What is Inno Setup?

Inno Setup version 4.0.10

Contacting Me

Inno Setup is a *free* installer for Windows programs. First introduced in 1997, Inno Setup today rivals and even surpasses many commercial installers in feature set and stability.

Key features:

- Support for all 32-bit Windows versions in use today -- Windows 95, 98, 2000, XP, Me, NT 4.0.
- Supports creation of a single EXE to install your program for easy online distribution. Disk spanning is also supported.
- Standard Windows 2000/XP-style wizard interface.
- Customizable setup types, e.g. Full, Minimal, Custom.
- Complete uninstall capabilities.
- Installation of files:
  Includes integrated "deflate" file compression (the same compression .zip files use) and also supports bzip2 compression. The installer has the ability to compare file version info, replace in-use files, use shared file counting, register DLL/OCX's and type libraries, and install fonts.
- Creation of shortcuts anywhere, including in the Start Menu and on the desktop.
- Creation of registry and .INI entries.
- Silent install and silent uninstall.
- Full source code is available (Borland Delphi 2.0-5.0).

Is it really free of charge, even for commercial use?

Yes, it may be used completely free of charge, even when deploying commercial applications.

(Note: "Completely free of charge" must not be confused with "completely free". Inno Setup is copyrighted software, *not* public domain software. There are some restrictions on distribution and use; see the LICENSE.TXT file for details.)

$jrsoftware: ishelp/isetup.rtf,v 1.147 2003/11/14 20:44:53 jr Exp $

Documentation Conventions

"Windows 98/NT 4+"	This is shorthand for "Windows 98, 2000, XP, NT 4.0, Me, and later."
"Windows NT"	Whenever Windows NT is mentioned, it includes Windows 2000 and XP (which are NT 5), unless otherwise indicated.
`monospaced text`	When you see monospaced text in the documentation, it refers to text you would type in a script file.

Creating Installations

Installations are created by means of *scripts*, which are ASCII text files with a format somewhat similar to .INI files. (No, it's not as complicated as you might be thinking!)

Scripts have an ".iss" (meaning Inno Setup Script) extension. The script controls every aspect of the installation. It specifies which files are to be installed and where, what shortcuts are to be created and what they are to be named, and so on.

Script files are usually edited from inside the Setup Compiler program. After you have finishing writing the script, the next and final step is select "Compile" in the Setup Compiler. What this does is create a complete, ready-to-run Setup program based on your script. By default, this is created in a directory named "Output" under the directory containing the script.

To give you an idea of how this all works, start the Setup Compiler, click *File* | *Open,* and select one of the script files in the Samples subdirectory located under the Inno Setup directory. (It may be helpful to use the sample scripts as a template for your own scripts.)

See also
Script Format Overview

Script Format Overview

Inno Setup Scripts are arranged into *sections.* Each section controls a different aspect of the installation. A section is started by specifying the name of the section enclosed in square brackets `[]`. Inside each section is any number of *entries.*

There are two different types of sections: those such as [Setup] whose entries contain directive names and values (in the form `Directive=Value`), and those such as [Files] whose entries are divided into parameters.

Here is an example:

```
[Setup]
AppName=My Program

[Files]
Source: "MYPROG.EXE"; DestDir: "{app}"
```

Note that it is legal in Inno Setup 3 to specify multiple sections of the same name.

You can put "comments" in the script (which are ignored by the compiler) by placing a semicolon at the beginning of a line. For example:

```
; This is a comment. I could put reminders to myself here...
```

A C-like `#include` directive is supported, which pulls in lines from a separate file into the script at the position of the `#include` directive. The syntax is:

```
#include "filename.txt"
```

If the filename is not fully qualified, the compiler will look for it in the same directory as the file containing the `#include` directive. The filename may be prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

See also

Parameters in Sections
Constants
[Setup] section
[Types] section
[Components] section
[Tasks] section
[Dirs] section
[Files] section
[Icons] section
[INI] section
[InstallDelete] section
[Languages] section
[Messages] section
[LangOptions] section
[Registry] section
[Run] section
[UninstallDelete] section
[UninstallRun] section
Pascal Scripting: Introduction

Parameters in Sections

All of the sections in a script, with the exception of `[Setup]`, `[Messages]`, and `[LangOptions]`, contain lines separated into *parameters*. The following is an example of a `[Files]` section:

```
[Files]
Source: "MYPROG.EXE"; DestDir: "{app}"
Source: "MYPROG.HLP"; DestDir: "{app}"
Source: "README.TXT"; DestDir: "{app}"; Flags: isreadme
```

Each parameter consists of a name, followed by a colon, and then a value. Unless otherwise noted, parameters are optional in that they assume a default value if they are not specified. Multiple parameters on a line are separated by semicolons, and can be listed in any order.

The value of a parameter is traditionally surrounded in double quotes (`"`) when it contains a user-defined string, such as a filename. Using quotes is not required, though, but by doing so it makes it possible to embed leading and trailing spaces in the value, as well as semicolons and double-quote characters.

To embed a double-quote character inside a quoted value, use two consecutive double-quote characters. For example:

```
"This "" contains "" embedded "" quotes"
```

The Setup Compiler would see that as:

```
This " contains " embedded " quotes
```

If you want the value of a parameter to be a single double-quote character, use four double-quote characters: `""""`. The outer two are needed to surround the string in quotes; the inner two are used to embed a single double-quote character.

Constants

The majority of the script entries can have *constants* embedded in them. These are predefined strings enclosed in brace characters `{ }`. Setup translates the constants to their literal values, depending on the user's choices and system configuration. For example, `{win}`, as described below, would translate to "C:\WINDOWS" on most systems.

A "{" character is treated as the start of the constant. If you want to use that actual character in a place where constants are supported, you must use two consecutive "{" characters.

When a backslash immediately follows a constant, Inno Setup automatically removes the backslash if the value of the constant ended in a backslash already. This way if a particular constant pointed to "C:\", `{constantname}\file` will translate to "C:\file" and not "C:\\file". If you want to prevent this from happening, enclose the backslash in `{ }` characters, e.g. `{app}{\}`.

The following is the list of supported constants.

Directory Constants

{app}
The application directory, which the user selects on the *Select Destination Directory* page of the wizard.
For example: If you used `{app}\MYPROG.EXE` on an entry and the user selected "C:\MYPROG" as the application directory, Setup will translate it to "C:\MYPROG\MYPROG.EXE".

{win}
The system's Windows directory.
For example: If you used `{win}\MYPROG.INI` on an entry and the system's Windows directory is "C:\WINDOWS", Setup will translate it to "C:\WINDOWS\MYPROG.INI".

{sys}
The system's Windows System directory (System32 on Windows NT platforms).
For example: If you used `{sys}\CTL3D32.DLL` on an entry and the system's Windows System directory is "C:\WINDOWS\SYSTEM", Setup will translate it to "C:\WINDOWS\SYSTEM\CTL3D32.DLL".

{src}
The directory in which the Setup files are located.
For example: If you used `{src}\MYPROG.EXE` on an entry and the user is installing from "S:\", Setup will translate it to "S:\MYPROG.EXE".

{sd}
System Drive. The drive Windows is installed on, typically "C:". On Windows NT platforms, this directory constant is equivalent to the *SystemDrive* environment variable.

{pf}
Program Files. The path of the system's Program Files directory, typically "C:\Program Files".

{cf}
Common Files. The path of the system's Common Files directory, typically "C:\Program Files\Common Files".

{tmp}
Temporary directory. This is *not* the value of the user's TEMP environment variable. It is a subdirectory of the user's temporary directory which is created at installation startup (with a name like "C:\WINDOWS\TEMP\IS-xxxxx.tmp"). All files and subdirectories in this directory are deleted when the setup program exits. This is primarily useful for extracting files that are to be executed in the [Run] section but aren't needed after the installation.

{fonts}
Fonts directory. Normally named "FONTS" under the Windows directory.

{dao}
DAO directory. This is equivalent to `{cf}\Microsoft Shared\DAO`.

Shell Folder Constants

Inno Setup supports another set of directory constants, referred to as *shell folder constants*. They can be used in the same way as the other directory constants.

The "user" constants below refer to the currently logged in user's profile. "common" constants refer to the *All Users* profile. When an installation is run on a Windows NT-platform system by a user without administrative privileges, all of the "common" constants are equivalent to the "user" constants.

Except where otherwise noted, shell folder constants work on all versions of Windows that Inno Setup supports, including Windows 95 and NT 4.0.

* = The "common" form of this constant is equivalent to the "user" form on Windows 9x/Me.

{group}
The path to the Start Menu folder, as selected by the user on Setup's *Select Start Menu Folder* wizard page. On Windows NT, this folder is always created under the *All Users* profile unless the user installing the application does not have administrative privileges, in which case it is created on the user's profile.

{localappdata}
The path to the local (nonroaming) Application Data folder.

{sendto}
The path to the current user's Send To folder. (There is no common Send To folder.)

{userappdata} & **{commonappdata}**
The path to the Application Data folder.

{userdesktop} & **{commondesktop}** *
The path to the desktop folder. It's recommended that desktop shortcuts be placed in `{userdesktop}`.

{userdocs} & **{commondocs}**
The path to the My Documents folder (or on NT 4.0, the Personal folder).

{userfavorites} & **{commonfavorites}**
The path to the Favorites folder. Usage of these constants requires a MinVersion setting of at least "`4.1, 4`". Currently only Windows 2000, Me, and later support `{commonfavorites}`; if used on previous Windows versions, it will translate to the same directory as `{userfavorites}`.

{userprograms} & **{commonprograms}** *
The path to the Programs folder on the Start Menu.

{userstartmenu} & **{commonstartmenu}** *
The path to the top level of the Start Menu.

{userstartup} & **{commonstartup}** *
The path to the Startup folder on the Start Menu.

{usertemplates} & **{commontemplates}**
The path to the Templates folder. Currently only Windows 2000, Me, and later support `{commontemplates}`; if used on previous Windows versions, it will translate to the same directory as `{usertemplates}`.

Other Constants

{\}
A backslash character. See the note at the top of this page for an explanation of what the difference between using `{\}` and only a `\` is.

{%*NAME*|*DefaultValue*}
Embeds the value of an environment variable.

- *NAME* specifies the name of the environment variable to use.
- *DefaultValue* determines the string to embed if the specified variable does not exist on the user's system.
- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *NAME* and *DefaultValue* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

Examples:
```
{%COMSPEC}
{%PROMPT|$P$G}
```

{cmd}
The full pathname of the system's standard command interpreter. On Windows NT/2000/XP, this is *Windows**System32*\cmd.exe. On Windows 9x/Me, this is *Windows*\COMMAND.COM. Note that the COMSPEC environment variable is not used when expanding this constant.

{computername}
The name of the computer the Setup program is running on (as returned by the *GetComputerName* function).

{drive:*Path*}
Extracts and returns the drive letter and colon (e.g. "C:") from the specified path. In the case of a UNC path, it returns the server and share name (e.g. "\\SERVER\SHARE").

- *Path* specifies the path.
- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *Path* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

Examples:
```
{drive:{src}}
{drive:c:\path\file}
{drive:\\server\share\path\file}
```

{groupname}
The name of the folder the user selected on Setup's *Select Start Menu Folder* wizard page. This differs from `{group}` in that it is only the name; it does not include a path.

{hwnd}
(Special-purpose) Translates to the window handle of the Setup program's background window.

{wizardhwnd}
(Special-purpose) Translates to the window handle of the Setup wizard window. This handle is set to '0' if the wizard window handle isn't available at the time the translation is done.

{ini:*Filename*,*Section*,*Key*|*DefaultValue*}
Embeds a value from an .INI file.

- *Filename* specifies the name of the .INI file to read from.
- *Section* specifies the name of the section to read from.
- *Key* specifies the name of the key to read.
- *DefaultValue* determines the string to embed if the specified key does not exist.
- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *Filename, Section,* and *Key* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

Example:
```
{ini:{win}\MyProg.ini,Settings,Path|{pf}\My Program}
```

{language}
The internal name of the selected language. See the [Languages] section documentation for more information.

{reg:HK*xxSubkeyName*,*ValueName*|*DefaultValue*}**
Embeds a registry value.

- HK*xx* specifies the root key; see the [Registry] section documentation for a list of possible root keys.
- *SubkeyName* specifies the name of the subkey to read from.
- *ValueName* specifies the name of the value to read; leave *ValueName* blank if you wish to read the "default" value of a key.
- *DefaultValue* determines the string to embed if the specified registry value does not exist, or is not a string type (REG_SZ or REG_EXPAND_SZ).
- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *SubkeyName, ValueName,* and *DefaultValue* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

Example:
```
{reg:HKLM\Software\My Program,Path|{pf}\My Program}
```

{param:*ParamName*|*DefaultValue*}
Embeds a command line parameter value.

- *ParamName* specifies the name of the command line parameter to read from.
- *DefaultValue* determines the string to embed if the specified command line parameter does not exist, or its value could not be determined.

- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *ParamName* and *DefaultValue* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

Example:
`{param:Path|{pf}\My Program}`

The example above translates to `c:\My Program` if the command line `"/Path=c:\My Program"` was specified.

{srcexe}
The full pathname of the Setup program file, e.g. "C:\SETUP.EXE".

{sysuserinfoname}
{sysuserinfoorg}
The name and organization, respectively, that Windows is registered to. This information is read from the registry.

{uninstallexe}
The full pathname of the uninstall program extracted by Setup, e.g. "C:\Program Files\My Program\unins000.exe". This constant is typically used in an [Icons] section entry for creating an Uninstall icon. It is only valid if `Uninstallable` is `yes` (the default setting).

{userinfoname}
{userinfoorg}
{userinfoserial}
The name, organization and serial number, respectively, that the user entered on the *User Information* wizard page (which can be enabled via the `UserInfoPage` directive). Typically, these constants are used in [Registry] or [INI] entries to save their values for later use.

{username}
The name of the user who is running Setup program (as returned by the *GetUserName* function).

Common Parameters

There are three optional parameters that are supported by all sections whose entries are separated into parameters. They are:

Languages

Description:
A space separated list of language names, telling Setup to which languages the entry belongs. If the end user selects a language from this list, the entry is processed (for example: the file is installed).

An entry without a Languages parameter is always installed, unless other parameters say it shouldn't.

Example:
```
Languages: en nl
```

MinVersion

Description:
A minimum Windows version and Windows NT version respectively for the entry to be processed. If you use "0" for one of the versions then the entry will never be processed on that platform. Build numbers and/or service pack levels may be included in the version numbers. This overrides any `MinVersion` directive in the script's `[Setup]` section.

An entry without a MinVersion parameter is always installed, unless other parameters say it shouldn't.

Example:
```
MinVersion: 4.0,4.0
```

OnlyBelowVersion

Description:
Basically the opposite of `MinVersion`. Specifies the minimum Windows and Windows NT version for the entry *not* to be processed. For example, if you put `4.1,5.0` and the user is running Windows 95 or NT 4.0 the entry *will* be processed, but if the user is running Windows 98 (which reports its version as 4.1) or Windows 2000 (which reports its version as NT 5.0), it will *not* be processed. Putting "0" for one of the versions means there is no upper version limit. Build numbers and/or service pack levels may be included in the version numbers. This overrides any `OnlyBelowVersion` directive in the script's `[Setup]` section.

An entry without an OnlyBelowVersion parameter is always installed, unless other parameters say it shouldn't.

Example:
```
OnlyBelowVersion: 4.1,5.0
```

Components and Tasks Parameters

There are two optional parameters that are supported by all sections whose entries are separated into parameters, except [Types], [Components] and [Tasks]. They are:

Components

Description:
A space separated list of component names, telling Setup to which components the entry belongs. If the end user selects a component from this list, the entry is processed (for example: the file is installed).

An entry without a Components parameter is always installed, unless other parameters say it shouldn't.

Example:
```
[Files]
Source: "MYPROG.EXE"; DestDir: "{app}"; Components: main
Source: "MYPROG.HLP"; DestDir: "{app}"; Components: help
Source: "README.TXT"; DestDir: "{app}"
```

Tasks

Description:
A space separated list of task names, telling Setup to which task the entry belongs. If the end user selects a task from this list, the entry is processed (for example: the file is installed).

An entry without a Tasks parameter is always installed, unless other parameters say it shouldn't.

The *Don't create any icons* checkbox doesn't control [Icons] entries that have a Task parameter since these have their own checkboxes. Therefore Setup will change the *Don't create any icons* text to *Don't create a Start Menu folder* if you have any icons with a Task parameter.

Example:
```
[Icons]
Name: "{group}\My Program"; Filename: "{app}\MyProg.exe"; Components: main;
Tasks: startmenu
Name: "{group}\My Program Help"; Filename: "{app}\MyProg.hlp"; Components:
help; Tasks: startmenu
Name: "{userdesktop}\My Program"; Filename: "{app}\MyProg.exe"; Components:
main; Tasks: desktopicon
```

[Setup] section

This section contains global settings used by the installer and uninstaller. Certain directives are required for any installation you create. Here is an example of a `[Setup]` section:

```
[Setup]
AppName=My Program
AppVerName=My Program version 1.4
DefaultDirName={pf}\My Program
DefaultGroupName=My Program
```

The following directives can be placed in the `[Setup]` section:

(**bold** = required)

Compiler-related

- Compression
- DiskClusterSize
- DiskSliceSize
- DiskSpanning
- DontMergeDuplicateFiles
- InternalCompressLevel
- OutputBaseFilename
- OutputDir
- ReserveBytes
- SlicesPerDisk
- SolidCompression
- SourceDir
- UseSetupLdr

Installer-related

Functional: These directives affect the operation of the Setup program, or are saved and used later by the uninstaller.

- AllowCancelDuringInstall
- AllowNoIcons
- AllowRootDirectory
- AllowUNCPath
- AlwaysRestart
- AlwaysShowComponentsList
- AlwaysShowDirOnReadyPage
- AlwaysShowGroupOnReadyPage
- AlwaysUsePersonalGroup
- **AppName**
- AppId
- AppMutex
- AppPublisher
- AppPublisherURL
- AppSupportURL
- AppUpdatesURL
- AppVersion
- **AppVerName**
- ChangesAssociations

- CreateAppDir
- CreateUninstallRegKey
- **DefaultDirName**
- DefaultGroupName
- DefaultUserInfoName
- DefaultUserInfoOrg
- DefaultUserInfoSerial
- DirExistsWarning
- DisableAppendDir
- DisableDirPage
- DisableFinishedPage
- DisableProgramGroupPage
- DisableReadyMemo
- DisableReadyPage
- DisableStartupPrompt
- EnableDirDoesntExistWarning
- ExtraDiskSpaceRequired
- InfoAfterFile
- InfoBeforeFile
- LanguageDetectionMethod
- LicenseFile
- MinVersion
- OnlyBelowVersion
- Password
- PrivilegesRequired
- RestartIfNeededByRun
- ShowLanguageDialog
- TimeStampsInUTC
- Uninstallable
- UninstallDisplayIcon
- UninstallDisplayName
- UninstallFilesDir
- UninstallLogMode
- UninstallRestartComputer
- UpdateUninstallLogAppName
- UsePreviousAppDir
- UsePreviousGroup
- UsePreviousSetupType
- UsePreviousTasks
- UsePreviousUserInfo
- UserInfoPage

Cosmetic: These directives are used only for display purposes in the Setup program.

