known as the application information key. Keys and values created under the \<version key> key are said to be created under the application information key.

InstallationInfo
: Uses company name, product name, and product version number to create the keys that together are referred to as the application information key. No values are set under the key until you call the RegDBSetAppInfo function (see below).

RegDBGetAppInfo
: Retrieves a value from under the application information key.

RegDBSetAppInfo
: Sets a value under the application information key.

{button ,AL(`Prepare my setup for uninstallation;The primary registry functions',0,`',`')} See also

The general registry-related functions

The general registry-related functions work with any registry key you specify, including the predefined keys handled by the special registry-related functions. These functions also work with the 16-bit registration database

RegDBCreateKeyEx
: Creates a key in the registry. Also allows you to associate a class object with a registry key (advanced users only).

RegDBDeleteKey
: Deletes the specified key from the registry.

RegDBDeleteValue
: Deletes a value from a specified registry key.

RegDBGetKeyValueEx
: Retrieves a value from under a key in the registry.

RegDBKeyExist
: Checks if a key exists.

RegDBQueryKey
: Queries a key for its subkeys and value names.

RegDBSetDefaultRoot
: Sets the root key.

RegDBSetKeyValueEx
: Sets registry entries.

unInstallShield will uninstall all keys under any key that is logged for uninstallation. Keys created automatically by InstallShield and keys created as a result of calling InstallationInfo and DeinstallStart are logged for uninstallation. When you call RegDBSetKeyValueEx to create keys above which there are no keys logged for uninstallation, your keys will not be uninstalled by unInstallShield, regardless of whether logging was enabled or not (you can Enable and Disable logging). However, when you call RegDBCreateKeyEx while logging is enabled to create a key above which there are no keys logged for uninstallation, your key *will be logged* for uninstallation. Any keys you create under the logged key will be uninstalled when the logged key is uninstalled.

InstallShield
www.installshield.com

The remote registry functions

InstallShield
www.installshield.com

These registry-related functions give you access to a remote registry, where you can create, delete, or retrieve registry keys, value names, and value pairs much as you would on a local registry.

RegDBConnectRegistry
    Opens a connection to a remote registry.

RegDBDisConnectRegistry
    Closes the connection to a remote registry.

These functions are intended for system administrators for network installations. If you are trying to open a registry on a remote Windows NT system, you must have administrator privileges. If you are trying to open a registry on a remote Windows 95 system, it must already have Remote Administration enabled.

When you access a remote registry with RegDBConnectRegistry, you can read and change keys and values only under HKEY_LOCAL_MACHINE and HKEY_USERS. When you call RegDBConnectRegistry, you must specify which root key you want to be able to edit. You must close and re-open the connection if you want to edit the other root key or one its subkeys. Also remember that you can only call the general registry-related functions to edit these keys and subkeys.

Since you set the root key by calling RegDBConnectRegistry, you cannot call RegDBSetDefaultRoot once you have established a connection to a remote registry. Once you have called RegDBDisConnectRegistry, all calls to registry-related functions will affect the local registry, and you can then call RegDBSetDefaultRoot to change the root key.

InstallShield
www.installshield.com

The 32-bit registry

The registry is a database containing system- and application-related information on Windows 95 and Windows NT systems. The registry stores all kinds of information, including the following:

- The name of the program associated with a given file extension
- The command line expression to be executed when the user opens a file from a shell application
- Application information such as company name, product name, and version number
- Uninstallation information allowing your application to be uninstalled easily without interfering with other software on the system
- The registry also makes it possible to integrate other applications that use OLE (Object Linking and Embedding). The database was designed to allow applications to share specific information.
- Path information allowing your application to run

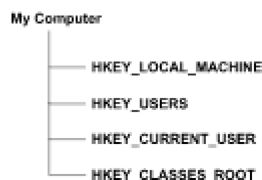
The registry in Windows 95 and Windows NT consists of several database files. The registry is depicted as a tree structure, much as the Windows File Manager or Windows 95 Explorer represents a file system.

The registry tree structure is similar to the paths on a PC file system, representing unique access pathways to system- and application-related information. A registry key is similar to a file on a PC file system, identifying a unique location for storing information.

Windows 3.1 does not have a registry. Rather, it has a registration database containing one key structure: HKEY_CLASSES_ROOT. The vast majority of system-, user-, and application-related information in Windows 3.1 is stored in .ini and .sys files. For more information, see The 16-bit registration database.

The base keys

The registry consists of a set of keys arranged hierarchically under the My Computer root key. Just under the root key are the four base keys that can be affected by a setup: HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_USER, and HKEY_CLASSES_ROOT



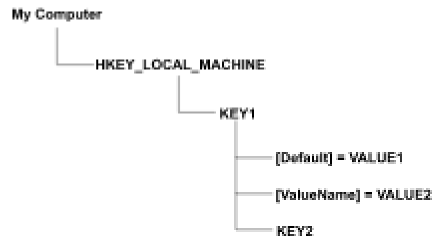
Windows 95 contains two base keys in addition to these four: HKEY_CURRENT_CONFIG and HKEY_DYN_DATA. Since these keys are not typically affected by setups, they are not discussed here.

The terms "key" and "subkey" are relative. In the registry, a key that is "below" another key can be referred to as a subkey or as a key, depending on how you wish to refer to it relative to another key in the registry hierarchy. The term "subkey" will not be used in this document unless a key's position relative to a higher-level key is important and needs to be expressed.

Keys, value names, and values

A key is a named location in the registry. A key can contain a subkey, a value name and value pair, and a default (unnamed) value.

A value name and value pair is a two-part data structure under a key. A value name identifies a value for storage under a key. A value is the data associated with a value name. When a value name is unspecified for a value, that value is the default value for that key. There can be only one default (unnamed) value under a key. For example, here is a sample key, a default (unnamed) value, and a named value, and a subkey:



Registry keys and their values are expressed in the following format, similar to the way paths are expressed:

```
HKEY_LOCAL_MACHINE\KEY1
       [Default] = VALUE1
       [ValueName] = VALUE2
```

Keys are separated by backslashes (\). Value name and value pairs are entered underneath the entire key expression. The value name is on the left side of the equal sign and the value is on the right.

{button ,AL(`The registry: 16- vs. 32-bit;The 16-bit registration database',0,`',`')} See also

InstallShield
www.installshield.com

The 16-bit registration database

Windows 95 and Windows NT both have full 32-bit registries in which applications store information. The registry is composed of several root-level keys with many additional keys under each root-level key. The Windows 95/NT 32-bit registry and the Windows 3.1 16-bit registration database have the root-level key HKEY_CLASSES_ROOT in common.

Windows 3.1 has a 16-bit registration database, which is composed of a default root key and many multiple subkeys under it. The registration database is used mainly to store file, application, and library registration information.

The 16-bit registration database and 32-bit registry also differ slightly in structure. The 32-bit registry allows each key to have a set of name-value pairs associated with it. The 16-bit registration database will store only one value per key. For purposes of compatibility, the 32-bit registry also has a default value for each key, corresponding to the 16-bit registration database's single value.

Viewing the registration database

To view the 16-bit registration database, from the Program Manager, open the File menu and choose Run. Enter `regedit /v` and click OK. The Registration Info Editor will open, displaying the contents of the 16-bit registration database.

{button ,AL(`The registry: 16- vs. 32-bit;The 32-bit registry',0,`',`')} See also

InstallShield
www.installshield.com

The registry: 16- vs. 32-bit

Windows 95 and Windows NT both have full 32-bit registries in which applications store information. The registry is composed of several root-level keys with many additional keys under each root-level key. The 32-bit Windows 95/NT registry and the Windows 3.1 16-bit registration database have the root-level key HKEY_CLASSES_ROOT in common.

Windows 3.1 has a 16-bit registration database, which is composed of a default root key and many multiple subkeys under it. The registration database is used mainly to store file registration and OLE information, so most applications do not need to modify this database during setup.

The 16-bit registration database and 32-bit registry also differ slightly in structure. The 32-bit registry allows each key to have a set of name-value pairs associated with it. The 16-bit registration database will only store one value per key. For purposes of compatibility, the 32-bit registry also has a default value for each key, corresponding to the 16-bit registration database's single value.

Registry-related functions

InstallShield has two types of registry-related functions—special and general.

The special registry-related functions modify specific keys located under HKEY_LOCAL_MACHINE and can only be used to modify the 32-bit registry, because the 16-bit registration database does not have a KEY_LOCAL_MACHINE key. Therefore, special registry-related functions will work only under Windows 95 or Window NT.

The general registry-related functions are designed to work with all registry keys, including those handled by the special registry-related functions. These functions can affect any key in the 32-bit registry or 16-bit registration database, but general registry-related functions can only work with keys under HKEY_CLASSES_ROOT in the 16-bit registration database. (See below for more information.)

The InstallShield engine under 16- and 32-bit platforms

The InstallScript registry-related functions work differently depending upon which platform you're targeting. You choose which platform you're targeting in the Project Wizard's Choose Target Platform panel. Be very careful to consider all platforms on which your setup will run when selecting a target platform.

The platform you choose to target determines which InstallShield engine file you ship with your setup. InstallShield automatically creates a setup with the correct engine file(s) when you build your setup through the Media Build Wizard. _inst16.ex_ is the 16-bit InstallShield engine file, and _inst32*x*.ex_ is the 32-bit InstallShield engine file (*x* is short for the specific 32-bit Windows platform you're targeting).

Depending on how you build your setup, which InstallScript registry-related functions you call, and the target platform, you will have one of the scenarios listed below. Since the 16- and 32-bit InstallShield engine files behave differently and the 32-bit registry and 16-bit registration database are different, there are a number of considerations when calling registry-related functions. You will have no problems if your setup will always run on the expected platforms, or if you write your setup script appropriately for a cross-platform setup, calling registry-related functions based upon the target platform. (See GetSystemInfo.)

Modifying the 16-bit registration database:

{button ,JI(`GETRES.HLP>(w95sec)',`Modifying_the_registration_database_with_16_bit_InstallShield')} with the 16-bit InstallShield engine (_inst16.ex_)

{button ,JI(`GETRES.HLP>(w95sec)',`Modifying_the_registration_database_with_32_bit_InstallShield')} with the 32-bit InstallShield engine (_inst32*x*.ex_)

Modifying the 32-bit registry:

{button ,JI(`GETRES.HLP>(w95sec)',`Modifying_the_registry_with_16_bit_InstallShield')} with the 16-bit InstallShield engine (_inst16.ex_)

{button ,JI(`GETRES.HLP>(w95sec)',`Modifying_the_registry_with_32_bit_InstallShield')} with the 32-bit InstallShield engine (_inst32*x*.ex_)

InstallShield
www.installshield.com

Modifying the registration database with 16-bit InstallShield

Since the 16-bit registration database has no HKEY_LOCAL_MACHINE root key, all special registry-related functions will fail, except for InstallationInfo and DeinstallStart which are necessary to call for uninstallation functionality.

When general registry-related functions under 16-bit InstallShield are used to modify the 16-bit registration database, they will adjust to the structure of the 16-bit registration database by ignoring certain parameters in the function calls. See Setting parameters to modify the 16-bit registration database for specific information as to which parameters each general registry-related function will ignore.

RegDBSetDefaultRoot works only with the 32-bit registry when using 32-bit InstallShield, because the 16-bit registration database has only one root key and cannot be changed. Any attempt to use this function with 16-bit InstallShield or to use it on a 16-bit registration database will fail.

Modifying the registration database with 32-bit InstallShield

Since the 16-bit registration database has no HKEY_LOCAL_MACHINE root key, all special registry-related functions will fail, except for InstallationInfo and DeinstallStart which are necessary to call for uninstallation functionality.

The 32-bit InstallShield engine (_inst32*x*.ex_) affects the 16-bit registration database in the same manner as the 16-bit engine. For information on calling the general registry-related functions under 32-bit InstallShield to modify the 16-bit registration database, see Setting parameters to modify the 16-bit registration database.

RegDBSetDefaultRoot works only with the 32-bit registry when using 32-bit InstallShield, because the 16-bit registration database has only one root key and cannot be changed. Any attempt to use this function with 16-bit InstallShield or to use it on a 16-bit registration database will fail.

Modifying the registry with 16-bit InstallShield

General registry-related functions called with the 16-bit InstallShield engine will modify the 32-bit Windows 95 or NT registry in exactly the same way that they modify the 16-bit Windows registration database.

In other words, the general registry-related functions under 16-bit InstallShield can add keys only under HKEY_CLASSES_ROOT, and the only value they can set or retrieve for a particular key is the default value. They will fail if called to modify any other part of the 32-bit registry. This functionality is included so that 16-bit OLE applications will install correctly under 32-bit Windows.

RegDBSetDefaultRoot works only with the 32-bit registry when using 32-bit InstallShield, because the 16-bit registration database has only one root key and cannot be changed. Any attempt to use this function with 16-bit InstallShield or to use it on a 16-bit registration database will fail.

{button ,AL(`Setting parameters to modify the 32-bit registry with 16-bit InstallShield',0,`',`')} See also

InstallShield
www.installshield.com

Modifying the registry with 32-bit InstallShield

The 32-bit InstallShield engine was specifically designed to work correctly in modifying the 32-bit Windows 95 and Windows NT registry in all cases.

Setting parameters to modify the 16-bit registration database

The 16-bit registration database does not support multiple named values under specific keys and does not allow those values to have a data type other than the default (REGDB_STRING). Thus, you are limited in setting or retrieving values for 16-bit registration database keys. Certain parameters will always return the same values. The following examples show you how to set parameters so that the functions will successfully modify a 16-bit registration database.

RegDBGetKeyValueEx (szKey, szName, nvType, svValue, nvSize);

The first two parameters should be set to the following values when calling this function:

szKey
This parameter should be set normally.

szName
This parameter must be set to "".

The last three parameters will then return the following values from the 16-bit registration database:

nvType
This parameter will always return REGDB_STRING.

svValue
This parameter will return the default value associated with *szKey*.

nvSize
This parameter will contain the size of the value returned in *svValue*.

RegDBSetKeyValueEx (szKey, szName, nType, szValue, nSize);

Here is what each parameter should contain before calling this function to set a value in the 16-bit registration database:

szKey
This parameter should be set normally.

szName
This parameter should be set to "".

nType
This parameter must be set to REGDB_STRING.

szValue
This parameter should contain the string value to be set.

nSize
This parameter can be set normally.

RegDBQueryKey (szSubKey, nItem, list);

szSubKey
This parameter can be set normally.

nItem
This parameter must be set to REGDB_KEYS. The function will fail if is set to REGDB_NAMES.

list
This parameter can be set normally.

InstallShield
www.installshield.com

Setting parameters to modify the 32-bit registry with 16-bit InstallShield

The 16-bit InstallShield engine will modify the 32-bit registry in the same way it would the 16-bit registration database, using registration database default values in all cases. The following examples of function syntax show which parameters need to be specified as the default ("").

All 16-bit InstallShield registry-related functions will only work with keys under HKEY_CLASSES_ROOT.

RegDBGetKeyValueEx (szKey, szName, nvType, svValue, nvSize);

The first two parameters should be set to the following values when calling this function:

szKey
This parameter should be set normally.

szName
This parameter should be set to "". (The default value for the appropriate key will be set by this function regardless of what this parameter is set to.)

The last three parameters will then return the following values from the 32-bit registry:

nvType
This parameter will always return REGDB_STRING.

svValue
This parameter will return the default value associated with *szKey*.

nvSize
This parameter will contain the size of the value returned in *svValue*.

RegDBSetKeyValueEx (szKey, szName, nType, szValue, nSize);

Here is what each parameter should contain before calling this function to set a value in the 32-bit registry using 16-bit InstallShield:

szKey
This parameter should be set normally.

szName
This parameter should be set to "". (The default value for the appropriate key will be set by this function regardless of what this parameter is set to.)

nType
This parameter should be set to REGDB_STRING.

szValue
This parameter should contain the string value to be set.

nSize
This parameter can be set normally.

RegDBQueryKey (szSubKey, nItem, list);

szSubKey
This parameter can be set normally.

nItem
This parameter must be set to REGDB_KEYS. The function will fail if is set to REGDB_NAMES.

list
This parameter can be set normally.

InstallShield
www.installshield.com

The primary registry functions

All setups use three InstallScript functions to provide information for the primary installation-related registry keys described in the previous section. The three primary registry key functions are InstallationInfo, DeinstallStart, and RegDBSetItem.

If you created your setup using the Project wizard, you do not need to call these functions. The script generated by the Project Wizard already includes these function calls.

InstallationInfo

szCompany, the first parameter to InstallationInfo, provides your company name for use as a key under the pre-existing key HKEY_LOCAL_MACHINE\Software. Setting szCompany to "Sample1Company" would result in the following entry:

```
HKEY_LOCAL_MACHINE\Software\Sample1Company
```

szProduct, the second parameter to InstallationInfo, is your application or product name. The value of szProduct gets written to the registry as a subkey of the company name key under HKEY_LOCAL_MACHINE\Software.

Setting szProduct to "Sample1App" would result in the following entry, assuming szCompany is set to "Sample1Company":

```
HKEY_LOCAL_MACHINE\Software\Sample1Company\Sample1App
```

The value in szProduct is also inserted into the first paragraph of message text in the Welcome dialog box.

szVersion, the third parameter to InstallationInfo, provides the application version number. The value of szVersion is written as a subkey of the product key. Continuing the previous key example, setting szVersion to 5.0 would yield the following entry:

```
[HKEY_LOCAL_MACHINE]\Software\Sample1Company\Sample1App\5.0
```

The InstallationInfo function's fourth parameter, szProductKey, is used to create the per application paths information key (App Paths key).

You must call InstallationInfo before you call DeinstallStart in your script. If you do not, DeinstallStart will fail because it needs the company name, product name, and version number placed in memory by InstallationInfo.

DeinstallStart

DeinstallStart enables unInstallShield functionality. DeinstallStart allows creation of the application uninstallation key under 32-bit Windows. DeinstallStart also initializes the uninstallation log file using the company name, product name, and version number placed in memory by the prior call to InstallationInfo, which is also necessary to call under 16-bit Windows. Call DeinstallStart immediately after calling InstallationInfo.

szDefLogPath, the first parameter to DeinstallStart, defines the full path to the uninstallation log file. InstallShield creates the uninstallation log file name and appends it to the path you provide in szDefLogPath. If the path does not exist, it is created for you whenever possible. The resulting path and file name expression is returned in the second parameter, svLogFile, and is used as the second half of the [UninstallString] value under the application uninstallation registry key.

InstallShield sets the first half of the value for [UninstallString]. When InstallShield transfers IsUninst.exe (32-bit platforms) or IsUn16.exe (16-bit platforms) to the target system, it records the file's location and uses it to construct the first half of the [UninstallString] value.

Assuming that InstallShield placed IsUninst.exe into the C:\Win95 folder, the complete [UninstallString] value might look like the one shown below. Notice that when the value of svLogFile is added to the [UninstallString] value in the registry, it is preceded by -f:

```
[UninstallString] = C:\Win95\IsUninst.exe
                   -fC:\Program Files\Company\Myapp1\Uninst.isu
```

szKey, the third parameter to DeinstallStart, provides the application-specific portion of the uninstallation key. InstallShield precedes the common portion of the key name. For example, setting szKey to "SampleDeinstKey" results in this key entry:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\
SampleDeinstKey
```

lStyle, the fourth parameter to DeinstallStart must be 0 (zero).

For more information about DeinstallStart, see The uninstallation log file.

RegDBSetItem

The RegDBSetItem function must be called after DeinstallStart to provide a value for [DisplayName] under the application uninstallation key. The value of [DisplayName] is displayed in the Windows 95 and Windows NT 4.0 Add/Remove Programs Properties sheet Install/Uninstall tab, allowing the application to be selected for removal.

nItem, the first parameter, must be the constant REGDB_UNINSTALL_NAME.

szValue, the second parameter, contains the name you wish to display in the Add/Remove Programs Properties sheet Install/Uninstall tab. Setting szValue to "Sample App" results in the following entry:

```
[DisplayName]=Sample App
```

InstallShield
www.installshield.com

Check whether a key exists

Call the RegDBKeyExist function to see if a specific key exists in the registry.

When creating keys, modifying keys, or querying information from keys, you may wish to test to see if the desired key already exists. Your setup can evaluate the results of the test and act accordingly. For example, your installation might determine if an earlier version of an application is installed by testing for the existence of a specific version number key.

InstallShield
www.installshield.com

Create registry keys

Call the RegDBCreateKeyEx function to create a registry key. If the key already exists, RegDBCreateKeyEx opens it.

By default, InstallShield creates the key under HKEY_CLASSES_ROOT. If you want to create a subkey under a different registry key, call the RegDBSetDefaultRoot function before calling RegDBCreateKeyEx.

Type a double backslash (\\) before each subkey. The subkey name can include several levels of subkeys, and InstallShield will create any subkeys all at once if they do not exist.

Keys created with RegDBCreateKeyEx are logged for uninstallation unless logging is disabled. Also note that RegDBCreateKeyEx does not associate a value with the key. RegDBSetKeyValueEx can create keys and values, but the keys are not logged for uninstallation unless they are created as subkeys under a key already logged.

InstallationInfo, DeinstallStart, and RegDBSetItem create keys in predefined places in the registry to store information about your application. For more information, see The primary registry functions.

InstallShield
www.installshield.com

Delete registry keys

Call the RegDBDeleteKey function to delete a specific key and its value from the registry. The RegDBDeleteKey function also deletes subkeys and their associated values.

InstallShield
www.installshield.com

Get information from the registry

Call the RegDBGetKeyValueEx function to get the value of a value name in a specified key.

Call the RegDBQueryKey function to query a key for its subkeys and value names.

RegDBGetKeyValueEx and RegDBQueryKey assume that the specified key is under HKEY_CLASSES_ROOT, unless you specify a different root key using RegDBSetDefaultRoot.

Call the RegDBGetAppInfo function to get information in the form of value name and value pairs from the application information key established for your main application and RegDBSetAppInfo. Before you call RegDBGetAppInfo, the application information must be set using InstallationInfo.

Call the RegDBGetItem function to retrieve information from the per application path key or the application uninstallation key. For more information about RegDBGetAppInfo or RegDBGetItem, see The special registry-related functions.

Set information in the registry

Call the RegDBSetKeyValueEx function to set a value to a value name under a specified key.

Call the RegDBSetAppInfo function to set information in the form of value name and value pairs to the application information key established for your main application. Before you call RegDBSetAppInfo, the application information must be set using InstallationInfo. For more information, see The special registry-related functions.

RegDBsetKeyValueEx assumes that the specified key is under HKEY_CLASSES_ROOT, unless you specify a different root key using RegDBSetDefaultRoot.

{button ,AL(`The primary registry functions',0,`',`')} See also

InstallShield
www.installshield.com

Delete a key's value

Call the RegDBDeleteValue function to delete a value from a specific key in the registry.

Get or set information in a remote registry

Call the RegDBConnectRegistry function to connect to a remote registry.

Call the RegDBDisConnectRegistry function to close the connection.

InstallShield
www.installshield.com

Register file extensions

Registering file extensions (making file associations) is a simple matter of calling certain InstallScript registry-related functions to write keys and values in the registry or registration database.

To register a file extension you must do three things:

1. Make a file extension key entry, such as:

   ```
   HKEY_CLASSES_ROOT\.zzz=zzzFile
   ```

2. Make an application (class) identification key entry, such as:

   ```
   HKEY_CLASSES_ROOT\zzzFile=ZZZ File
   ```

3. Make shell command key entries, such as the following:

   ```
   HKEY_CLASSES_ROOT\zzzFile\shell\open\command=Notepad.exe %1
   HKEY_CLASSES_ROOT\zzzFile\shell\print\command=Notepad.exe /P %1
   ```

Follow the above instructions to register the .zzz file extension and associate Notepad with the .zzz file extension. To see a complete example script that calls the necessary InstallScript registry-related functions to complete steps 1-3, click here.

InstallShield
www.installshield.com

Registering file extensions example script