- AppCopyright
- BackColor
- BackColor2
- BackColorDirection
- BackSolid
- FlatComponentsList
- ShowComponentSizes
- ShowTasksTreeLines
- UninstallStyle
- WindowShowCaption
- WindowStartMaximized
- WindowResizable

- WindowVisible
- WizardImageBackColor
- WizardImageFile
- WizardSmallImageBackColor
- WizardSmallImageFile

Obsolete

- AdminPrivilegesRequired
- AlwaysCreateUninstallIcon
- Bits
- CompressLevel
- DisableDirExistsWarning
- MessagesFile
- OverwriteUninstRegEntries
- UninstallIconName
- WizardStyle

[Types] section

This section is optional. It defines all of the setup types Setup will show on the *Select Components* page of the wizard. During compilation a set of default setup types is created if you define components in a [Components] section but don't define types. If you are using the default (English) messages file, these types are the same as the types in the example below.

Here is an example of a `[Types]` section:

```
[Types]
Name: "full"; Description: "Full installation"
Name: "compact"; Description: "Compact installation"
Name: "custom"; Description: "Custom installation"; Flags: iscustom
```

The following is a list of the supported parameters:

Name *(Required)*

Description:
The internal name of the type. Used as parameter for components in the [Components] section to instruct Setup to which types a component belongs.

Example:
```
Name: "full"
```

Description *(Required)*

Description:
The description of the type, which can include constants. This description is shown during installation.

Example:
```
Description: "Full installation"
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

iscustom
Instructs Setup that the type is a custom type. Whenever the end user manually changes the components selection during installation, Setup will set the setup type to the custom type. Note that if you don't define a custom type, Setup will only allow the user to choose a setup type and he/she can no longer manually select/unselect components.

Example:
```
Flags: iscustom
```

Common Parameters

[Components] section

This section is optional. It defines all of the components Setup will show on the *Select Components* page of the wizard for setup type customization.

By itself a component does nothing: it needs to be 'linked' to other installation entries. See Components and Tasks Parameters.

Here is an example of a `[Components]` section:

```
[Components]
Name: "main"; Description: "Main Files"; Types: full compact; Flags: fixed
Name: "help"; Description: "Help Files"; Types: full
Name: "help\english"; Description: "English"; Types: full
Name: "help\dutch"; Description: "Dutch"; Types: full
```

The example above generates four components: A "main" component which gets installed if the end user selects a type with name "full" or "compact" and a "help" component which has two child components and only gets installed if the end user selects the "full" type.

The following is a list of the supported parameters:

Name *(Required)*

Description:
The internal name of the component.

The total number of \ or / characters in the name of the component is called the level of the component. Any component with a level of 1 or more is a child component. The component listed before the child component with a level of 1 less than the child component, is the parent component. Other components with the same parent component as the child component are sibling components.

A child component can't be selected if its parent component isn't selected. A parent component can't be selected if none of it child components are selected and it doesn't install anything itself.

If sibling components have the `exclusive` flag, only one of them can be selected.

Example:
`Name: "help"`

Description *(Required)*

Description:
The description of the component, which can include constants. This description is shown to the end user during installation.

Example:
`Description: "Help Files"`

Types

Description:
A space separated list of types this component belongs to. If the end user selects a type from this list, this component will be installed.

If the `fixed` flag isn't used (see below), any custom types (types using the `iscustom` flag) in this list are ignored by Setup.

Example:
`Types: full compact`

ExtraDiskSpaceRequired

Description:

The extra disk space required by this component, similar to the ExtraDiskSpaceRequired directive for the [Setup] section.

Example:
```
ExtraDiskSpaceRequired: 0
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

exclusive
Instructs Setup that this component is mutually exclusive with sibling components that also have the `exclusive` flag.

fixed
Instructs Setup that this component can not be manually selected or unselected by the end user during installation.

restart
Instructs Setup to ask the user to restart the system if this component is installed, regardless of whether this is necessary (for example because of [Files] section entries with the `restartreplace` flag). Like AlwaysRestart but per component.

disablenouninstallwarning
Instructs Setup not to warn the user that this component will not be uninstalled after he/she deselected this component when it's already installed on his/her machine.

Depending on the complexity of your components, you can try to use the [InstallDelete] section and this flag to automatically 'uninstall' deselected components.

Example:
```
Flags: fixed
```

Common Parameters

[Tasks] section

This section is optional. It defines all of the user-customizable tasks Setup will perform during installation. These tasks appear as check boxes and radio buttons on the *Select Additional Tasks* wizard page.

By itself a task does nothing: it needs to be 'linked' to other installation entries. See Components and Tasks Parameters.

Here is an example of a `[Tasks]` section:

```
[Tasks]
Name: desktopicon; Description: "Create a &desktop icon"; GroupDescription:
"Additional icons:"; Components: main
Name: desktopicon\common; Description: "For all users"; GroupDescription:
"Additional icons:"; Components: main; Flags: exclusive
Name: desktopicon\user; Description: "For the current user only";
GroupDescription: "Additional icons:"; Components: main; Flags: exclusive
unchecked
Name: quicklaunchicon; Description: "Create a &Quick Launch icon";
GroupDescription: "Additional icons:"; Components: main; Flags: unchecked
Name: associate; Description: "&Associate files"; GroupDescription: "Other
tasks:"; Flags: unchecked
```

The following is a list of the supported parameters:

Name *(Required)*

Description:
The internal name of the task.

The total number of \ or / characters in the name of the task is called the level of the task. Any task with a level of 1 or more is a child task. The task listed before the child task with a level of 1 less than the child task, is the parent task. Other tasks with the same parent task as the child task are sibling tasks.

A child task can't be selected if its parent task isn't selected. A parent task can't be selected if none of it child tasks are selected and it doesn't install anything itself.

If sibling tasks have the `exclusive` flag, only one of them can be selected.

Example:
```
Name: "desktopicon"
```

Description *(Required)*

Description:
The description of the task, which can include constants. This description is shown to the end user during installation.

Example:
```
Description: "Create a &desktop icon"
```

GroupDescription

Description:
The group description of a group of tasks, which can include constants. Consecutive tasks with the same group description will be grouped below a text label. The text label shows the group description.

Example:
```
GroupDescription: "Additional icons"
```

Components

Description:
A space separated list of components this task belongs to. If the end user selects a component from this list, this task will be shown. A task entry without a Components parameter is always shown.

Example:
`Components: main`

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

checkedonce
Instructs Setup that this task should be unchecked initially when Setup finds a previous version of the same application is already installed. This flag cannot be combined with the `unchecked` flag.

If the `UsePreviousTasks [Setup]` section directive is `no`, this flag is effectively disabled.

exclusive
Instructs Setup that this task is mutually exclusive with sibling tasks that also have the `exclusive` flag.

restart
Instructs Setup to ask the user to restart the system at the end of installation if this task is selected, regardless of whether it is necessary (for example because of [Files] section entries with the `restartreplace` flag). Like AlwaysRestart but per task.

unchecked
Instructs Setup that this task should be unchecked initially. This flag cannot be combined with the `checkedonce` flag.

Example:
`Flags: unchecked`

Common Parameters

[Dirs] section

This optional section defines any additional directories Setup is to create *besides* the application directory the user chooses, which is created automatically. Creating subdirectories off the main application directory is a common use for this section.

Note that you aren't required to explicitly create directories before installing files to them using the [Files] section, so this section is primarily useful for creating empty directories.

Here is an example of a `[Dirs]` section:

```
[Dirs]
Name: "{app}\data"
Name: "{app}\bin"
```

The example above will, after Setup creates the application directory, create two subdirectories off the application directory.

The following is a list of the supported parameters:

Name *(Required)*

Description:
The name of the directory to create, which normally will start with one of the directory constants.

Example:
```
Name: "{app}\MyDir"
```

Attribs

Description:
Specifies additional attributes for the directory. This can include one or more of the following: `readonly`, `hidden`, `system`. If this parameter is not specified, Setup does not assign any special attributes to the directory.

If the directory already exists, the specified attributes will be combined with the directory's existing attributes.

Example:
```
Attribs: hidden system
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

deleteafterinstall
Instructs Setup to create the directory as usual, but then delete it once the installation is completed (or aborted) if it's empty. This can be useful when extracting temporary data needed by a program executed in the script's [Run] section.

This flag will not cause directories that already existed before installation to be deleted.

uninsalwaysuninstall
Instructs the uninstaller to always attempt to delete the directory if it's empty. Normally the uninstaller will only try to delete the directory if it didn't already exist prior to installation.

uninsneveruninstall
Instructs the uninstaller to not attempt to delete the directory. By default, the uninstaller deletes any directory specified in the [Dirs] section if it is empty.

Example:
```
Flags: uninsneveruninstall
```

Components and Tasks Parameters

Common Parameters

[Files] section

This optional section defines any files Setup is to install on the user's system.

Here is an example of a `[Files]` section:

```
[Files]
Source: "CTL3DV2.DLL"; DestDir: "{sys}"; Flags: onlyifdoesntexist
uninsneveruninstall
Source: "MYPROG.EXE"; DestDir: "{app}"
Source: "MYPROG.HLP"; DestDir: "{app}"
Source: "README.TXT"; DestDir: "{app}"; Flags: isreadme
```

See the *Remarks* section at the bottom of this topic for some important notes.

The following is a list of the supported parameters:

Source *(Required)*

Description:
The name of the *source file*. The compiler will prepend the path of your installation's source directory if you do not specify a fully qualified pathname.

This can be a wildcard to specify a group of files in a single entry. When a wildcard is used, all files matching it use the same options.

When the flag `external` is specified, `Source` must be the full pathname of an existing file (or wildcard) on the distribution media or the user's system (e.g. "{src}\license.ini").

Constants may only be used when the `external` flag is specified, because the compiler does not do any constant translating itself.

Examples:
```
Source: "MYPROG.EXE"
Source: "Files\*"
```

DestDir *(Required)*

Description:
The directory where the file is to be installed on the user's system. The will almost always begin with one of the directory constants. If the specified path does not already exist on the user's system, it will be created automatically, and removed automatically during uninstallation if empty.

Examples:
```
DestDir: "{app}"
DestDir: "{app}\subdir"
```

DestName

Description:
This parameter specifies a new name for the file when it is installed on the user's system. By default, Setup uses the name from the `Source` parameter, so in most cases it's not necessary to specify this parameter.

Example:
```
DestName: "MYPROG2.EXE"
```

CopyMode

Description:
You should not use this parameter in any new scripts. This parameter was deprecated and replaced by flags in Inno Setup 3.0.5:

`CopyMode: normal -> Flags: promptifolder`
`CopyMode: alwaysskipifsameorolder ->` no flags
`CopyMode: onlyifdoesntexist -> Flags: onlyifdoesntexist`
`CopyMode: alwaysoverwrite -> Flags: ignoreversion`
`CopyMode: dontcopy -> Flags: dontcopy`

What was `CopyMode: alwaysskipifsameorolder` is now the default behavior. (The previous default was `CopyMode: normal`.)

Attribs

Description:
Specifies additional attributes for the file. This can include one or more of the following: `readonly`, `hidden`, `system`. If this parameter is not specified, Setup does not assign any special attributes to the file.

Example:
```
Attribs: hidden system
```

FontInstall

Description:
Tells Setup the file is a font that needs to be installed. The value of this parameter is the name of the font as stored in the registry or WIN.INI. This must be exactly the same name as you see when you double-click the font file in Explorer. Note that Setup will automatically append " (TrueType)" to the end of the name.

If the file is not a TrueType font, you must specify the flag `fontisnttruetype` in the Flags parameter.

It's recommended that you use the flags `onlyifdoesntexist` and `uninsneveruninstall` when installing fonts to the {fonts} directory.

To successfully install a font on Windows 2000/XP, the user must be a member of the Power Users or Administrators groups. On Windows NT 4.0 and earlier, anyone can install a font.

Example:
```
Source: "OZHANDIN.TTF"; DestDir: "{fonts}"; FontInstall: "Oz Handicraft
BT"; Flags: onlyifdoesntexist uninsneveruninstall
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

allowunsafefiles
Disables the compiler's automatic checking for unsafe files. It is strongly recommended that you DO NOT use this flag, unless you are absolutely sure you know what you're doing.

comparetimestamp
(Not recommended; see below)
Instructs Setup to proceed to comparing time stamps if the file being installed already exists on the user's system, and at least one of the following conditions is true:

1. The existing file and the file being installed have the same version number (as determined by the files' version info).
2. Neither the existing file nor the file being installed has version info.
3. The `ignoreversion` flag is also used on the entry.

If the existing file has an older time stamp than the file being installed, the existing file will

replaced. Otherwise, it will not be replaced.

Use of this flag is *not recommended* except as a last resort, because there are at least two issues that may affect you. First, NTFS partitions store time stamps in UTC (unlike FAT partitions), which causes local time stamps -- what Inno Setup works with -- to shift whenever a user changes their system's time zone or when daylight saving time goes into or out of effect. This can create a situation where files are replaced when the user doesn't expect them to be, or not replaced when the user expects them to be. A separate problem, but one with a similar outcome, can occur if an installation was compiled on an NTFS partition and the files are installed to a FAT partition, because on FAT partitions time stamps only have a 2-second resolution.

confirmoverwrite
Always ask the user to confirm before replacing an existing file.

deleteafterinstall
Instructs Setup to install the file as usual, but then delete it once the installation is completed (or aborted). This can be useful for extracting temporary data needed by a program executed in the script's [Run] section.

This flag will not cause existing files that weren't replaced during installation to be deleted.

This flag cannot be combined with the `isreadme`, `regserver`, `regtypelib`, `restartreplace`, `sharedfile`, or `uninsneveruninstall` flags.

dontcopy
Don't copy the file to the user's system. This flag is useful if the file is handled by the [Code] section exclusively.

external
This flag instructs Inno Setup not to statically compile the file specified by the `Source` parameter into the installation files, but instead copy from an existing file on the distribution media or the user's system. See the `Source` parameter description for more information.

fontisnttruetype
Specify this flag if the entry is installing a *non-TrueType* font with the `FontInstall` parameter.

ignoreversion
Don't compare version info at all; replace existing files regardless of their version number.

This flag should only be used on files private to your application, *never* on shared system files.

isreadme
File is the "README" file. Only *one* file in an installation can have this flag. When a file has this flag, the user will asked if he/she would like to view the README file after the installation has completed. If Yes is chosen, Setup will open the file, using the default program for the file type. For this reason, the README file should always end with an extension like .txt, .wri, or .doc.

Note that if Setup has to restart the user's computer (as a result of installing a file with the flag `restartreplace` or if the `AlwaysRestart [Setup]` section directive is `yes`), the user will not be given an option to view the README file.

noregerror
When combined with either the `regserver` or `regtypelib` flags, Setup will not display any error message if the registration fails.

onlyifdestfileexists
Only install the file if a file of the same name already exists on the user's system. This flag may be useful if your installation is a patch to an existing installation, and you don't want files to be installed that the user didn't already have.

onlyifdoesntexist
Only install the file if it doesn't already exist on the user's system.

overwritereadonly
Always overwrite a read-only file. Without this flag, Setup will ask the user if an existing read-only file should be overwritten.

promptifolder
By default, when a file being installed has an older version number (or older time stamp, when the `comparetimestamp` flag is used) than an existing file, Setup will not replace the existing file. (See the *Remarks* section at the bottom of this topic for more details.) When this flag is used, Setup will ask the user whether the file should be replaced, with the default answer being to keep the existing file.

recursesubdirs
Instructs the compiler to also search for the `Source` filename/wildcard in subdirectories under the `Source` directory. This flag cannot be combined with the `external` flag.

regserver
Register the OLE server (a.k.a. ActiveX control). With this flag set, Setup will locate and execute the DLL/OCX's DllRegisterServer export. The uninstaller calls DllUnregisterServer. When used in combination with `sharedfile`, the DLL/OCX will only be unregistered when the reference count reaches zero.

See the *Remarks* at the bottom of this topic for more information.

regtypelib
Register the type library (.tlb). The uninstaller will unregister the type library (unless the flag `uninsneveruninstall` is specified). As with the `regserver` flag, when used in combination with `sharedfile`, the file will only be unregistered by the uninstaller when the reference count reaches zero.

See the *Remarks* at the bottom of this topic for more information.

restartreplace
This flag is generally useful when replacing core system files. If the file existed beforehand and was found to be locked resulting in Setup being unable to replace it, Setup will register the file (either in WININIT.INI or by using MoveFileEx, for Windows and Windows NT respectively) to be replaced the next time the system is restarted. When this happens, the user will be prompted to restart the computer at the end of the installation process.

To maintain compatibility with Windows 95/98 and Me, long filenames should not be used on an entry with this flag. Only "8.3" filenames are supported. (Windows NT platforms do not have this limitation.)

IMPORTANT: The *restartreplace* flag will only successfully replace an in-use file on Windows NT platforms if the user has administrative privileges. If the user does not have administrative privileges, this message will be displayed: "RestartReplace failed: MoveFileEx failed; code 5." Therefore, when using *restartreplace* it is highly recommended that you have your installation require administrative privileges by setting "PrivilegesRequired=admin" in the [Setup] section.

sharedfile
Uses Windows' shared file counting feature (located in the registry at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs). This enables a file to be shared between applications, without worrying about it being inadvertently removed. Each time the file is installed, the *reference count* for the file is incremented. When an application using the file is uninstalled, the reference count is decremented. If the count reaches zero, the file is deleted (with the user's confirmation).

Most files installed to the Windows System directory should use this flag, including .OCX and .DPL (Delphi 3 package) files. One of the few exceptions is MFC DLLs, which should *not* use this flag. Instead, the flags `onlyifdoesntexist` and `uninsneveruninstall` should be used. Or if installing the latest version of a particular MFC DLL is an issue, use the flags

`uninsneveruninstall` and `restartreplace`.

skipifsourcedoesntexist
This flag only has an effect when the `external` flag is also used. It instructs the installer to silently skip over the entry if the source file does not exist, instead of displaying an error message.

uninsremovereadonly
When uninstalling the file, remove any read-only attribute from the file before attempting to delete it.

uninsrestartdelete
When this flag is used and the file is in use at uninstall time, the uninstaller will queue the file to be deleted when the system is restarted, and at the end of the uninstallation process ask the user if he/she wants to restart. This flag can be useful when uninstalling things like shell extensions which cannot be programmatically stopped. Note that administrative privileges are required on Windows NT/2000/XP for this flag to have an effect.

uninsneveruninstall
Never uninstall this file. This flag should be used sparingly, and is usually used in combination with the `onlyifdoesntexist` flag. It is meant to be used when installing a very common shared file, such as CTL3DV2.DLL or an MFC DLL, because you wouldn't want the uninstaller to delete the file since other applications make use of it also.

Example:
```
Flags: isreadme
```

Components and Tasks Parameters

Common Parameters

Remarks

If a file already exists on the user's system, it by default will be replaced according to the following rules:

1. If the existing file is an older version than the file being installed (as determined by the files' version info), the existing file will be replaced.
2. If the existing file is the same version as the file being installed, the existing file will not be replaced.
3. If the existing file is a newer version than the file being installed, or if the existing file has version info but the file being installed does not, the existing file will not be replaced.
4. If the existing file does not have version info, it will be replaced.

Certain flags such as `onlyifdoesntexist`, `ignoreversion`, and `promptifolder` alter the aforementioned rules.

Setup registers all files with the `regserver` or `regtypelib` flags as the last step of installation. However, if the `[Setup]` section directive `AlwaysRestart` is `yes`, or if there are files with the `restartreplace` flag, all files get registered on the next reboot (by creating an entry in Windows' *RunOnce* registry key).

When files with a .HLP extension (Windows help files) are uninstalled, the corresponding .GID and .FTS files are automatically deleted as well.

[Icons] section

This optional section defines any shortcuts Setup is to create in the Start Menu and/or other locations, such as the desktop.

Here is an example of an `[Icons]` section:

```
[Icons]
Name: "{group}\My Program"; Filename: "{app}\MYPROG.EXE"; WorkingDir:
"{app}"
Name: "{group}\Uninstall My Program"; Filename: "{uninstallexe}"
```

The following is a list of the supported parameters:

Name *(Required)*

Description:
The name and location of the shortcut to create. Any of the shell folder constants or directory constants may be used in this parameter.

Keep in mind that shortcuts are stored as literal files so any characters not allowed in normal filenames can't be used here. Also, because it's not possible to have two files with the same name, it's therefore not possible to have two shortcuts with the same name.

Examples:
```
Name: "{group}\My Program"
Name: "{group}\Subfolder\My Program"
Name: "{userdesktop}\My Program"
Name: "{commonprograms}\My Program"
Name: "{commonstartup}\My Program"
```

Filename *(Required)*

Description:
The command line filename for the shortcut, which normally begins with a directory constant.

Examples:
```
Filename: "{app}\MYPROG.EXE"
Filename: "{uninstallexe}"
```

Parameters

Description:
Optional command line parameters for the shortcut, which can include constants.

Example:
```
Parameters: "/play filename.mid"
```

WorkingDir

Description:
The working (or *Start In*) directory for the shortcut, which is the directory in which the program is started from. If this parameter is not specified or is blank, Windows will use a default path, which varies between the different Windows versions. This parameter can include constants.

Example:
```
WorkingDir: "{app}"
```

HotKey

Description:
The hot key (or "shortcut key") setting for the shortcut, which is a combination of keys with which the program can be started.

Note: If you change the shortcut key and reinstall the application, Windows may continue to recognize old shortcut key(s) until you log off and back on or restart the system.

Example:
```
HotKey: "ctrl+alt+k"
```

Comment

Description:
Specifies the *Comment* (or "description") field of the shortcut, which determines the popup hint for it in Windows 2000, Me, and later. Earlier Windows versions ignore the comment.

Example:
```
Comment: "This is my program"
```

IconFilename

Description:
The filename of a custom icon (located on the user's system) to be displayed. This can be an executable image (.exe, .dll) containing icons or a .ico file. If this parameter is not specified or is blank, Windows will use the file's default icon. This parameter can include constants.

Example:
```
IconFilename: "{app}\myicon.ico"
```

IconIndex

Default:
0

Description:
Zero-based index of the icon to use in the file specified by `IconFilename`.

If `IconIndex` is non-zero and `IconFilename` is not specified or is blank, it will act as if `IconFilename` is the same as `Filename`.

Example:
```
IconIndex: 0
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

closeonexit
When this flag is set, Setup will set the "Close on Exit" property of the shortcut. This flag only has an effect if the shortcut points to an MS-DOS application (if it has a .pif extension, to be specific). If neither this flag nor the `dontcloseonexit` flags are specified, Setup will not attempt to change the "Close on Exit" property.

createonlyiffileexists
When this flag is set, the installer will only try to create the icon if the file specified by the `Filename` parameter exists.

dontcloseonexit
Same as `closeonexit`, except it causes Setup to uncheck the "Close on Exit" property.

runmaximized
When this flag is set, Setup sets the "Run" setting of the icon to "Maximized" so that the program will be initially maximized when it is started.

runminimized

When this flag is set, Setup sets the "Run" setting of the icon to "Minimized" so that the program will be initially minimized when it is started.

uninsneveruninstall
Instructs the uninstaller not to delete the icon.

useapppaths
When this flag is set, specify just a filename (no path) in the `Filename` parameter, and Setup will retrieve the pathname from the "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths" registry key and prepend it to the filename automatically.

Example:
```
Flags: runminimized
```

Components and Tasks Parameters

Common Parameters

[INI] section

This optional section defines any .INI file entries you would like Setup to set on the user's system.

Here is an example of an `[INI]` section:

```
[INI]
Filename: "{win}\MYPROG.INI"; Section: "InstallSettings"; Flags:
uninsdeletesection
Filename: "{win}\MYPROG.INI"; Section: "InstallSettings"; Key:
"InstallPath"; String: "{app}"
```

The following is a list of the supported parameters:

Filename *(Required)*

Description:
The name of the .INI file you want Setup to modify, which can include constants. If this parameter is blank, it writes to WIN.INI in the system's Windows directory.

Example:
`Filename: "{win}\MYPROG.INI"`

Section *(Required)*

Description:
The name of the section to create the entry in, which can include constants.

Example:
`Section: "Settings"`

Key

Description:
The name of the key to set, which can include constants. If this parameter is not specified or is blank, no key is created.

Example:
`Key: "Version"`

String

Description:
The value to assign to the key, which can use constants. If this parameter is not specified, no key is created.

Example:
`String: "1.0"`

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

createkeyifdoesntexist
Assign to the key only if the key name doesn't already exist.

uninsdeleteentry
Delete the entry when the program is uninstalled. This can be combined with the `uninsdeletesectionifempty` flag.

uninsdeletesection
When the program is uninstalled, delete the entire section in which the entry is located. It

obviously wouldn't be a good idea to use this on a section that is used by Windows itself (like some of the sections in WIN.INI). You should only use this on sections private to your application.

uninsdeletesectionifempty
Same as `uninsdeletesection`, but deletes the section only if there are no keys left in it. This can be combined with the `uninsdeleteentry` flag.

Example:
```
Flags: uninsdeleteentry
```

Components and Tasks Parameters

Common Parameters

[InstallDelete] section

This optional section is identical in format to the [UninstallDelete] section, except its entries are processed as the first step of *installation.*

[Languages] section

Inno Setup supports multilingual installations. The [Languages] section defines the languages to make available to the Setup program.

Setup determines the default language to use for its messages in the following order:

1. It searches for a language whose `LanguageID` setting (normally specified in the [LangOptions] section of the language's .isl file) matches both the primary language identifier and sublanguage identifier of the current user's UI language or locale (depending on the setting of LanguageDetectionMethod).
2. If none is found, it searches for just a primary language identifier match. If two or more available languages have the same primary language identifier, it goes with the first one listed in the [Languages] section.
3. If none is found, it defaults to the first language specified in the [Languages] section.

If the ShowLanguageDialog [Setup] section directive is set to `yes` (the default), a *Select Language* dialog will be displayed which gives the user an opportunity to override the language Setup chose.

The following is an example of a `[Languages]` section. It defines two languages: English, based on the standard Default.isl file, and Dutch, based on a third-party translation.

```
[Languages]
Name: "en"; MessagesFile: "compiler:Default.isl"
Name: "nl"; MessagesFile: "Dutch.isl"
```

Name *(Required)*

Description:
The internal name of the language. This can used as a prefix on [LangOptions] or [Messages] section entries to have the entries apply to only one language. The {language} constant returns the internal name of the selected language.

Example:
```
Name: "en"
```

MessagesFile *(Required)*

Description:
Specifies the name(s) of file(s) to read the default messages from. The file(s) must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

When multiple files are specified, they are read in the order they are specified, thus the last message file overrides any messages in previous files.

See the [Messages] section help topic for details on the format of .isl files.

Examples:
```
MessagesFile: "compiler:Dutch.isl"
MessagesFile: "compiler:Default.isl,compiler:MyMessages.isl"
```

LicenseFile

Description:
Specifies the name of an optional license agreement file, in .txt or .rtf (rich text) format, which is displayed before the user selects the destination directory for the program. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

Example:
```
LicenseFile: "license-Dutch.txt"
```

InfoBeforeFile

Description:
Specifies the name of an optional "readme" file, in .txt or .rtf (rich text) format, which is displayed before the user selects the destination directory for the program. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

Example:
```
InfoBeforeFile: "infobefore-Dutch.txt"
```

InfoAfterFile

Description:
Specifies the name of an optional "readme" file, in .txt or .rtf (rich text) format, which is displayed after a successful install. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

This differs from `isreadme` files in that this text is displayed as a page of the wizard, instead of in a separate Notepad window.

Example:
```
InfoAfterFile: "infoafter-Dutch.txt"
```

[Messages] section

A [Messages] section is used to define the messages displayed by the Setup program and uninstaller. Normally, you need not create a [Messages] section in your script file, since all messages are, by default, pulled in from the file *Default.isl* included with Inno Setup (or whichever file is specified by a [Languages] section entry).

However, particular messages can be overridden by creating a [Messages] section in your script file. To do this, first you will need to know the ID of the message you want to change. This can be easily found by searching Default.isl. For example, say you wanted to change the "&Next >" button on the wizard to read "&Forward >". The ID of this message is "ButtonNext", so you would create a [Messages] section like this:

```
[Messages]
ButtonNext=&Forward >
```

Some messages take arguments such as %1 and %2. You can rearrange the order of the arguments (i.e. move the %2 before a %1) and also duplicate arguments if needed (i.e. "%1 ... %1 %2"). On messages with arguments, use two consecutive "%" characters to embed a single "%". "%n" creates a line break.

If you wish to translate all of Inno Setup's text to another language, instead of modifying Default.isl or overriding each message in every script you create, make a copy of Default.isl with another name like *MyTranslation.isl.* On any installation you wish to use MyTranslation.isl, create a [Languages] section entry pointing to the file.

In cases where there are multiple [Languages] section entries, specifying a [Messages] section entry in your script (as opposed to an .isl file) will by default override that message for all languages. To apply a [Messages] section entry to only one language, prefix it with the language's internal name followed by a period. For example:

```
en.ButtonNext=&Forward >
```

Special-purpose IDs

The special-purpose `BeveledLabel` message can be used to specify a line of text that is shown in the lower left corner of the wizard window and uninstaller window. The following is an example:

```
[Messages]
BeveledLabel=Inno Setup
```

[LangOptions] section

A [LangOptions] section is used to define the language-specific settings, such as fonts, used by the Setup program and uninstaller. Normally, you need not create a [LangOptions] section in your script file, since the language-specific settings are, by default, pulled in from the file *Default.isl* included with Inno Setup (or whichever file is specified by a [Languages] section entry).

The following is an example of a `[LangOptions]` section. (The settings listed below are the defaults.)

```
[LangOptions]
LanguageName=English
LanguageID=$0409
DialogFontName=MS Shell Dlg
DialogFontSize=8
DialogFontStandardHeight=13
TitleFontName=Arial
TitleFontSize=29
WelcomeFontName=Verdana
WelcomeFontSize=12
CopyrightFontName=Arial
CopyrightFontSize=8
```

`LanguageName` is the name of the language. It is displayed in the list of available languages on the *Select Language* dialog in a multilingual installation.

`LanguageID` is the numeric "language identifier" of the language. See http://msdn.microsoft.com/library/en-us/intl/nls_238z.asp for a list of valid language identifiers. This is used for the purpose of auto-detecting the most appropriate language to use by default, so be sure it is set correctly. It should always begin with a "$" sign, since language identifiers are in hexadecimal.

`DialogFontName` and `DialogFontSize` specify the font name and point size to use in dialogs. If the specified font does not exist on the user's system, 8-point *MS Shell Dlg* or *MS Sans Serif* will be substituted (whichever one is found first).

`DialogFontStandardHeight` affects how the dialogs are scaled. If you are using a font with a height other than 13 pixels (at 96 DPI), you may need to adjust `DialogFontStandardHeight`, otherwise the dialogs may appear inflated or shrunk.

`TitleFontName` and `TitleFontSize` specify the font name and point size to use when displaying the application name on the background window (only visible when `WindowVisible=yes`). If the specified font does not exist on the user's system, 29-point *Arial* will be substituted.

`WelcomeFontName` and `WelcomeFontSize` specify the font name and point size to use at the top of Welcome page. If the specified font does not exist on the user's system, 12-point *MS Shell Dlg* or *MS Sans Serif* will be substituted (whichever one is found first).

`CopyrightFontName` and `CopyrightFontSize` specify the font name and point size to use when displaying the `AppCopyright` message on the background window (only visible when `WindowVisible=yes`). If the specified font does not exist on the user's system, 8-point *Arial* will be substituted.

In cases where there are multiple [Languages] section entries, specifying a [LangOptions] section directive in your script (as opposed to an .isl file) will by default override that directive for all languages. To apply a [LangOptions] section directive to only one language, prefix it with the language's internal name followed by a period. For example:

```
en.LanguageName=English
```

[Registry] section

This optional section defines any registry keys/values you would like Setup to create, modify, or delete on the user's system.

By default, registry keys and values created by Setup are not deleted at uninstall time. If you want the uninstaller to delete keys or values, you must include one of the `uninsdelete*` flags described below.

The following is an example of a `[Registry]` section.

```
[Registry]
Root: HKCU; Subkey: "Software\My Company"; Flags: uninsdeletekeyifempty
Root: HKCU; Subkey: "Software\My Company\My Program"; Flags: uninsdeletekey

Root: HKLM; Subkey: "Software\My Company"; Flags: uninsdeletekeyifempty
Root: HKLM; Subkey: "Software\My Company\My Program"; Flags: uninsdeletekey

Root: HKLM; Subkey: "Software\My Company\My Program"; ValueType: string;
ValueName: "InstallPath"; ValueData: "{app}"
```

The following is a list of the supported parameters:

Root *(Required)*

Description:
The root key. This must be one of the following:

`HKCR`	(HKEY_CLASSES_ROOT)
`HKCU`	(HKEY_CURRENT_USER)
`HKLM`	(HKEY_LOCAL_MACHINE)
`HKU`	(HKEY_USERS)
`HKCC`	(HKEY_CURRENT_CONFIG)

Example:
`Root: HKCU`

Subkey *(Required)*

Description:
The subkey name, which can include constants.

Example:
`Subkey: "Software\My Company\My Program"`

ValueType

Description:
The data type of the value. This must be one of the following:

```
none
string
expandsz
multisz
dword
binary
```

If `none` (the default setting) is specified, Setup will create the key but *not* a value. In this case the `ValueName` and `ValueData` parameters are ignored.
If `string` is specified, Setup will create a string (REG_SZ) value.
If `expandsz` is specified, Setup will create an expand-string (REG_EXPAND_SZ) value. This data type is primarily used on Windows NT, but is supported by Windows 95/98.

If `multisz` is specified, Setup will create an multi-string (REG_MULTI_SZ) value.
If `dword` is specified, Setup will create an integer (REG_DWORD) value.
If `binary` is specified, Setup will create a binary (REG_BINARY) value.

Example:
```
ValueType: string
```

ValueName

Description:
The name of the value to create, which can include constants. If this is blank, it will write to the "Default" value. If the `ValueType` parameter is set to `none`, this parameter is ignored.

Example:
```
ValueName: "Version"
```

ValueData

Description:
The data for the value. If the `ValueType` parameter is `string`, `expandsz`, or `multisz`, this is a string that can include constants. If the data type is `dword`, this can be a decimal integer (e.g. "123") or a hexadecimal integer (e.g. "$7B"). If the data type is `binary`, this is a sequence of hexadecimal bytes in the form: "00 ff 12 34". If the data type is `none`, this is ignored.

On a `string`, `expandsz`, or `multisz` type value, you may use a special constant called `{olddata}` in this parameter. `{olddata}` is replaced with the previous data of the registry value. The `{olddata}` constant can be useful if you need to append a string to an existing value, for example, `{olddata};{app}`. If the value does not exist or the existing value isn't a string type, the {olddata} constant is silently removed. {olddata} will also be silently removed if the value being created is a `multisz` type but the existing value is not a multi-string type (i.e. it's REG_SZ or REG_EXPAND_SZ), and vice versa.

On a `multisz` type value, you may use a special constant called `{break}` in this parameter to embed line breaks (nulls).

Example:
```
ValueData: "1.0"
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

createvalueifdoesntexist
When this flag is specified, Setup will create the value *only* if a value of the same name doesn't already exist. This flag has no effect if the data type is `none`, or if you specify the *deletevalue* flag.

deletekey
When this flag is specified, Setup will first try deleting the entire key if it exists, including all values and subkeys in it. If `ValueType` is not `none`, it will then create a new key and value.

To prevent disasters, this flag is ignored during installation if `Subkey` is blank or contains only backslashes.

deletevalue
When this flag is specified, Setup will first try deleting the value if it exists. If `ValueType` is not `none`, it will then create the key if it didn't already exist, and the new value.

dontcreatekey
When this flag is specified, Setup will not attempt to create the key or any value if the key did not

already exist on the user's system. No error message is displayed if the key does not exist.

Typically this flag is used in combination with the `uninsdeletekey` flag, for deleting keys during uninstallation but not creating them during installation.

noerror
Don't display an error message if Setup fails to create the key or value for any reason.

preservestringtype
This is only applicable when the `ValueType` parameter is `string` or `expandsz`. When this flag is specified and the value did not already exist or the existing value isn't a string type (REG_SZ or REG_EXPAND_SZ), it will be created with the type specified by `ValueType`. If the value did exist and is a string type, it will be replaced with the same value type as the pre-existing value.

uninsclearvalue
When the program is uninstalled, set the value's data to a null string (type REG_SZ). This flag cannot be combined with the `uninsdeletekey` flag.

uninsdeletekey
When the program is uninstalled, delete the entire key, including all values and subkeys in it. It obviously wouldn't be a good idea to use this on a key that is used by Windows itself. You should only use this on keys private to your application.

To prevent disasters, this flag is ignored during installation if `Subkey` is blank or contains only backslashes.

uninsdeletekeyifempty
When the program is uninstalled, delete the key if it has no values or subkeys left in it. This flag can be combined with `uninsdeletevalue`.

To prevent disasters, this flag is ignored during installation if `Subkey` is blank or contains only backslashes.

uninsdeletevalue
Delete the value when the program is uninstalled. This flag can be combined with `uninsdeletekeyifempty`.

NOTE: In Inno Setup versions prior to 1.1, you could use this flag along with the data type `none` and it would function as a "delete key if empty" flag. This technique is no longer supported. You must now use the `uninsdeletekeyifempty` flag to accomplish this.

Example:
```
Flags: uninsdeletevalue
```

Components and Tasks Parameters

Common Parameters

[Run] & [UninstallRun] sections

The [Run] section is optional, and specifies any number of programs to execute after the program has been successfully installed, but before the Setup program displays the final dialog. The [UninstallRun] section is optional as well, and specifies any number of programs to execute as the first step of *uninstallation.* Both sections share an identical syntax, except where otherwise noted below.

Programs are executed in the order they appear in the script. While processing a [Run]/[UninstallRun] entry, Setup/Uninstall will wait until the program has terminated before proceeding to the next one, unless the `nowait`, `shellexec`, or `waituntilidle` flags are used.

Note that by default, if a program executed in the [Run] section queues files to be replaced on the next reboot (by calling MoveFileEx or by modifying wininit.ini), Setup will detect this and prompt the user to restart the computer at the end of installation. If you don't want this, set the RestartIfNeededByRun directive to `no`.

The following is an example of a `[Run]` section.

```
[Run]
Filename: "{app}\INIT.EXE"; Parameters: "/x"
Filename: "{app}\README.TXT"; Description: "View the README file"; Flags:
postinstall shellexec skipifsilent
Filename: "{app}\MYPROG.EXE"; Description: "Launch application"; Flags:
postinstall nowait skipifsilent unchecked
```

The following is a list of the supported parameters:

Filename *(Required)*

Description:
The program to execute, or file/folder to open. If `Filename` is not an executable (.exe or .com) or batch file (.bat or .cmd), you *must* use the `shellexec` flag on the entry. This parameter can include constants.