```
   STRING szProgName;

program

   // Create file-extension registry key, .zzz.
   RegDBCreateKeyEx( ".zzz", "" );

   // Assign the value "zzzFile" to the .zzz key.
   RegDBSetKeyValueEx( ".zzz", "", REGDB_STRING, "zzzFile", -1 );

   // Create the application (class) identification key, zzzFile.
   RegDBCreateKeyEx( "zzzFile", "" );

   // Assign the value "ZZZ File" to the zzzFile key.
   RegDBSetKeyValueEx( "zzzFile", "", REGDB_STRING, "ZZZ File", -1 );

   // Assign the open command expression (szProgName) to the shell\open\command
   // subkey of the zzzFile key.
   szProgName = WINDIR ^ "Notepad.exe %1";
   RegDBSetKeyValueEx( "zzzFile\\shell\\open\\command", "", REGDB_STRING,
                       szProgName, -1 );

   // Assign the print command expression (szProgName) to the shell\print\command
   // subkey of the zzzFile key.
   szProgName = WINDIR ^ "Notepad.exe /p %1";
   RegDBSetKeyValueEx( "zzzFile\\Shell\\print\\command", "",REGDB_STRING,
                       szProgName, -1 );

endprogram

   // Source file:  Is5gr006.rul
```

Merge registration (.reg) files

Call the LaunchApp function to launch the registry editor program (Regedit.exe) from your setup.

If you also want to merge several .reg files when you call the LaunchApp function, pass all of the files to be merged as the second parameter, leaving a space between the file names. For example:

```
LaunchApp(WINDIR ^ "Regedit.exe", "/s Setup.reg Ole2.reg Test.reg");
```

When launching Regedit, it is important to call the LaunchApp or LaunchAppAndWait with the NOWAIT option instead of calling the LaunchAppAndWait function with the WAIT option. Using the WAIT option will cause the setup to hang to approximately 90 seconds after the merging has completed since the registry editor does not have a valid window handle when run in silent mode. Because the registry editor automatically locks out any other processes from executing until it has completed when run in silent mode, your setup will not continue until the registry file has been merged.

Changes made by merging .reg files are not logged by InstallShield and will not be removed by unInstallShield. If you are including uninstallation functionality with your setup, you should instead use the InstallShield registry functions to make the desired changes.

Remember that the registry editor program is named differently on different platforms. If you are developing a script to run on multiple platforms, call the GetSystemInfo function to determine the operating system and then use the appropriate name depending upon that platform.

Platform	Registry editor program
Windows 3.1	Regedit.exe
Windows 95	Regedit.exe
Windows NT	Regedt32.exe

InstallShield
www.installshield.com

Enter a hexadecimal (type DWORD) value into the registry

Call the RegDBSetKeyValueEx function with the REGDB_NUMBER option. Pass the decimal equivalent of the value in the fourth parameter.

For example, to create this value:

```
..\TestKey\Key1 = 0x0000003e8(1000)
```

call RegDBSetKeyValueEx as shown:

```
   RegDBSetKeyValueEx ("TestKey", "Key1", REGDB_NUMBER, "1000", -1);
```

InstallShield
www.installshield.com

Set or get multi-line strings in the Windows NT registry

The Windows 95 registry does not support multi-line strings. If you use the multi-line string option under Windows 95, a binary value will be placed in or read from the registry.

Setting multi-line strings

Call the RegDBSetKeyValueEx function. The szValue parameter should contain the substrings to be added, separated by null characters. You must add two null characters to the end of the main string to denote its end. To create such a string, follow these steps:

1. Initialize the string to contain each of the substrings, separated by space characters.
2. Assign the ASCII value for a null string to the string positions after each line.
3. Assign an additional null character (ASCII value 0) to the end of the string.

For example:

```
// svValue is the string read in by RegDBGetKeyValueEx
svValue = "This is line one. This is line two. This is line three. ";
svValue[17] = 0;
svValue[35] = 0;
svValue[55] = 0;
svValue[56] = 0;
```

Also, the last parameter to RegDBSetKeyValueEx, nSize, cannot be -1. It must be set to the amount of characters that will be copied into the registry, including the two null characters at the end of the string.

Retrieving multi-line strings

Call the RegDBGetKeyValueEx function. The substrings will be returned in a main string, separated by null characters. To extract them, use the following code to read the strings into a string list and display them in an SdShowInfoList dialog box:

```
// svValue is the string read in by RegDBGetKeyValueEx
listID = ListCreate( STRINGLIST );
if (listID != LIST_NULL) then
   StrGetTokens( listID, svValue, "" );
   SdShowInfoList( szTitle, szMsg, listID );
endif;
```

How Windows NT security permissions affect a setup

There are many permutations and combinations of user permissions that an administrator can implement. There is nothing that InstallShield can do to override security permissions. Since an administrator can create keys at most levels, it is recommended that a user with administrator privileges perform all setups.

The information below applies to the default security permissions, but some Windows NT systems may be set up with different security permissions.

Default security permissions

Creation/Reading/Deletion of Registry Keys	Guest	Administrator
HKEY_CURRENT_USER	Full	Full
HKEY_USER	Read-Only	Full
HKEY_CLASSES_ROOT	Full	Full
HKEY_LOCAL_MACHINE\SECURITY	None	None
HKEY_LOCAL_MACHINE\SAM	None	None
HKEY_LOCAL_MACHINE\SYSTEM	Read-Only	Full
HKEY_LOCAL_MACHINE\SOFTWARE	Full	Full
HKEY_LOCAL_MACHINE\SOFTWARE\ SECURE	Read-Only	Full
HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\SECURE	Read-Only	Full
HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\WINDOWS NT	Creation-Only	Full
HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\ProgramGroup	Read-Only	Full
HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\Windows	Full	Full

Other privileges

	Guest	Administrator
Create PERSONAL Program Groups	Yes	Yes
Create COMMON Program Groups	No	Yes
Add/Modify Icons in PERSONAL Program Groups	Yes	Yes
Add/Modify Icons in COMMON Program Groups	No	Yes
Create PERSONAL Add/Remove Program - Uninstall Strings	Yes	Yes
Create COMMON Add/Remove Program - Uninstall Strings	No	Yes

Set an environment variable under Windows NT

This information is kept in the Windows registry under Windows NT. Call the InstallScript registry-related functions to modify these variables.

To set a variable for the current user, call RegDBSetKeyValueEx to create a value under HKEY_CURRENT_USER\Environment. The name of this value should be the same as the name of the environment variable that you are setting.

To create a system-wide environment variable, call RegDBSetKeyValueEx to create a value under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment. The name of this value should be the same as the name of the environment variable that you are setting.

Then call the SendMessage function with the WM_WININICHANGE option to broadcast to all applications that a registry change has been made.

Not all applications respond to a WM_WININICHANGE message. If you need to be sure that all applications are aware of the new environment variable, you will need to reboot the system by calling the SdFinishReboot or the RebootDialog functions at the end of setup.

In order to set system-wide environment variables, the current user must have administrator privileges. Otherwise, the attempt to add a new system-wide environment variable will fail.

{button ,AL(`How Windows NT security permissions affect a setup',0,`',`')} See also

InstallShield
www.installshield.com

Overview: Program Folders and Items

InstallShield provides several functions you can use to create and manipulate folder windows, folder items, shortcuts, and icons.

Be aware that there are several differences in the way that the Program Manager and Explorer shells handle program groups, menus, icons, and shortcuts. Depending on the target platform, you may need to call different InstallScript functions or change your function calls.

In addition, there are some differences between the 16- and 32-bit InstallShield engines. (You choose which engine to distribute in the Media Build Wizard - Platforms panel.) The 16-bit InstallShield engine cannot create shortcuts (link files) or long file names.

{button ,AL(`The InstallScript functions for creating and modifying program folders and items;The Program Manager vs. the Explorer shell',0,`',`')} See also

InstallShield
www.installshield.com

The Program Manager vs. the Explorer shell

The Program Manager shell, which is used on Windows 3.1 and Windows NT 3.51, handles access to applications differently than does the Explorer shell, which is used on Windows 95 and Windows NT 4.0. Under the Program Manager shell you create program groups which are visible on the desktop. The program groups are populated with program item icons, which launch applications.

Explorer uses program folders and program items, but they are typically placed on the Start menu and its submenus. Icons placed on the Start menu and its submenus can launch applications directly, open program folders containing icons to launch applications, or open submenus containing icons.

There may be times when you need to know whether Program Manager or Explorer is the shell on the target system. For example, if the shell is Program Manager, you may wish to create a program group with program items, whereas if the shell is Explorer, you may wish to add a shortcut to the Start Programs menu.

{button ,AL(`Determine the current shell',0,`',`')} See also

Determine the current shell

Call the GetSystemInfo function with the OSMAJOR option to determine whether the target system is running the Program Manager or Explorer shell.

For an example script that calls GetSystemInfo to determine if the shell is Explorer before adding an icon, see Sample script for setting up uninstallation.

The reason that QueryShellMgr is not used to determine whether Explorer or Program Manager is the shell is that there is no guarantee that the shell manager name will be either "Explorer.exe" or "Progman.exe." A third-party shell could be running, making tests based on "Explorer.exe" and "Progman.exe" invalid.

InstallShield
www.installshield.com

The InstallScript functions for creating and modifying program folders and items

These functions are available for determining shell information and creating and manipulating program folders and items:

AddFolderIcon
  Adds an icon to a specified folder.

CreateProgramFolder
  Creates a new folder on the target system.

DeleteFolderIcon
  Removes an icon or item from a program folder.

DeleteProgramFolder
  Removes a program folder from the target system.

ExitProgMan
  Closes the Program Manager.

GetFolderNameList
  Retrieves a list of program folder names and a list of shortcuts in a folder under Windows 95 or Windows NT 4.0.

GetGroupNameList
  Retrieves a list of all program folder names that exist in the Program Manager or Windows 95 shell.

GetItemNameList
  Retrieves all the program items from a specified program folder.

GetSystemInfo
  Retrieves various information about the target system.

ProgDefGroupType
  Designates program groups as either Personal or Common in a Windows NT environment.

QueryProgGroup
  Queries information about the specified group.

QueryProgItem
  Queries information about the specified program item.

QueryShellMgr
  Retrieves the current shell manager.

ReloadProgGroup
  Reloads the program group if changes have been made to it.

ReplaceFolderIcon
  Replaces an icon in a specified folder.

SelectFolder
  Creates a dialog box that contains a list of valid program folders that the end user can select.

ShowGroup
  Displays the specified program group.

ShowProgramFolder
  Displays all folders under a specified program folder.

InstallShield
www.installshield.com

Get a list of program folders

Call the GetGroupNameList function to retrieve all program folder names that exist under the Program Manager shell or in the Start Programs menu under the Explorer shell.

The following example uses GetGroupNameList to fill a string list with folder names, and then displays the list using the SdShowInfoList function.

```
   #include "sddialog.h"
   STRING svGroupName, szTitle, szMsg;
   LIST listID;

program

   listID = ListCreate(STRINGLIST);

   // Put all of the group names in the list.
   GetGroupNameList(listID);

   // Display the group names.
   szTitle = "Displaying GetGroupNameList Results";
   szMsg = "Groups listed by GetGroupNameList:\n";
   SdShowInfoList(szTitle, szMsg, listID);

   ListDestroy(listID);

endprogram

   // Only include code for SdShowInfoList function
   // to reduce compiled file size and compilation time.
   #define  SD_SINGLE_DIALOGS 1
   #define  SD_SHOWINFOLIST   1
   #include "sddialog.rul"

   // Source file:  Is5gr007.rul
```

When the target system is running the Explorer shell, you can call the GetFolderNameList function to list shortcuts and subfolders in a specified folder. GetFolderNameList also works when Program Manager is the shell, but it is not recommended because of certain limitations (refer to its description in the InstallScript Language Reference help file).

The example below uses GetFolderNameList to display the shortcuts and submenus in the Start menu under the Explorer shell.

```
   #include "sddialog.h"

   STRING  szFolderName;
   NUMBER  nResult;
   LIST    listShortcutsID, listSubFoldersID;

program

   // Create lists for shortcuts and folder names.
   listShortcutsID = ListCreate (STRINGLIST);
   if (listShortcutsID = LIST_NULL) then
      MessageBox ("Failed to create items list!", WARNING);
   endif;

   listSubFoldersID = ListCreate (STRINGLIST);
   if (listSubFoldersID = LIST_NULL) then
       MessageBox ("Failed to create subfolders list!", WARNING);
   endif;
```

```
   // Place the Start menu's shortcuts and submenus
   // into the lists.
   nResult = GetFolderNameList (FOLDER_STARTMENU, listShortcutsID,
                                 listSubFoldersID);
   if (nResult < 0) then
      MessageBox ("GetFolderNameList failed!", SEVERE);
   endif;

   // Display the lists.
   SdShowInfoList ("", "Shortcuts on Start menu:\n", listShortcutsID);
   SdShowInfoList ("", "Submenus on the Start menu:\n", listSubFoldersID);

endprogram

   // Only include code for SdShowInfoList function
   // to reduce compiled file size and compile time.
   #define  SD_SINGLE_DIALOGS 1
   #define  SD_SHOWINFOLIST   1
   #include "sddialog.rul"

   // Source file:  Is5gr008.rul
```

Create a program folder

Call the CreateProgramFolder function to add a program group to the desktop under the Program Manager shell, or to the Start Programs menu under the Explorer shell.

When creating a folder Windows NT, call the ProgDefGroupType function first to establish the group as either COMMON or PERSONAL. COMMON is the default if ProgDefGroupType is not called.

To see an example that also shows how to use CreateProgramFolder, see Sample script for setting up uninstallation.

It is not necessary to check if the program folder exists. You can call the CreateProgramFolder function regardless of whether the folder exists on the target system. The AddFolderIcon function will also automatically create the specified program folder if it does not exist.

InstallShield
www.installshield.com

Delete a program folder

Call the DeleteProgramFolder function to delete an existing program group, folder, or menu. (You cannot delete the Programs folder with DeleteProgramFolder.) When the DeleteProgramFolder function deletes a program folder, all program items and shortcuts in the folder are also deleted. Always ask for confirmation before deleting a program folder, since the user may have added additional program items to it.

The example below creates a folder on the desktop and adds a shortcut to it. Then, the shortcut and the folder are deleted, provided the user responds affirmatively to the AskYesNo dialog boxes.

```
   STRING szItemName, szCommandLine, szWorkingDir, szIconPath;
   STRING szShortCutKey, szFolderName;
   NUMBER nIcon, nFlag;

program

   Disable(BACKGROUND);

   szFolderName = FOLDER_DESKTOP ^ "Test";
   szItemName    = "Test (Notepad)";
   szCommandLine = WINDIR ^ "Notepad.exe";
   szWorkingDir = WINDIR;
   szIconPath = "";
   nIcon = 0;
   szShortCutKey = "";
   nFlag = REPLACE;

   // Add the szItemName shortcut to the szFolderName folder.
   // If the szFolderName folder does not exist, it will be created.
   AddFolderIcon (szFolderName, szItemName, szCommandLine,
                  szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag);

   // Open the folder to show the shortcut.
   ShowProgramFolder(szFolderName, SW_SHOW);

   // Ask the user if you can delete shortcut, then folder. Delete them
   // if answers are "Yes."  Exit if a deletion fails.
   if (AskYesNo('Delete the "' + szItemName + '" shortcut?', NO) = YES) then
      if (DeleteFolderIcon(szFolderName, szItemName) < 0) then
         MessageBox("Could not delete shortcut!", SEVERE);
         abort;
      endif;
   endif;

   Delay(2);

   if (AskYesNo('Delete the folder "' + szFolderName + '"?', NO) = YES) then
      if (DeleteProgramFolder(szFolderName) < 0) then
         MessageBox("Could not delete folder!", SEVERE);
         abort;
      endif;
   endif;

endprogram

   // Source file:  Is5gr009.rul
```

InstallShield
www.installshield.com

Get information about a program folder

Call the QueryProgGroup function to check for the existence of a specific program group under the Program Manager shell.

If InstallShield finds the program group, the QueryProgGroup function returns its attributes. The attributes include the path of the group, as well as the number of items it contains. Under Windows NT, QueryProgGroup also determines whether the folder is COMMON or PERSONAL.

InstallShield
www.installshield.com

Display a program folder

Call the ShowGroup function to bring a particular group window to the front. The ShowGroup function displays or moves a folder window. Call the ShowGroup function at the end of the setup to make the new program folder and its program items visible.

Call the ShowProgramFolder function under the Explorer shell to display a program folder at the end of the setup.

InstallShield
www.installshield.com

Refresh a program group

The ReloadProgGroup function instructs the Program Manager to unload and reload the group file. If you make modifications to program group files directly and wish to view the changes immediately, calling the ReloadProgGroup function forces the Program Manager to update the group.

Advanced developers can modify group files directly using the ReloadProgGroup function.

Add an item to a program folder

Call the AddFolderIcon function to add a program item to a folder. If the specified folder does not exist, it is created. Otherwise, the AddFolderIcon function adds the program item to the existing program folder. You can also specify program item attributes, including the application's command line, location, icon folder, shortcut key, and minimize flag.

Delete an item

Call the DeleteFolderIcon function to delete an existing program item.

Always call the ShowGroup or the ShowProgramFolder function to bring the program items into view before calling the DeleteFolderIcon function.

InstallShield
www.installshield.com

Get information about an item

Call the QueryProgItem function to check for the existence of a specific program item. If InstallShield finds the program item, the QueryProgItem function returns the item's attributes, such as the application's command line, location, icon folder, shortcut key, and minimize flag.

To see an example script that uses QueryProgItem, click here. This script tests whether a program item exists. If the item exists, its attributes are displayed. ShowGroup is called to open the specified folder prior to calling QueryProgItem on one of the folder's items. Last, the program item's name is changed and then restored.

InstallShield
www.installshield.com

Replace an item or change its properties

Call the ReplaceFolderIcon function to replace or update an existing item in a folder. The ReplaceFolderIcon function can change every attribute of the program item, including the item's name, command line, location, icon folder, icon index, shortcut key, and minimize flag.

The ReplaceFolderIcon function cannot move an item from one folder to another. It can only replace one item with another and/or change an item's attributes.

You can use special characters such as '"abc\\test.exe ( ), , ss,"' in the first three parameters of ReplaceFolderIcon. Just be sure to enclose the double quotation marks within single quotation marks.

Click here for an example script using ReplaceFolderIcon to change a program item's attributes.

Getting and changing an item's properties example script

```
   STRING szFolder, szItem, szNewItem, svCmdLine;
   STRING svWrkDir, svIconPath, svShortCutKey;
   NUMBER nvIconIndex, nvMinimizeFlag;

program

   Disable(BACKGROUND);

   szFolder = FOLDER_STARTMENU ^ "Programs" ^ "Accessories";
   szItem   = "Paint";

   ShowGroup (szFolder, SW_NORMAL);

   // Test the existence of the szItem in szFolder. If not found, exit.
   if (QueryProgItem(szFolder, szItem, svCmdLine, svWrkDir,
                    svIconPath, nvIconIndex, svShortCutKey,
                    nvMinimizeFlag) < 0) then
      MessageBox("QueryProgItem failed on 'Shortcuts'", SEVERE);
      abort;
   endif;

   // Display information about szItem.
   SprintfBox(INFORMATION, "",
              "Command Line: %s\n\nWorking Dir: %s\n\n" +
              "Icon Path: %s\n\nIcon Index: %ld\n\nShortcut key: %s",
              svCmdLine, svWrkDir, svIconPath, nvIconIndex,
              svShortCutKey);

   // Now change the name of szItem to szNewItem.
   szNewItem = "New " + szItem;
   ReplaceFolderIcon(szFolder, szItem, szNewItem, svCmdLine, svWrkDir,
                    svIconPath, nvIconIndex, svShortCutKey, REPLACE);

   ShowGroup (szFolder, SW_NORMAL);
   Delay(2);
   MessageBox("Icon name changed.", INFORMATION);

   // Now change the name of szNewItem icon back to the original szItem.
   ReplaceFolderIcon(szFolder, szNewItem, szItem, svCmdLine, svWrkDir,
                    svIconPath, nvIconIndex, svShortCutKey, REPLACE);

endprogram

   // Source file:  Is5gr010.rul
```

InstallShield
www.installshield.com

Long file names

Windows 95 and Windows NT support the use of long file names, allowing users to give folders and files more meaningful designations. The term "long file name" refers to both long file names and long paths.

Since all 16-bit and some 32-bit applications cannot recognize long file names, InstallShield provides functions for working with long file names:

LongPathFromShortPath
Creates a long file name from a short file name.

LongPathToQuote
Encloses a long file name with double quotes.

LongPathToShortPath
Creates a short file name from a long file name.

Long file names and double quotation marks

Under Windows 95, if you pass a long file name containing one or more space characters to the command line (such as in a DOS box or in the Command Line field in an icon's properties sheet) you must enclose the long file name in double quotation marks. The quotation marks convert the long file name to a string literal, allowing the command line to receive it as a single argument. Otherwise, the command line recognizes the space character as a delimiter.

Failing to use double quotation marks around a long path containing a space character causes the command line to misinterpret the command.

Double quotation marks must be removed from long file names (using LongPathToQuote) before they can be converted to short file names (using LongPathToShortPath).

Long file names and 16-bit applications

You cannot pass long file names as command line parameters to 16-bit applications. 16-bit applications require the short file name versions of long file names to function correctly.

Use the InstallScript long file name functions to perform the conversion when required. For example, if your setup writes file names to .ini files and 16-bit applications are expected to use these file names, you must convert them to short file names.

Long file names contain names longer than the conventional 8.3 (8 characters plus 3-character extension) short file name limit. Long file names allow the use of all the characters used in short file names.

In addition, long file names can contain plus signs (+), commas (,), semicolons (;), equal signs (=), left and right square brackets ([ ]), and spaces. Leading and trailing spaces are ignored. A fully qualified long file name can be up to 260 characters long.

Windows 95 and Windows NT create a short file name for every long file name. The short file name consists of the first six characters of the long file name, a tilde (~), and a number.

If you're creating a 16-bit setup (by selecting Windows 3.1 & 3.11 in the Media Build Wizard - Platforms panel), you cannot use long file names in your setup or setup script.

InstallShield
www.installshield.com

Adding icons to the Control Panel

How you add an icon to the Control Panel depends on your target platform and whether the icon you are installing launches a 16- or 32-bit program.

16-bit programs

Windows 3.1 and Windows 95

Place an entry in the [MMCPL] section of the Control.ini file. (See Adding an entry to the [MMCPL] section of Control.ini, below.)

Windows NT

Since the Windows NT Control Panel does not support 16-bit Control Panel applications, you must instead launch 16-bit programs from program folders.

32-bit programs with a .cpl file

Windows 95 or Windows NT

If your application has a .cpl file, install it in the Windows\System folder. The icon will automatically be loaded and displayed when the Control Panel is opened. Consult Microsoft documentation for more information about creating a .cpl file.

Windows 3.1

Since the Windows 3.1 Control Panel does not support 32-bit control panel applications, the application must be launched from a normal program folder.

32-bit programs without a .cpl file

Windows 95

Place an entry in the [MMCPL] section of the Control.ini file (see below).

Windows NT

Add an entry into the registry under HKEY_CURRENT_USER\Control Panel\MMCPL (see below).

Windows 3.1

The Windows 3.1 Control Panel does not support 32-bit control panel applications, therefore under this platform the application must be launched from a normal program folder.

Adding an entry to the [MMCPL] section of Control.ini (Windows 3.1 and Windows 95 only)

The Control.ini file can be found in the Windows folder. The key should have the following format:

```
<keyname>=<full path and file name of .cpl or .dll file>
```

The keyname will be ignored but must be unique for each Control Panel icon specified in this section. If two entries have the same keyname, only the first entry will display in the Control Panel.

For example, this call to the WriteProfString function would place an ODBC icon in the Control Panel:

```
WriteProfString( WINDIR ^ "Control.ini", "MMCPL", "ODBC",
                 szODBCTargetDir ^ "Odbcinst.dll" );
```

When using this method, the .dll or .cpl file does not need to be located in the Windows\System folder.

If you specify a DLL in this path, it must have the appropriate functions to be called by Control.exe. Consult Microsoft documentation for more information about creating Control Panel DLLs.

Adding an entry to the MMCPL key in the registry (Windows NT only)

The MMCPL key is found under HKEY_CURRENT_USER\Control Panel in the 32-bit registry. The value should have this format:

<keyname>=<full path and file name of .cpl or .dll file>

The keyname will be ignored but must be unique for each Control Panel icon specified in this section. If two entries have the same keyname, only the first entry will be used.

For example, the code below adds an ODBC icon to the Control Panel by calling the RegDBSetKeyValueEx function:

```
RegDBSetDefaultRoot(HKEY_CURRENT_USER);
RegDBSetKeyValueEx("Control Panel\\MMCPL", "ODBC",
                   REGDB_STRING, szODBCTargetDir ^ "Odbcinst.dll",-1)
```

When using this method, the .dll or .cpl file does not need to be located in the Windows\System folder.

If you specify a DLL in this path, it must have the appropriate functions to be called by Control.exe. Consult Microsoft documentation for more information about creating Control Panel DLLs.

Overview: Handling Files and Folders During Setup

InstallShield provides many functions you can use to manipulate text files, binary files, and folders. There are also several InstallScript functions you can call to perform specific tasks in batch (.bat) files, configuration (.sys) files, and initialization (.ini) files. For more information, refer to the individual subtopics, described below.

If you're creating a 16-bit setup (by selecting Windows 3.1 & 3.11 in the Media Build Wizard - Platforms panel), you cannot use long file names in your setup or setup script.

Basic File Management

Covers copying, deleting, finding, or renaming an existing file, as well as getting and setting a file's attributes.

Text Files

Explains working with text files and manipulating strings in them.

Binary Files

Explains working with binary files and reading and writing bytes to them.

Folders and Paths

Tells you how to create, delete and query folders, disks, and their contents.

Batch (.bat) Files

Explains modifying batch files.

Configuration (.sys) Files

Explains modifying configuration files.

Initialization (.ini) Files

Explains modifying initialization files.

InstallShield
www.installshield.com

The InstallScript basic file management functions

File management refers to the handling of existing files: copying, deleting, finding, or renaming a file, as well as retrieving and setting a file's attributes. Call these InstallScript functions to handle files during setup:

CopyFile
Copies a file from the source to the target folder.

DeleteFile
Removes a file in the target folder.

FindAllFiles
Searches a folder for all the files that match the specified criteria.

FindFile
Searches a folder for a specified file.

GetFileInfo
Retrieves file attributes, date, time, and size information for a file.

Is
Retrieves information about a file.

ParsePath
Allows you to separate a full path string without using the string manipulation functions.

RenameFile
Renames the given file, possibly to a different folder.

SetFileInfo
Sets file attributes: date, time, and size information for a file.

XCopyFile
Copies a file or files from the source to the target folder, including subfolders if desired.

Copy files

Call the CopyFile function to copy a single file from the source folder to the target folder. You can also rename a file when you call CopyFile.

In this example, a file named Myfile.txt is copied from the source folder to the target folder and renamed Mynewfil.txt:

```
CopyFile("Myfile.txt", "Mynewfil.txt");
```

Call the XCopyFile function to copy a file or multiple files and include any subfolders. You cannot rename a file when you call XCopyFile.

The following example illustrates copying a file from the source folder named Myfile.exe to the target folder. Subfolders are included in the copy operation.

```
XCopyFile("Myfile.exe", " Myfile.exe ", INCLUDE_SUBDIR);
```

InstallShield uses the SRCDIR and TARGETDIR system variables to find the source and target paths. You can change the values of SRCDIR and TARGETDIR with an assignment statement. Call the VarSave and VarRestore functions if you only want to change the values of SRCDIR and TARGETDIR temporarily.

InstallShield
www.installshield.com

Delete a file

Call the DeleteFile function to remove a specified file from the target folder. The DeleteFile function uses the contents of the TARGETDIR system variable as the path for the file. DeleteFile cannot delete read-only, system, or hidden files.

This example deletes Myfile.txt, if found:

```
if FindFile (szPath, "Myfile.txt", svResult) = 0 then
   TARGETDIR = szPath;
    DeleteFile ("Myfile.txt");
endif;
```

Rename a file

Call the RenameFile function to change the name of a file. RenameFile uses the contents of SRCDIR as the path for the file you are renaming, and the contents of TARGETDIR as the path for the renamed file. You cannot use wild card characters in this function.

To rename the file in the source folder and have it remain there, verify that SRCDIR and TARGETDIR contain the same path information. If SRCDIR and TARGETDIR have different values, the file is moved and renamed.

The example below renames C:\Program\Help Files\Usage.hlp to C:\Program\Help Files\Reference.hlp. Notice the use of the VarSave and VarRestore functions to change the SRCDIR and TARGETDIR values only temporarily for the file renaming.

```
VarSave (SRCTARGETDIR);
SRCDIR = "C:\\Program\\Help Files";
TARGETDIR = "C:\\Program\\Help Files";
RenameFile ("Usage.hlp", "Reference.hlp");
VarRestore (SRCTARGETDIR);
```

InstallShield
www.installshield.com

Find files

Call the FindFile function to search a folder for a specified file or for a given file specification. FindFile returns only the first file that matches the file specification, not all occurrences. For example, to make sure that Win.ini is present in C:\Win95:

```
if (FindFile("C:\\Win95", "Win.ini ", svResult) < 0) then
   MessageBox("File not found", INFORMATION);
endif;
```

Call the FindAllFiles function to search a folder and its subfolders for a file specification. The following example finds all .ini files in the Windows folder and below and displays the file names in message boxes.

```
nResult = FindAllFiles("D:\\Windows", "*.ini", svFileName,
                       RESET);
while (nResult = 0)
   MessageBox(svFileName, INFORMATION);
   nResult = FindAllFiles("D:\\Windows", "*.ini",
                          svFileName, CONTINUE);
endwhile;
```

InstallShield
www.installshield.com

Get a file's attributes

Call the GetFileInfo function to see the attributes, access rights, date, time, or size information for a file. You can only request one type of information each time you call GetFileInfo. For example, f you want to know both the size and access rights of a particular file, you need to call GetFileInfo twice.

The sample code below gets the file attributes and creation date of C:\Example\Myfile.txt and displays the results:

```
GetFileInfo("C:\\Example\\Myfile.txt", FILE_ATTRIBUTE,
            nvResult, svResult);
MessageBox(svResult, INFORMATION);
GetFileInfo("C:\\Example\\Myfile.txt", FILE_DATE,
            nvResult, svResult);
MessageBox(nvResult, INFORMATION);
```

InstallShield
www.installshield.com

Set a file's attributes

Call the SetFileInfo function to change the attributes, access rights, file creation date, or time of a particular file. You can only set one type of information when you call the SetFileInfo function. For example, if you want to change both the creation date and access rights of a particular file, you need to call SetFileInfo twice.

This example sets the file attribute of Myfile.txt to read-only and the creation date to September 15, 1994.

```
SetFileInfo("C:\\Example\\Myfile.txt", FILE_ATTRIBUTE,
            FILE_ATTR_READONLY, "");
szDate = "1994/09/15";
SetFileInfo("C:\\Example\\Myfile.txt", FILE_DATE,
            0, szDate);
```

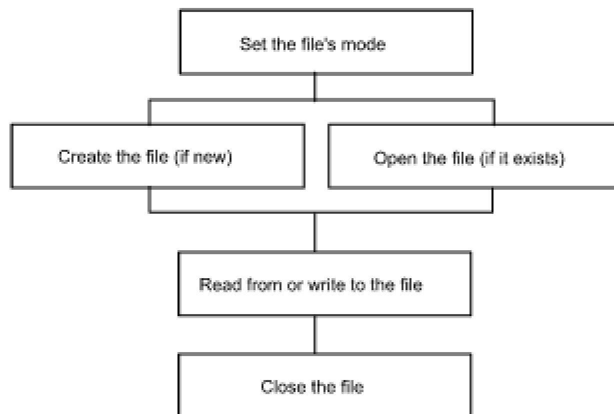
InstallShield
www.installshield.com

Overview: Working with Text Files

Text files contain ASCII information you can read, write, or print. Each line of a text file ends with the ASCII sequence for carriage return-line feed.

You may want to use text files in the script for a variety of reasons: to store information for a message box, to store a series of strings, to retrieve file names from another text file, and so on.

There are four steps to follow when working with text files.



These steps are detailed in these topics:

{button ,JI(`GETRES.HLP',`Setting_a_text_file_s_mode')} Set the file's mode

{button ,JI(`GETRES.HLP>(w95sec)',`Creating_a_new_text_file')} Create the file (if new)

{button ,JI(`GETRES.HLP>(w95sec)',`Opening_a_text_file')} Open the file (if it exists)

{button ,JI(`GETRES.HLP>(w95sec)',`Reading_from_a_text_file')} Read from the file

{button ,JI(`GETRES.HLP>(w95sec)',`Writing_to_a_text_file')} Write to the file

{button ,JI(`GETRES.HLP>(w95sec)',`Closing_a_text_file')} Close the file

InstallShield
www.installshield.com

Setting a text file's mode

Before you open or create a file, you must specify the mode of the file by calling the OpenFileMode function.

Use the FILE_MODE_NORMAL option to open a text file in read-only mode with the file pointer at the beginning of the file. If you are writing to the file, use the FILE_MODE_APPEND option to open the file in append mode with the file pointer at the end of the file, allowing you to add lines to the file.

When you create a file, InstallShield automatically opens it in append mode (even if you specify FILE_MODE_NORMAL) and OF_SHARE_DENY_NONE mode, meaning that other programs have read and write access to the files.

InstallShield
www.installshield.com

Creating a text file

To create an empty file, call OpenFileMode with either the FILE_MODE_NORMAL or the FILE_MODE_APPEND option. Then, call the CreateFile function to create the file.

For example, use this script fragment to create a text file named Myfile.txt in the C:\Example folder:

```
OpenFileMode(FILE_MODE_APPEND);
CreateFile(nvFileHandle, "C:\\Example", "Myfile.txt");
```

When you create a file, InstallShield automatically opens the file in append mode (even if you specify FILE_MODE_NORMAL) and OF_SHARE_DENY_NONE mode, meaning that other programs have read and write access to the files. If you have just created a particular file, the file pointer is at the beginning of the empty file. Do not attempt to open a newly created file with the OpenFile function.

Opening a text file

You must open a text file before reading from or writing to it. Before you can open an existing file, you must set the file's mode.

After you specify the mode, open the file by calling the OpenFile function.

For example, use this code to open C:\Example\Myfile.txt in append mode:

```
OpenFileMode(FILE_MODE_APPEND);
OpenFile(nvFileHandle, "C:\\Example", "Myfile.txt");
```

If you have just created a file using the CreateFile function, do not open the file. It is already open in append mode.

{button ,AL(`Creating a text file;Setting a text file mode',0,`',`')} See also

InstallShield
www.installshield.com

Reading from a text file

Call the GetLine function to read a line of text from a text file opened in read-only mode (FILE_MODE_NORMAL).

The first time you call GetLine it retrieves the first line in the file. Subsequent GetLine function calls retrieve subsequent lines in the file. The file pointer automatically increments one line each time. The string returned by the GetLine function does not contain the carriage return and line feed found at the end of the line.

The following example illustrates using the GetLine function to read the entire contents of a file named Myfile.txt, line by line. Notice that the file is opened in read-only mode.

```
   STRING svLine;
   NUMBER nFlag, nvFileHandle;

program

   OpenFileMode(FILE_MODE_NORMAL);
   OpenFile(nvFileHandle, "C:\\Example", "Myfile.txt");

   while (nFlag = 0)
      nFlag = GetLine(nvFileHandle, svLine);
      SprintfBox(INFORMATION, "", "Line is:\n%s", svLine);
   endwhile;

   CloseFile(nvFileHandle);

endprogram

   // Source file:  Is5gr011.rul
```

Writing to a text file

Call the WriteLine function to write a line to the end of a text file. You must open an existing file in FILE_MODE_APPEND mode in order to write to it. If you are writing to a newly created file, do not set the file mode or open the file—the CreateFile function automatically opens the new file in append mode, allowing you to write to the file.

The WriteLine function always places the new line of text at the end of the file. WriteLine adds a carriage return and a line feed at the end of each line it writes in the text file.

The following example illustrates using WriteLine function to add a line of text to an existing text file named Myfile.txt. Notice that the file is opened in append mode.

```
OpenFileMode(FILE_MODE_APPEND);
OpenFile(nvFileHandle, "C:\\Example", "Myfile.txt");
WriteLine(nvFileHandle, "Line I am adding to my file.");
```

InstallShield
www.installshield.com

Closing a text file

When you finish reading from or writing to the text file, you must call the CloseFile function to close the file. Once you call CloseFile, you can no longer read from or write to the file unless you set the mode and open the file again.

The following example illustrates closing a file after reading from it.

```
   STRING svLine;
   NUMBER nFlag, nvFileHandle;

program

   OpenFileMode(FILE_MODE_NORMAL);
   OpenFile(nvFileHandle, "C:\\Example", "Myfile.txt");

   while (nFlag = 0)
      nFlag = GetLine(nvFileHandle, svLine);
      SprintfBox(INFORMATION, "", "Line is:\n%s", svLine);
   endwhile;

   CloseFile(nvFileHandle);

endprogram

   // Source file:  Is5gr011.rul
```

InstallShield
www.installshield.com

The InstallScript functions for working with text files

These InstallScript functions are available for working with text files:

CloseFile
Closes an open file.

CreateFile
Creates a new file.

FileCompare
Compares the size, date, or version of two files.

FileGrep
Searches a text file for a specified string.

FileDeleteLine
Removes a line from a text file.

FileInsertLine
Inserts a line into a text file.

GetLine
Retrieves a line of text from the opened file.

OpenFile
Opens a file for input or output operations.

OpenFileMode
Sets the mode in which a subsequent OpenFile function call opens a file.

WriteLine
Writes a line of text to the end of a file opened in FILE_MODE_APPEND mode.

InstallShield
www.installshield.com

Search for a string in a text file

InstallShield
www.installshield.com

Call the FileGrep function to determine if a specific string exists in a text file. If it does, FileGrep returns the entire line and its line number. FileGrep does not work on binary files.

FileGrep does not require you to manually open or close the file. It automatically opens and closes the requested file.

InstallShield
www.installshield.com

Compare a file's size, date, or version

InstallShield
www.installshield.com

Call the FileCompare function to compare the size, date, or version of two files. Ver.dll, a Windows DLL, must reside on the target system in order to compare file version information.

Delete lines in a text file

Call the FileDeleteLine function to remove a specific line or a range of lines in a text file. FileDeleteLine does not work on binary files.

When using FileDeleteLine, there is no need to open or close the file with OpenFile or CloseFile.

InstallShield
www.installshield.com

Insert a line into a text file

InstallShield
www.installshield.com

Call the FileInsertLine function to insert a line using a line number in a text file. FileInsertLine does not work on binary files.

When using FileInsertLine, there is no need to open or close the file with OpenFile or CloseFile.

InstallShield
www.installshield.com

Replace a line in a text file

InstallShield
www.installshield.com

1. Call the FileGrep function to find the line (as a string). FileGrep returns the line number where the string was found.
2. Call the FileDeleteLine function to delete the line you want to replace.
3. Call the FileInsertLine function to insert the new line into the text file.

The following example illustrates these three functions in a script. It detects a line in a file using FileGrep, deletes the line using FileDeleteLine, and inserts a new string using FileInsertLine.

```
   STRING svLine;
   NUMBER nvLineNumber;

program

   // Set the source path.
   SRCDIR = "C:\\Example";

   FileGrep("Myfile.txt", "Line", svLine, nvLineNumber, RESTART);
   FileDeleteLine("Myfile.txt", nvLineNumber, nvLineNumber);
   FileInsertLine("Myfile.txt", "Insert this line with FileInsertLine.",
                  nvLineNumber, AFTER);

endprogram

   // Source file:  Is5gr012.rul
```

The InstallScript functions for manipulating strings

InstallShield provides you with several powerful functions for manipulating strings:

NumToStr
  Converts a number to a string.

StrCompare
  Compares one string to another.

StrFind
  Finds a string within another string.

StrGetTokens
  Gets a token from a string based on specified delimiters.

StrLength
  Determines the length of a string.

StrRemoveLastSlash
  Removes the last backslash from a text path.

StrSub
  Returns a substring of a given length.

StrToLower
  Converts a string to all lowercase.

StrToNum
  Converts a string to a number.

StrToUpper
  Converts a string to all uppercase.

For all functions which concern the relative positions of characters and substrings, the first character of a string is in position 0.

{button ,AL(`Overview: Working with Text Files',0,`',`')} See also

Convert a string to a number

Call the StrToNum function when you want to convert a string to a number, allowing it to be used in arithmetic expressions.

This code fragment converts the string "144000" into the numeric value 144000.

```
StrToNum(nvVar, "144000");
```

StrToNum will fail on a string with alphabetic characters.

InstallShield
www.installshield.com

Convert a number to a string

Call the NumToStr function when you want to convert a number to a string.

For example, if you wanted to display a numeric result in a MessageBox, which requires a string variable, you could call the NumToStr function to convert it:

```
nNum3 = nNum1 + nNum2;
NumToStr (svString, nNum3);
MessageBox ("The result is:  " + svString, INFORMATION);
```

InstallShield
www.installshield.com

Change the case of a string

Call the StrToLower function when you want to convert a string to all lowercase characters.

This code fragment converts the string "This Is A Test String" into "this is a test string" and assigns it to svTarget:

```
StrToLower(svTarget, "This Is A Test String");
```

Call the StrToUpper function when you want to convert a string to all uppercase characters.

This code fragment converts the string "This Is A Test String" into "THIS IS A TEST STRING" and assigns it to svTarget:

```
StrToUpper(svTarget, "This Is A Test String");
```

InstallShield
www.installshield.com

Comparing strings

Call the StrCompare function to perform a string comparison on two strings. Depending on the results of the comparison, the StrCompare function returns values in the following ranges:

- A value less than zero (<0) indicates that the first string is less than the second string. StrCompare also returns a negative value if the string you are searching for is not found.
- If the return value equals zero (=0) indicates that the two strings are identical.
- A value greater than zero (>0) indicates that the first string is greater than the second string.

This example compares the strings "This Is A Test String" and "This Is B Test String":

```
lResult = StrCompare("This Is A Test String",
                     "This Is B Test String");
SprintfBox(INFORMATION, "StrCompare Example",
           "StrCompare returned %ld.", lResult);
// The StrCompare function returns <0 since the first string is
// smaller than the second string.
```

InstallShield
www.installshield.com

Find a string within another string

Call the StrFind function to determine whether a string contains another string.

If it succeeds, StrFind returns a number which indicates the location of the first character in the substring, if found. The first position in the string is position 0.

This example looks for the string "Test String" in "This Is A Test String".

```
StrFind("This Is A Test String", "Test String");
```

{button ,JI(`GETRES.HLP>(w95sec)',`String_Operators@Langref.hlp')}   See also

InstallShield
www.installshield.com

Find a string's length

Call the StrLength function to determine the length of a string. StrLength returns the length in bytes.

This example returns the length of the string "This Is A Test String."

```
szString = "This Is A Test String";
lReturn = StrLength(szString);
SprintfBox(INFORMATION, "StrLength Example",
           "szString is %ld bytes long.", lReturn);
// This function returns 21. Save this value to another
// variable if you intend to use it later in the script.
```

InstallShield
www.installshield.com

Retrieve a substring from a string

Call the StrSub function to retrieve a substring of a specified length from another string,.

This example retrieves 6 bytes from the string, starting at byte 3.

```
StrSub(svSubStr, "D:\\Windev\\Include", 3, 6);
// svSubStr now contains "Windev"
```

InstallShield
www.installshield.com

Parse a string

InstallShield
www.installshield.com

Call the StrGetTokens function to parse a string into a list of tokens.

Note that if the delimiter character is the first character of the string, the first element of the list will be set to NULL. The string will then be parsed until the next delimiter character is reached. If the last character of the string happens to be the delimiter character, a null string will be placed at the end of the list.

This example parses the string specified in szString into the list listID.