Example:
`Filename: "{app}\INIT.EXE"`

Description

Description:
Valid only in a `[Run]` section. The description of the entry, which can include constants. This description is used for entries with the `postinstall` flag. If the description is not specified for an entry, Setup will use a default description. This description depends on the type of the entry (normal or `shellexec`).

Example:
`Description: "View the README file"`

Parameters

Description:
Optional command line parameters for the program, which can include constants.

Example:
`Parameters: "/x"`

WorkingDir

Description:
The directory in which the program is started from. If this parameter is not specified or is blank, it uses the directory from the `Filename` parameter; if `Filename` does not include a path, it will use a default directory. This parameter can include constants.

Example:
```
WorkingDir: "{app}"
```

StatusMsg

Description:
Valid only in a `[Run]` section. Determines the message displayed on the wizard while the program is executed. If this parameter is not specified or is blank, a default message of "Finishing installation..." will be used. This parameter can include constants.

Example:
```
StatusMsg: "Installing BDE..."
```

RunOnceId

Description:
Valid only in an `[UninstallRun]` section. If the same application is installed more than once, "run" entries will be duplicated in the uninstall log file. By assigning a string to `RunOnceId`, you can ensure that a particular `[UninstallRun]` entry will only be executed once during uninstallation. For example, if two or more "run" entries in the uninstall log have a `RunOnceId` setting of "DelService", only the latest entry with a `RunOnceId` setting of "DelService" will be executed; the rest will be ignored. Note that `RunOnceId` comparisons are case-sensitive.

Example:
```
RunOnceId: "DelService"
```

Flags

Description:
This parameter is a set of extra options. Multiple options may be used by separating them by spaces. The following options are supported:

hidewizard
If this flag is specified, the wizard will be hidden while the program is running.

nowait
If this flag is specified, it will not wait for the process to finish executing before proceeding to the next [Run] entry, or completing Setup. Cannot be combined with `waituntilidle`.

postinstall
Valid only in an [Run] section. Instructs Setup to create a checkbox on the *Setup Completed* wizard page. The user can uncheck or check this checkbox and thereby choose whether this entry should be processed or not. Previously this flag was called `showcheckbox`.

If Setup has to restart the user's computer (as a result of installing a file with the flag `restartreplace` or if the `AlwaysRestart [Setup]` section directive is `yes`), there will not be an opportunity for the checkbox to be displayed and therefore the entry will never be processed.

The `isreadme` flag for entries in the [Files] section is now obsolete. If the compiler detects a entry with an `isreadme` flag, it strips the `isreadme` flag from the [Files] entry and inserts a generated [Run] entry at the head of the list of [Run] entries. This generated [Run] entry runs the README file and has flags `shellexec`, `skipifdoesntexist`, `postinstall` and `skipifsilent`.

runhidden
If this flag is specified, it will launch the program in a hidden window. Never use this flag when executing a program that may prompt for user input.

runmaximized
If this flag is specified, it will launch the program or document in a maximized window.

runminimized
If this flag is specified, it will launch the program or document in a minimized window.

shellexec
This flag is required if `Filename` is not a directly executable file (an .exe or .com file). When this flag is set, `Filename` can be a folder or any registered file type -- including .hlp, .doc, and so on. The file will be opened with the application associated with the file type on the user's system, the same way it would be if the user double-clicked the file in Explorer.

When using a folder name in `Filename` it's recommended that you follow it by a backslash character (e.g. "{group}\"), to ensure that a program of the same name is not executed.

There is one drawback to using the `shellexec` flag: it cannot wait until the process terminates. Therefore, it always works as if the `nowait` flag was specified.

skipifdoesntexist
If this flag is specified in the [Run] section, Setup won't display an error message if `Filename` doesn't exist.

If this flag is specified in the [UninstallRun] section, the uninstaller won't display the "some elements could not be removed" warning if `Filename` doesn't exist.

skipifnotsilent
Valid only in an [Run] section. Instructs Setup to skip this entry if Setup is not running (very) silent.

skipifsilent
Valid only in an [Run] section. Instructs Setup to skip this entry if Setup is running (very) silent.

unchecked
Valid only in an [Run] section. Instructs Setup to initially uncheck the checkbox. The user can still check the checkbox if he/she wishes to process the entry. This flag is ignored if the `postinstall` flag isn't also specified.

waituntilidle
If this flag is specified, it will pause until the process is waiting for user input with no input pending, instead of waiting for the process to terminate. (This calls the *WaitForInputIdle* Win32 function.) Cannot be combined with `nowait`.

Example:
```
Flags: postinstall nowait skipifsilent
```

Components and Tasks Parameters

Common Parameters

[UninstallDelete] section

This optional section defines any additional files or directories you want the uninstaller to delete, besides those that were installed/created using [Files] or [Dirs] section entries. Deleting .INI files created by your application is one common use for this section. The uninstaller processes these entries as the last step of uninstallation.

Here is a example of a `[UninstallDelete]` section:

```
[UninstallDelete]
Type: files; Name: "{win}\MYPROG.INI"
```

The following is a list of the supported parameters:

Type *(Required)*

Description:
Specifies what is to be deleted by the uninstaller. This must be one of the following:

files
The `Name` parameter specifies a name of a particular file, or a filename with wildcards.

filesandordirs
Functions the same as `files` except it matches directory names also, and any directories matching the name are deleted including all files and subdirectories in them.

dirifempty
When this is used, the `Name` parameter must be the name of a directory, but it cannot include wildcards. The directory will only be deleted if it contains no files or subdirectories.

Example:
```
Type: files
```

Name *(Required)*

Description:
Name of the file or directory to delete.

NOTE: Don't be tempted to use a wildcard here to delete all files in the {app} directory. I strongly recommend against doing this for two reasons. First, users usually don't appreciate having their data files they put in the application directory deleted without warning (they might only be uninstalling it because they want to move it to a different drive, for example). It's better to leave it up to the end users to manually remove them if they want. Also, if the user happened to install the program in the wrong directory by mistake (for example, C:\WINDOWS) and then went to uninstall it there could be disastrous consequences. So again, **DON'T DO THIS!**

Example:
```
Name: "{win}\MYPROG.INI"
```

Components and Tasks Parameters

Common Parameters

Frequently Asked Questions

The Frequently Asked Questions is now located in a separate document. Please click the "Inno Setup FAQ" shortcut created in the Start Menu when you installed Inno Setup, or open the "isfaq.htm" file in your Inno Setup directory.

For the most recent Frequently Asked Questions, go to http://www.jrsoftware.org/isfaq.php

Wizard Pages

Below is a list of all the wizard pages Setup may potentially display, and the conditions under which they are displayed.

- **Welcome**
  Always shown.
- **License Agreement**
  Shown if LicenseFile is set. Users may proceed to the next page only if the option "I accept the agreement" is selected.
- **Password**
  Shown if Password is set. Users may proceed to the next page only after entering the correct password.
- **Information**
  Shown if InfoBeforeFile is set.
- **User Information**
  Shown if UserInfoPage is set to `yes`.
- **Select Destination Directory**
  Shown by default, but can be disabled via DisableDirPage.
- **Select Components**
  Shown if there are any [Components] entries.
- **Select Start Menu Folder**
  Shown if there are any [Icons] entries, but can be disabled via DisableProgramGroupPage.
- **Select Tasks**
  Shown if there are any [Tasks] entries, unless the [Tasks] entries are all tied to components that were not selected on the *Select Components* page.
- **Ready to Install**
  Shown by default, but can be disabled via DisableReadyPage.
- **Preparing to Install**
  Normally, Setup will never stop on this page. The only time it will is if Setup determines it can't continue. Currently, the only time this can happen is if one or more files specified in the [Files] section were queued (by some other installation) to be replaced or deleted on the next restart. In this case, it tells the user they need to restart their computer and then run Setup again. Note that this check is performed on silent installations too, but any messages are displayed in a message box instead of inside a wizard page.
- **Installing**
  Shown during the actual installation process.
- **Information**
  Shown if InfoAfterFile is set.
- **Setup Completed**
  Shown by default, but can be disabled in some cases via DisableFinishedPage.

Installation Order

Once the actual installation process begins, this is the order in which the various installation tasks are performed:

- [InstallDelete] is processed.
- The entries in [UninstallDelete] are stored in the uninstall log (which, at this stage, is stored in memory).
- The application directory is created, if necessary.
- [Dirs] is processed.
- A filename for the uninstall log is reserved, if necessary.
- [Files] is processed. (File registration does not happen yet.)
- [Icons] is processed.
- [INI] is processed.
- [Registry] is processed.
- Files that needed to be registered are now registered, unless the system needs to be restarted, in which case no files are registered until the system is restarted.
- The *Add/Remove Programs* entry for the program is created, if necessary.
- The entries in [UninstallRun] are stored in the uninstall log.
- The uninstaller EXE and log are finalized and saved to disk.
- If ChangesAssociations was set to `yes`, file associations are refreshed now.
- [Run] is processed, except for entries with the `postinstall` flag, which get processed after the *Setup Completed* wizard page is shown.

All entries are processed by the installer in the order they appear in a section.

Changes are undone by the uninstaller in the *opposite* order in which the installer made them. This is because the uninstall log is parsed from end to beginning.

In this example:

```
[INI]
Filename: "{win}\MYPROG.INI"; Section: "InstallSettings"; Flags:
uninsdeletesectionifempty
Filename: "{win}\MYPROG.INI"; Section: "InstallSettings"; Key: "InstallPath";
String: "{app}"; Flags: uninsdeleteentry
```

the installer will first record the data for first entry's `uninsdeletesectionifempty` flag in the uninstall log, create the key of the second entry, and then record the data for the `uninsdeleteentry` flag in the uninstall log. When the program is uninstalled, the uninstaller will first process the `uninsdeleteentry` flag, deleting the entry, and then the `uninsdeletesectionifempty` flag.

Miscellaneous Notes

- If Setup detects a shared version of Windows on the user's system where the Windows System directory is write protected, the {sys} directory constant will translate to the user's Windows directory instead of the System directory.
- To easily auto update your application, first make your application somehow detect a new version of your Setup.exe and make it locate or download this new version. Then, to auto update, start your Setup.exe from your application with for example the following command line:

  ```
  /SP- /silent /noicons "/dir=c:\Program Files\My Program"
  ```

  After starting setup.exe, exit your application as soon as possible. Note that to avoid problems with updating your .exe, Setup has an auto retry feature when it is silent or very silent.

  Optionally you could also use the `skipifsilent` and `skipifnotsilent` flags and make your application aware of a '/updated' parameter to for example show a nice message box to inform the user that the update has completed.

Command Line Compiler Execution

- Scripts can also be compiled by the Setup Compiler from the command line. Command line usage is as follows:

  compiler /cc <*script name*>

  Example:

  compil32 /cc "c:\isetup\samples\my script.iss"

  As shown in the example above, filenames that include spaces must be enclosed in quotes.

  Running the Setup Compiler from the command line does not suppress the normal progress display or any error messages. The Setup Compiler will return an exit code of 0 if the compile was successful, 1 if the command line parameters were invalid, or 2 if the compile failed.

- Alternatively, you can compile scripts using the console-mode compiler, ISCC.exe.

  Example:

  iscc "c:\isetup\samples\my script.iss"

  As shown in the example above, filenames that include spaces must be enclosed in quotes.

  ISCC will return an exit code of 0 if the compile was successful, 1 if the command line parameters were invalid or an internal error occurred, or 2 if the compile failed.

- The Setup Script Wizard can be started from the command line. Command line usage is as follows:

  compiler /wizard <*wizard name*> <*script name*>

  Example:

  compil32 /wizard "MyProg Script Wizard" "c:\temp.iss"

  As shown in the example above, wizard names and filenames that include spaces must be enclosed in quotes.

  Running the wizard from the command line does not suppress any error messages. The Setup Script Wizard will return an exit code of 0 if there was no error and additionally it will save the generated script file to the specified filename, 1 if the command line parameters were invalid, or 2 if the generated script file could not be saved. If the user cancelled the Setup Script Wizard, an exit code of 0 is returned and no script file is saved.

Setup Command Line Parameters

The Setup program accepts optional command line parameters. These can be useful to system administrators, and to other programs calling the Setup program.

/SP-
Disables the *This will install... Do you wish to continue?* prompt at the beginning of Setup. Of course, this will have no effect if the `DisableStartupPrompt [Setup]` section directive was set to `yes`.

/SILENT, /VERYSILENT
Instructs Setup to be silent or very silent. When Setup is silent the wizard and the background window are not displayed but the installation progress window is. When a setup is very silent this installation progress window is not displayed. Everything else is normal so for example error messages during installation are displayed and the startup prompt is (if you haven't disabled it with DisableStartupPrompt or the '/SP-' command line option explained above)

If a restart is necessary and the '/NORESTART' command isn't used (see below) and Setup is silent, it will display a *Reboot now?* message box. If it's very silent it will reboot without asking.

/NOCANCEL
Prevents the user from cancelling during the installation process, by disabling the Cancel button and ignoring clicks on the close button. Useful along with /SILENT.

/NORESTART
When combined with /SILENT or /VERYSILENT, instructs Setup not to reboot even if it's necessary.

/LOADINF="*filename*"
Instructs Setup to load the settings from the specified file after having checked the command line. This file can be prepared using the '/SAVEINF=' command as explained below.

Don't forget to use quotes if the filename contains spaces.

/SAVEINF="*filename*"
Instructs Setup to save installation settings to the specified file.

Don't forget to use quotes if the filename contains spaces.

/LANG=*language*
Specifies the language to use. *language* specifies the internal name of the language as specified in a [Languages] section entry.

When a valid /LANG parameter is used, the *Select Language* dialog will be suppressed.

/DIR="*x*:*dirname*"
Overrides the default directory name displayed on the *Select Destination Directory* wizard page. A fully qualified pathname must be specified. If the `[Setup]` section directive `DisableDirPage` was set to `yes`, this command line parameter is ignored.

/GROUP="*folder name*"
Overrides the default folder name displayed on the *Select Start Menu Folder* wizard page. If the `[Setup]` section directive `DisableProgramGroupPage` was set to `yes`, this command line parameter is ignored.

/NOICONS
Instructs Setup to initially check the *Don't create any icons* check box on the *Select Start Menu Folder* wizard page.

/COMPONENTS="*comma separated list of component names*"
Overrides the default components settings. Using this command line parameter causes Setup to automatically select a custom type.

Setup Exit Codes

Beginning with Inno Setup 3.0.3, the Setup program may return one of the following exit codes:

0 Setup was successfully run to completion.

1 Setup failed to initialize.

2 The user clicked Cancel in the wizard before the actual installation started, or chose "No" on the opening "This will install..." message box.

3 A fatal error occurred while preparing to move to the next installation phase (for example, from displaying the pre-installation wizard pages to the actual installation process). This should never happen except under the most unusual of circumstances, such as running out of memory or Windows resources.

4 A fatal error occurred during the actual installation process.

Note: Errors that cause an Abort-Retry-Ignore box to be displayed are not fatal errors. If the user chooses *Abort* at such a message box, exit code 5 will be returned.

5 The user clicked Cancel during the actual installation process, or chose *Abort* at an Abort-Retry-Ignore box.

Before returning an exit code of 1, 3, or 4, an error message explaining the problem will normally be displayed.

Future versions of Inno Setup may return additional exit codes, so applications checking the exit code should be programmed to handle unexpected exit codes gracefully. Any non-zero exit code indicates that Setup was not run to completion.

Uninstaller Command Line Parameters

The uninstaller program (unins???.exe) accepts optional command line parameters. These can be useful to system administrators, and to other programs calling the uninstaller program.

/SILENT

When specified, the uninstaller will not ask the user any questions or display a message stating that uninstall is complete. Shared files that are no longer in use are deleted automatically without prompting. Any critical error messages will still be shown on the screen.

/NORESTART

When combined with /SILENT, the prompt to restart the computer will also be suppressed.

Uninstaller Exit Codes

Beginning with Inno Setup 4.0.8, the uninstaller will return a non-zero exit code if the user cancels or a fatal error is encountered. Programs checking the exit code to detect failure should not check for a specific non-zero value; any non-zero exit code indicates that the uninstaller was not run to completion.

Note that at the moment you get an exit code back from the uninstaller, some code related to uninstallation might still be running. Because Windows doesn't allow programs to delete their own EXEs, the uninstaller creates and spawns a copy of itself in the TEMP directory. This "clone" performs the actual uninstallation, and at the end, terminates the original uninstaller EXE (at which point you get an exit code back), deletes it, then displays the "uninstall complete" message box (if it hasn't been suppressed with /SILENT).

Unsafe Files

As a convenience to new users who are unfamiliar with which files they should and should not distribute, the Inno Setup compiler will display an error message if one attempts to install certain "unsafe" files using the [Files] section. These files are listed below.

(Note: It is possible to disable the error message by using a certain flag on the [Files] section entry, but this is NOT recommended.)

Any DLL file from own Windows System directory

You should not deploy any DLLs out of your own Windows System directory because most of them are tailored for your own specific version of Windows, and will not work when installed on other versions. Often times a user's system will be **rendered unbootable** if you install a DLL from a different version of Windows. Another reason why it's a bad idea is that when you install programs on your computer, the DLLs may be replaced with different/incompatible versions, and were you not to notice this and take action, it could also lead to problems on users' systems when you build new installations.

Instead of deploying the DLLs from your Windows System directory, you should find versions that are specifically deemed "redistributable". Redistributable DLLs typically work on more than one version of Windows. To find redistributable versions of the Visual Basic and Visual C++ run-time DLLs, see the Inno Setup FAQ.

If you have a DLL residing in the Windows System directory that you are **absolutely sure** is redistributable, copy it to your script's source directory and deploy it from there instead.

ADVAPI32.DLL, COMDLG32.DLL, GDI32.DLL, KERNEL32.DLL, SHELL32.DLL, USER32.DLL

These are all core components of Windows and must never be deployed with an installation. Users may only get new versions of these DLLs by installing a new version of Windows or a service pack or hotfix for Windows.

COMCAT.DLL version 5.0

Version 5.0 of COMCAT.DLL must not be redistributed because it does not work on Windows 95 or NT 4.0. If you need to install COMCAT.DLL, use version 4.71 instead.
Reference: http://support.microsoft.com/support/kb/articles/Q201/3/64.ASP

COMCTL32.DLL

Microsoft does not allow separate redistribution of COMCTL32.DLL (and for good reason - the file differs between platforms), so you should never place COMCTL32.DLL in a script's [Files] section. You can however direct your users to download the COMCTL32 update from Microsoft, or distribute the COMCTL32 update along with your program.
Reference: http://www.microsoft.com/permission/copyrgt/cop-soft.htm#COM
Reference: http://www.microsoft.com/downloads/details.aspx?FamilyID=cb2cf3a2-8025-4e8f-8511-9b476a8d35d2&DisplayLang=en

CTL3D32.DLL, Windows NT-specific version

Previously, on the "Installing Visual Basic 5.0 & 6.0 Applications" How-To page there was a version of CTL3D32.DLL included in the zip files. At the time I included it, I was not aware that it only was compatible with Windows NT. Now if you try to install that particular version of CTL3D32.DLL you must use a `MinVersion` setting that limits it to Windows NT platforms only. (You shouldn't need to install CTL3D32.DLL on Windows 9x anyway, since all versions have a 3D look already.)

SHLWAPI.DLL

This is a component of Internet Explorer which is also used by Windows Explorer. Replacing it may prevent Explorer from starting. If your application depends on this DLL, or a recent version of it, then your users will need to install a recent version of Internet Explorer to get it.