```
szString = "C:\\Dos|D:\\Windows|G:\\Help\\Myhelp|F:\\Myapp";
szDelimiterSet = "|";
listID = ListCreate( STRINGLIST );
StrGetTokens( listID, szString, szDelimiterSet );
// This function populates the list with these tokens:
// C:\Dos
// D:\Windows
// G:\Help\Myhelp
// F:\Myapp
```

Remove a trailing backslash from a path string

Call the StrRemoveLastSlash function to remove the last backslash in a path or directory string.

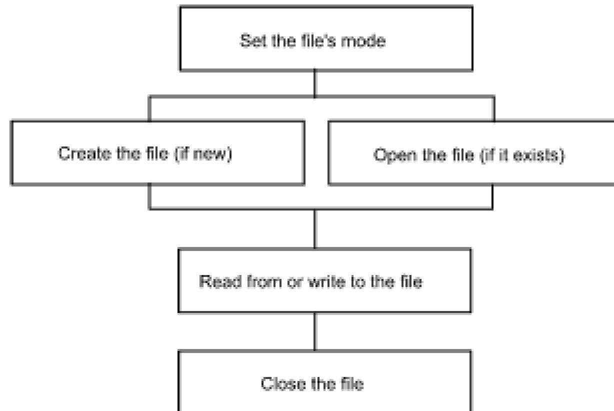
InstallShield
www.installshield.com

Overview: Working with Binary Files

InstallShield
www.installshield.com

Unlike text files, binary files do not use a carriage return and line feed character at the end of each line. Binary files are considered a series of numbers rather than characters. They can have many different file extensions, including .exe, .dll, .bin, and .msg.

There are four steps to follow when working with binary files:



These steps are detailed in these topics:

{button ,JI(`GETRES.HLP>(w95sec)',`Setting_a_binary_file_s_mode')} Set the file's mode

{button ,JI(`GETRES.HLP>(w95sec)',`Creating_a_binary_file')} Create the file (if new)

{button ,JI(`GETRES.HLP>(w95sec)',`Opening_a_binary_file')} Open the file (if it exists)

{button ,JI(`GETRES.HLP>(w95sec)',`Reading_from_a_binary_file')} Read from the file

{button ,JI(`GETRES.HLP>(w95sec)',`Writing_to_a_binary_file')} Write to the file

{button ,JI(`GETRES.HLP>(w95sec)',`Closing_a_binary_file')} Close the file

InstallShield
www.installshield.com

The InstallScript functions for working with binary files

InstallShield
www.installshield.com

These functions are available for working with binary files:

CloseFile
Closes an open file.

CopyBytes
Copies a specified number of bytes to a binary file.

CreateFile
Creates a new file name with the given name for output.

OpenFile
Opens a file for input.

OpenFileMode
Sets the mode to binary.

ReadBytes
Reads a specified number of bytes from a binary file.

SeekBytes
Allows you to position the file pointer in a binary file.

WriteBytes
Writes a specified number of bytes to a binary file at the current file pointer location.

InstallShield
www.installshield.com

Setting a binary file's mode

InstallShield
www.installshield.com

Before you create or open a binary file, you must specify the binary mode. Call the OpenFileMode function to specify FILE_MODE_BINARY (read/write) or FILE_MODE_BINARYREADONLY (read-only) mode. Use the latter when on a CD-ROM or read-only drive.

InstallShield
www.installshield.com

Creating a binary file

InstallShield
www.installshield.com

Once you have set the mode, call the CreateFile function to create the file. If a file with the same name exists in the target folder, InstallShield overwrites the existing file, deleting its contents.

For example, this code creates a new binary file named Myfile.bin in the C:\Example folder:

```
OpenFileMode(FILE_MODE_BINARY);
CreateFile(nvFileHandle, "C:\\Example", "Myfile.bin");
```

When you create a binary file, InstallShield automatically opens it in read/write mode (FILE_MODE_APPEND) and in OF_SHARE_DENY_NONE mode. If you have just created a binary file, you can use read and write functions immediately. Do not attempt to open a newly created file.

Opening a binary file

Before you can open a binary file, you must specify the binary file mode by calling the OpenFileMode function to specify FILE_MODE_BINARY (read/write) or FILE_MODE_BINARYREADONLY (read-only) mode. Use the latter when on a CD-ROM or read-only drive.

After you have specified the binary mode, call the OpenFile function to gain access to the binary file.

For example, this code opens a file named Myfile.bin in the C:\Example folder in read/write mode:

```
OpenFileMode(FILE_MODE_BINARY);
OpenFile(nvFileHandle, "C:\\Example", "Myfile.bin");
```

If you have just created a file with the CreateFile function, do not open the file. It is already open in read/write mode.

Reading from a binary file

Call the ReadBytes function to read a specific number of bytes from a binary file. Unless you adjust the pointer before using the ReadBytes function for the first time, it will retrieves bytes from the beginning of the file. Bytes are retrieved from the current position of the pointer, which is usually the beginning of the file or the end point of the previous ReadBytes call.

This example demonstrates calling the ReadBytes function to retrieve the first 75 bytes in the binary file Myfile.bin:

```
OpenFileMode(FILE_MODE_BINARY);
OpenFile(nvFileHandle, "C:\\Example", "Myfile.bin");
ReadBytes(nFileHandle, svBytes, 0, 75);
```

{button ,AL(`Get a byte from a string;Move the file pointer;Set a byte in a string',0,`',`')} See also

InstallShield
www.installshield.com

Writing to a binary file

InstallShield
www.installshield.com

Call the WriteBytes function to write a specified number of bytes to a binary file. WriteBytes also allows you to indicate where you want to add the new bytes. If you do not specify a location where you want to write the data, it is automatically added at the beginning of the file.

The example below writes a user name into a binary data file named C:\Mydatdir\Myfile.dat. The user name has been stored in szUserName. This example adds the string starting at position 170, using the SeekBytes function to set the current position in the file.

```
// Set the pointer to byte 170.
SeekBytes(nvFileHandle, 170, FILE_BIN_START);

// Retrieve the length of szUserName string (needed by WriteBytes).
nLength = StrLength(szUserName);

// Write the string to the file starting at the current position.
WriteBytes(nvFileHandle, szUserName, 0, nLength);
```

WriteBytes overwrites existing data in the file. If WriteBytes reaches the current end of file, it will append data to the file.

Call the CopyBytes function when you want to copy a specified number of bytes to a binary file. You can specify the offset index (starting point) in the source and destination strings. You can call the CopyBytes function to copy a portion of a string to another string.

{button ,AL(`Get a byte from a string;Move the file pointer;Set a byte in a string',0,`',`')} See also

InstallShield
www.installshield.com

Closing a binary file

InstallShield
www.installshield.com

When you finish reading from or writing to a binary file, call the CloseFile function to close the file.

Once you close the file with the CloseFile function, you can no longer read from or write to the file unless you set the file mode and open it again.

This example illustrates closing a binary file after reading from it.

```
OpenFileMode(FILE_MODE_BINARY);
OpenFile(nvFileHandle, "C:\\Example", "Myfile.bin");
SeekBytes(nFileHandle, 100, FILE_BIN_START);
ReadBytes(nFileHandle, svBytes, 0, 75);
CloseFile(nFileHandle);
```

Move the file pointer

Call the SeekBytes function to move the pointer's position in a binary file. The ReadBytes function leaves the pointer at the end point of the previous read operation. The SeekBytes function allows you to set the pointer anywhere in the binary file.

The sample code below moves the pointer 100 bytes from the beginning of the file, and then reads 75 bytes from the file:

```
SeekBytes(nFileHandle, 100, FILE_BIN_START);
ReadBytes(nFileHandle, svBytes, 0, 75);
```

Get a byte from a string

Assign a specific character of the string to a CHAR variable. For example, to see if the first position (element [0]) in svString is "L", use this code:

```
svString = "Language";
cVal = svString[0];
if cVal = "L" then
   MessageBox ('"L" is the first character in svString.', INFORMATION);
endif;
```

Set a byte in a string

Assign a CHAR variable to a specific character in the string. For example, to change "lex" to "hex", use this code:

```
svString = "lex";
cVal     = "h";
svString[0] = cVal;
MessageBox (svString + " is the new value of svString.",
            INFORMATION);
```

InstallShield
www.installshield.com

The InstallScript functions for handling folders and paths

InstallShield provides these functions that allow you to create, delete, and query folders, paths, disks, and their contents:

ChangeDirectory
Sets the specified folder as the current folder.

CreateDir
Creates a set of new folders with the given path.

DeleteDir
Deletes the specified folder and, optionally, all of its contents.

ExistsDir
Checks to see if the given folder exists.

ExistsDisk
Checks to see if the given disk exists in the system.

FindAllDirs
Finds all folders located under a given folder.

FindAllFiles
Finds all files that match the given file specifications. (Searches all subfolders, starting from the given folder.)

GetDir
Retrieves only the path from a full file specification.

GetDisk
Retrieves only the drive letter from a full file specification.

GetDiskSpace
Retrieves the amount of free disk space for a specific disk drive.

GetValidDrivesList
Retrieves a list of valid disk drives available on the target system.

InstallShield
www.installshield.com

Create a folder

Call the CreateDir function to create one or more folders on the target drive.

The CreateDir function may fail if the path is illegal, if the drive or any of its folders is write-protected, if the drive name is invalid, or if you do not have network privileges to create folders.

This example illustrates creating a folder called Temp_2 under the Windows folder:

```
CreateDir(WINDIR ^ "Temp_2");
```

{button ,AL(`How Windows NT security permissions affect a setup',0,`',`')} See also

Delete a folder

Call the DeleteDir function to delete one or more folders on the target system.

The first line of code below deletes an empty folder. The second line deletes a folder, its subfolders, and all files within the specified folder or folders.

```
DeleteDir("C:\\Win95\\Temp_2", ONLYDIR);
DeleteDir("D:\\Win95", ALLCONTENTS);
```

InstallShield
www.installshield.com

Make a folder the current folder

Call the ChangeDirectory function when you want to set the specified folder as the current folder.

When you set a folder as the current folder, you cannot delete it using the DeleteDir function. Specify a different folder as the current folder, then delete the desired folder.

InstallShield
www.installshield.com

Check if a folder exists

Call the ExistsDir function to check for the existence of a specified folder on the target system.

This example checks for the existence of the C:\Win95\Temp folder.

```
ExistsDir("C:\Win95\Temp");
```

InstallShield
www.installshield.com

Check if a drive exists

Call the ExistsDisk function to check for the existence of a specified disk drive on the target system.

This example checks for the existence of the B drive on the target system. The ExistsDisk function returns one of two values, EXISTS and NOTEXISTS.

```
if (ExistsDisk("B") = EXISTS) then
   MessageBox("Yes, the B drive exists", INFORMATION);
endif;
```

InstallShield
www.installshield.com

Find a folder

Call the FindAllDirs function to search a folder (and, if specified, all its subfolders) and return a string list of subfolders.

In this example, FindAllDirs returns all folders and subfolders on the C drive.

```
listDirs = ListCreate (STRINGLIST);
FindAllDirs ("C:\\", INCLUDE_SUBDIR, listDirs);
```

Parse a path or file name

Call the ParsePath function to separate a full path string into its component parts without using the string manipulation functions.

Call the GetDir function to retrieve only the path expression from a full file specification.

Call the GetDisk function to retrieve only the drive letter from a full file specification.

InstallShield
www.installshield.com

Get the amount of free disk space

Call the GetDiskSpace function to determine how much free disk space is available on a particular drive..

This example asks the user for a destination folder, calls the GetDisk function to retrieve the disk drive, and then calls GetDiskSpace to check how much free space is available on that drive:

```
AskPath(szMsg, szPath, svResultPath);

GetDisk(svResultPath, svDisk);
szDrive = svDisk;


GetDiskSpace(szDrive);
```

{button ,AL(`Parse a path or file name',0,`',`')}   See also

InstallShield
www.installshield.com

Find all drives on the target system

Call the GetValidDrivesList function to retrieve a list of valid disk drives available on the target system.

The InstallScript functions for working with the path buffer

InstallShield provides these functions for reading, writing, and editing the path buffer:

PathAdd
  Adds a directory to the path buffer.

PathDelete
  Deletes a directory from the path buffer.

PathFind
  Finds a directory in the path buffer.

PathGet
  Retrieves the current value of the path buffer.

PathMove
  Rearranges the path buffer.

PathSet
  Sets the current value of the path buffer.

InstallShield
www.installshield.com

Overview: Working with Batch Files

When you work with batch and configuration files, you can treat the files as normal text files, or you can use over 30 InstallScript functions designed specifically to modify batch and configuration files.

Ez and advanced batch functions

There are two classes of InstallScript batch file functions, Ez and advanced. Ez functions are quick and easy to use because they have many preprogrammed features. If you need greater flexibility and control over the changes you need to make to configuration and batch files, use the advanced functions.

When you use the Ez functions, you do not need to open the Autoexec.bat file and load it into memory before making changes. The Ez functions perform these operations for you. Also, if you use an Ez function to add a folder to the PATH statement, InstallShield does not duplicate a an already existing path.

Although the advanced batch and configuration functions give you more control, you are responsible for performing error checking. In addition, you must individually load the Autoexec.bat file using the BatchFileLoad function.

InstallShield does not display a message if a batch function fails; InstallShield returns a value < 0. The return value tells you if the function performed successfully. Remember to check the return value. You can display a message based on the result.

{button ,AL(`Autoexec.bat;The InstallScript advanced batch file functions;The InstallScript basic file management functions;The InstallScript Ez batch file functions',0,`',`')} See also

InstallShield
www.installshield.com

Autoexec.bat

Autoexec.bat is a batch file that automatically runs when the computer starts. The Autoexec.bat file executes a series of commands that sets up the computer for a particular environment.

Call the InstallScript Ez and advanced batch file functions to modify Autoexec.bat. InstallShield checks the target system and automatically determines which file to change. If you need to know specifically which file these functions will modify on the target system, call the BatchGetFileName function.

All batch functions use reference keys so that you can specify the relative placement of the line you're modifying. A reference key is a unique word or command that InstallShield uses to identify the statement. In the excerpt from Autoexec.bat provided below, the reference keys in each line below are highlighted in bold.

```
@D:\WINDOWS\AD_WRAP.COM
@ECHO OFF
D:\WINDOWS\SMARTDRV.EXE /Q
SET MOUSE=D:\WINDOWS
LOADHIGH D:\WINDOWS\MOUSE
PROMPT $P$G
PATH=C:\;D:\;D:\EXTRAWIN;D:\WINDOWS;
SET COMSPEC=C:\COMMAND.COM
SET DIRCMD=/OE/P
CALL C:\NW\STARTNW.BAT
D:\EXTRAWIN\DLCINTFC
```

Lines 4 and 5 have the same reference key, MOUSE. There is no conflict since line 4's statement identifies an environment variable and line 5's statement identifies a command. InstallScript batch file functions can differentiate between a command and an environment variable.

InstallShield
www.installshield.com

The InstallScript Ez batch file functions

Ez functions change the default Autoexec.bat file. InstallShield automatically determines the fully qualified path and name of the default Autoexec.bat file on the target system.

EzBatchAddPath
Adds a path directory to the PATH statement in the Autoexec.bat file. Also adds a value to any environment variable in the Autoexec.bat file.

EzBatchAddString
Adds a line of text to the Autoexec.bat file. The line can be positioned relative to another line in the file.

EzBatchReplace
Replaces a statement in the Autoexec.bat file.

InstallShield
www.installshield.com

The default batch file

During setup initialization, InstallShield stores the fully-qualified name of the Autoexec.bat file that was executed by the system during the boot sequence. In help topics, this internal variable is referred to as the default batch file. You can determine the current default batch file by calling the BatchGetFileName function. All of the Ez batch functions operate on the default batch file, as does BatchFileLoad when it is passed a null string.

The value assigned to the default batch file can be changed by calling BatchSetFileName. To use Ez batch functions to edit a file other than the Autoexec.bat file that was executed by the system during the boot sequence, simply call BatchSetFileName, passing to it the name of the file you want to edit. Although that call will change the value of the default batch file as it is known within InstallShield, it has no effect outside of the setup itself. When the system is rebooted, it will process the Autoexec.bat file in the boot folder in the usual manner.

InstallShield
www.installshield.com

The InstallScript advanced batch file functions

Advanced batch file functions provide greater flexibility and control for modifying batch files. When using these functions, you need to load and save the batch files you wish to modify. InstallShield automatically identifies a default system Autoexec.bat and loads it if you do not specify a batch file name. You must save the Autoexec.bat file using BatchFileSave in order for the changes to be written to the file.

BatchAdd
Adds an environment variable to a batch file.

BatchDeleteEx
Deletes a line identified by the key in the batch file.

BatchFileLoad
Loads a batch file into memory for editing using batch functions.

BatchFileSave
Saves the batch file loaded in memory to disk; you can also create a backup file with the BatchFileSave function.

BatchFind
Finds items in a batch file.

BatchGetFileName
Retrieves the path and name of the current default boot up Autoexec.bat file.

BatchMoveEx
Moves an item in a batch file.

BatchSetFileName
Determines which file the batch file functions will act upon.

Do not mix the Ez functions with Advanced functions. For example, if you are loading and saving an Autoexec.bat file using BatchFileLoad and BatchFileSave, do not use any Ez functions between the load and the save.

Load a batch file into memory

Before you use any advanced functions to change an Autoexec.bat file, you must load the file. To load the Autoexec.bat file, call the BatchFileLoad function with a null string in the szBatchFile parameter:

```
BatchFileLoad ("");
```

The statement shown above automatically loads the default system Autoexec.bat file. If you want to load a different batch file, enter the path and file name in the szBatchFile parameter. For example, if you want to load a file named C:\Mypath\Myautoex.bat, type:

```
BatchFileLoad("C:\\Mypath\\Myautoex.bat");
```

Do not call BatchFileLoad to load a batch file if you are using the Ez batch file functions to make changes to batch files. The Ez batch file functions load and save the file automatically.

Save changes to a batch file

Call the BatchFileSave function to save a batch file loaded in memory.

Do not call BatchFileSave to save a batch file if you are using the Ez batch file functions to make changes to batch files. The Ez batch file functions load and save the file automatically.

{button ,AL(`Load a batch file into memory',0,`',`')} See also

InstallShield
www.installshield.com

Save a backup copy of a batch file

Call the BatchFileSave function and specify an alternate file name in the szBackupFile parameter.

For example, to save the unmodified Autoexec.bat as Autoexec.bak, use this code:

```
BatchFileLoad ("");
BatchFileSave ("Autoexec.bak");
// You can now load the Autoexec.bat file as shown below:
BatchFileLoad ("");
```

Do not call BatchFileSave to save or rename a batch file if you are using the Ez batch file functions to make changes to batch files. The Ez batch file functions load and save the file automatically.

Add or modify an environment variable

If you are modifying a batch file with the InstallScript Ez batch file functions, call the EzBatchAddPath function to modify an environment variable. Or, you can call the EzBatchAddString function to add a new variable. Do not mix the Ez and advanced batch file functions.

Once you load a file with the BatchFileLoad function, you must use the advanced batch functions. To add a statement to the Autoexec.bat file, call the BatchAdd function. The BatchAdd function gives you complete control over what you add to the Autoexec.bat file.

For example, if you want to add the environment variable TEMP to the Autoexec.bat file, you would want to add a statement like this:

```
SET TEMP=C:\Windows\Temp
```

To add the TEMP variable after the PATH statement:

```
SET PATH=C:\Win32s
```

Use this code in your setup script:

```
   BatchAdd("TEMP", "C:\\Windows\\Temp", "PATH", AFTER);
```

When you set an environment variable, do not end the statement with a semicolon.

The BatchAdd function automatically adds the word "SET" before the environment variable and the equal sign (=) after the environment variable. The BatchAdd function does not add anything if szKey is a command statement (specified with the COMMAND option).

When you are finished making the changes to the Autoexec.bat file, call the BatchFileSave function to save the changes to disk. When you save the file, you can also create a backup file.

{button ,AL(`Set an environment variable under Windows NT',0,`',`')} See also

InstallShield
www.installshield.com

Delete lines in a batch file

Call the BatchDeleteEx function to delete a line (or multiple occurrences of a line) in a batch file. Make sure that you have loaded the batch file with the BatchFileLoad function.

BatchDeleteEx is an advanced batch file function. Do not mix the Ez and advanced batch file functions.

Overview: Working with Configuration Files

InstallShield provides functions that allow you to load a configuration file into memory, changing numeric values in configuration files, and installing a device driver, and checking for the existence of a reference key.

Do not delete a device driver that is presently loaded into memory. Other applications may use the driver you are removing. Remember to restart Windows any time you add a device driver to the target system.

When you use InstallScript functions to change configuration files, InstallShield checks the target system and automatically determines which file to change. If you need to know specifically which file these functions will modify on the target system, call the ConfigGetFileName function.

All configuration functions use reference keys. A reference key is a word or command in a configuration function that uniquely identifies the statement.

The reference keys are highlighted in bold:

```
DOS=HIGH,UMB
BREAK=ON
FCBS=1
LASTDRIVE=E
STACKS=0,0
DEVICE=D:\WINDOWS\HIMEM.SYS
DEVICE=D:\WINDOWS\EMM386.EXE L=B00-B7F NOEMS
DEVICEHIGH SIZE=160 C:\DOS\SETVER.EXE
DEVICEHIGH SIZE=3D70 C:\DXMC0MOD.SYS 001
SHELL=C:\COMMAND.COM C:\ /E:2048 /P
FILES=50
BUFFERS=15
```

The Ez and advanced configuration file functions

Like the InstallScript batch file functions, there are two groups of configuration file functions: Ez and advanced. The Ez configuration file functions perform specific tasks with a minimum amount of effort on your part.

The advanced configuration file functions offer you more flexibility and control, but you have to do more work in your setup script. For example, you must always load Config.sys and save the changes when using the advanced functions.

There may be cases when the more general file modification functions FileGrep and FileInsertLine will serve your needs better than the configuration functions. FileGrep, which works on line-oriented text files with lines shorter than 512 bytes, allows you to specify a search string, and returns the line number of the first line in the specified file that contains the search string. FileInsertLine can then use the line number returned by FileGrep to let you add, replace, or append a line relative to that line number.

InstallShield
www.installshield.com

The default system configuration file

During setup initialization, InstallShield stores the fully-qualified name of the Config.sys file that was executed by the system during the boot sequence. In help topics, this internal variable is referred to as the default system configuration file. You can determine the current default system configuration file by calling the ConfigGetFileName function. All of the Ez configuration functions operate on the default system configuration file, as does ConfigFileLoad when it is passed a null string.

The value assigned to the default system configuration file can be changed by calling ConfigSetFileName. To use Ez configuration functions to edit a file other than the Config.sys file that was executed by the system during the boot sequence, simply call ConfigSetFileName, passing to it the name of the file you want to edit. Although that call will change the value of the default system configuration file as it is known within InstallShield, it has no effect outside of the setup itself. When the system is rebooted, it will process the Config.sys file in the boot folder in the usual manner.

InstallShield
www.installshield.com

The InstallScript Ez configuration file functions

Use the functions listed in the table below to modify the Config.sys file. When using Ez functions you do not need to load and save the configuration files.

EzConfigAddDriver
Adds a device driver statement to the default Config.sys file.

EzConfigAddString
Adds a statement or line of text to the default Config.sys file.

EzConfigGetValue
Retrieves the value of a Config.sys parameter such as FILES, BUFFERS, etc.

EzConfigSetValue
Sets the value of a Config.sys parameter such as FILES, BUFFERS, etc.

InstallShield
www.installshield.com

The InstallScript advanced configuration file functions

The advanced configuration file functions provide greater flexibility and control when modifying configuration files. When using advanced configuration functions, you need to load and save the configuration file you wish to modify. InstallShield automatically identifies a default system Config.sys file and loads it if no other configuration file name is specified. You must save the configuration file in order for the changes to be written to the file.

ConfigAdd
Adds a statement to the configuration file.

ConfigDelete
Deletes an item in the configuration file.

ConfigFileLoad
Loads the specified Config.sys file into memory for editing.

ConfigFileSave
Saves the Config.sys file loaded in memory to disk. The original file is renamed to the backup file name specified. If you do not specify a backup file name, InstallShield replaces the original file with the modified file.

ConfigFind
Searches for an item in the Config.sys file.

ConfigGetFileName
Retrieves the fully qualified path and name of the default bootup Config.sys file used to boot the target system.

ConfigGetInt
Retrieves a value from the Config.sys file.

ConfigMove
Moves an item in the Config.sys file.

ConfigSetFileName
Specifies the path and file name of the Config.sys file to work with.

ConfigSetInt
Sets a value in the Config.sys file.

Do not mix the Ez functions with advanced functions. For example, if you are loading and saving a Config.sys file using ConfigFileLoad and ConfigFileSave, do not use any Ez functions between the load and the save.

InstallShield
www.installshield.com

Load a configuration file into memory

Before you use any advanced functions to change a Config.sys file, you must load the file into memory. To load the default Config.sys file, call the ConfigFileLoad function with a null string in the szConfigFile parameter:

```
ConfigFileLoad("");
```

The statement shown above automatically loads the default Config.sys file. If you want to load a different configuration file, enter the path and file name in the szConfigFile parameter. For example, if you want to load a file named C:\Mypath\Myconfig.sys, type:

```
ConfigFileLoad("C:\\Mypath\\Myconfig.sys");
```

You do not need to load Config.sys into memory if you are using the Ez configuration file functions. The Ez configuration file functions load and save the file automatically.

InstallShield
www.installshield.com

Save a backup copy of Config.sys

Before you make changes to the configuration file, you may want to back up the original file as Config.old. Call the ConfigFileSave function with the alternate file name in the szConfigFile parameter to back up the original file. For example:

```
ConfigFileLoad ("");
ConfigFileSave("Config.old");
// You can now load the Config.sys file as shown below.
ConfigFileLoad("");
```

InstallShield
www.installshield.com

Get the name of the default Config.sys

Call the ConfigGetFileName function to retrieve the path and name of the default system Config.sys file. For example:

```
ConfigGetFileName(svFileName);
MessageBox (svFileName, INFORMATION);
ConfigFileLoad (svFileName);
```

The three statements shown above retrieve the name of the default configuration file, display it to you, and then load the default file into memory.

InstallShield
www.installshield.com

Change a numeric value in Config.sys

Call the EzConfigSetValue function to check and reset commands that have numeric values in the Config.sys file. The EzConfigSetValue function does not add a space between the command and the value.

For example, if Config.sys must have FILES set to 50 (`FILES=50`), type:

```
EzConfigSetValue("FILES", 15);
```

Use the following format to change the number of buffers in Config.sys. If you must have at least 18 buffers, type:

```
EzConfigSetValue("BUFFERS", 18);
```

Do not mix the Ez functions with advanced functions. For example, if you are loading and saving a Config.sys file using ConfigFileLoad and ConfigFileSave, do not use any Ez functions between the load and the save.

Install a device driver

Call the EzConfigAddDriver function to install a Config.sys file. You must restart Windows any time you add a device driver to the target system. Call the RebootDialog or the SdFinishReboot function to allow the user to restart Windows once setup is over.

Suppose the driver is Mydriver.sys and you want Himem.sys loaded before Mydriver.sys because the driver uses Himem.sys. Add the line shown below to the Config.sys file:

```
DEVICE=C:\Myapplication\Mydriver.sys
```

The statement shown below is the statement you will use to add Mydriver.sys after Himem.sys in the Config.sys file:

```
   EzConfigAddDriver("C:\\Myapplication\\Mydriver.sys",
                     "Himem.sys",
                     AFTER);
```

Do not mix the Ez functions with advanced functions. For example, if you are loading and saving a Config.sys file using ConfigFileLoad and ConfigFileSave, do not use any Ez functions between the load and the save.

{button ,AL(`Install device drivers into an initialization file',0,`',`')}   See also

InstallShield
www.installshield.com

Check if a reference key exists

Call the ConfigFind function to determine whether the Config.sys file contains a certain environment variable or command.

Suppose you wanted to check if the environment variable RAMDRIVE exists in the Config.sys file. The ConfigFind function looks from the top of the file for RAMDRIVE. If it is found, a message box displays the result for you.

```
ConfigFind("RAMDRIVE", svResult, RESTART);
MessageBox(svResult, INFORMATION);
```

Do not mix the Ez functions with advanced functions. For example, if you are loading and saving a Config.sys file using ConfigFileLoad and ConfigFileSave, do not use any Ez functions between the load and the save.

InstallShield
www.installshield.com

Share.exe and Vshare.386

Share.exe is an MS-DOS terminate and stay resident (TSR) program providing file sharing and updating of in-memory information about resources.

With the advent of Windows for Workgroups 3.1, Vshare.386 took over as the Windows file sharing mechanism. When properly loaded, Vshare.386 overrides Share.exe, if present. Vshare.386 is loaded with the DEVICE=Vshare.386 statement in the [386Enh] section of System.ini. Windows 95 took file sharing a step further by incorporating Vshare.386 functionality into the operating system kernel. Under Windows 95 file sharing is enabled by means of the DEVICE=*VSHARE statement in the [386Enh] section of System.ini.

For more information, see the example scripts and descriptions for:

Detecting and adding Share.exe to the Autoexec.bat file

{button ,JI(`GETRES.HLP>(w95sec)',`Detecting_and_adding_Share_exe_to_Autoexec_bat')} Detecting and adding Share.exe to the Autoexec.bat file

{button ,JI(`GETRES.HLP>(w95sec)',`Changing_Share_exe')} Changing Share.exe parameters

{button ,JI(`GETRES.HLP>(w95sec)',`Detecting_and_adding_Vshare_386')} Detecting and adding Vshare.386

InstallShield
www.installshield.com

Detecting and adding Share.exe to Autoexec.bat

Before adding Share.exe to the target system, determine if it is already installed. Remember that the Autoexec.bat file may not be the only location you need to search for the existence of Share.exe. The user may be running Share.exe from a different batch file.

Call the GetSystemInfo function with the SHARE option to determine if the target system has Share.exe loaded.

This example illustrates using the GetSystemInfo function to detect the existence of Share.exe on the target system.

```
   BOOL nResult;
   STRING svResult;

program

   GetSystemInfo (SHARE, nResult, svResult);

   if (nResult = TRUE)  then
      SprintfBox (INFORMATION, "Share Information",
                  "Share.exe is loaded");
   else
      SprintfBox (INFORMATION, "Share Information",
                  "Share.exe is not loaded");
   endif;

endprogram

   // Source file:  Is5gr013.rul
```

If Share.exe is not loaded, nResult = FALSE. You can use the following section of code to install Share.exe on the target system. The section of script performs the following:

- Searches for Share.exe in the Autoexec.bat file. If it is found, a message is displayed announcing that it exists. If Share.exe is not found, the rest of the script section is performed.
- Searches for the presence of Share.exe by searching all the folders specified in the PATH environment variable.
- If Share.exe is not found in the Autoexec.bat file, the section of script shown below adds the Share.exe path to the target system's Autoexec.bat file.

```
   NUMBER nResult, ListID, nCheck;
   STRING szPath, szDelimiter, szString, szResult;

program

   BatchFileLoad("");

   // szPath parameter contains the value of the reference
   // key found in the batch file.
   nResult = BatchFind("Share.exe", szPath, RESTART|COMMAND);
   if (szPath != "") then
      MessageBox("Share.exe is already present" +
                 " in your Autoexec.bat",
                 INFORMATION);
      abort;
   endif;

   GetEnvVar("PATH", szPath);

   // MessageBox(szPath, INFORMATION);
   ListID = ListCreate (STRINGLIST);
```

```
   szDelimiter = ";";
   StrGetTokens (ListID, szPath, szDelimiter);

   nCheck = ListGetFirstString (ListID, szString);
   while ( nCheck = 0 )
      FindFile(szString, "Share.exe", szResult);
      if (szResult = "Share.exe") then
         SprintfBox(INFORMATION, "SHARE",
                    "Share.exe was found in the directory %s, so we will" +
                    " add Share.exe to your Autoexec.bat file.",
                    szString);
         szString = szString ^ "Share.exe";
         BatchAdd("", szString, "WIN", BEFORE | COMMAND);
         BatchFileSave("Autoexec.*");
         abort;
      endif;

      nCheck = ListGetNextString(ListID, szString);
   endwhile;

   ListDestroy(ListID);

endprogram

   // Source file:  Is5gr014.rul
```

Changing Share.exe

There may be times when you need to change a parameter of Share.exe to meet the application's needs. This is simply a matter of modifying Share.exe values in the Autoexec.bat file.

The example below searches Autoexec.bat for a line containing Share.exe. (The FileGrep function looks in SRCDIR for the file to search, and you can set SRCDIR to whatever value you need. The example uses the default.) If the example fails to find a line with Share.exe in it, it exits. If Share.exe is found, the example finds the /L: parameter in that line and tests to see if its value is less than 500. If the existing /L: parameter is less than 500, the example parses a copy of the existing line, places the tokens into a list, and replaces the existing /L: parameter with /L:500. Next, the example builds a new version of the line from the tokens stored in the list. Finally, the example uses the new line to replace the existing line in Autoexec.bat. This example assumes that Share.exe currently exists on the target system.

```
   STRING  szBatchFile, svReturnLine, szPath, szDelimiterSet, szString;
   STRING  szCommand, szSwitch, szTest, svResult, szFinal;
   NUMBER  nvLineNumber, ListID, nCheck, nVar, nResult, nPos;
   BOOL    bFirst;

program

   // Specify the file to search/modify.  The functions called in this example
   // will look in SRCDIR for the file.
   szBatchFile = "Autoexec.bat";
   szCommand = "Share.exe";
   szSwitch = "/L:";

   // Search szBatchFile (in SRCDIR) for the first line containing "Share.exe".
   // Return the line in svReturnLine, and line number in nvLineNumber.
   nResult = FileGrep(szBatchFile, szCommand,
                      svReturnLine, nvLineNumber, RESTART);

   // If FileGrep failed to find "Share.exe", inform user.  Also handle other
   // error return values FileGrep can return.
   if (nResult < 0) then
      MessageBox (szCommand + " was not found in " + szBatchFile, INFORMATION);
   endif;

   // Create a string list which you will use to hold the parsed parts
   // of svReturnLine.
   ListID = ListCreate(STRINGLIST);

   // Parse svReturnLine into tokens stored in a string list so you can find
   // szSwitch and so you can build the new (replacement) line.
   szDelimiterSet = " ";
   StrGetTokens(ListID, svReturnLine, szDelimiterSet);

   // Start with the first string token in ListID and begin testing for the
   // presence of szSwitch.
   nCheck = ListGetFirstString (ListID, szString);
   while (nCheck = 0)
      nPos = StrFind(szString, szSwitch);
      if (nPos = 0) then
         // If you find szSwitch, get the three characters that follow
         // it and see if they are greater than or equal to 500.
         StrSub(svResult, szString, 3, 3);
         StrToNum(nVar, svResult);

            if (nVar>=500) then
               // No need to change the command line.
               MessageBox("Change of SHARE parameter not required.",
```

```
                      INFORMATION);
            abort;
            // No need for an "else" because you exit if nVar>=500.
         endif;

      // If you find szSwitch followed by a value less than 500, replace
      // the string token in ListID with szSwitch followed by 500.
      ListCurrentString(ListID, szString);
      ListDeleteString(ListID);
      ListAddString(ListID, szSwitch + "500", BEFORE);
   endif;

   // If you didn't find szSwitch in the string token, get the
   // next one and loop to check for szSwitch again.
   nCheck = ListGetNextString(ListID, szString);

// Loop until nCheck is 0 (until no more string tokens in ListID).
endwhile;

// Initialize a string variable you will use to build the new line.
// Also initialize a Boolean variable to control the while loop.
szFinal = "";
bFirst = TRUE;

// Get the first string token in ListID.
nCheck = ListGetFirstString(ListID, szString);

// Begin building szFinal, one token at a time.
// While tokens are found and bFirst is TRUE...
while (nCheck = 0)
   if (bFirst) then
      // Place the first token into szFinal.
      szFinal = szString;
      bFirst  = FALSE;
   else
      // After adding first token (above), you can now
      // add remaining tokens, with a space between each one.
      szFinal = szFinal + " " + szString;
   endif;
   nCheck = ListGetNextString(ListID, szString);
endwhile;

// Display original line and new line.
MessageBox(szCommand + "command line has been changed from\n\n" +
"  " + svReturnLine + "\n\nto\n\n" + "  " + szFinal, INFORMATION);

nResult = FileInsertLine(szBatchFile, szFinal, nvLineNumber, REPLACE);

if (nResult <0) then
   MessageBox("FileInsertLine failed.", INFORMATION);
endif;

endprogram

// Source file:  Is5gr015.rul
```

InstallShield
www.installshield.com

Detecting and adding Vshare.386

Your setup might need to verify that Vshare.386 is present. If it is not found, then you will have to install it and make an entry in System.ini system. The example script below demonstrates checking for the presence of the Vshare.386 file and the DEVICE=Vshare.386 entry in System.ini, and adding an entry if required.

Since the VSHARE functionality is internal to Windows 95, you typically do not need to check if it is present. If for any reason the DEVICE=*VSHARE entry from System.ini were missing or removed, you could add or replace it with a call to AddProfString as in the example below.

```
   STRING szGrepStr, svReturnLine;
   STRING szTest, szResult, szFinal, szFile;
   NUMBER ListID, nCheck, nVar, nvLineNumber;
   BOOL bGrep;

program

   // Determine if Vshare.386 is present in the Windows\System folder
   // and act accordingly.
   if( FindFile( WINSYSDIR , "Vshare.386", szFile) = 0) then
      MessageBox("Found " + szFile, INFORMATION);
   else
      MessageBox("Could not find Vshare.386. Exiting...", INFORMATION);
      abort;
   endif;

   // Save the current values of SRCDIR and TARGETDIR.
   VarSave(SRCTARGETDIR);

   // Set SRCDIR to WINDIR because FileGrep uses SRCDIR as
   // the location for the file it searches.
   SRCDIR = WINDIR;

   // Check System.ini for a VSHARE device entry. If not found as
   // device=vshare.386, then display message accordingly.
   szGrepStr= "device=vshare.386";
   if ( FileGrep("System.ini", szGrepStr, svReturnLine,
                 nvLineNumber, RESTART) >= 0 ) then
      SprintfBox(INFORMATION, "Vshare.386 Entry Info.",
                 "%s is line number %ld", svReturnLine, nvLineNumber);
      // Restore SRCDIR and TARGETDIR to original values.
      VarRestore(SRCTARGETDIR);
      abort;
   else
      // Add Vshare.386 entry to System.ini if user okays it.
      if (AskYesNo("Okay to add Vshare.386 entry to System.ini?", NO) = YES) then
         AddProfString( WINDIR ^ "System.ini" , "386Enh",
                        "device", "vshare.386" );
      endif;
   endif;

   // Restore SRCDIR and TARGETDIR to original values.
   VarRestore(SRCTARGETDIR);

endprogram
```

```
// Source file:  Is5gr016.rul
```

InstallShield
www.installshield.com

Overview: Initialization Files

InstallShield
www.installshield.com

Initialization, or profile, files make it possible for you to provide user-specific information for networked applications. Initialization files are used in addition to the Autoexec.bat file, and provide more detailed information. You can store profile, configuration, national language, font, and window driver information in an initialization file. There are three types of initialization (.ini) files:

- Win.ini
- System.ini
- Private .ini files

Sections in .ini files follow this format:

```
[section name]
keyname1=value1
keyname2=value2
keyname3=value3
```

The `[section name]` is the name of the application. Place brackets on either side of the section (application) name. The next three lines are keynames with their related values. The keyname, equal sign, and the value must all be on the same line.

Win.ini

Win.ini allows you to set up user-specific information and characteristics on the computer. Items such as the default font, printer drivers, network settings, etc., are included in the initialization file. The items are then used as defaults in any Windows-based application. You can insert or change information in any Windows-supplied section, or you can add your own section(s) to the Win.ini file using InstallScript functions.

System.ini

System.ini contains global information Windows uses when it starts up. You will find keyboard information, device driver aliases, non-Windows application information, etc., in the System.ini file

Private initialization files

You can create your own initialization files that set up characteristics for the application. You can include items such as fonts, Windows drivers, and any other information in a private .ini file.

InstallShield
www.installshield.com

The InstallScript initialization file functions

InstallShield
www.installshield.com

InstallShield's initialization file functions obtain information from and copy information to initialization and profile files. An initialization file is a special ASCII file that contains keyname-value pairs. These InstallScript functions also allow you to access and update private initialization file and system initialization files.

AddProfString
: Adds a non-unique key in a section of the .ini file.

GetProfInt
: Returns an integer value in a section from the .ini file.

GetProfString
: Returns a character string from a section of the .ini file.

ReplaceProfString
: Replaces a non-unique key in a section of the .ini file.

WriteProfString
: Writes a character string to a profile .ini file.

InstallShield
www.installshield.com

Write to an initialization file

InstallShield
www.installshield.com

Call the WriteProfString function to write a string to a specific .ini file. If the file does not exist, then WriteProfString creates it for you. The WriteProfString function checks for the existence of the key you specify; if it finds the key, it replaces the value. Otherwise, it adds the key and value to the section. If the section you specify does not exist, the WriteProfString function creates it for you.

You can use the WriteProfString function to write integer values to the .ini file by converting the integer to a string before writing.

Suppose you wanted to set the value of TIMEOUT to be 100 in My.ini. The section you wanted to add this keyname to looks like this before your call to WriteProfString:

```
[MYSECT]
mouse.drv = lmouse.drv
```

Then call WriteProfString to add your keyname-value pair to this section:

```
WriteProfString("My.ini", "MYSECT", "TIMEOUT", "100");
```

The section looks like this after your call to WriteProfString:

```
[MYSECT]
mouse.drv = lmouse.drv
timeout=100
```

InstallShield
www.installshield.com

Add a line to an initialization file

InstallShield
www.installshield.com

Call the AddProfString function to add a profile string to an .ini file. The AddProfString function adds the `KEY=VALUE` line to the top of the section you specify; it does not check for the existence of the key and then replace it. The AddProfString function is useful when you require multiple instances of the key with different values (such as the DEVICE keyword).

This example illustrates how to use the AddProfString function to add the keyname DEVICE with a value of T:\Msvc\Bin\Mmd.386 in the [386Enh] section of an .ini file. Before the AddProfString function adds the line, the section was expressed as shown below:

```
[386Enh]
device=*vmcpd
device=*reboot
```

Call AddProfString like this to add the desired line:

```
   AddProfString("My.ini", "386Enh", "device", "T:\\Msvc\\Bin\\Mmd.386");
```

After the AddProfString function adds the line, the section looks like this:

```
[386Enh]
device=T:\MSVC\BIN\mmd.386
device=*vmcpd
device=*reboot
```

InstallShield
www.installshield.com

Get integer values from an initialization file

InstallShield
www.installshield.com

Call the GetProfInt function to retrieve an integer value from an .ini file.

The following example retrieves the value of `TIMEOUT` located in the `[MYSECTION]` section of the D:\Mydir\My.ini file.

```
GetProfInt("D:\\Mydir\\My.ini", "MYSECTION", "TIMEOUT", nvValue);
```

InstallShield
www.installshield.com

Get string values from an initialization file

InstallShield
www.installshield.com

Call the GetProfString function to retrieve a string value from an .ini file.

The following example gets the string value associated with the `DEVICE` keyname located in the `[MYSECT]` section of the D:\Mydir\My.ini file. Remember that the GetProfString function will only retrieve the value of the first instance of the keyword.

```
GetProfString("D:\\Mydir\\My.ini", "MYSECT", "DEVICE", svResult);
```

InstallShield
www.installshield.com

Change a string in an initialization file

InstallShield
www.installshield.com

Call the ReplaceProfString function to replace a profile string in an .ini file. The ReplaceProfString function allows you to replace values of duplicate keys (non-unique keys) such as DEVICE.

The following example replaces the value of the keyname `DEVICE` in the `[MYSECT]` section of the My.ini file with another value. Assume that the My.ini file looks like this before calling the ReplaceProfString function:

```
device=T:\Msvc\Bin\Mmd.386
device=*vmcpd
device=*reboot
```

Call the ReplaceProfString function:

```
ReplaceProfString("My.ini", "MYSECT", "device",
                  "T:\\Msvc\\Bin\\Mmd.386", "T:\\Msvc\\Bin\\Cvwl.386");
```

This is how the section would look after calling ReplaceProfString:

```
device=T:\MSVC\BIN\cvwl.386
device=*vmcpd
device=*reboot
```

InstallShield
www.installshield.com

Check if a keyname exists

InstallShield
www.installshield.com

Call the GetProfString function to see if the keyname exists and what its value is.

For example, suppose you wanted to see if the `[boot]` section of Myuser.ini, shown below, contained the line `386grabber=vga.3gr`.

```
[boot]
shell=progman.exe
mouse.drv=mouse.drv
network.drv=netware.drv
keyboard.drv=keyboard.drv
system.drv=system.drv
fonts.fon=vgasys.fon
display.drv=supervga.drv
drivers=mmsystem.dll
```

First call GetProfString to see if the keyname is found. If it is not present, call the AddProfString function to create the keyname-value pair. If the keyname is present but its value is not vga.3gr, then call the ReplaceProfString function to replace the current value.

```
   lReturn = GetProfString("C:\\Mydir\\Myuser.ini", "boot",
                           "386grabber", svResult);

   if (lReturn < 0) then
      AddProfString("C:\\Mydir\\Myuser.ini", "boot",
                    "386grabber", "vga.3gr");
   elseif (svResult != "vga.3gr") then
      ReplaceProfString("C:\\Mydir\\Myuser.ini", "boot", "386grabber",
                        svResult, "vga.3gr");
   endif;
```

This is how this section of Myuser.ini looks after the above statements execute:

```
[boot]
shell=progman.exe
mouse.drv=mouse.drv
network.drv=netware.drv
keyboard.drv=keyboard.drv
system.drv=system.drv
fonts.fon=vgasys.fon
display.drv=supervga.drv
drivers=mmsystem.dll
386grabber=vga.3gr
```

InstallShield
www.installshield.com

Install device drivers into an initialization file

InstallShield
www.installshield.com

Call the WriteProfString function to install a device driver into an initialization file. Remember to restart Windows any time you add a device driver. The example below installs the driver Vshare.386 into the `[386Enh]` section of a System.ini file.