Credits

The following is a list of those who have contributed significant code to the Inno Setup project, or otherwise deserve special recognition:

Jean-loup Gailly & Mark Adler: Creators of the zlib compression library that Inno Setup uses.

Julian Seward: Creator of the bzlib compression library that Inno Setup uses.

?: Most of the disk spanning code (1.09). (Sorry, I somehow managed to lose your name!)

Vince Valenti: Most of the code for the "Window" `[Setup]` section directives (1.12.4).

Joe White: Code for ChangesAssociations `[Setup]` section directive (1.2.?).

Jason Olsen: Most of the code for appending to existing uninstall logs (1.3.0).

Martijn Laan: Code for Rich Edit 2.0 & URL detection support (1.3.13); silent uninstallation (1.3.25); system image list support in drive and directory lists (1.3.25); silent installation (2.0.0); `[Types]`, `[Components]` and `[Tasks]` sections (2.0.0); `postinstall` flag (2.0.0); `[Code]` section (4.0.0); Subcomponents and subtasks support (4.0.0).

Alex Yackimoff: Portions of TNewCheckListBox (4.0.0).

Carlo Kok: Innerfuse Pascal Script (4.0.0).

Creators of SynEdit: The syntax-highlighting editor used in the Compiler (2.0.0).

If I have left anyone out, please don't hesitate to let me know.

Contacting Me

The latest versions of Inno Setup and other software I've written can be found on my web site at: http://www.jrsoftware.org/

For information on contacting me and obtaining technical support for Inno Setup, go to this page: http://www.jrsoftware.org/contact.php

[Setup]: Bits

Valid values: 32

Description:
Obsolete in 1.3. Pre-1.3 versions of Inno Setup had a 16-bit version, and the `Bits` directive was checked by the Compiler to determine if the correct Compiler was being used to compile the script. Since the newer versions of Inno Setup are available in a 32-bit version only, you are no longer required to set this directive. If, however, `Bits` is set to "16", the Compiler will fail with an error message.

[Setup]: UseSetupLdr

Valid values: `yes` or `no`

Default value: `yes`

Description:
This tells the Setup Compiler which type of Setup to create. If this is `yes`, it compiles all setup data into a single EXE. If this is `no`, it compiles the setup data into at least three files: SETUP.EXE, SETUP.0, and SETUP-1.BIN. The only reason you would probably want to use `no` is for debugging purposes.

Note: Do not use `UseSetupLdr=no` on an installation which uses disk spanning (`DiskSpanning=yes`). When `UseSetupLdr` is `yes`, the setup program is copied to and run from the user's TEMP directory. This does not happen when `UseSetupLdr` is `no`, and could result in errors if Windows tries to locate the setup.exe file on the disk and can't find it because a different disk is in the drive.

[Setup]: BackColor, BackColor2

Valid values: A value in the form of `$bbggrr`, where `rr`, `gg`, and `bb` specify the two-digit intensities (in hexadecimal) for red, green, and blue respectively. Or it may be one of the following predefined color names: clBlack, clMaroon, clGreen, clOlive, clNavy, clPurple, clTeal, clGray, clSilver, clRed, clLime, clYellow, clBlue, clFuchsia, clAqua, clWhite.

Default value: `clBlue` for `BackColor`,
`clBlack` for `BackColor2`

Description:
The `BackColor` directive specifies the color to use at the top (or left, if `BackColorDirection=lefttoright`) of the setup window's gradient background. `BackColor2` specifies the color to use at the bottom (or right).

The setting of `BackColor2` is ignored if `BackSolid=yes`.

Examples:

```
BackColor=clBlue
BackColor2=clBlack

BackColor=$FF0000
BackColor2=$000000
```

[Setup]: BackColorDirection

Valid values: `toptobottom` or `lefttoright`

Default value: `toptobottom`

Description:
This determines the direction of the gradient background on the setup window. If `BackColorDirection` is `toptobottom`, it is drawn from top to bottom; if it is `lefttoright`, it is drawn from left to right.

[Setup]: BackSolid

Valid values: `yes` or `no`

Default value: `no`

Description:
This specifies whether to use a solid or gradient background on the setup window. If this is `yes`, the background is a solid color (the color specified by `BackColor`; `BackColor2` is ignored).

[Setup]: AppName

Description:
This required directive specifies the title of the application you are installing. Do not include the version number, as the `AppVerName` directive is for that purpose. `AppName` is shown throughout the installation process, in places like the upper-left corner of the Setup screen, and in the wizard.

Example: `AppName=My Program`

[Setup]: AppVerName

Description:
The value of this required directive should be the same (or similar to) the value of AppName, but it should also include the program's version number.

Example: `AppVerName=My Program version 3.0`

[Setup]: AppId

Default value: If `AppId` is not specified or is blank, the Compiler uses the value of the `AppName` directive for `AppId`.

Description:
The value of `AppId` is stored inside uninstall log files (unins???.dat), and is checked by subsequent installations to determine whether it may append to a particular existing uninstall log. Setup will only append to an uninstall log if the `AppId` of the existing uninstall log is the same as the current installation's `AppId`. For a practical example, say you have two installations -- one entitled *My Program* and the other entitled *My Program 1.1 Update.* To get My Program 1.1 Update to append to My Program's uninstall log, you would have to set `AppId` to the same value in both installations.

`AppId` also determines the actual name of the Uninstall registry key, to which Inno Setup tacks on "_is1" at the end. (Therefore, if `AppId` is "MyProgram", the key will be named "MyProgram_is1".) Pre-1.3 versions of Inno Setup based the key name on the value of `AppVerName`.

AppId is a not used for display anywhere, so feel free to make it as cryptic as you desire.

Example: `AppId=MyProgram`

[Setup]: AppMutex

Description:
This directive is used to prevent the user from installing new versions of an application while the application is still running, and to prevent the user from uninstalling a running application. It specifies the names of one or more named mutexes (multiple mutexes are separated by commas), which Setup and Uninstall will check for at startup. If any exist, Setup/Uninstall will display the message: "[Setup or Uninstall] has detected that [AppName] is currently running. Please close all instances of it now, then click OK to continue, or Cancel to exit."

Use of this directive requires that you add code to your application which creates a mutex with the name you specify in this directive. Examples of creating a mutex in Delphi, C, and Visual Basic are shown below. The code should be executed during your application's startup.

Delphi:

```
CreateMutex(nil, False, 'MyProgramsMutexName');
```

C:

```
CreateMutex(NULL, FALSE, "MyProgramsMutexName");
```

Visual Basic (submitted by Peter Young):

```
'Place in Declarations section:
Private Declare Function CreateMutex Lib "kernel32" _
        Alias "CreateMutexA" _
       (ByVal lpMutexAttributes As Long, _
        ByVal bInitialOwner As Long, _
        ByVal lpName As String) As Long

'Place in startup code (Form_Load or Sub Main):
CreateMutex 0&, 0&, "MyProgramsMutexName"
```

It is not necessary to explicitly destroy the mutex object upon your application's termination; the system will do this automatically. Nor is it recommended that you do so, because ideally the mutex object should exist until the process completely terminates.

Note that mutex name comparison in Windows is *case sensitive.*

See the topic for CreateMutex in the MS SDK help for more information on mutexes.

Example: `AppMutex=MyProgramsMutexName`

[Setup]: AppCopyright

Description:
This is optional, and is only used to display a copyright message in the bottom-right corner of Setup's background window.

Note that the copyright message will only be seen if WindowVisible is `yes`.

Example: `AppCopyright=Copyright © 1997 My Company, Inc.`

[Setup]: AppPublisher, AppPublisherURL, AppSupportURL, AppUpdatesURL, AppVersion

Description:
These are all used for display purposes on the "Support" dialog of the *Add/Remove Programs* Control Panel applet in Windows 2000/XP. Setting them is optional, and will have no effect on earlier Windows versions.

Example:
```
AppPublisher=My Company, Inc.
AppPublisherURL=http://www.mycompany.com/
AppVersion=1.5
```

[Setup]: DefaultDirName

Description:
This value of this required directive is used for the default directory name, which is used in the *Select Destination Directory* page of the wizard. Normally it is prefixed by a directory constant.

If UsePreviousAppDir is `yes` (the default) and Setup finds a previous version of the same application is already installed, it will substitute the default directory name with the directory selected previously.

Example:

If you used:
```
DefaultDirName={sd}\MYPROG
```
In Setup, this would typically display:
C:\MYPROG

If you used:
```
DefaultDirName={pf}\My Program
```
In Setup, this would typically display:
C:\Program Files\My Program

[Setup]: Uninstallable

Valid values: `yes` or `no`

Default value: `yes`

Description:
This determines if Inno Setup's automatic uninstaller is to be included in the installation. If this is `yes` the uninstaller is included. If this is `no`, no uninstallation support is included, requiring the end-user to manually remove the files pertaining to your application.

[Setup]: MinVersion

Format: *a.bb*,*c.dd*, where *a.bb* is the Windows version, and *c.dd* is the Windows NT version.

Default value: 4,4

Description:
This directive lets you specify a minimum version of Windows or Windows NT that your software runs on. To prevent your program from running on Windows or Windows NT, specify "0" for one the minimum versions. Build numbers and/or service pack levels may be included in the version numbers.

If the user's system does not meet the minimum version requirement, Setup will give an error message and exit.

[Setup]: OnlyBelowVersion

Format: *a.bb*,*c.dd*, where *a.bb* is the Windows version, and *c.dd* is the Windows NT version.

Default: 0,0

Description:
This directive lets you specify a minimum version of Windows or Windows NT that your software *will not* run on. Specifying "0" for one of the versions means there is no upper version limit. Build numbers and/or service pack levels may be included in the version numbers.

This directive is essentially the opposite of MinVersion.

[Setup]: AdminPrivilegesRequired

Valid values: `yes` or `no`

Default value: `no`

Description:
Obsolete in 3.0.4. While `AdminPrivilegesRequired` is still recognized by the compiler, PrivilegesRequired supersedes and overrides it.

When set to `yes`, Setup will give an error message at startup ("You must be logged in as an administrator when installing this program") if the user doesn't have administrative privileges. This only applies to Windows NT platforms.

[Setup]: PrivilegesRequired

Valid values: `none`, `poweruser`, or `admin`

Default value: `none`

Description:
This directive specifies the minimum user privileges required to run the installation. When set to `poweruser` or `admin`, Setup will give an error message at startup (e.g. "You must be logged in as an administrator when installing this program") if the user doesn't have at least Power User or administrative privileges, respectively. This only applies to Windows NT platforms.

[Setup]: DisableAppendDir

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `yes`, Setup won't automatically append the last component of the path from DefaultDirName to directories the user double-clicks on the *Select Destination Directory* wizard page. In addition, it sets the directory list box's initial directory to `DefaultDirName` (if the directory exists) instead of one level up.

[Setup]: EnableDirDoesntExistWarning

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `yes`, Setup will display a message box if the directory the user selects doesn't exist. Usually you will also set `DirExistsWarning=no` when this is `yes`.

[Setup]: AlwaysCreateUninstallIcon

Description:
Obsolete in 3.0. This directive is no longer supported. If you wish to create an Uninstall icon, use the new `{uninstallexe}` constant in the `Filename` parameter of an [Icons] section entry.

[Setup]: ExtraDiskSpaceRequired

Default value: `0`

Description:
Normally, the disk space requirement displayed on the wizard is calculated by adding up the size of all the files in the [Files] section. If you want to increase the disk space display for whatever reason, set `ExtraDiskSpaceRequired` to the amount of bytes you wish to add to this figure. (1048576 bytes = 1 megabyte)

[Setup]: CompressLevel

Valid values: `0` through `9`

Default value: `7`

Description:
Obsolete in 2.0.17. While `CompressLevel` is still recognized by the compiler, Compression supersedes and overrides it.

This is a number from 0 to 9 specifying how much compression to use on the files. 0 is no compression, 9 is maximum compression.

[Setup]: Compression

Valid values: `zip`
`zip/1` through `zip/9`
`bzip`
`bzip/1` through `bzip/9`
`none`

Default value: `zip`

Description:
This specifies the type of compression to use on the files, and optionally the level of compression from 1 to 9, where 9 is maximum compression. Higher numbers are slower.

`zip` is the type of compression used by .zip files ("deflate"). It is relatively fast. If a compression level isn't specified, it defaults to 7.

`bzip` is the type of compression used by the bzip2 compressor. It almost always compresses better than `zip` but can be significantly slower in both compression and decompression. It also requires about 3 MB more memory during decompression. If a compression level isn't specified, it defaults to 9.

`none` specifies that no compression be used.

[Setup]: SolidCompression

Valid values: `yes` or `no`

Default value: `no`

Description:
If `yes`, solid compression will be enabled. This causes all files to be compressed at once instead of separately. This can result a much greater overall compression ratio if your installation contains many files with common content, such as text files. Be sure to also use `Compression=bzip`, since the default `zip` compression doesn't benefit too much from solid compression (as it works with smaller blocks).

The disadvantage to using solid compression is that because all files are compressed into a single compressed stream, Setup can no longer randomly access the files. This can decrease performance. If a certain file isn't going to be extracted on the user's system, it has to decompress the data for that file anyway (into memory) before it can decompress the next file. And if, for example, there was an error while extracting a particular file and the user clicks Retry, it can't just seek to the beginning of that file's compressed data; since all files are stored in one stream, it has seek to the very beginning. If disk spanning was enabled, the user would have to re-insert disk 1.

Thus, it is not recommended that solid compression be enabled on huge installs (say, over 100 MB) or on disk-spanned installs. It is primarily designed to save download time on smaller installs distributed over the Internet.

[Setup]: InternalCompressLevel

Valid values: 0 through 9

Default value: 9

Description:
This is a number from 0 to 9 specifying how much compression to use on internal structures used by Setup. 0 is no compression, 9 is maximum compression. Higher numbers are slower. Generally, there is no reason to change this from the default setting of 9.

[Setup]: CreateAppDir

Valid values: `yes` or `no`

Default value: `yes`

Description:
If this is set to `no`, no directory for the application will be created, the *Select Destination Directory* wizard page will not be displayed, and the {app} directory constant is equivalent to the {win} directory constant. If the uninstall feature is enabled when `CreateAppDir` is `no`, the uninstall data files are created in the system's Windows directory.

[Setup]: CreateUninstallRegKey

Valid values: `yes` or `no`

Default value: `yes`

Description:
If this is set to `no`, Setup won't create an entry in the *Add/Remove Programs* Control Panel applet. This can be useful if your installation is merely an update to an existing application and you don't want another entry created, but don't want to the disable the uninstall features entirely (via `Uninstallable=no`).

When this directive is set to `no`, UpdateUninstallLogAppName is usually set to `no` as well.

[Setup]: OverwriteUninstRegEntries

Description:
Obsolete in 1.3. This directive is no longer supported and is ignored. In Inno Setup 1.3.6 and later, it functions as if the `OverwriteUninstRegEntries` directive of prior versions was set to 1 (which was the default setting).

[Setup]: DirExistsWarning

Valid values: `auto`, `yes`, or `no`

Default value: `auto`

Description:
When set to `auto`, the default setting, Setup will show a "The directory ... already exists. Would you like to install to that directory anyway?" message if the user selects a directory that already exists on the *Select Destination Directory* wizard page, except when another version of the same application is already installed and the selected directory is the same as the previous one (only if `UsePreviousAppDir` is `yes`, the default setting).

When set to `yes`, Setup will always display the "Directory Exists" message when the user selects an existing directory.

When set to `no`, Setup will never display the "Directory Exists" message.

[Setup]: DisableDirExistsWarning

Valid values: `yes` or `no`

Default value: `no`

Description:
Obsolete in 1.3.6. Use DirExistsWarning instead.

`DisableDirExistsWarning` is still recognized by the compiler, however. It translates `DisableDirExistsWarning=no` to `DirExistsWarning=auto`, and `DisableDirExistsWarning=yes` to `DirExistsWarning=no`. If both `DisableDirExistsWarning` and `DirExistsWarning` directives are specified, `DirExistsWarning` takes precedence.

[Setup]: DisableDirPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will not show the *Select Destination Directory* wizard page. In this case, it will always use the default directory name.

[Setup]: DisableFinishedPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will not show the *Setup Completed* wizard page, and instead will immediately close the Setup program once the installation process finishes. This may be useful if you execute a program in the [Run] section using the `nowait` flag, and don't want the *Setup Completed* window to remain in the background after the other program has started.

Note that the `DisableFinishedPage` directive is ignored if a restart of the computer is deemed necessary, or if a file is assigned to the `InfoAfterFile [Setup]` section directive. In those cases, the *Setup Completed* wizard page will still be displayed.

[Setup]: DisableProgramGroupPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will not show the *Select Start Menu Folder* wizard page. In this case, it uses the folder name specified by the DefaultGroupName [Setup] section directive, or "(Default)" if none is specified.

[Setup]: DisableReadyMemo

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will not show a list of settings on the *Ready to Install* wizard page. Otherwise the list is shown and contains information like the chosen setup type and the chosen components.

[Setup]: DisableReadyPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will not show the *Ready to Install* wizard page.

[Setup]: UserInfoPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is set to `yes`, Setup will show a *User Information* wizard page which asks for the user's name, organization and possibly a serial number. The values the user enters are stored in the `{userinfoname}`, `{userinfoorg}` and `{userinfoserial}` constants. You can use these constants in [Registry] or [INI] entries to save their values for later use.

The DefaultUserInfoName, DefaultUserInfoOrg and DefaultUserInfoSerial directives determine the default name, organization and serial number shown. If UsePreviousUserInfo is `yes` (the default) and Setup finds that a previous version of the same application is already installed, it will use the name, organization and serial number entered previously instead.

[Setup]: DefaultUserInfoName

Default value: `{sysuserinfoname}`

Description:
Specifies the default name shown on the *User Information* wizard page. This can include constants.

[Setup]: DefaultUserInfoOrg

Default value: `{sysuserinfoorg}`

Description:
Specifies the default organization shown on the *User Information* wizard page. This can include constants.

[Setup]: DefaultUserInfoSerial

Description:
Specifies the default serial number shown on the *User Information* wizard page. This can include constants.

[Setup]: AlwaysUsePersonalGroup

Valid values: `yes` or `no`

Default value: `no`

Description:
Normally on Windows NT platforms, Inno Setup's `{group}` constant points to the All Users start menu if the user has administrative privileges. If this directive is set to `yes`, it always uses current user's profile.

[Setup]: OutputBaseFilename

Default value: `setup`

Description:
This directive allows you to assign a different name for the resulting Setup file(s), so you don't have to manually rename them after running the Setup Compiler.

Note: If `UseSetupLdr` is set to `no`, the names of the resulting files SETUP.0 and SETUP.MSG will not change since they are hard-coded names.

Example: `OutputBaseFilename=MyProg100`

[Setup]: UninstallFilesDir

Default value: `{app}`

Description:
Specifies the directory where the "unins*.*" files for the uninstaller are stored.

Note: You should not assign a different value here on a new version of an application, or else Setup won't find the uninstall logs from the previous versions and therefore won't be able to append to them.

Example: `UninstallFilesDir={app}\uninst`

[Setup]: UninstallDisplayIcon

Description:
This lets you specify a particular icon file (either an executable or an .ico file) to display for the Uninstall entry in the *Add/Remove Programs* Control Panel applet on Windows 2000/XP. The filename will normally begin with a directory constant.