```
WriteProfString(WINDIR ^ "System.ini", "386Enh",
                "DEVICE", "vshare.386");
```

Or, you can call the AddProfString function to install device drivers into an initialization file. Use the AddProfString function when you're adding non-unique keys, such as those found in the `[386Enh]` section of the System.ini file.

For example, to install the device driver "mmd.386," use the following:

```
AddProfString(WINDIR ^ "System.ini", "386Enh",
              "device", "T:\\Msvc\\Bin\\mmd.386");
```

{button ,AL(`Install a device driver',0,`',`')} See also

InstallShield
www.installshield.com

Get all keynames in an entire section

InstallShield
www.installshield.com

Call the GetProfString function with a null string in the third parameter to get all keynames under a given section. The keynames are separated with null characters ("").

This example gets all the keynames under the `[colors]` section in Win.ini. The keynames are then placed into a list and displayed one at a time:

```
szPath = WINDIR ^ "Win.ini";
// Get a section of the Win.ini file.
GetProfString(szPath, "colors", "", svResult);

// Create a string list.
listID = ListCreate(STRINGLIST);

// Retrieve each token from svResult and place it in the list.
szDelimiterSet = "";
StrGetTokens(listID, svResult, szDelimiterSet);

// Retrieve each token and display it.
nCheck = ListGetFirstString(listID, szString);
while (nCheck = 0)
   SprintfBox(INFORMATION, "[colors] Keynames",
              "Keyname: '%s'", szString);
   nCheck = ListGetNextString(listID, szString);
endwhile;
```

InstallShield
www.installshield.com

Overview: Network Installations

InstallShield is your complete solution for creating setups for all Windows platforms. InstallShield also offers special features for installing your product over a network connection.

In addition, there are restrictions and issues involved in running a setup over a network. This section contains information to help you handle these situations.

InstallShield
www.installshield.com

InstallShield Silent

InstallShield Silent allows automated electronic software distribution, also known as silent installation. With InstallShield Silent, there is no need for a user to monitor the installation and provide input via dialog boxes. An InstallShield Silent installation runs on its own, without any end-user intervention.

You must launch InstallShield Silent with the Setup.exe -s command line parameter. To comply with Windows 95 logo requirements, a silent installation must create a response file in which the default installation options are selected.

You can run your setup with the Setup.exe -r parameter to select setup options and automatically record the InstallShield Silent Response File, or you can create your own from scratch. To view a real world example of a response file, refer to the Setup.iss file located on InstallShield installation's Disk1. In addition, a description of the response file format is provided in Create the response file.

Steps to create a silent installation

Follow these steps to create a silent installation:

1. Create the setup
2. Create the response file
3. Play back the silent installation
4. Check for errors

InstallShield
www.installshield.com

Create the setup

Create the setup program in a typical manner, either by modifying an InstallShield template project or by creating a setup with the Project Wizard. Compile and test the script in the normal (non-silent) manner.

You can easily modify your setup script logic to include flow control based on whether or not InstallShield Silent is running. InstallShield provides a system variable called MODE which contains the installation's current mode. The MODE variable will contain one of the following constants:

- SILENTMODE—Indicates the installation is running in silent mode.
- NORMALMODE—Indicates the installation is running in normal mode.

You can use the MODE system variable in if statements to control the flow of your script based on mode, as shown below.

```
if (MODE = SILENTMODE) then
   // Perform silent installation actions and events.  .  . ;
else
   // Perform normal installation actions and events. . . ;
endif;
```

All InstallShield built-in and Sd dialogs automatically handle the values stored in the InstallShield Silent Response File (.iss file). If you are creating custom dialogs, you will need to call SilentReadData to handle the dialog box's return values in silent mode.

After you have created or modified the setup program, the next step in creating a silent installation is to create the response file.

Create the response file

A normal (non-silent) installation receives the necessary input from the user in the form of responses to dialog boxes. However, a silent installation does not prompt the user for input. A silent installation must get its user input from a different source. That source is the InstallShield Silent Response File (.iss file).

A response file contains information similar to that which an end user would enter as responses to dialog boxes when running a normal setup. InstallShield Silent reads the necessary input from the response file at run time.

The format of response files resembles that of an .ini file, but response files have .iss extensions. A response file is a plain text file consisting of sections containing data entries.

There are two ways in which you can create an InstallShield Silent Response File: you can run the setup and have InstallShield record and create the response file for you, or you can write the response file from scratch.

Recording a response file

You have the option of letting InstallShield create the response file for you. Simply run your setup with the Setup.exe -r command line parameter. InstallShield will record all your installation choices in Setup.iss and place the file in the Windows folder.

All InstallShield built-in and Sd dialog box functions are designed to write values into the Setup.iss file when InstallShield runs in record mode (Setup -r). If you are creating custom dialogs, you will need to call SdMakeName and SilentWriteData to add sections and dialog data to the response file when setup runs in record mode. Refer to the Sd dialogs' source code in the <InstallShield location>\Include folder for examples of using these functions to write to Setup.iss. Please read the following section for more information about what data to add to Setup.iss when calling SdMakeName and SilentWriteData.

Manually creating a response file

You can also create the response file completely by hand. As mentioned, the Setup.iss file is similar to an .ini file. The sections of an InstallShield response file must be in the following order:

1. Silent Header Section
2. Application Header Section
3. Dialog Sequence Section
4. Dialog Data Sections (one per dialog)

Section names are contained in square brackets, as in `[InstallShield Silent]`.

Data entries follow their section names, and consist of `<name=value>` pairs, as in `Dlg0=Welcome-0`.

Follow these steps to create the response file:

1. Create a text file named Setup.iss using any text editor.
2. Enter the silent header into Setup.iss.
3. Enter the application header into Setup.iss.
4. Enter the dialog sequence into Setup.iss.
5. Enter the dialog data into Setup.iss.
6. Save and close Setup.iss.

A sample response file is included to help familiarize you with the format.

InstallShield
www.installshield.com

Sample response file

The following response file is the .iss file for a silent InstallShield installation. It is located on distribution Disk1.

```
[InstallShield Silent]
Version=v5.00.000
File=Response File

[Application]
Name=InstallShield5 Professional
Version=5.00.000
Company=InstallShield

[DlgOrder]
Dlg0=SdBitmap-0
Count=8
Dlg1=Welcome-0
Dlg2=SdRegisterUser-0
Dlg3=AskDestPath-0
Dlg4=SetupType-0
Dlg5=SelectFolder-0
Dlg6=SdStartCopy-0
Dlg7=SdFinish-0

[SdBitmap-0]
Result=1

[Welcome-0]
Result=1

[SdRegisterUser-0]
szName=John Doe
szCompany=InstallShield Corporation
Result=1

[AskDestPath-0]
szPath=C:\Program Files\InstallShield\InstallShield5 Professional
Result=1

[SetupType-0]
Result=301
szDir=C:\Program Files\InstallShield\InstallShield5 Professional

[SelectFolder-0]
szResultFolder=InstallShield
Result=1

[SdStartCopy-0]
Result=1

[SdFinish-0]
Result=1
bOpt1=1
bOpt2=0
```

InstallShield
www.installshield.com

Response file silent header

All response files begin with a response file silent header. The response file silent header allows InstallShield to identify the file as a legitimate InstallShield response file. It also helps to verify that the response file corresponds to an installation created with the proper version of InstallShield.

The format of the silent header is shown below. Enter the following lines at the beginning of your Setup.iss file:

```
[InstallShield Silent]
Version=v5.00.000
File=Response File
```

The `Version=v5.00.000` line indicates the version of the InstallShield Silent response file, *not* the version of InstallShield. Use v5.00.000 in all response files. Future versions of InstallShield that use later response file versions will be able to read earlier response file versions, so response file backward compatibility will be maintained.

InstallShield
www.installshield.com

Response file application header

The response file application header is the second block of information in the response file, immediately following the response file silent header. The response file application header allows installation developers to identify response files visually. It is not used by the installation script or by Setup.exe. Installation developers can use the application header to identify exactly which installation the response file is for, since it is often difficult to determine this by looking at the dialog data.

The format of the application header is shown below. The values assigned to Name, Version, and Company are derived from the values written to the registry in the call to the InstallationInfo function in your installation script. Enter the following lines into your Setup.iss file below the silent header:

```
[Application]
Name=<ProductKey from InstallationInfo>
Version=<VersionKey from InstallationInfo>
Company=<CompanyKey from InstallationInfo>
```

InstallShield
www.installshield.com

Response file dialog sequence

The third block of information in the response file, after the silent header and the application header, is the response file dialog sequence. The dialog sequence section lists all dialogs you would need to use in a normal installation (including custom dialog boxes), in the order in which they would appear. The dialogs are listed under the section heading [DlgOrder].

The dialog numbering sequence begins at 0. There is no limit to the number of dialogs you can list.

The order and the number of dialogs is significant. When InstallShield Silent runs, if either the number or the order of the dialogs does not match the order or the number of the dialogs in the non-silent installation, the silent installation fails and the log file records the failure. Make one entry for each occurrence of a dialog. A given dialog may be listed more than once if it appears more than once in the installation.

The format for a dialog sequence entry is Dlg<#>=<DialogIdentifier>. Dlg<#> consists of the letters Dlg and a sequence number. The first dialog in the installation is Dlg0. Each dialog after that increments the value of <#> by one.

<DialogIdentifier> is in the form functionname-<#>, where functionname is the name of the function that created the dialog, and <#> represents the sequential order of that particular dialog in the installation.

For custom dialog boxes, you can use any unique alphanumeric name for the functionname portion of <DialogIdentifier>. For example, you could refer to a custom dialog box as MyDialog. If the tenth dialog box in the installation were the second occurrence of the custom dialog box MyDialog, there would be an entry in the dialog sequence section thus:

```
Dlg9=MyDialog-1
```

The <DialogIdentifier> expression will be used to identify the dialog data section for the dialog.

Always end the dialog sequence section with a statement of the form Count=<number of dialogs> that specifies the exact number of dialogs listed in the dialog sequence section. Count every entry. Since dialog numbering begins with 0, the value of Count will always be 1 greater than the highest <#> value for a dialog sequence.

The example below lists two dialogs, the Welcome dialog and the AskOptions dialog. Enter your dialog sequence list into Setup.iss as shown in the example below.

```
[DlgOrder]
Dlg0=Welcome-0
Dlg1=AskOptions-0
Count=2
```

Response file dialog data

The last block of information in a response file is the response file dialog data. The response file dialog data is a collection of sections containing the values returned by each dialog identified in the dialog sequence section. Each dialog has its own section. The values listed are the same values that the dialog returns in a normal, user input-driven installation. You can also create dialog data sections for custom dialog boxes.

Dialog data section format

```
[<DialogIdentifier>]
Result=value
Keyname1=value
Keyname2=value
```

The format for a dialog data section is shown above. The [<DialogIdentifier>] section header identifies the specific dialog and is followed by the dialog data entry list. <DialogIdentifier> is the same expression used to list the dialog box in the dialog sequence section.

Data entry items are in the format keyname=value. The keyname is a name for a value returned by a dialog and recorded in the response file. All dialogs, including custom dialogs, return a value for the keyname Result, reflecting the button that was clicked to end or exit the dialog. Many dialogs will also set or return a value in a variable.

For example, in a non-silent installation the AskDestPath dialog returns the destination location in the svDir parameter. The line in the script might look like the following:

```
nResult = AskDestPath( szTitle, szMsg, svDir, 0 );
```

The corresponding dialog data entry in the Setup.iss file for a silent installation might look like the following:

```
[AskDestPath-0]
Result=1
szPath=C:\Program Files\InstallShield\InstallShield5
```

In the above example, Result=1 is equivalent to clicking the Next button in the dialog, and szPath=C:\Program Files\InstallShield\InstallShield5 is the value returned in the svDir parameter of AskDestPath.

The name of the variable used in the script is *meaningless* relative to the Setup.iss file. However, in the Setup.iss dialog data sections, each built-in and Sd dialog has its own set of keynames which map to its parameters. The keynames are important and must be exact as defined for each dialog. Refer to Dialog data keynames list, below.

When using custom dialog boxes, you must create a dialog data entry for the Result keyname for each dialog box, plus entries for any other values set or returned by the custom dialog boxes. Use the Dialog data keynames list, below, as an indicator of how to create keyname=value expressions for your custom dialog boxes. Call GetProfString or GetProfInt to read the dialog data information for the custom dialog. GetProfString and GetProfInt allow you to specify the .iss file, the section, the keyname, and they return the value assigned to that keyname.

Result standard values

All dialogs, including custom dialogs, return a keyname value called Result, indicating which push button was clicked to end the dialog. Unless otherwise indicated, the Result standard values are:

- 12 for the Back button
- 1 for the Next or OK button

Recording component and subcomponent selections

Some dialog box functions allow the user to select components and subcomponents. There are three kinds of dialog data keyname entries used to record component and subcomponent selections in response files: type, count, and <#> (described below).

Every set of component selections and every set of subcomponent selections has one type keyname entry, one count keyname entry, and as many <#> keyname entries as are required to document each individual component or subcomponent selection.

When creating keynames to record component selections, precede the type, count, and <#> keyname entries with the word "Component", thus:

```
Component-type
Component-count
Component-0
```

When creating keynames to record subcomponent selections, precede the type, count, and <#> keyname entries with the name of the component to which the subcomponents belong, thus:

```
Program Files-type
Program Files-count
Program Files-0
Program Files-1
```

To create complete value entries, use the equal sign to attach the values to the keynames. (The types of values assigned to each kind of keyname are described below.) The following example shows complete value entries for two components, Program Files and Binary Files, and two subcomponents of Program Files, Executables and Support Elements:

```
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Binary Files
Program Files-type=string
Program Files-count=2
Program Files-0=Executables
Program Files-1=Support Elements
```

Type keyname entry

The type keyname indicates the data type of the components or subcomponents list. Since InstallShield dialog boxes currently use only string lists for components and subcomponents lists, type is always equal to "string", as in `Component-type=string`. Future versions may use number lists, in which case type could equal "number".

Count keyname entry

Count is equal to the number of selections for a given component or subcomponent entry. For example, if two components were selected for installation in the ComponentDialog dialog box, the count dialog data entry would be `Component-count=2`. If two subcomponents of the Program Files component were selected, there would be a `Program Files-count=2` entry.

<#> keyname entry

The number portion of the <#> keyname entry is simply a sequential (one-up) number, beginning with 0, that numbers each recorded component or subcomponent selection. Since numbering begins with 0, the greatest number value will always be one less than the value of count.

The value assigned to a <#> keyname entry is the selected component's or subcomponent's visible name (the string passed as the second parameter to ComponentAddItem when the components or subcomponents list was built).

For example, assume the ComponentDialog dialog box offers the user a component selection of Program Files, Help Files, Sample Files, and Utilities. If the user selects Program Files and Help Files, then the dialog data section for that instance of the ComponentDialog dialog box will have two list item entries and will look something like this:

```
[ComponentDialog-0]
szDir=C:\MYAPP\FILES
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Help Files
Result=1
```

The following example shows how subcomponent list selections are recorded. The example is for an instance of the SdComponentMult dialog box. The example shows that two components—Program Files and Help Files—were selected. It also shows that four subcomponents were chosen—Main Files, Aux. Files, Main Help, and Tutorial Files. Main Files and Aux. Files are subcomponents of Program Files, and Main Help and Tutorial Files subcomponents of

Help Files.

```
[SdComponentMult-0]
Component-type=string
Component-count=2
Component-0=Program Files
Component-1=Help Files
Program Files-type=string
Program Files-count=2
Program Files-0=Main Files
Program Files-1=Aux. Files
Help Files-type=string
Help Files-count=2
Help Files-0=Main Help
Help Files-1=Tutorial Files
Result=1
```

Dialog data keynames list

The dialog data keynames for the InstallShield dialogs are listed in the table below. The first column contains the dialog names. The second column lists the keynames applicable to each dialog. The third column contains descriptions of the values associated with the keynames.

Dialog	Keyname	Description
AskDestPath-<#>	Result	Standard values
	szPath	The path which is set in the edit field
AskOptions-<#>	Result	Standard values

AskOptions is special because the number of return values can change based on the script call. You can use a sequence of keynames of the form Sel-<#>, where <#> is the number of the selection variable. Numbering begins with 0. The number of Sel-<#> entries should match the number of variables (check boxes/radio buttons) in the particular call to AskOptions. Refer to the following examples:

Dialog	Keyname	Description
	Sel-0	Maps to first selection variable in AskOptions
	Sel-1	Maps to second selection variable in AskOptions
	Sel-2	Maps to third selection variable in AskOptions
	and so on...	
AskPath-<#>	Result	Standard values
	szPath	The path entered in the Destination Directory edit field
AskText-<#>	Result	Standard values
	szText	The text from the edit field
AskYesNo-<#>	Result	1 = User selected "Yes"
		0 = User selected "No"
ComponentDialog-<#>	Result	Standard values

	szDir	The path entered in the Destination Directory edit field
	Component-type	String (the only value currently allowed)
	Component-count	The total number of component selections
	Component-<#>	The selected item's number in the list (numbering begins with 0)
MessageBox-<#>	Result	1 = OK button clicked
RebootDialog-<#>	Result	0 (Result is always 0.)
	Choice	Indicates the "final" reboot selection made before rebooting the machine. Maps to the radio buttons and has these possible values: 601 = "Yes, I want to restart Windows now." (16-bit only) 602 = "Yes, I want to restart my computer now." 0 = "No, I will restart my computer later."
SdAskDestPath-<#>	Result	Standard values
	szDir	The path entered in the Destination Directory edit field
SdAskOptions-<#>	Result	Standard values
	Component-type	String (the only value currently allowed)
	Component-count	The total number of component selections
	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdAskOptionsList-<#>	Result	Standard values
	Component-type	String (the only value currently allowed)
	Component-count	The total number of component selections
	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdBitmap-<#>	Result	Standard values

SdComponentDialog-<#>	Result	Standard values
	szDir	The path entered in the Destination Directory edit field
	Component-type	String (the only value currently allowed)
	Component-count	The total number of component selections
	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentDialog2-<#>	Result	Standard values
	Component-type, <subcomponent>-type	String (the only value currently allowed)
	Component-count, <subcomponent>-count	The total number of component or subcomponent selections
	Component-<#>, <subcomponent>-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentDialogAdv-<#>	Result	Standard values
	szDir	The path entered in the Destination Directory edit field
	Component-type	String (the only value currently allowed)
	Component-count	The total number of component selections
	Component-<#>	The selected item's number in the list (numbering begins with 0)
SdComponentMult-<#>	Result	Standard values
	Component-type, <subcomponent>-type	String (the only value currently allowed)
	Component-count, <subcomponent>-count	The total number of component or subcomponent selections
	Component-number, <subcomponent>-<#>	The selected item's number in the list (numbering begins with 0)
SdConfirmNewDir-<#>	Result	1 = User selected "Yes" 0 = User selected "No"
SdConfirmRegistration-<#>	Result	1 = User selected "Yes" 0 = User selected "No"
SdDisplayTopics-<#>	Result	Standard values

SdFinish-<#>	Result	1 = Finish
	bOpt1	1 = "Yes, I want to view the README file." is checked 0 = "Yes, I want to view the README file." is not checked
	bOpt2	1 = "Yes, I want to launch [app name]" is checked 0 = "Yes, I want to launch [app name]" is not checked
SdFinishReboot-<#>	Result	1 = Finish
	BootOption	0 = Do not restart Windows or the machine 2 = Restart Windows 3 = Reboot the machine
SdLicense-<#>	Result	12 = Back 1 = User selected "Yes"
SdOptionsButtons-<#>	Result	12 = Back Or, when the Next button is clicked: 101 = Option button one (top) is selected 102 = Option button two is selected 103 = Option button three is selected 104 = Option button four is selected
SdRegisterUser-<#>	Result	Standard values
	szName	The text entered in the Name field
	szCompany	The text entered in the Company field
SdRegisterUserEx-<#>	Result	Standard values
	szName	The text entered in the Name field
	szCompany	The text entered in the Company field
	szSerial	The text entered in the Serial (number) field
SdSelectFolder-<#>	Result	Standard values
	szFolder	The folder name entered in the Program Folder field

SdSetupType-<#>	Result	12 = Back
		Or, when the Next button is clicked:
		301 = Typical radio button is currently selected
		302 = Compact radio button is currently selected
		303 = Custom radio button is currently selected
	szDir	The path entered in the Destination Directory edit field
SdShowDlgEdit1-<#>	Result	Standard values
	szEdit1	The text entered in the svEdit1 field
SdShowDlgEdit2-<#>	Result	Standard values
	szEdit1	The text entered in the svEdit1 field
	szEdit2	The text entered in the svEdit2 field
SdShowDlgEdit3-<#>	Result	Standard values
	szEdit1	The text entered in the svEdit1 field
	szEdit2	The text entered in the svEdit2 field
	szEdit3	The text entered in the svEdit3 field
SdShowFileMods-<#>	Result	Standard values
	nSelection	The "Choose what you want Setup to do" radio button selection:
		101 = Let Setup modify the AUTOEXEC.BAT file.
		102 = Save the required changes to AUTOEXEC.NEW file.
		103 = Do not make any changes.
SdShowInfoList-<#>	Result	Standard values
SdStartCopy-<#>	Result	Standard values
SdWelcome-<#>	Result	Standard values
SelectDir-<#>	Result	Standard values
	szDir	The path selected by this dialog
SelectFolder-<#>	Result	Standard values

	szResultFolder	The folder selected by this dialog
SetupType-<#>	Result	12 = Back Or, when the Next button is clicked: 301 = Typical radio button is currently selected 302 = Compact radio button is currently selected 303 = Custom radio button is currently selected
SprintfBox-<#>	Result	1 = OK button clicked
Welcome-<#>	Result	Standard values

Play back the silent installation

After you have created the installation and the response file, you are ready to run the installation in silent mode using InstallShield Silent. When running an installation in silent mode, be aware that no messages are displayed. Instead, a log file named Setup.log captures installation information, including whether the installation was successful. You can review the log file and determine the result of the installation.

To launch InstallShield Silent, run Setup.exe with the -s option.

InstallShield also provides the -f1 and -f2 switches so you can specify the name and location of the response file and the location of the log file. See the examples under Setup usage examples.

To verify if a silent installation succeeded, look at the `ResultCode` value in the `[ResponseResult]` section of Setup.log. InstallShield writes an appropriate return value after the `ResultCode` keyname.

InstallShield
www.installshield.com

Setup.log

Setup.log is the default name for the silent installation log file, and its default location is Disk1 (in the same folder as Setup.ins). You can specify a different name and location for Setup.log using the -f1 and -f2 switches with Setup.exe
.

The Setup.log file contains three sections. The first section, `[InstallShield Silent]`, identifies the version of InstallShield Silent used in the silent installation. It also identifies the file as a log file.

The second section, `[Application]`, identifies the installed application's name and version, and the company name.

The third section, `[ResponseResult]`, contains the result code indicating whether or not the silent installation succeeded. An integer value is assigned to the `ResultCode` keyname in the `[ResponseResult]` section. InstallShield places one of the following return values after the `ResultCode` keyname:

0	Success.
-1	General error.
-2	Invalid mode.
-3	Required data not found in the Setup.iss file.
-4	Not enough memory available.
-5	File does not exist.
-6	Cannot write to the response file.
-7	Unable to write to the log file.
-8	Invalid path to the InstallShield Silent response file.
-9	Not a valid list type (string or number).
-10	Data type is invalid.
-11	Unknown error during setup.
-12	Dialogs are out of order.
-51	Cannot create the specified folder.
-52	Cannot access the specified file or folder.
-53	Invalid option selected.

The Setup.log file for a successful silent installation of InstallShield is shown below.

```
[InstallShield Silent]
Version=v5.00.000
File=Log File

[Application]
Name=InstallShield5
Version=5.00.000
Company=InstallShield

[ResponseResult]
ResultCode=0
```

InstallShield
www.installshield.com

Setup.exe and command line parameters

Setup.exe is the main InstallShield executable; it performs setup initialization and launches the appropriate InstallShield engine file (_inst*xxx*.ex_) to execute the setup script (Setup.ins) on the target system. You must ship this file on Disk1 of your distribution disks. If desired, you can rename the program file and distribute it freely with its new name.

Command line parameters

Following is the list of command line parameters that can be used with Setup.exe. These switches are optional, but note that either a slash (/) or a dash (-) must precede the command line parameters. Separate multiple command line parameters with a space. But do not put a space inside a command line parameter (for example, `/r /fInstall.ins` is valid, but not `/r/f Install.ins`).

When using long path and file name expressions with switches, enclose the expressions in double quotation marks. The enclosing double quotes tell the operating system that spaces within the quotation marks are not to be treated as command line delimiters.

Setup.exe command line parameters are not case sensitive; upper- or lowercase letters can be used.

You can pass command line parameters directly to Setup.exe, or you can place the command line parameters in Setup.ini.

Parameter	Description
-f<path\CompiledScript>	The alternate compiled script can be specified using this option. Unless the compiled script (.ins file) also resides in the same directory as that of Setup.exe, the full path to the compiled script must be specified. _setup.dll must also reside in the same directory as your .ins file. For example, setup -ftest.ins will launch setup using Test.ins instead of Setup.ins.
-f1<path\ResponseFile>	The alternate location and name of the response file (.iss file) can be specified using this option. If this option is used when running InstallShield Silent, the response file is read from the folder/file specified by<path\ResponseFile>. If an alternate compiled script is specified using the -f switch, the -f1 switch entry must follow the -f switch entry.
-f2<path\LogFile>	The alternate location and name of the log file created by InstallShield Silent can be specified by using this option. By default, Setup.log log file is created and stored in the same directory as that of Setup.exe. If an alternate compiled script is specified using the -f switch, the -f2 switch entry must follow the -f switch entry.
-d	Run setup in debug mode. The -d switch also includes a <pathonly> option for specifying the path of the Setup.rul file. For more information, refer to the Visual Debugger help file.
-m<file name>	Causes InstallShield to generate a Management Information Format (.mif) file automatically at the end of the installation. Do not include a path—the .mif file is always placed in the Windows folder. <file name> is optional. If you do not specify a file name, the resulting file will be called Status.mif.
-m1<serial number>	Tells InstallShield to place the indicated serial number in the created .mif file.
-r	Causes Setup.exe automatically to generate a silent installation file (.iss file), which is a record of the installation input, in the Windows folder.
-s	Run InstallShield Silent to execute a silent installation .
-SMS	The SMS switch prevents a network connection and the Setup.exe from closing before the installation is complete. This switch works with installations originating from a Windows NT server over a network. Please note that SMS must be uppercase; this is a case-sensitive switch.

User-defined command line parameters

Along with the command line parameters listed above, -c, -e, -q, -t, -x, -z and -z1, are command line parameters reserved by InstallShield. User redefinition of these command line parameters, both upper- and lowercase, can cause errors.

The user defined command line parameters must be specified before specifying the predefined Setup options, if they are applicable.

{button ,AL(`Setup usage examples;Setup.exe error messages;Setup.ini',0,`',`')} See also

InstallShield
www.installshield.com

Setup usage examples

The following examples illustrate the use of Setup.exe, including use of the command line switches -s, -d, -f, -f1, and -f2:

`setup`

Launches Setup and tries to load Setup.ins from the same directory that contains Setup.exe.

`setup -fTest.ins`

Launches Setup and tries to load Test.ins from the same directory that contains Setup.exe.

`setup -fC:\Mydir\Test.ins`

Launches Setup and tries to load Test.ins from the C:\Mydir folder.

`setup -d`

Launches the InstallShield Visual Debugger and tries to load Setup.ins from the same folder that contains Setup.exe.

`setup -dC:\Mydir\Test`

Launches the InstallShield Visual Debugger, tries to load Setup.ins from the same folder that contains Setup.exe, and looks for the Setup.rul file in the C:\Mydir\Test folder.

`setup -d -fC:\Mydir\Test.ins`

Launches the InstallShield Visual Debugger and tries to load Test.ins from the C:\Mydir folder.

`setup -s`

Launches InstallShield Silent and tries to load Setup.ins and Setup.iss from the folder containing Setup.exe. The log file Setup.log is created in the same folder.

`setup -s -f1C:\Mydir\Mydir.iss`

Launches InstallShield Silent, tries to load Setup.ins from the same folder, and uses Mydir.iss (from the C:\Mydir folder) as the response file. This example also creates the log file Setup.log in the same folder as that of the response file (C:\Mydir).

If you specify an alternate compiled script using the -f switch and you place -f1 before -f in the command line, Setup will ignore the -f1 switch and will create the response file (the .iss file) in the Windows folder.

`setup -s -f1C:\Mydir\Mydir.iss -fC:\Mydir\Mydir.ins`

Will not launch InstallShield Silent because the -f1 switch is used before the -f switch, and -f1 is therefore ignored. No log file will be generated. However, the -fC:\Mydir\Mydir.ins portion of the command line is executed.

`setup -s -fC:\Mydir\Mydir.ins -f1C:\Mydir\Mydir.iss`

Launches InstallShield Silent, tries to load Mydir.ins from the C:\Mydir folder, uses Mydir.iss from the C:\Mydir folder, and generates the log file Setup.log in the C:\Mydir folder.

`setup -s -fC:\Mydir\Mydir.ins -f1C:\Mydir\Mydir.iss -f2C:\Mydir\Mydir.log`

Launches InstallShield Silent, tries to load Mydir.ins from the C:\Mydir folder, uses Mydir.iss from the C:\Mydir folder, and generates the log file Mydir.log in the C:\Mydir folder.

A few notes on using Setup.exe:

- Do not leave a space between command line switches and options.
- If you specify an alternate compiled script using the -f switch, always use the -f switch before specifying an -f1 or -f2 switch.
- When InstallShield Silent runs, a log file is created in the folder as the response file. The log file has the default name Setup.log if the -f2 switch is not provided along with -f1.

- If the -f1 switch is not used when running InstallShield Silent, Setup looks for the response file Setup.iss in the same folder as Setup.exe. A log file is created in the same folder.
- Make sure that _user1.cab and _setup.dll are stored in the same folder as the compiled script. That is, if Setup.ins resides in the C:\Mydir folder, then _user1.cab and _setup.dll must also reside in C:\Mydir.
- Setup.exe command line switches/options are not case sensitive.

Setup.exe error messages

The Setup.exe program may produce error messages if it cannot start properly. In most cases you'll encounter these messages when a severe error occurs. Rarely will your end users see these messages.

When you get an error message, it appears in a message box. Every error message has a number. These are InstallShield system error messages and there is no way to suppress them in your script.

Number	Error Message

Error 101 — **Setup is unable to find a hard disk location to store temporary files. Make at least 500KB of free disk space available and then try the installation again.**

Setup.exe is unable to find an appropriate location to copy temporary files. Setup.exe intelligently searches all possible locations for free space to copy temporary files.

Check to see enough free space is available and temporary location is not write protected. Check the environment variable TEMP. Make sure it points to a folder with ample free space.

Error 102 — **Setup is unable to find a compressed library file required to proceed with the installation. Check to make sure all required files are located with the Setup program.**

Make sure that _inst*xxx*.ex_ (the InstallShield engine file) is in the source folder. *xxx* will vary depending on the operating system. For example, InstallShield for Windows NT (Intel) and Windows 95 require the file _inst32i.ex_, where 32 is for 32 bit and *i* is for Intel.

Error 103 — **Setup is unable to find _setup.dll, which is needed to complete the installation. Restart your system and try again.**

Check to make sure that _setup.dll is located with the Setup program.

Error 104 — **Setup is unable to run an intermediate file needed to proceed with the installation.**

Same as Error 105. This error message refers to a different intermediate file.

Make sure that you are running a normal installation of Windows. If you cannot correct this error and need to contact InstallShield Technical Support, please give us information about any special configurations you've defined, and complete details of your environment.

Error 105 — **Setup is unable to run an intermediate file needed to proceed with the installation.**

Setup.exe launches the installation program after copying it to the hard disk. You'll see Error 105 when Setup.exe cannot launch the installation program from the hard disk. This message appears only rarely.

Make sure that you are running a normal installation of Windows. If you have need to contact Technical Support, please give us complete information on your environment and on any special configurations you've defined.

Error 107 — **Setup is unable to locate the script file <%s> which is needed to complete the installation.**

Setup was unable to find a script file named Setup.ins or optionally another script file that was specified on the command line.

Check to make sure that you have a compiled script file named Setup.ins co-located with Setup.exe. If you are using the -f option to specify another script file, make sure the path and the name of that file are correct.

Error 110 — **Setup was started with a command line argument that contained an incomplete parameter bad_parameter.**

Check to make sure that the command line argument is in double quotes or single quotes—you cannot mix and match.

Error 111 — **Insufficient memory available to run Setup. Close all other applications to make more memory available and try to run Setup again.**

If after closing applications this message continues to appear, reboot your system.

Error 112 — **Setup is unable to locate the layout file '…\Layout.bin' which is needed to complete the installation.**

The Layout.bin file must be present in the same folder as Setup.exe.

Error 201 **Setup is unable to initialize the installation program (Install.exe).**

InstallShield was internally unable to create a main window or initialize its window classes. Or there is not enough memory to register or create its internal window classes.

This error is highly unlikely under normal circumstances. If it should occur, reboot the machine and try again. If it persists, contact Technical Support.

Error 202 **Setup is unable to initialize the installation program.**

Setup.exe was internally unable to load the script file due to low memory conditions, or it was unable to create its internal data structures, window classes or other control classes. The script file was contaminated or damaged, or was not compiled properly.

This error is highly unlikely. If it does occur, (1) recompile the script file, (2) reboot the machine, and (3) try running InstallShield again with the new script file.

Error 301 **Setup was unable to start up the installation program.**

Make sure that you have a good copy of Setup.exe on your source disks. Check also to see if there's enough space on the target disk.

Error 401 **String variable is not large enough for string. Please check string declarations.**

InstallShield was trying to copy a text string into a string variable. However, the text string was larger than the length you declared for that string variable. Before copying a string into a string variable, InstallShield checks the length of the string variable. There may be instances when InstallShield may be unable to detect such string overwrites. Check the declared length of the string variables. Increase the length to the maximum allowed value. Check the logic to make sure that you do not have a loop which generates strings longer than 512 bytes for Windows 3.1 and 1,024 bytes for Windows NT and Windows 95.

Error 420 **Setup is unable to copy the installation support file <filename> to a temporary location.**

_user1.cab and _sys1.cab are InstallShield's setup resources files. InstallShield automatically decompresses and copies _user1.cab and _sys1.cab to a temporary location on the user's hard drive. You'll see this error message if InstallShield cannot find an acceptable temporary location where it can copy the support files, or if there was not enough free disk space to copy the files.

Make sure you have enough free disk space and that the environment variable TEMP points to a valid location with enough available disk space.

Error 421 **Setup is unable to copy the installation support file _user1.cab to a temporary location. Make more space available and try again.**

Setup.exe cannot copy the support files to a temporary location.

Check to see that enough free space and appropriate write privileges are available.

Error 422 **Setup is unable to expand the installation support file <filename>.**

After copying the _user1.cab or _sys1.cab file to a temporary location, Setup.exe could not decompress the support file.

Make sure there is enough free space on the drive that you're using for temporary storage.

Error 423 **Setup is unable to load the installation script file.**

Setup.exe is unable to load the script file into memory for processing.

Error 424 **Setup has encountered an internal stack overflow error. Close all applications, restart the system and try the installation again.**

Setup.exe displays this severe error when InstallShield's internal stack exceeds its limits. It's highly unlikely that you will see this error message.

Recompile your script and try the installation again under a normal windows installation.

Error 425 **Setup has encountered an incomplete return statement in the script. Check your script for unmatched return statements.**

You cannot use the return statement in a script unless it follows a call statement. If a return statement is encountered without a call statement, then the return statement is considered unmatched and you'll see this error message.

Scan your script file to make sure each return statement is encountered only after a call statement. Fix the problem and recompile your script.

Error 426 **Setup is unable to find the installation script file: script_filename.**

Setup.exe was unable to find a script file named Setup.ins or, optionally, another script file that was specified on the command line.

Error 427 **Setup is unable to load the installation script file: script_filename. The script may be from a previous version or corrupted.**

Setup.exe is unable to load the script file into memory for processing. It's possible that the script file has been corrupted or is otherwise incomplete.

Recompile your script file.

Error 432 **Setup has detected that unInstallShield is in use. Please close unInstallShield and restart setup.**

InstallShield needs unInstallShield to be shut down so that the engine can update IsUninst.exe or IsUninst16.exe with a newer version if necessary. Close unInstallShield, and then run Setup.exe again.

Error 440 **Setup has detected a possible infinite loop in the script with function function_name. Make sure your are handling the error returns codes properly.**

Setup.exe has detected the possibility there's an infinite loop in the way a function is used. ComponentMoveData and other similar functions have a specific usage that you must follow. Otherwise, it's possible that the script could end up in an infinite loop.

For every InstallScript, there's a typical example in code of the correct way to use it. If you get this error message, make sure you've structured your code the same as the example in the InstallScript Language Reference.

If you're processing lists in a while loop, make sure that you have created the list with the ListCreate function, and that you haven't destroyed the list with the ListDestroy function.

Error 502 **Setup is unable to initialize the installation program. The script file may be bad.**

The script file is corrupted or not enough free memory is available to execute the script file.

Recompile the script.

Error 701 **A division by zero error was detected in the script. Installation will continue.**

In checking your script, Setup.exe detected a division by zero. Installation will continue.

Examine the logic of your script. Check every division operation and make sure there's no possibility of a zero value in the denominator.

Error 702 **An internal error has occurred. Insufficient memory to allocate buffer.**

InstallShield was internally unable to allocate memory to execute an instruction.

A highly unlikely error. Before contacting Technical Support about this, make sure you can repeatedly duplicate the error message. Before you call, send your script and installation disks to InstallShield Technical Support..

Error 703 **An internal read error has occurred on script_filename. Unable to load installation instructions.**

Setup.exe detects that it cannot read the script file from memory.

Recompile the script file and use the new compiled script file. Reboot the machine and try running Setup.exe again.

Error 704 **Script_filename file has become corrupted. Unable to load installation instructions.**

This message appears if Setup.exe detects that script file is corrupted, or if·Setup.exe is unable to load the script file into memory for processing.

Run Windows in standard or enhanced mode, or make more memory available to Windows by removing any memory-resident programs or network drivers.

Error 3000 **Error messages ranging from 3000 to 3021 are internal memory-related error conditions.**

Low memory conditions exist. Internal string variables were overwritten, or the compiled script file is

corrupted. Severe error conditions exist.

Compile your installation again and then test it. Make sure that you are not exceeding the limits of the string variables that you declared in your script.

InstallShield
www.installshield.com

Specify UNC paths in my setup script

If you are using UNC (Universal Naming Convention) paths in InstallScript strings, in order to create a double backslash in the form:

```
\\<computername>\<sharename>\<filename>
```

You will need to use two backslash escape sequences to specify the double backslashes required by the string. Therefore, you would use four backslashes "\\\\" at the beginning of the string.

UNC paths are fully supported on 32-bit platforms. But be aware that the 16-bit InstallShield engine (_inst16.ex_) does not fully support UNC paths.

InstallShield
www.installshield.com

Running setup from a network server

InstallShield has been successfully used to install thousands of applications from network servers. However, you should be aware of some unique considerations when running a setup from a network server:

Long file names

When running installations from Novell networks, except Novell NetWare Client 32 for Windows 95, you must use short names for all paths and file names in the setup folder. The Novell server will not return file names longer than 8.3 characters (eight characters with a three-character file extension), forcing InstallShield to return a "Setup Initialization Error."

Also be aware that Setup.exe is itself a 16-bit program (necessary to be able to launch cross-platform installations), so it cannot accept long file names .

Waiting for the executable to terminate

Launch Setup.exe with the -SMS switch over network connections. Ordinarily, Setup.exe launches an InstallShield engine file (_inst*xxx*.ex_) and terminates, which would cause many servers to assume that setup had completed. Using the -SMS switch forces Setup.exe to wait for the setup to complete in its entirety before terminating.

Registry functions

InstallShield provides you with functions that allow you to connect to the registry on a networked system (a "remote registry") and edit keys and values on it. For more information, see How do I get or set information in a remote registry.

InstallShield
www.installshield.com

Create a setup .mif file

Back Office Logo certification requires that a Management Information Format (.mif) file be created for every installation. It uses this file to integrate information about the application. The .mif file should be called <app name>.mif and placed in the Windows folder. InstallShield does not create an .mif file for a 16-bit setup, since they are not required.

InstallShield can generate a Management Information Format (.mif) file automatically at the end of a setup. There are two ways you can create a setup .mif file using InstallShield, each described below.

For testing purposes, you can create an .mif file when you run your setup from the InstallShield IDE. From the Build menu, select Settings. In the Setup Settings dialog, select the Generate MIF File check box. Customize any desired settings. Select Run Setup from the Build menu. Your setup will generate an .mif file only when you run it from the IDE.

Create an [Mif] section in Setup.ini

You can create a section in the Setup.ini file with the following entries:

```
[Mif]
Type=
FileName=
SerialNo=
```

The section name must be [Mif]. The keynames are explained in The [Mif] Section of Setup.ini.

The following is an example of a Setup.ini file for a setup that will automatically create an .mif file (IShield.mif):

```
[Startup]
AppName=InstallShield5
FreeDiskSpace=1984

[Mif]
Type=SMS
FileName=IShield
SerialNo=IS50-32XYZ-12345
```

Pass Setup.exe the -m command line parameter

You can launch Setup.exe with the -m command line parameter as follows:

```
setup -m<file name>
```

This command line expression tells InstallShield to generate an .mif file. <file name> is optional. If you do not specify a file name, the resulting file will be called Status.mif.

As with all command line parameters for Setup.exe, no space characters should be included between the letter m and the file name. Also, the file name should not have a file extension or a path—it must always have the .mif file extension and be placed in the Windows folder.

The Setup.exe parameter -m1 tells InstallShield to place the indicated serial number in the .mif file:

```
-m1<serial number>
```

The following example creates an .mif file (IShield.mif) using Setup.exe command line parameters:

```
setup -mIShield -m1IS30-PX32-12345
```

Script modifications

If the script aborts the installation for any reason and the default exit handler is not called, the script must use the

abort keyword instead of the exit keyword. Using the abort keyword informs InstallShield that the setup did not complete successfully and that an error code should be placed in the .mif file. If the exit statement is instead used to exit the installation, the .mif file will falsely indicate that the installation was successful.

Sample .mif file

Click here to see a sample .mif file generated by InstallShield. The values listed in **bold** are the values added specifically for your setup.

Uninstallation

unInstallShield does not automatically create .mif files. To create a Back Office logo-compliant uninstallation, you can call a custom DLL function to create the appropriate .mif file. For more information regarding the custom DLL function option of InstallShield, see How do I call a custom DLL function from unInstallShield.

A current limitation of using this method is that it is not directly possible to determine whether the uninstallation was successful. Therefore, the custom DLL function must either check for application specific information to determine this, or assume and report success in the .mif file.

InstallShield
www.installshield.com

Sample .mif file

```
////////////////////////////////////////////////////////
//                                                    //
//     InstallShield MIF Generator, Version 1.0      //
//     Copyright 1996 InstallShield Corporation      //
//       InstallShield MIF Format Version 1.0        //
//          http://www.installshield.com              //
//                                                    //
////////////////////////////////////////////////////////

START COMPONENT
  NAME = "WORKSTATION"
  START GROUP
    NAME = "ComponentID"
    ID = 1
    CLASS = "DMTF|ComponentID|1.0"
    START ATTRIBUTE
      NAME = "Manufacturer"
      ID = 1
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(255)
      VALUE = "InstallShield"
    END ATTRIBUTE
    START ATTRIBUTE
      NAME = "Product"
      ID = 2
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(255)
      VALUE = "InstallShield5"
    END ATTRIBUTE
    START ATTRIBUTE
      NAME = "Version"
      ID = 3
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(255)
      VALUE = "5.00.000"
    END ATTRIBUTE
    START ATTRIBUTE
      NAME = "Serial Number"
      ID = 4
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(255)
      VALUE = "IS30-32P55-12345"
    END ATTRIBUTE
    START ATTRIBUTE
      NAME = "Installation"
      ID = 5
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(64)
      VALUE = "1997021116535000.000000+000"
    END ATTRIBUTE
  END GROUP
  START GROUP
    NAME = "InstallStatus"
```