If the file you specify contains multiple icons, you may append the suffix ",*n*" to specify an icon index, where *n* is the zero-based numeric index.

If this directive is not specified or is blank, Windows will select an icon itself, which may not be the one you prefer.

Examples:
```
UninstallDisplayIcon={app}\MyProg.exe
UninstallDisplayIcon={app}\MyProg.exe,1
```

[Setup]: UninstallDisplayName

Description:
This lets you specify a custom name for the program's entry in the *Add/Remove Programs* Control Panel applet. The value may include constants. If this directive is not specified or is blank, Setup will use the value of `[Setup]` section directive `AppVerName` for the name.

Due to limitations of Windows 9x's *Add/Remove Programs* Control Panel applet, the value of `UninstallDisplayName` will be trimmed if it exceeds 63 characters.

Example:
```
UninstallDisplayName=My Program
```

[Setup]: UninstallIconName

Description:
Obsolete in 3.0. This directive is no longer supported. If you wish to create an Uninstall icon, use the new `{uninstallexe}` constant in the `Filename` parameter of an [Icons] section entry.

[Setup]: UninstallLogMode

Valid values: `append`, `new`, or `overwrite`

Default value: `append`

Description:
`append`, the default setting, instructs Setup to append to an existing uninstall log when possible.

`new`, which corresponds to the behavior in pre-1.3 versions of Inno Setup, instructs Setup to always create a new uninstall log.

`overwrite` instructs Setup to overwrite any existing uninstall logs from the same application instead of appending to them (this is *not* recommended). The same rules for appending to existing logs apply to overwriting existing logs.

Example: `UninstallLogMode=append`

[Setup]: UninstallRestartComputer

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `yes`, the uninstaller will always prompt the user to restart the system at the end of a successful uninstallation, regardless of whether it is necessary (e.g., because of `[Files]` section entries with the `uninsrestartdelete` flag).

[Setup]: UpdateUninstallLogAppName

Valid values: `yes` or `no`

Default value: `yes`

Description:
If `yes`, when appending to an existing uninstall log, Setup will replace the `AppName` field in the log with the current installation's `AppName`. The `AppName` field of the uninstall log determines the title displayed in the uninstaller. You may want to set this to `no` if your installation is merely an upgrade or add-on to an existing program, and you don't want the title of the uninstaller changed.

[Setup]: DefaultGroupName

Description:
The value of this directive is used for the default Start Menu folder name on the *Select Start Menu Folder* page of the wizard. If this directive is blank or isn't specified, it will use "(Default)" for the name.

Keep in mind that Start Menu folders are stored as literal directories so any characters not allowed in normal directory names can't be used in Start Menu folder names.

Example: `DefaultGroupName=My Program`

[Setup]: DisableStartupPrompt

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this is set to `yes`, Setup will not show the *This will install... Do you wish to continue?* prompt.

This setting has no effect if `UseSetupLdr` is set to `no`.

[Setup]: DiskSpanning

Valid values: `yes` or `no`

Default value: `no`

Description:
If set to `yes`, the disk spanning feature will be enabled. Instead of storing all the compressed file data inside SETUP.EXE, the compiler will split it into multiple SETUP-*.BIN files -- known as "slices" -- suitable for copying onto separate floppy disks, CD-ROMs, or DVD-ROMs. Each generated slice contains a number in its name which indicates the disk onto which it should be copied. (For example, SETUP-2.BIN should be placed on disk 2.) The generated SETUP.EXE always goes on disk 1 along with the SETUP-1*.BIN file.

The size of each slice and the number of slices to create for each disk are determined by the values of the DiskSliceSize and SlicesPerDisk `[Setup]` section directives, respectively. Other disk spanning-related directives that you may want to tweak include DiskClusterSize and ReserveBytes.

Note that it is required that you set this directive to `yes` if the compressed size of your installation exceeds 2,100,000,000 bytes, even if you don't intend to place the installation onto multiple disks. (The installation will still function correctly if all the SETUP-*.BIN files are placed on the same disk.)

[Setup]: DiskSliceSize

Valid values: `262144` through `2100000000`

Default value: `1457664` (the size of a 1.44MB floppy)

Description:
This specifies the maximum number of bytes per disk slice (SETUP-*.BIN file). Normally, this should be set to the total number of bytes available on the disk media divided by the value of the `SlicesPerDisk [Setup]` section directive, which defaults to 1.

This directive is ignored if disk spanning is not enabled using the `DiskSpanning [Setup]` section directive.

[Setup]: DiskClusterSize

Default value: `512` (the standard cluster size for floppy disks)

Description:
This specifies the cluster size of the disk media. The Setup Compiler needs to know this in order to properly fill each disk to capacity.

This directive is ignored if disk spanning is not enabled using the `DiskSpanning [Setup]` section directive.

[Setup]: SlicesPerDisk

Valid values: `1` through `26`

Default value: `1`

Description:
The number of SETUP-*.BIN files to create for each disk. If this is 1 (the default setting), the files will be named SETUP-*x*.BIN, where *x* is the disk number. If this is greater than 1, the files will be named SETUP-*xy*.BIN, where *x* is the disk number and *y* is a unique letter.

One reason why you may need to increase this from the default value of 1 is if the size of your disk media exceeds 2,100,000,000 bytes -- the upper limit of the `DiskSliceSize [Setup]` section directive. If, for example, your disk media has a capacity of 3,000,000,000 bytes, you can avoid the 2,100,000,000-byte disk slice size limit by setting `SlicesPerDisk` to `2` and `DiskSliceSize` to `1500000000` (or perhaps slightly less, due to file system overhead).

[Setup]: ReserveBytes

Default value: `0`

Description:
This specifies the minimum number of free bytes to reserve on the first disk. This is useful if you have to copy other files onto the first disk that aren't part of the setup program, such as a Readme file.

The Setup Compiler rounds this number up to the nearest cluster.

This directive is ignored if disk spanning is not enabled using the `DiskSpanning [Setup]` section directive.

[Setup]: DontMergeDuplicateFiles

Valid values: `yes` or `no`

Default value: `no`

Description:
Normally two file entries referring to the same source file will be compressed and stored only once. If you have a bunch of identical files in your installation, make them point to the same source file in the script, and the size of your installation can drop significantly. If you wish to disable this feature for some reason, set this directive to `yes`.

[Setup]: AllowCancelDuringInstall

Valid values: `yes` or `no`

Default value: `yes`

Description:
Setting this to `no` prevents the user from cancelling during the actual installation process, by disabling the Cancel button and ignoring clicks on the close button. This has the same effect as passing /NOCANCEL to Setup on the command line.

[Setup]: AllowNoIcons

Valid values: `yes` or `no`

Default value: `no`

Description:
This is used to determine whether Setup should display a *Don't create any icons* check box, which allows the user to skip creation of program icons. If it is `no` the check box will not be displayed; if it is `yes` it will be displayed.

[Setup]: AllowRootDirectory

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `no`, the default, the user will not be allowed to enter a root directory (such as "C:\") on the *Select Destination Directory* page of the wizard.

[Setup]: AllowUNCPath

Valid values: `yes` or `no`

Default value: `yes`

Description:
If set to `no`, the user will not be allowed to enter a UNC path (such as "\\server\share") on the *Select Destination Directory* page of the wizard. This was the default behavior in Inno Setup 2.0.17 and earlier.

[Setup]: AlwaysRestart

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `yes`, Setup will always prompt the user to restart the system at the end of a successful installation, regardless of whether this is necessary (for example, because of `[Files]` section entries with the `restartreplace` flag).

[Setup]: RestartIfNeededByRun

Valid values: `yes` or `no`

Default value: `yes`

Description:
When set to `yes`, and a program executed in the [Run] section queues files to be replaced on the next reboot (by calling MoveFileEx or by modifying wininit.ini), Setup will detect this and prompt the user to restart the computer at the end of installation.

[Setup]: MessagesFile

Description:
Obsolete in 4.0. This directive is no longer supported. Use the new [Languages] section to specify a custom messages file.

[Setup]: LicenseFile

Description:
Specifies the name of an optional license agreement file, in .txt or .rtf (rich text) format, which is displayed before the user selects the destination directory for the program. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

If the user selects a language for which the `LicenseFile` parameter is set, this directive is effectively ignored. See the [Languages] section documentation for more information.

Example: `LicenseFile=license.txt`

[Setup]: InfoBeforeFile

Description:
Specifies the name of an optional "readme" file, in .txt or .rtf (rich text) format, which is displayed before the user selects the destination directory for the program. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

If the user selects a language for which the `InfoBeforeFile` parameter is set, this directive is effectively ignored. See the [Languages] section documentation for more information.

Example: `InfoBeforeFile=infobefore.txt`

[Setup]: InfoAfterFile

Description:
Specifies the name of an optional "readme" file, in .txt or .rtf (rich text) format, which is displayed after a successful install. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

This differs from `isreadme` files in that this text is displayed as a page of the wizard, instead of in a separate Notepad window.

If the user selects a language for which the `InfoAfterFile` parameter is set, this directive is effectively ignored. See the [Languages] section documentation for more information.

Example: `InfoAfterFile=infoafter.txt`

[Setup]: ChangesAssociations

Valid values: `yes` or `no`

Default value: `no`

Description:
When set to `yes`, Setup will tell Explorer to refresh its file associations information at the end of the installation, and Uninstall will do the same at the end of uninstallation.

If your installation creates a file association but doesn't have `ChangesAssociations` set to `yes`, the correct icon for the file type likely won't be displayed until the user logs off or restarts the computer.

[Setup]: UsePreviousAppDir

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is `yes`, the default, at startup Setup will look in the registry to see if the same application is already installed, and if so, it will use the directory of the previous installation as the default directory presented to the user in the wizard.

Note that only Inno Setup 1.3.1 and later save the directory in the registry, so Setup will not "see" applications installed with older Inno Setup versions.

If `Uninstallable` is `no`, this directive is effectively forced to `no`.

[Setup]: UsePreviousGroup

Valid values: yes or no

Default value: yes

Description:
When this directive is yes, the default, at startup Setup will look in the registry to see if the same application is already installed, and if so, it will use the Start Menu folder name of the previous installation as the default Start Menu folder name presented to the user in the wizard.

Note that only Inno Setup 1.3.10 and later save the Start Menu folder name in the registry, so Setup will not reuse the Start Menu folder name of applications installed with older Inno Setup versions.

If Uninstallable is no, this directive is effectively forced to no.

[Setup]: UsePreviousSetupType

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is `yes`, the default, at startup Setup will look in the registry to see if the same application is already installed, and if so, it will use the setup type and component settings of the previous installation as the default settings presented to the user in the wizard.

If `Uninstallable` is `no`, this directive is effectively forced to `no`.

[Setup]: UsePreviousTasks

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is `yes`, the default, at startup Setup will look in the registry to see if the same application is already installed, and if so, it will use the task settings of the previous installation as the default settings presented to the user in the wizard.

If `Uninstallable` is `no`, this directive is effectively forced to `no`.

[Setup]: UsePreviousUserInfo

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is `yes`, the default, at startup Setup will look in the registry to see if the same application is already installed, and if so, it will use the name, organization and serial number entered previously as the default settings presented to the user on the *User Information* wizard page.

If `Uninstallable` is `no`, this directive is effectively forced to `no`.

[Setup]: Password

Description:
Specifies a password you want to prompt the user for at the beginning of the installation.

When using a password, it's important to keep in mind that since no encryption is used and the source code to Inno Setup is freely available, it would not be too difficult for an experienced individual to remove the password protection from an installation. Use this only as a "deterrent" to keep unauthorized people out of your installations.

The password itself is not stored as clear text; it's stored as a 32-bit hash.

[Setup]: WizardImageFile

Default value: `compiler:WIZMODERNIMAGE.BMP`

Description:
Specifies the name of the bitmap file to display on the left side of the wizard in the Setup program. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

256-color bitmaps may not display correctly in 256-color mode, since it does not handle palettes. The maximum size of the bitmap is 164x314 pixels. Note that if Windows is running with Large Fonts, the area on the wizard for the bitmap will be larger.

Example: `WizardImageFile=myimage.bmp`

[Setup]: WindowShowCaption

Valid values: `yes` or `no`

Default value: `yes`

Description:
If set to `no`, Setup will be truly "full screen" -- it won't have a caption bar or border, and it will be on top of the taskbar.

This directive has no effect if `WindowVisible` is not set to `yes`.

[Setup]: WindowStartMaximized

Valid values: `yes` or `no`

Default value: `yes`

Description:
If set to `yes`, the Setup program's background window will initially be displayed in a maximized state, where it won't cover over the taskbar.

This directive has no effect if `WindowVisible` is not set to `yes`.

[Setup]: WindowResizable

Valid values: `yes` or `no`

Default value: `yes`

Description:
If set to `no`, the user won't be able to resize the Setup program's background window when it's not maximized.

This directive has no effect if `WindowVisible` is not set to `yes`.

[Setup]: WindowVisible

Valid values: `yes` or `no`

Default value: `no`

Description:
If set to `yes`, there will be a gradient background window displayed behind the wizard.

Note that this is considered a legacy feature; it likely will be removed at some point in the future.

[Setup]: WizardImageBackColor

Valid values: A value in the form of `$bbggrr`, where `rr`, `gg`, and `bb` specify the two-digit intensities (in hexadecimal) for red, green, and blue respectively. Or it may be one of the following predefined color names: clBlack, clMaroon, clGreen, clOlive, clNavy, clPurple, clTeal, clGray, clSilver, clRed, clLime, clYellow, clBlue, clFuchsia, clAqua, clWhite.

Default value: `$400000`

Description:
This directive specifies the background color used to fill the unused space around the wizard bitmap (which is specified by WizardImageFile).

[Setup]: WizardSmallImageBackColor

Valid values: A value in the form of `$bbggrr`, where `rr`, `gg`, and `bb` specify the two-digit intensities (in hexadecimal) for red, green, and blue respectively. Or it may be one of the following predefined color names: clBlack, clMaroon, clGreen, clOlive, clNavy, clPurple, clTeal, clGray, clSilver, clRed, clLime, clYellow, clBlue, clFuchsia, clAqua, clWhite.

Default value: `clWhite`

Description:
This directive specifies the background color used to fill the unused space around the small wizard bitmap (which is specified by WizardSmallImageFile).

[Setup]: SourceDir

Description:
Specifies a new source directory for the script.

Example: `SourceDir=c:\files`

[Setup]: OutputDir

Default value: `Output`

Description:
Specifies the "output" directory for the script, which is where the Setup Compiler will place the resulting SETUP.* files. By default, it creates a directory named "Output" under the directory containing the script for this.

If `OutputDir` is not a fully-qualified pathname, it will be treated as being relative to `SourceDir`. Setting `OutputDir` to `.` will result in the files being placed in the source directory.

Example: `OutputDir=c:\output`

[Setup]: WizardStyle

Valid values: `modern`

Default value: `modern`

Description:
Obsolete in 3.0. Inno Setup 2.x supported an alternate wizard style called "classic". Support for the "classic" style has been dropped in Inno Setup 3.0.

[Setup]: UninstallStyle

Valid values: `modern` or `classic`

Default value: `modern`

Description:
If this is set to `modern`, Setup will use the 'modern' uninstaller style which looks exactly like the 'modern' wizard style. If this is set to `classic`, Setup will display the 'classic' uninstaller style.

[Setup]: WizardSmallImageFile

Default value: `compiler:WIZMODERNSMALLIMAGE.BMP`

Description:
Specifies the name of the bitmap file to display in the upper right corner of the wizard window. This file must be located in your installation's source directory when running the Setup Compiler, unless a fully qualified pathname is specified or the pathname is prefixed by "compiler:", in which case it looks for the file in the Compiler directory.

256-color bitmaps may not display correctly in 256-color mode, since it does not handle palettes. The maximum size of the bitmap is 55x55 pixels.

Example: `WizardSmallImageFile=mysmallimage.bmp`

[Setup]: AlwaysShowComponentsList

Valid values: `yes` or `no`

Default value: `yes`

Description:
If this directive is set to `yes`, Setup will always show the components list for customizable setups. If this is set to `no` Setup will only show the components list if the user selected a custom type from the type list.

[Setup]: AlwaysShowDirOnReadyPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this directive is set to `yes`, Setup will always show the selected directory in the list of settings on the *Ready to Install* wizard page. If this is set to `no`, Setup will not show the selected directory if `DisableDirPage` is `yes`.

[Setup]: AlwaysShowGroupOnReadyPage

Valid values: `yes` or `no`

Default value: `no`

Description:
If this directive is set to `yes`, Setup will always show the selected Start Menu folder name in the list of settings on the *Ready to Install* wizard page. If this is set to `no`, Setup will not show the selected Start Menu folder name if `DisableProgramGroupPage` is `yes`.

If no Start Menu folder is going to be created by Setup, this directive is effectively ignored.

[Setup]: FlatComponentsList

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is set to `yes`, Setup will use 'flat' checkboxes for the components list. Otherwise Setup will use '3D' checkboxes.

[Setup]: ShowComponentSizes

Valid values: `yes` or `no`

Default value: `yes`

Description:
When this directive is set to `yes`, Setup will show the size of a component in the components list. Depending on the largest component, Setup will display sizes in kilobytes or in megabytes.

[Setup]: ShowTasksTreeLines

Valid values: `yes` or `no`

Default value: `no`

Description:
When this directive is set to `yes`, Setup will show 'tree lines' between parent and sub tasks.

[Setup]: ShowLanguageDialog

Valid values: `yes`, `no`, or `auto`

Default value: `yes`

Description:
When set to `yes` and there are multiple [Languages] section entries, a *Select Language* dialog will be displayed to give the user an opportunity to override the language Setup chose by default. See the [Languages] section documentation for more information.

When set to `no`, the dialog will never be displayed.

When set to `auto`, the dialog will only be displayed if Setup does not find a language identifier match.

[Setup]: LanguageDetectionMethod

Valid values: `uilanguage`, `locale`, `none`

Default value: `uilanguage`

Description:
When set to `uilanguage`, Setup will determine the default language to use by checking the user's "UI language" (by calling GetUserDefaultUILanguage(), or on Windows versions where that function is unsupported, by reading the registry). This is the method that Microsoft recommends. The "UI language" is the language used in Windows' own dialogs. Thus, on an English edition of Windows, English will be the default, while on a Dutch edition of Windows, Dutch will be the default. On the MUI edition of Windows, the default will be the currently selected UI language.

When set to `locale`, Setup will determine the default language to use by calling GetUserDefaultLangID(). This function returns the setting of "Your locale" in Control Panel's Regional Options. It should however be noted that since Windows 2000 the "Your locale" option is not intended to affect languages; it is only documented to affect "numbers, currencies, times, and dates".

When set to `none`, Setup will use the first language specified in the [Languages] section as the default language.

[Setup]: TimeStampsInUTC

Valid values: `yes` or `no`

Default value: `no`

Description:
By default, time stamps on files referenced by [Files] section entries are saved and restored as local times. This means that if a particular file has a time stamp of 01:00 local time at compile time, Setup will extract the file with a time stamp of 01:00 local time, regardless of the user's time zone setting or whether DST is in effect.

If `TimeStampsInUTC` is set to `yes`, time stamps will be saved and restored in UTC -- the native time format of Win32 and NTFS. In this mode, a file with a time stamp of 01:00 local time in New York will have a time stamp of 06:00 local time when installed in London.

Notes on "yes" and "no"

For compatibility with previous Inno Setup versions, `1` and `0` may be used in place of `yes` and `no`, respectively.

Additionally, it allows `true` and `false` to be used in place of `yes` and `no`.