```
    ID = 2
    CLASS = "MICROSOFT|JOBSTATUS|1.0"
    START ATTRIBUTE
      NAME = "Status"
      ID = 1
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(32)
      VALUE = "Success"
    END ATTRIBUTE
    START ATTRIBUTE
      NAME = "Description"
      ID = 2
      ACCESS = READ-ONLY
      STORAGE = SPECIFIC
      TYPE = STRING(255)
      VALUE = "Installation Successful."
    END ATTRIBUTE
  END GROUP
END COMPONENT
```

Overview: Localization

InstallShield supports many powerful features that allow you to easily customize your setup for global distribution. Using these features, you can create a single setup that is localized in any number of languages and can handle conditional installation of language-specific files.

In order to localize a setup in a given language, you must purchase support for that language, such as an Export Edition or International Edition, from InstallShield Corporation. To see which languages are supported in your edition of InstallShield, open the About box under the Help menu. Click the Files button. The folders under Redistributable Files reflect the supported languages. (Check the InstallShield Web site frequently for the availability of international editions of InstallShield5.)

If, for example, you've purchased InstallShield5 Professional Edition for English but no additional language support, you can create only an English setup. If you also include setup resources or language-specific files for a second or third language, InstallShield will still only create an English setup and will not include the non-English files.

Localizing a setup primarily involves separating code from language-specific resources and files. You may also need to include separate files, depending on the language the setup is running in.

InstallShield handles localization by dividing it into the following distinct tasks:

{button ,JI(`Getres.HLP>(w95sec)',`Create_a_string_table_for_each_supported_language')} Create a string table for each supported language

{button ,JI(`Getres.HLP>(w95sec)',`Include_my_language_specific_resource_files')} Include your language-specific resource files

{button ,JI(`Getres.HLP>(w95sec)',`Include_localized_InstallShield_files')} Include localized InstallShield files

{button ,JI(`Getres.HLP>(w95sec)',`Filter_language_dependent_application_files')} Filter any language-dependent application files

---

{button ,AL(`How InstallShield determines which language the setup should run in;Specify which languages I want to include in my built setup;Tips for internationalizing a setup',0,`',`')} See also

InstallShield
www.installshield.com

Create a string table for each supported language

InstallShield
www.installshield.com

String tables allow you to access strings for a given language from your setup script. That is, you can keep the code for your setup completely separate from any language-specific strings you may want to display during setup; simply refer to the string by means of string identifiers. When you use string identifiers in your setup script, they must be preceded by the at sign (@).

Many dialog boxes in the InstallShield IDE also let you enter a string identifier for text that will be displayed during setup.

String identifiers are defined in the string tables stored in the InstallShield IDE's Resources pane. You can associate a string value and a comment with each string identifier. You can have unique string values in each string table for the same string identifier.

When you include a language in your setup project, InstallShield creates a string table for that language and includes in it all existing string identifiers. Then, it is your responsibility to translate the strings into the target language.

The string tables themselves are stored in text files called Value.shl stored in the default location: My Installations\<project name>\String Tables\<language ID-language name>. If you need to, you can send these files to a translator and then return them to the correct folders. The translated strings will now appear in the Resources tab and will be included in your setup the next time you build it.

When you build your setup with the Media Build Wizard, InstallShield includes the string tables for the target languages you select.

The string table that is used during the setup depends upon which language the setup is running in. For more information, see How InstallShield determines which language the setup should run in.

{button ,AL(`Associate string identifiers with values',0,`',`')} See also

InstallShield
www.installshield.com

Set a default language for my setup

InstallShield
www.installshield.com

You can create a single setup for as many languages as are supported. However, only one language can serve as the "default" language for your setup. InstallShield needs to know which language is the default in order to determine which language to launch the setup in.

If you have purchased InstallShield for distribution to only one language, then that language is always the default since it is the only language in which you can build a setup.

If you have purchased InstallShield for more than one language and are building a setup for distribution to more than one language, you can select the default language in the Media Build Wizard - Languages panel. The first language that you click is listed as the default at the bottom of the panel. (If you have not purchased support for more than one language, you will not see the Languages panel at all.)   You can confirm your selection in the Media Build Wizard - Summary panel.

{button ,AL(`Specify which languages I want to include in my built setup',0,`',`')}   See also

InstallShield
www.installshield.com

Tips for internationalizing a setup

InstallShield
www.installshield.com

Consider these general points when designing your setup:

- The goal of international design is a localized product that is international in scope and readily adaptable to specific areas of the world.
- The key to international design is resource and code separation, plus country and language independence.
- Internationalizing your setup requires a design that is simple and modular.
- Creating a worldwide specification package means incorporating international requirements into the installation specifications from the beginning.
- Make bitmaps and icons culturally sensitive. What may be acceptable in one country could be misleading or offensive in another.
- English strings are usually shorter than equivalent text strings in other languages. Translated strings grow an average of 30-40%. This implies that both static and temporary storage areas will increase in size.
- When designing prompts, use only one-half of the available space to allow for expansion.
- Try to avoid hard coding element positioning and size on the screen, since these items may change when the element is translated.

InstallShield
www.installshield.com

Choose which languages I want to include in my project

InstallShield
www.installshield.com

If you are creating your setup with the Project Wizard, you can choose which languages your setup supports in the Project Wizard - Specify Languages panel. If you're creating a setup in a different manner or you want to add a language after you've made your selections in the Project Wizard, follow the instructions below to include new languages in your setup project.

1. Close your setup project by selecting Close on the File menu.
2. In the Projects window, right-click on the icon for the project you want to modify. A popup menu opens.
3. Select Properties. The Setup Properties window opens.
4. Click the Add button. The Add Language dialog opens.
5. Select the languages you wish to add. (You can choose from any of the supported languages; for more information, see the note in Overview: Localization.)   Click OK. A confirmation dialog opens.
6. Click Yes to add the language selections to your project.
7. Click OK to exit the Add Language dialog. The next time you open your project, you will find folders in the Resources pane and in the Setup Files pane where you can place resources for the new languages.

Not all languages that you include in your project will be included in your setup. When you build your setup with the Media Build Wizard, you choose which languages you want to create your setup in. For more information, see How do I specify which languages I want to include in my built setup.

How InstallShield determines which language the setup should run in

Even though you can localize a setup in as many languages as you've purchased support for, InstallShield will run a setup in only one language. During setup initialization InstallShield determines which language to launch the setup in depending on the languages you include in the setup or the end user's selection, as described below. The language code for the language that setup is running in is stored in the SELECTED_LANGUAGE system variable.

All Windows language IDs are listed in <InstallShield location>\Include\SdLang.h. If you include this file in your setup script, you can compare the value of SELECTED_LANGUAGE to the constants defined in SdLang.h.

If your setup has support for only one language, then it will always run in that language.

If you selected more than one language when you built your setup, then one of those languages is considered the default. When InstallShield initializes, it determines the default language for the target system. If one of the languages contained in your setup matches the system's default language, then InstallShield launches the setup in the target system's language.

If the target system's default language is not present in your setup, then InstallShield launches the setup in the language that you chose as the default when you built your setup.

The Language dialog

You also have the option of enabling the Language dialog so that your end user can choose which language the setup should run in. For instructions on enabling the Language dialog in your setup, see Let my user select which language the setup should run in.

InstallShield will launch setup in the language chosen by the end user.

{button ,AL(`Appending to an existing uninstallation log file;Determining the default language of the target system',0,`',`')} See also

Specify which languages I want to include in my built setup

If you have support for more than one language, you can select the languages that you want to include in your setup in the Media Build Wizard - Languages panel. You can confirm your selection in the Media Build Wizard - Summary panel.

If you have not purchased support for more than one language, you will not see the Languages panel at all. For more information, see the note in Overview: Localization.

You can build a setup for distribution in as many languages as you have support for, but InstallShield will only run a setup in one language. For more information, see How InstallShield determines which language the setup should run in.

{button ,AL(`Choose which languages I want to include in my project;Set a default language for my setup',0,`',`')}
See also

InstallShield
www.installshield.com

Let my user select which language the setup should run in

InstallShield
www.installshield.com

If you are building a setup with multiple languages, you can let your end user select which language the setup should run in. If you enable the Language dialog box, InstallShield displays a Language dialog during setup initialization.

The text in the dialog is displayed according to the target system's default language. So, if setup is initializing on a French system, the text in the Language dialog and the language selections will be in French; Japanese on a Japanese system.

If the target system's default language is present in your setup, then that language will be the default selection in the Language dialog. If the target system's default language is *not* present in your setup, then the language that you chose as the default when you built your setup will be the default selection in the Language dialog.

InstallShield launches setup in the language chosen by the end user. To enable the Language dialog

1. In the Media Build Wizard - Build Type panel, click the Advanced button. The Advanced Build Type property sheet opens.
2. Click the Setup Tab.
3. Click the Enable language dialog check box.
4. Complete the rest of the Media Build Wizard's panels to build your setup.

When it builds your setup, InstallShield creates the following keyname-value pair in the [Startup] section of Setup.ini: `EnableLangDlg=Y`. If you want to change this value later, you can edit Setup.ini and change the line to read `EnableLangDlg=N`.

{button ,AL(`How InstallShield determines which language the setup should run in',0,`',`')} See also

Include my language-specific resource files

When you choose which languages you want to include in your setup project, InstallShield creates folders for each language in the Setup Files pane.

Many DLLs and bitmap files can go into the Language Independent folders. However, if you have DLLs or bitmaps that you want to access during setup that are specific for a certain language, place those files under the folder for the specific target language.

When you build your setup with the Media Build Wizard, you can specify which languages you want to include in your built setup. For any given language that you choose, its resource files are accessed only if the setup runs in that language.

{button ,AL(`Include localized InstallShield files',0,`',`')} See also

InstallShield
www.installshield.com

Include localized InstallShield files

When you purchase support for a given language from InstallShield Corporation, you obtain localized versions of the following files:

- IsUninst.exe (32-bit) and IsUn16.exe (16-bit)–the unInstallShield executable files
- _setup.dll—contains InstallShield initialization resources
- _isres.dll—contains InstallShield resources for dialog boxes and user interface objects, including the built-in dialog boxes and the Sd dialog boxes
- Lang.dat—contains resources for the Language dialog box

Including these files in your setup means that all InstallShield and unInstallShield dialog boxes and error messages appear in the desired language. You must do three things in order for the correct localized InstallShield files to be accessed during setup:

1. You must include the specified language in your setup project. Including a language in your project means that you can build a setup that can be localized in that language when you run the Media Build Wizard.
2. When you build your setup with the Media Build Wizard, you must include the specified language in your built setup. For example, to include the localized InstallShield resource files for both English and French, select English and French in the Media Build Wizard - Languages panel. Both English and French resource files will be available to your completed setup.
3. You have to determine which language you want setup to run in, depending on the target system's default language. You can also give the end user the option of selecting which language setup will run in. For more information, see How InstallShield determines which language the setup should run in.

{button ,AL(`Include my language-specific resource files',0,`',`')} See also

InstallShield
www.installshield.com

Filter language-dependent application files

InstallShield
www.installshield.com

Mark any language-dependent file groups

If you're creating an international setup, doubtless you have some application files or system DLLs that are language specific.

Remember that all files in a file group share the same properties, so you should organize language-dependent files into unique file groups.

Next, mark the file groups as language dependent.

Filter those language-dependent files you do not want to install

Call the ComponentFilterLanguage function to exclude any files that you do not want installed. Call ComponentFilterLanguage before transferring your files with the ComponentMoveData function.

You can call ComponentFilterLanguage conditionally depending on the target system's default language, end-user selections, the language that the setup is running in, or any other factor you want to take into consideration. If you want to find out the target system's default language, call the GetSystemInfo function with the LANGUAGE option.

For more information, see Filter language-dependent files based on the target system's language.

Include only the desired file groups in your setup

When you build your setup with the Media Build Wizard, you can specify which languages you want to include in your built setup. For any given language that you choose, its language-dependent file groups are included in the completed setup's file media library. But if you have not purchased support for the language that your file groups are associated with, those file groups will not be included in your file media library. (For more information, see the note in Overview: Localization.)

{button ,AL(`Determining the default language of the target system;Filter language-dependent files based on the target systems language',0,`',`')}   See also

Filter language-dependent files based on the target system's language

To find out the language of the target system, call the GetSystemInfo function with the LANGUAGE option. GetSystemInfo returns the name of the target system's default language in the svResult parameter.

GetSystemInfo also returns a hexadecimal value in the nvResult parameter indicating the target system's language. If you're running a 16-bit setup, nvResult contains the default system language. If you're running a 32-bit setup, it contains the locale setting. These values are identical to the constants defined in SdLang.h. If you include this file in your script, you can compare the value of nvResult to these constants.

To filter out language-specific files, call the ComponentFilterLanguage function, as shown in the example script below.

The example is for a setup that targets English, German, and both French sublanguages. English is the default language, which means that English files are installed if the target machine is English or if the target system's language is not French, German, or English.

The hexadecimal values returned by GetSystemInfo are sublanguage IDs, which are generally not passable as parameters to ComponentFilterLanguage. Therefore, you should also check for sublanguages in your switch statement. (The only exceptions to this rule are the sublanguages for which InstallShield resources are available: French - Standard (0x040c), French - Canadian (0x0c0c), Portuguese - Brazilian (0x0416), and Portuguese - Standard (0x0816).)

```
// Filter out all language-specific file groups.
ComponentFilterLanguage (MEDIA, ISLANG_ALL, TRUE);

// Retrieve the target system's default language or locale setting.
GetSystemInfo (LANGUAGE, nvResult, svResult);

// Install language-specific files if the target system is running
// in the files' language.
switch (nvResult)
   case ISLANG_FRENCH_CANADIAN:
      ComponentFilterLanguage (MEDIA, ISLANG_FRENCH_CANADIAN, FALSE);
   case ISLANG_FRENCH_STANDARD,
        ISLANG_FRENCH_BELGIAN,
        ISLANG_FRENCH_SWISS,
        ISLANG_FRENCH_LUXEMBOURG:
      ComponentFilterLanguage (MEDIA , ISLANG_FRENCH_STANDARD, FALSE);
   case ISLANG_GERMAN_STANDARD,
        ISLANG_GERMAN_SWISS,
        ISLANG_GERMAN_AUSTRIAN,
        ISLANG_GERMAN_LUXEMBOURG,
        ISLANG_GERMAN_LIECHTENSTEIN:
      ComponentFilterLanguage (MEDIA, ISLANG_GERMAN, FALSE);
   default:
      ComponentFilterLanguage (MEDIA, ISLANG_ENGLISH, FALSE);
endswitch;
```

{button ,AL(`Mark application files as language dependent;Determining the default language of the target system',0,`',`')} See also

Determining the default language of the target system

Call the GetSystemInfo function with the LANGUAGE option to determine the language of the target system. GetSystemInfo returns a language ID constant in the nvResult parameter and the name of the language in the svResult parameter. It is more useful to use the language identifier constant returned in nvResult, because unlike the name of the language it is platform- and language-independent.

GetSystemInfo returns different values in the svResult and nvResult parameters, depending on the target platform and whether you're running a 16- or 32-bit setup. (You choose which type of setup to create in the Media Build Wizard - Platforms panel.) The following sections detail how GetSystemInfo retrieves the system's language under different platforms.

nvResult - language ID

When called with the LANGUAGE option, GetSystemInfo returns a hexadecimal value indicating the language of the target system in the nvResult parameter. This language ID value is identical to one of the constants defined in SdLang.h. The nvResult value is taken from the following locations in 16- and 32-bit setups:

16-bit setups

GetSystemInfo returns the language ID for the language version of Windows that is installed, regardless of the current locale setting in the Regional Settings or International icon in the Control Panel.

For example, on English Windows, ISLANG_ENGLISH_UNITEDSTATES is returned regardless of the current locale setting. On French Windows, ISLANG_FRENCH_STANDARD is returned.

During setup initialization, InstallShield always uses the 16-bit method to determine the default system language, even in a 32-bit setup.

32-bit setups

GetSystemInfo returns the language ID for the current locale setting in the Regional Settings or International icon in the Control Panel. Note that this value will differ from the language ID for the language version of Windows if the user has changed the locale setting in the Control Panel.

Also, note that Windows NT allows each user to have a personal locale setting. Under this platform GetSystemInfo will always return the language identifier for the default system locale setting, which can only be set by a system administrator. Any locale settings set by the current user will not affect the information retrieved by the setup.

For example, on English Windows 95 with the locale set to English (British), GetSystemInfo will return ISLANG_ENGLISH_UNITEDKINGDOM. On French Windows with the locale set to French (Canadian), the function will return ISLANG_FRENCH_CANADIAN.

svResult - language name

When called with the LANGUAGE option, GetSystemInfo returns the name of the default system language in the svResult parameter. The svResult value is taken from the following locations in 16- and 32-bit setups:

16-bit setups

GetSystemInfo returns the fully-localized name of the language version of Windows that is installed, regardless of the current locale setting in the Regional Settings or International icon in the Control Panel.

Note that this string is stored by Windows and may be slightly different even on Windows platforms that have the same language version. For example, on English Windows 95, "English (United States)" is returned in this parameter, but on English Windows NT, "U.S. English" is returned. On French Windows 95 or French Windows NT "Francais (Standard)" is returned.

32-bit setups

GetSystemInfo returns the English name of the current locale setting in the Regional Settings or International icon in the Control Panel. This value will differ from the name of the language version of Windows if the user has changed the locale setting in the Control Panel.

Remember that Windows NT allows each user to have a personal locale setting. Under this platform GetSystemInfo will always return the name of default system locale setting, which can only be set by a system administrator. Any locale settings set by the current user will not affect the information retrieved by the setup.

Note that this string is always the English name of the language, regardless of the language version of Windows installed. For example, on French Windows 95 with a locale set to French (Standard), GetSystemInfo will return "French" (not "Francais (Standard)") in the svResult parameter.

InstallShield
www.installshield.com

Overview: Extensibility

InstallShield
www.installshield.com

InstallShield provides functions that you can use to extend your setup by accessing DLLs and Windows APIs and launching child setups and other applications from your setup.

You can call functions in a DLL or Windows API for use in the installation. You can pass data back and forth between the script and the DLL. You can also pass pointers to data in a DLL.

You can launch another setup from within the main setup program. You can set the main setup program to wait for the launched setup to complete, or you can run both setups simultaneously.

You can launch an external application from within a setup. For example, you may want to launch a tutorial or a Readme file at the end of the setup. In order for you to read the Readme file, you must launch Notepad.exe or some other program, instructing it to load the Readme file.

You can also display a custom dialog box in your setup. To register the dialog box with InstallShield and use the resources stored in a DLL, call the EzDefineDialog or the DefineDialog function.   For complete information about displaying a custom dialog box in your setup, see the “Custom Dialog Boxes” section under “End-user dialog boxes” in the Getting Results help file.

InstallShield
www.installshield.com

Requirements for external functions

1. If your DLL function was created using C++, you need to wrap the function in an extern "C" statement. This is because C++ compilers mangle function names, meaning that the InstallShield prototype will be invalid because the function name used will not be recognized. A good way to wrap the function definitions in a C++ file is to use #ifdef logic as follows:

   ```
   #ifdef __cplusplus
   extern "C" {
   #endifdef

   // The function definition goes here.

   #ifdef __cplusplus
   }
   #endifdef
   ```

2. If your function declaration uses `_declspec (_dllexport)`, then you must list the function name in the .def file to prevent name mangling.
3. Use the following table to match data types between your InstallScript prototype and external functions:

External type	InstallScript type
int	INT
int *	BYREF INT or POINTER
pointer to 4-byte, non-int number	BYREF LONG or POINTER
4-byte, non-int number	LONG
char	CHAR
lpstr	STRING

4. Avoid unsigned 4-byte numerical types in your external functions. The largest positive value InstallShield can handle is a signed LONG that is +2,147,483,647. If the external function returns an unsigned 4-byte type/value greater than +2,147,483,647, InstallShield will interpret it as a negative value.
5. Use the Pascal calling convention in your external functions.

{button ,AL(`Call a DLL function;Call a Windows API',0,`',`')} See also

Call a DLL function

1. In the IDE, place the DLL in the appropriate <target language>\<target platform> folder in the Setup Files pane.
2. In the script, before the program block, prototype the function using the following syntax:

```
prototype ReturnType DLLName.FunctionName( ParamType1, ParamType2, ... )
```

   For example:

```
prototype BOOL MyDLL.MyFunction( INT, INT, INT );
```

   You can specify the return type to be one of the following InstallScript data types: BOOL, CHAR, HWND, INT, LONG, LPSTR, NUMBER, POINTER, or SHORT. If a return type is not specified, InstallShield assumes that the DLL function returns a 4-byte value.

3. Load the DLL by calling UseDLL. For example:

```
UseDLL( SUPPORTDIR ^ "MyDLL.dll" );
```

   You do not have to load _isuser.dll, _isres.dll, or Windows API DLLs.

4. Call the function as you would any other. For example:

```
bResult = MyFunction( nInt1, nInt2, nInt3 );
```

5. After all script calls to the DLL have been made, unload the DLL by calling UnUseDLL. For example:

```
UnUseDLL( SUPPORTDIR ^ "MyDLL.dll" );
```

There are three rules you must remember when calling DLL functions from your setup script:

- The maximum length of the DLL name is 33 characters; the maximum length of the function name is 80 characters.
- InstallShield cannot accept a composite parameter (that is, a parameter with a width exceeding four bytes) when calling a DLL. However, a parameter can be a pointer that points to a composite structure.
- Call 16-bit DLLs when installing on 16-bit platforms; call 32-bit DLLs when installing on 32-bit platforms.

{button ,JI(`Langref.hlp',`Pointers')} See also

You cannot directly prototype an external function in your setup script if you are using the InstallShield Free Edition. Instead, you must call the CallDLLFx function.

For a list of differences between the Free and Professional editions, click here.

Call a Windows API

1. Before the program block, prototype the function using the following syntax:

   ```
   prototype ReturnType DLLName.FunctionName( ParamType1, ParamType2, ... )
   ```

   For example:

   ```
   prototype INT User.LoadString( INT, SHORT, BYREF STRING, INT );
   ```

   Use Dumpbin.exe with the /EXPORTS option to determine which functions reside in any of the Windows API DLLs Kernel32.dll, User32.dll, and GDI32.dll. Consult a Windows programming reference such as the Microsoft Development Library (MSDL) to determine the return type and parameter types for a function.

   You can specify the return type to be one of the following InstallScript data types: BOOL, CHAR, HWND, INT, LONG, LPSTR, NUMBER, POINTER, or SHORT. If a return type is not specified, InstallShield assumes that the DLL function returns a 4-byte value.

2. Call the function as you would any other. For example:

   ```
   nResult = LoadString( iInstance, nStringID, szMyString, MAX_SIZE );
   ```

Refer to Winsub.h and Winsub.rul, in <InstallShield location>\Include, for examples of Windows Application Programming Interface (API) prototyping. In fact, you may wish to use Winsub.h and Winsub.rul if they provide the desired API interface.

{button ,JI(`Langref.hlp',`Pointers')} See also

Launch another setup

Calling DoInstall

The preferred way to launch a setup created with InstallShield is to call the DoInstall function, as described below:

1. Create a complete setup project for the child (launched) setup; all the files generated by InstallShield are necessary for the child setup to run properly.
2. In the parent (launching) setup project, place all the files for the child setup in the appropriate <target language>\<target platform> folder in the Setup Files pane.
3. In the parent setup script, call DoInstall as in the following example:

```
DoInstall (SUPPORTDIR ^ "Setup.ins", "", WAIT);
```

You should call DoInstall only to launch a setup created with the same major and minor version (and preferably build number) of InstallShield as the setup launching it. Calling DoInstall to launch a setup created with any other version of InstallShield will not be successful.

Calling LaunchAppAndWait

You can also launch a complete setup by calling the LaunchApp or the LaunchAppAndWait function.

For example, if you placed the child setup's files in the Setup Files pane, you would call LaunchAppAndWait as follows:

```
LaunchAppAndWait (SUPPORTDIR ^ "Setup.exe", "");
```

If, for example, you left the child setup's files on a CD-ROM, you would call LaunchAppAndWait as follows:

```
LaunchAppAndWait (SRCDISK ^ "Launched Setup Folder\\Setup.exe", "");
```

Alternatively, you could insert links to the child setup's files into your file groups, install the files onto the target system, and then launch the setup from the target location. (Of course, this method will leave the setup files on the target system until the parent setup was uninstalled.)

Launch another application

Call LaunchAppAndWait as in the following example:

```
LaunchAppAndWait( "Notepad.exe", TARGETDIR ^ "UserInfo.txt", WAIT );
```