Appending to Existing Uninstall Logs

When a new version of an application is installed over an existing version, instead of creating a new uninstall log file (unins???.dat), Setup will by default look for and append to an existing uninstall log file that belongs to the same application and is in the same directory. This way, when the application is uninstalled, changes made by all the different installations will be undone (starting with the most recent installation).

The uninstaller will use the messages from the most recent installation of the application. However, there is an exception: if an installation was built with an older version of Inno Setup that included an older version of the uninstaller than the existing one on the user's system, neither the existing uninstaller nor its messages will be replaced. In this case the uninstall log will still be appended to, though, since the file format is backward compatible.

The application name displayed in the uninstaller will be the same as the value of the `[Setup]` section directive AppName from the most recent installation, unless UpdateUninstallLogAppName is set to `no`.

The uninstall log-appending feature is new to Inno Setup 1.3. If you wish to disable it, set the `[Setup]` section directive UninstallLogMode.

Note: Setup can only append to uninstall log files that were created by an Inno Setup 1.3.1 (or later) installation.

Same Application

"Same application" refers to two separate installations that share the same AppId setting (or if `AppId` is not set, the same AppName setting).

Source Directory

By default, the Setup Compiler expects to find files referenced in the script's `[Files]` section `Source` parameters, and files referenced in the `[Setup]` section, under the same directory the script file is located if they do not contain fully qualified pathnames. To specify a different source directory, create a SourceDir directive in the script's `[Setup]` section.

Using Build Number and/or Service Pack Levels

The version numbers in `MinVersion` and `OnlyBelowVersion` can include build numbers and/or service pack levels. Examples: `5.0.2195`, `5.0sp1`, `5.0.2195sp1`. If a build number is not specified or is zero, Setup will not check the build number. If a service pack level is not specified or is zero, Setup interprets it as meaning "no service pack."

Windows Versions

Windows versions:

4.0.950	Windows 95
4.0.1111	Windows 95 OSR 2 & OSR 2.1
4.0.1212	Windows 95 OSR 2.5
4.1.1998	Windows 98
4.1.2222	Windows 98 Second Edition
4.9.3000	Windows Me

Windows NT versions:

4.0.1381	Windows NT 4.0
5.0.2195	Windows 2000
5.01.2600	Windows XP
5.02.3790	Windows Server 2003

Note that there is normally no need to specify the build numbers (i.e. you may simply use "4.1" for Windows 98).

Pascal Scripting: Introduction

The Pascal scripting feature (modern Delphi-like Pascal) adds lots of new possibilities to customize your Setup at run-time. Some examples:

- Support for aborting setup startup under custom conditions.
- Support for adding custom wizard pages to Setup at run-time.
- Support for extracting and calling DLL or other files from the Pascal script before, during or after the installation.
- Support for scripted constants that can do anything the normal constants, the read-from-registry, read-from-ini and read-from-commandline constants can do + more.
- Support for run-time removal of types, components and/or tasks under custom conditions.
- Support for conditional installation of `[Files]`, `[Registry]`, `[Run]` etc. entries based on custom conditions.
- Lots of support functions to do from the Pascal script just about everything Inno Setup itself does/can do + more.

An integrated run-time debugger to debug your custom Pascal script is also available.

The scripting engine used by Inno Setup is Innerfuse Pascal Script by Carlo Kok from Innerfuse. Like Inno Setup, Innerfuse Pascal Script is freely available and comes with source. See http://www.carlo-kok.com/ifps3.php for more information.

See also
Creating the [Code] section
Event Functions
Scripted Constants
Check Parameters
Examples
Support Functions Reference
Support Classes Reference

Pascal Scripting: Creating the [Code] Section

The `[Code]` section is an optional section that specifies a Pascal script. A Pascal script can be used to customize Setup in many ways. Note that creating a Pascal script is not easy and requires experience with Inno Setup and knowledge about programming in Pascal or at least a similar programming language.

The "Code*.iss" files in the "Examples" subdirectory in your Inno Setup directory contain various example [Code] sections. Please study them carefully before trying to create your own Pascal script.

Pascal Scripting: Event functions

The Pascal script can contain several event functions which are called at appropiate times. These are:

- `function InitializeSetup(): Boolean;`
  return False to abort Setup
- `procedure InitializeWizard();`
  Use this event function to make changes to the wizard or wizard pages at startup. You can't use the `InitializeSetup` event function for this since at the time it is triggered, the wizard form does not yet exist.
- `procedure DeInitializeSetup();`
- `procedure CurStepChanged(CurStep: Integer);`
- `function NextButtonClick(CurPage: Integer): Boolean;`
  return False to surpress the click on the Next button
- `function BackButtonClick(CurPage: Integer): Boolean;`
  return False to surpress the click on the Back button
- `function SkipCurPage(CurPage: Integer): Boolean;`
  return True to skip the CurPage page
- `procedure CurPageChanged(CurPage: Integer);`
- `function CheckPassword(Password: String): Boolean;`
  If Setup finds the `CheckPassword` event function in the Pascal script, it automatically displays the *Password* page and calls `CheckPassword` to check passwords. Return True to accept the password and False to reject it. When using a password, it's important to keep in mind that since no encryption is used and the source code to Inno Setup is freely available, it would not be too difficult for an experienced individual to remove the password protection from an installation. Use this only as a "deterrent" to keep unauthorized people out of your installations.
- `function NeedRestart(): Boolean;`
  return True to instruct Setup to prompt the user to restart the system at the end of a successful installation.
- `function UpdateReadyMemo(Space, NewLine, MemoUserInfoInfo, MemoDirInfo, MemoTypeInfo, MemoComponentsInfo, MemoGroupInfo, MemoTasksInfo: String): String;`
  If Setup finds the `UpdateReadyMemo` event function in the Pascal script, it is called automatically when the *Ready to Install* wizard page becomes the active page. It should return the text to be displayed in the settings memo on the *Ready to Install* wizard page as a single string with lines separated by the `NewLine` parameter. Parameter `Space` contains a string with spaces. Setup uses this string to indent settings. The other parameters contain the (possibly empty) strings that Setup would have used as the setting sections. The `MemoDirInfo` parameter for example contains the string for the *Selected Directory* section.
- `procedure RegisterPreviousData(PreviousDataKey: Integer);`
  To store user settings entered on custom wizard pages, place a `RegisterPreviousData` event function in the Pascal script and call `SetPreviousData(PreviousDataKey, ...)` inside it, once per setting.
- `function CheckSerial(Serial: String): Boolean;`
  If Setup finds the `CheckSerial` event function in the Pascal script, a serial number field will automatically appear on the User Info wizard page (which must be enabled using `UserInfoPage=yes` in your [Setup] section!). Return True to accept the serial number and False to reject it. When using serial numbers, it's important to keep in mind that since no encryption is used

and the source code to Inno Setup is freely available, it would not be too difficult for an experienced individual to remove the serial number protection from an installation. Use this only as a convienience to the end user and double check the entered serial number (stored in the `{userinfoserial}` constant) in your application.

Here's the list of constants used by these functions:

CurStep values
```
csStart, csWizard, csCopy, csFinished, csTerminate
```

CurPage values
```
wpWelcome, wpLicense, wpPassword, wpInfoBefore, wpUserInfo, wpSelectDir,
wpSelectComponents, wpSelectProgramGroup, wpSelectTasks, wpReady,
wpPreparing, wpInstalling, wpInfoAfter, wpFinished
```

None of these functions are required to be present in a Pascal script.

`SkipCurPage` isn't called if Setup already determined that the page should be skipped or for the *Welcome* page or for the *Installing* page.

Pascal Scripting: Scripted Constants

The Pascal script can contain several functions which are called when Setup wants to know the value of a scripted `{code:...}` constant. The called function has to have 1 String parameter named `Default` which is used to pass a default value. It has to return a String value.

The syntax of a `{code:...}` constant is: **{code:*FunctionName*|*DefaultValue*}**

- *FunctionName* specifies the name of the Pascal script function.
- *DefaultValue* determines the string to embed if the specified function does not exist.
- If you wish to include a comma, vertical bar ("|"), or closing brace ("}") inside the constant, you must escape it via "%-encoding." Replace the character with a "%" character, followed by its two-digit hex code. A comma is "%2c", a vertical bar is "%7c", and a closing brace is "%7d". If you want to include an actual "%" character, use "%25".
- *DefaultValue* may include constants. Note that you do *not* need to escape the closing brace of a constant as described above; that is only necessary when the closing brace is used elsewhere.

  Example:
  `DefaultDirName={code:MyConst|{pf}}`\My Program

Here is an example of a `[Code]` section containing the `MyConst` function used above.

```
[Code]
program Setup;

function MyConst(Default: String): String;
begin
  Result := ExpandConstant('{pf}');
end;

begin
end.
```

See also
Constants

Pascal Scripting: Check Parameters

There is one optional parameter that is supported by all sections whose entries are separated into parameters. This is:

Check

Description:
The name of the check function in the Pascal script file that determines whether an entry has to be processed or not. May include one parameter that Setup should pass to the check function. This parameter may include constants.

Example:

```
[Files]
Source: "MYPROG.EXE"; DestDir: "{app}"; Check: MyProgCheck
Source: "A\MYFILE.TXT"; DestDir: "{app}"; Check: MyDirCheck({app}\A)
Source: "B\MYFILE.TXT"; DestDir: "{app}"; Check: MyDirCheck({app}\B)
```

Here is an example of `[Code]` section containing the check functions used above.

```
[Code]
program Setup;

var
  MyProgChecked: Boolean;
  MyProgCheckResult: Boolean;

function MyProgCheck(): Boolean;
begin
  if not MyProgChecked then begin
    MyProgCheckResult := MsgBox('Script.MyProg:' #13#13 'Do you want to
install MyProg.exe?', mbConfirmation, MB_YESNO) = idYes;
    MyProgChecked := True;
  end;
  Result := MyProgCheckResult;
end;

function MyDirCheck(DirName: String): Boolean;
begin
  Result := DirExists(DirName);
end;

begin
end.
```

The check function has to have the same prototype as in the example above: no parameters or 1 String parameter depending on the setting of the Check parameter and a Boolean return value. If the check function returns True, the entry is processed otherwise it's skipped.

Setup might call the check function several times, even if there's only one entry that uses the check function. If your function performs a lenghty piece of code, you can optimize it by performing the code only once and 'caching' the result in a global variable.

The check function isn't called if Setup already determined from the entry's Components and/or Tasks parameter that it shouldn't be processed.

Pascal Scripting: Examples

The Pascal Scripting example scripts are located in separate files. Open one of the "Code*.iss" files in the "Examples" subdirectory in your Inno Setup directory.

Pascal Scripting: Support Functions Reference

Here's the list of support functions that can be called from within the Pascal script.

Note: only use `CallDllProc` on DLL functions that use the standard calling convention. If you want to pass/receive a string to or from a DLL function, use the `CastStringToInteger` or `CastIntegerToString` function in your Pascal script. See below for the prototype of these functions.

Setup Info functions

```
function GetCmdTail: String;
function ParamCount: Integer;
function ParamStr(Index: Integer): String;

function ActiveLanguage: String;

function SetupMessage(const ID: TSetupMessageID): String;

function WizardDirValue: String;
function WizardGroupValue: String;
function WizardNoIcons: Boolean;
function WizardSetupType(const Description: Boolean): String;
function WizardSelectedComponents(const Descriptions: Boolean): String;
function WizardSelectedTasks(const Descriptions: Boolean): String;
function WizardSilent: Boolean;

function ExpandConstant(const S: String): String;
function ExpandConstantEx(const S: String; const CustomConst, CustomValue:
String): String;

function ShouldProcessEntry(const Components, Tasks: String):
TShouldProcessEntryResult;

function ExtractTemporaryFile(const FileName: String): Boolean;

function GetPreviousData(const ValueName, DefaultValueData: String):
String;
function SetPreviousData(const PreviousDataKey: Integer; const ValueName,
ValueData: String): Boolean;

function Terminated: Boolean;
```

System functions

```
function IsAdminLoggedOn: Boolean;
function IsPowerUserLoggedOn: Boolean;
function UsingWinNT: Boolean;

function InstallOnThisVersion(const MinVersion, OnlyBelowVersion: String):
Integer;

function GetEnv(const EnvVar: String): String;
function GetUserNameString: String;
function GetComputerNameString: String;

function FindWindowByClassName(const ClassName: String): Longint;
function FindWindowByWindowName(const WindowName: String): Longint;
function SendMessage(const Wnd, Msg, WParam, LParam: Longint): Longint;
function PostMessage(const Wnd, Msg, WParam, LParam: Longint): Boolean;
```

```
function SendNotifyMessage(const Wnd, Msg, WParam, LParam: Longint):
Boolean;
function RegisterWindowMessage(const Name: String): Longint;
function SendBroadcastMessage(const Msg, WParam, LParam: Longint): Longint;

function PostBroadcastMessage(const Msg, WParam, LParam: Longint): Boolean;

function SendBroadcastNotifyMessage(const Msg, WParam, LParam: Longint):
Boolean;

procedure CreateMutex(const Name: String);
function CheckForMutexes(Mutexes: String): Boolean;
```

String functions

```
function Chr(B: Byte): Char;
function Ord(C: Char): Byte;
function Copy(S: String; Indx, Count: Integer): String;
function Length(s: String): Longint;
function Lowercase(s: string): String;
function StrGet(S: String; I: Integer): Char;
function StringOfChar(c: Char; I : Longint): String;
function StrSet(c: Char; I: Integer; var s: String): Char;
function Uppercase(s: string): String;
procedure Delete(var S: String; Indx, Count: Integer);
procedure Insert(Source: String; var Dest: String; Indx: Integer);
procedure StringChange(var S: String; const FromStr, ToStr: String);
function Pos(SubStr, S: String): Integer;
function AddQuotes(const S: String): String;
function RemoveQuotes(const S: String): String;
function ConvertPercentStr(var S: String): Boolean;

function CompareText(const S1, S2: string): Integer;
function CompareStr(const S1, S2: string): Integer;

function Format1(const Format, S1: String): String;
function Format2(const Format, S1, S2: String): String;
function Format3(const Format, S1, S2, S3: String): String;
function Format4(const Format, S1, S2, S3, S4: String): String;

function Trim(const S: string): String;
function TrimLeft(const S: string): String;
function TrimRight(const S: string): String;

function StrToIntDef(s: string; def: Longint): Longint;
function StrToInt(s: string): Longint;
function IntToStr(i: Longint): String;

function AddBackslash(const S: String): String;
function RemoveBackslashUnlessRoot(const S: String): String;
function RemoveBackslash(const S: String): String;
```

```
function AddPeriod(const S: String): String;
function ExtractFileExt(const FileName: string): String;
function ExtractFileDir(const FileName: string): String;
function ExtractFilePath(const FileName: string): String;
function ExtractFileName(const FileName: string): String;
function ExtractFileDrive(const FileName: string): String;
function ExtractRelativePath(const BaseName, DestName: String): String;
function ExpandFileName(const FileName: string): String;
function ExpandUNCFileName(const FileName: string): String;

function GetTimeString: String;
function GetDateString: String;
function GetDateTimeString: String;

procedure SetLength(var S: String; L: Longint);
procedure CharToOemBuff(var S: String);
procedure OemToCharBuff(var S: String);

function SysErrorMessage(ErrorCode: Integer): String;
```

Array functions

```
function GetArrayLength(var Arr: Array): Longint;
procedure SetArrayLength(var Arr: Array; I: Longint);

function Low(var U: Array): Longint;
function High(var U: Array): Longint;
```

File System functions

```
function DirExists(const Name: String): Boolean;
function FileExists(const Name: String): Boolean;
function FileOrDirExists(const Name: String): Boolean;
function FileSize(const Name: String; var Size: Integer): Boolean;
function DiskFree(Drive: Char): Integer;
function DiskSize(Drive: Char): Integer;

function FileSearch(const Name, DirList: string): String;
function FindFirst(const FileName: String): String;
function FindNext: String;

function GetCurrentDir: String;
function SetCurrentDir(const Dir: string): Boolean;
function GetWinDir: String;
function GetSystemDir: String;
function GetTempDir: String;
function GetShellFolder(Common: Boolean; const ID: TShellFolderID): String;


function GetShortName(const LongName: String): String;
function GenerateUniqueName(Path: String; const Extension: String): String;
```

```
function GetVersionNumbers(const Filename: String; var VersionMS,
VersionLS: Cardinal): Boolean;
function GetVersionNumbersString(const Filename: String; var Version:
String): Boolean;
```

File functions

```
function InstExec(const Filename, Params: String; WorkingDir: String; const
WaitUntilTerminated, WaitUntilIdle: Boolean; const ShowCmd: Integer; var
ResultCode: Integer): Boolean;
function InstShellExec(const Filename, Params: String; WorkingDir: String;
const ShowCmd: Integer; var ErrorCode: Integer): Boolean;

function RenameFile(const OldName, NewName: string): Boolean;
function ChangeFileExt(const FileName, Extension: string): String;
function FileCopy(const ExistingFile, NewFile: String; const FailIfExists:
Boolean): Boolean;
function DeleteFile(const FileName: string): Boolean;
procedure DelayDeleteFile(const Filename: String; const Tries: Integer);

function LoadStringFromFile(const FileName: String; var S: String):
Boolean;
function LoadStringsFromFile(const FileName: String; var S:
TArrayOfString): Boolean;
function SaveStringToFile(const FileName, S: String; const Append:
Boolean): Boolean;
function SaveStringsToFile(const FileName: String; const S: TArrayOfString;
const Append: Boolean): Boolean;

function CreateDir(const Dir: string): Boolean;
procedure ForceDirectories(Dir: string);
function RemoveDir(const Dir: string): Boolean;
function DelTree(const Path: String; const IsDir, DeleteFiles,
DeleteSubdirsAlso: Boolean): Boolean;

function CreateShellLink(const Filename, Description, ShortcutTo,
Parameters, WorkingDir, IconFilename: String; const IconIndex, ShowCmd:
Integer): Boolean;

procedure RegisterServer(const Filename: String; const FailCriticalErrors:
Boolean);
function UnregisterServer(const Filename: String; const FailCriticalErrors:
Boolean): Boolean;
procedure RegisterTypeLibrary(const Filename: String);
function UnregisterTypeLibrary(const Filename: String): Boolean
procedure IncrementSharedCount(const Filename: String; const
AlreadyExisted: Boolean);
function DecrementSharedCount(const Filename: String): Boolean;
procedure RestartReplace(const TempFile, DestFile: String);
procedure UnregisterFont(const FontName, FontFilename: String);
```

```
function ModifyPifFile(const Filename: String; const CloseOnExit: Boolean):
Boolean;
```

Registry functions

```
function RegKeyExists(const RootKey: Integer; const SubKeyName: String):
Boolean;
function RegValueExists(const RootKey: Integer; const SubKeyName,
ValueName: String): Boolean;

function RegGetSubkeyNames(const RootKey: Integer; const SubKeyName:
String; var Names: TArrayOfString): Boolean;

function RegQueryStringValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;
function RegQueryMultiStringValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;
function RegQueryDWordValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultDWord: Cardinal): Boolean;
function RegQueryBinaryValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;

function RegWriteStringValue (const RootKey: Integer; const SubKeyName,
ValueName, Value: String): Boolean;
function RegWriteMultiStringValue (const RootKey: Integer; const
SubKeyName, ValueName, Value: String): Boolean;
function RegWriteDWordValue (const RootKey: Integer; const SubKeyName,
ValueName: String; const Value: Cardinal): Boolean;
function RegWriteBinaryValue (const RootKey: Integer; const SubKeyName,
ValueName, Value: String): Boolean;
```

INI File functions

```
function IniKeyExists(const Section, Key, Filename: String): Boolean;
function IsIniSectionEmpty(const Section, Filename: String): Boolean;

function GetIniBool(const Section, Key: String; const Default: Boolean;
const Filename: String): Boolean
function GetIniInt(const Section, Key: String; const Default, Min, Max:
Longint; const Filename: String): Longint;
function GetIniString(const Section, Key, Default, Filename: String):
String;

function SetIniBool(const Section, Key: String; const Value: Boolean; const
Filename: String): Boolean;
function SetIniInt(const Section, Key: String; const Value: Longint; const
Filename: String): Boolean;
function SetIniString(const Section, Key, Value, Filename: String):
Boolean;

procedure DeleteIniSection(const Section, Filename: String);
procedure DeleteIniEntry(const Section, Key, Filename: String);
```

Custom Wizard Page functions

```
procedure ScriptDlgPageSetCaption(const Caption: String);
procedure ScriptDlgPageSetSubCaption1(const SubCaption1: String);
procedure ScriptDlgPageSetSubCaption2(const SubCaption2: String);
procedure ScriptDlgPageShowBackButton(const Show: Boolean);

procedure ScriptDlgPageOpen;

function InputDir(const AppendDir: String; var Value: String): Boolean;
function InputFile(const Prompt, Filter, DefaultExtension: string; var
Value: string): Boolean;
function InputFileArray(const Prompts, Filters, DefaultExtensions:
TArrayOfString; var Values: TArrayOfString): Boolean;
function InputOptionArray(const Prompts: TArrayOfString; var Values:
TArrayOfString; const Exclusive, ListBox: Boolean): Boolean;
function InputOption(const Prompt: String; var Value: String): Boolean;
function InputQuery(const Prompt: String; var Value: String): Boolean;
function InputQueryArray(const Prompts: TArrayOfString; var Values:
TArrayOfString): Boolean;
function InputQueryArrayEx(const Prompts: TArrayOfString; const
PasswordChars: TArrayOfChar; var Values: TArrayOfString): Boolean;
function OutputMsg(const Msg: String; const WaitUntilClick: Boolean):
Boolean;
function OutputMsgMemo(const Prompt, Msg: String): Boolean;
procedure OutputProgress(const Msg1, Msg2: String; const Progress,
MaxProgress: Longint);
function ScriptDlgPageProcessCustom(): Boolean;
procedure ScriptDlgPageClearCustom();

procedure ScriptDlgPageClose(const FullRestore: Boolean);
```

Dialog functions

```
function MsgBox(const Text: String; const Typ: TMsgBoxType; const Buttons:
Integer): Integer;
function GetOpenFileName(const Prompt: String; var FileName: String; const
InitialDirectory, Filter, DefaultExtension: String): Boolean;
function BrowseForFolder(const Prompt: String; var Directory: String):
Boolean;
function ExitSetupMsgBox: Boolean;
```

Explicit DLL functions

```
function LoadDLL(const DLLName: String; var ErrorCode: Integer): Longint;
function CallDLLProc(const DLLHandle: Longint; const ProcName: String;
const Param1, Param2: Longint; var Result: Longint): Boolean;
function FreeDLL(const DLLHandle: Longint): Boolean;

function CastStringToInteger(var S: String): Longint;
function CastIntegerToString(const L: Longint): String;
```

Other functions

```
procedure Sleep(const Milliseconds: LongInt);
function Random(const Range: Integer): Integer;
procedure Beep;

procedure BringToFrontAndRestore;
```

Here's the list of constants used by these functions:

```
CurStep values
csStart, csWizard, csCopy, csFinished, csTerminate

CurPage values
wpWelcome, wpLicense, wpPassword, wpInfoBefore, wpUserInfo, wpSelectDir,
wpSelectComponents, wpSelectProgramGroup, wpSelectTasks, wpReady,
wpPreparing, wpInstalling, wpInfoAfter, wpFinished

TMsgBoxType
mbInformation, mbConfirmation, mbError, mbCriticalError

MsgBox - Buttons flags
MB_OK, MB_OKCANCEL, MB_ABORTRETRYIGNORE, MB_YESNOCANCEL, MB_YESNO,
MB_RETRYCANCEL, MB_DEFBUTTON1, MB_DEFBUTTON2, MB_DEFBUTTON3,
MB_SETFOREGROUND

MsgBox - return values
IDOK, IDCANCEL, IDABORT, IDRETRY, IDIGNORE, IDYES, IDNO

TGetShellFolderID
sfDesktop, sfStartMenu, sfPrograms, sfStartup, sfSendTo, sfFonts,
sfAppData, sfDocs, sfTemplates, sfFavorites, sfLocalAppData

Reg* - RootKey values
HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS,
HKEY_PERFORMANCE_DATA, HKEY_CURRENT_CONFIG, HKEY_DYN_DATA,
HKCR, HKCU, HKLM, HKU, HKCC

TShouldProcessEntryResult
srNo, srYes, srUnknown

InstallOnThisVersion - return values
irInstall, irNotOnThisPlatform, irVerTooLow, irVerTooHigh, irInvalid

TSetupMessageID
Use 'msg' + the message name. Example: SetupMessage(msgSetupAppTitle)

Inst*Exec - ShowCmd values
SW_SHOWNORMAL, SW_SHOWMAXIMIZED, SW_SHOWMINNOACTIVE, SW_HIDE
```

Support function: GetCmdTail

Prototype:

```
function GetCmdTail: String;
```

Description:
Returns all command line parameters passed to Setup as a single string.

Remarks:
none

Support function: ParamCount

Prototype:

```
function ParamCount: Integer;
```

Description:
Returns the number of command line parameters passed to Setup.

Remarks:
none

Support function: ParamStr

Prototype:

```
function ParamStr(Index: Integer): String;
```

Description:
Returns the Index-th command line parameter passed to Setup.

Remarks:
none

Support function: ActiveLanguage

Prototype:

```
function ActiveLanguage: String;
```

Description:
Returns the name of the active language.

Remarks:
none

Support function: SetupMessage

Prototype:

```
function SetupMessage(const ID: TSetupMessageID): String;
```

Description:
Returns the value of the specified message.

Remarks:
none

Support function: WizardDirValue

Prototype:

```
function WizardDirValue: String;
```

Description:
Returns the application directory selected by the user.

Remarks:
none

Support function: WizardGroupValue

Prototype:

```
function WizardGroupValue: String;
```

Description:
Returns the start menu folder selected by the user.

Remarks:
none

Support function: WizardNoIcons

Prototype:

```
function WizardNoIcons: Boolean;
```

Description:
Returns the 'don't create any icons' setting selected by the user.

Remarks:
none

Support function: WizardSetupType

Prototype:

```
function WizardSetupType(const Description: Boolean): String;
```

Description:

Returns the name or description of the setup type selected by the user.

Remarks:

none

Support function: WizardSelectedComponents

Prototype:

```
function WizardSelectedComponents(const Descriptions: Boolean): String;
```

Description:

Returns a comma-separated list of names or descriptions of the components selected by the user.

Remarks:

none

Support function: WizardSelectedTasks

Prototype:

```
function WizardSelectedTasks(const Descriptions: Boolean): String;
```

Description:

Returns a comma-separated list of names or descriptions of the tasks selected by the user.

Remarks:

none

Support function: WizardSilent

Prototype:

```
function WizardSilent: Boolean;
```

Description:

Returns True if Setup is running silently, False otherwise.

Remarks:

none

Support function: ExpandConstant

Prototype:

```
function ExpandConstant(const S: String): String;
```

Description:
Changes all constants in S to their values. For example, ExpandConstant('{srcexe}') is changed to the filename of Setup.

Remarks:
none

Support function: ExpandConstantEx

Prototype:

```
function ExpandConstantEx(const S: String; const CustomConst, CustomValue:
String): String;
```

Description:

Changes all constants in S to their values. Additionally, any constant equal to CustomConst will be changed to CustomValue.

Remarks:

none

Support function: ShouldProcessEntry

Prototype:

```
function ShouldProcessEntry(const Components, Tasks: String):
TShouldProcessEntryResult;
```

Description:

Returns srYes if an entry with the specified Components and Tasks parameters should be installed.

Remarks:

none

Support function: ExtractTemporaryFile

Prototype:

```
function ExtractTemporaryFile(const FileName: String): Boolean;
```

Description:
Extracts the specified file from the [Files] section to a temporary directory. To find the location of the temporary directory use ExpandConstant('{tmp}').

The extracted files are automatically deleted when Setup exits.

Returns True if the file was extracted successfully and False if it wasn't extracted successfully or if the file wasn't found or if the file was found be couldn't be processed because of its 'MinVersion' and/or 'OnlyBelowVersion' parameters.

Remarks:
Use 'CopyMode: dontcopy' in the [Files] section to tell Setup to skip the file during the normal file copying stage.

Support function: GetPreviousData

Prototype:

```
function GetPreviousData(const ValueName, DefaultValueData: String): String;
```

Description:

Gets a value that was previously stored using SetPreviousData.

Remarks:

none

Support function: SetPreviousData

Prototype:

```
function SetPreviousData(const PreviousDataKey: Integer; const ValueName,
ValueData: String): Boolean;
```

Description:

Sets a value that can be restored later using GetPreviousData. Call SetPreviousData inside a RegisterPreviousData event function.

Remarks:

none

Support function: Terminated

Prototype:

```
function Terminated: Boolean;
```

Description:
Returns True if Setup is terminating, False otherwise.

Remarks:
none

Support function: IsAdminLoggedOn

Prototype:

```
function IsAdminLoggedOn: Boolean;
```

Description:
Returns True if an administrator is logged onto the system. Always returns True on Windows 95/98/ME.

Remarks:
none

Support function: IsPowerUserLoggedOn

Prototype:

```
function IsPowerUserLoggedOn: Boolean;
```

Description:
Returns True if a Power User is logged onto the system. Always returns True on Windows 95/98/ME.

Remarks:
none

Support function: UsingWinNT

Prototype:

```
function UsingWinNT: Boolean;
```

Description:
Returns True if system is running any version of Windows NT.

Remarks:
none

Support function: InstallOnThisVersion

Prototype:

```
function InstallOnThisVersion(const MinVersion, OnlyBelowVersion: String):
Integer;
```

Description:

Returns irInstall if an entry with the specified MinVersion and OnlyBelowVersion parameters should be installed.

Remarks:

none

Support function: GetEnv

Prototype:

```
function GetEnv(const EnvVar: String): String;
```

Description:
Gets the value of the specified environment variable.

Remarks:
none

Support function: GetUserNameString

Prototype:

```
function GetUserNameString: String;
```

Description:
Retrieves the name of the user currently logged onto the system.

Remarks:
none

Support function: GetComputerNameString

Prototype:
```
function GetComputerNameString: String;
```

Description:
Retrieves the computer name of the current system.

Remarks:
none

Support function: FindWindowByClassName

Prototype:

```
function FindWindowByClassName(const ClassName: String): Longint;
```

Description:

Retrieves a handle to the top-level window whose class name match the specified string. This function does not search child windows. This function does not perform a case-sensitive search.

Remarks:

none

Support function: FindWindowByWindowName

Prototype:

```
function FindWindowByWindowName(const WindowName: String): Longint;
```

Description:

Retrieves a handle to the top-level window whose window name match the specified string. This function does not search child windows. This function does not perform a case-sensitive search.

Remarks:

none

Support function: SendMessage

Prototype:

```
function SendMessage(const Wnd, Msg, WParam, LParam: Longint): Longint;
```

Description:
Sends the specified message to the specified window. Does not return until the window procedure has processed the message.

Remarks:
none

Support function: PostMessage

Prototype:

```
function PostMessage(const Wnd, Msg, WParam, LParam: Longint): Boolean;
```

Description:
Posts the specified message to the specified window, returning immediately.

Remarks:
none

Support function: SendNotifyMessage

Prototype:

```
function SendNotifyMessage(const Wnd, Msg, WParam, LParam: Longint): Boolean;
```

Description:
not yet available

Remarks:
none

Support function: RegisterWindowMessage

Prototype:

```
function RegisterWindowMessage(const Name: String): Longint;
```

Description:
The RegisterWindowMessage function defines a new window message that is guaranteed to be unique throughout the system. The returned message value can be used when calling the SendBroadcastMessage or PostBroadcastMessage function.

Remarks:
none

Support function: SendBroadcastMessage

Prototype:

```
function SendBroadcastMessage(const Msg, WParam, LParam: Longint): Longint;
```

Description:
Sends the specified message to top-level windows in the system. Does not return until all window procedure have processed the message.
The specified message must be unique. Use RegisterWindowMessage to get such a message.

Remarks:
none

Support function: PostBroadcastMessage

Prototype:

```
function PostBroadcastMessage(const Msg, WParam, LParam: Longint): Boolean;
```

Description:

Posts the specified message to top-level windows in the system, returning immediately.
The specified message must be unique. Use RegisterWindowMessage to get such a message.

Remarks:

none

Support function: SendBroadcastNotifyMessage

Prototype:

```
function SendBroadcastNotifyMessage(const Msg, WParam, LParam: Longint):
Boolean;
```

Description:
not yet available

Remarks:
none

Support function: CreateMutex

Prototype:

```
procedure CreateMutex(const Name: String);
```

Description:
Creates a mutex with the specified name.

Remarks:
none

Support function: CheckForMutexes

Prototype:

```
function CheckForMutexes(Mutexes: String): Boolean;
```

Description:

Returns True if any of the mutexes in the comma-separated Mutexes string exist.

Remarks:

none

Support function: Chr

Prototype:

```
function Chr(B: Byte): Char;
```

Description:
Returns the character with the specified ordinal value.

Remarks:
none

Support function: Ord

Prototype:
```
function Ord(C: Char): Byte;
```

Description:
Returns the ordinal value of the specified character.

Remarks:
none

Support function: Copy

Prototype:

```
function Copy(S: String; Indx, Count: Integer): String;
```

Description:

Returns a string containing Count characters starting with at S[Index].
If Index is larger than the length of S, Copy returns an empty string.
If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

Remarks:

none

Support function: Length

Prototype:

```
function Length(s: String): Longint;
```

Description:
Returns the length of the specified string.

Remarks:
none

Support function: Lowercase

Prototype:
```
function Lowercase(s: string): String;
```

Description:
Returns a string with the same text as the string passed in S, but with all letters converted to lowercase.

Remarks:
none

Support function: StrGet

Prototype:

```
function StrGet(S: String; I: Integer): Char;
```

Description:
Returns the I-th character in string S.

Remarks:
none

Support function: StringOfChar

Prototype:

```
function StringOfChar(c: Char; I : Longint): String;
```

Description:
Returns a string of length I with all characters set to character C.

Remarks:
none

Support function: StrSet

Prototype:

```
function StrSet(c: Char; I: Integer; var s: String): Char;
```

Description:
Set the I-th character in string S to character C.

Remarks:
none

Support function: Uppercase

Prototype:
```
function Uppercase(s: string): String;
```

Description:
Returns a string containing the same text as S, but with all letters converted to uppercase.

Remarks:
none

Support function: Delete

Prototype:

```
procedure Delete(var S: String; Indx, Count: Integer);
```

Description:

Removes a substring of Count characters from string S starting at S[Index].
If Index is larger than the length of S, no characters are deleted. If Count specifies more characters than remain starting at the S[Index], Delete removes the rest of the string.

Remarks:

none

Support function: Insert

Prototype:
```
procedure Insert(Source: String; var Dest: String; Indx: Integer);
```

Description:
Merges Source into S at the position S[index].

Remarks:
none

Support function: StringChange

Prototype:

```
procedure StringChange(var S: String; const FromStr, ToStr: String);
```

Description:

Change all occurances in S of FromStr to ToStr.

Remarks:

none

Support function: Pos

Prototype:

```
function Pos(SubStr, S: String): Integer;
```

Description:
Searches for Substr within S and returns an integer value that is the index of the first character of Substr within S.
If Substr is not found, Pos returns zero.

Remarks:
none

Support function: AddQuotes

Prototype:

```
function AddQuotes(const S: String): String;
```

Description:

Adds a quote (") character to the left and right sides of the string if the string contains a space and it didn't have quotes already. This is primarily used when spawning another process with a long filename as one of the parameters.

Remarks:

none

Support function: RemoveQuotes

Prototype:

```
function RemoveQuotes(const S: String): String;
```

Description:

Opposite of AddQuotes; removes any quotes around the string.

Remarks:

none

Support function: ConvertPercentStr

Prototype:

```
function ConvertPercentStr(var S: String): Boolean;
```

Description:
Expands all %-encoded characters in the string (see RFC 2396). Returns True if all were successfully expanded.

Remarks:
none

Support function: CompareText

Prototype:

```
function CompareText(const S1, S2: string): Integer;
```

Description:

Compares the strings S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. The CompareText function is not case sensitive.

Remarks:

none

Support function: CompareStr

Prototype:

```
function CompareStr(const S1, S2: string): Integer;
```

Description:

Compares S1 to S2, with case-sensitivity. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2.

Remarks:

none

Support function: Format1

Prototype:

```
function Format1(const Format, S1: String): String;
```

Description:

Returns the Format string with the first %s in the Format string replaced by the S1 string.

Remarks:

none

Support function: Format2

Prototype:

```
function Format2(const Format, S1, S2: String): String;
```

Description:

Returns the Format string with the first %s in the Format string replaced by the S1 string and the second %s replaced by the S2 string.

Remarks:

none

Support function: Format3

Prototype:

```
function Format3(const Format, S1, S2, S3: String): String;
```

Description:

Returns the Format string with the first %s in the Format string replaced by the S1 string, the second %s replaced by the S2 string, etc.

Remarks:

none

Support function: Format4

Prototype:

```
function Format4(const Format, S1, S2, S3, S4: String): String;
```

Description:

Returns the Format string with the first %s in the Format string replaced by the S1 string, the second %s replaced by the S2 string, etc.

Remarks:

none

Support function: Trim

Prototype:
```
function Trim(const S: string): String;
```

Description:
Trims leading and trailing spaces and control characters from the given string S.

Remarks:
none

Support function: TrimLeft

Prototype:
```
function TrimLeft(const S: string): String;
```

Description:
Trims leading spaces and control characters from the given string S.

Remarks:
none

Support function: TrimRight

Prototype:
```
function TrimRight(const S: string): String;
```

Description:
Trims trailing spaces and control characters from the given string S.

Remarks:
none

Support function: StrToIntDef

Prototype:

```
function StrToIntDef(s: string; def: Longint): Longint;
```

Description:
The StrToInt function converts the string passed in S into a number. If S does not represent a valid number, StrToInt returns the number passed in Def.

Remarks:
none

Support function: StrToInt

Prototype:

```
function StrToInt(s: string): Longint;
```

Description:
The StrToInt function converts the string passed in S into a number.

Remarks:
Use of StrToIntDef instead of StrToInt is recommended.

Support function: IntToStr

Prototype:

```
function IntToStr(i: Longint): String;
```

Description:
The IntToStr function converts an integer into a string containing the decimal representation of that number.

Remarks:
none

Support function: AddBackslash

Prototype:

```
function AddBackslash(const S: String): String;
```

Description:

Adds a trailing backslash to the string, if one wasn't there already.
But if S is an empty string, the function returns an empty string.

Remarks:

none

Support function: RemoveBackslashUnlessRoot

Prototype:

```
function RemoveBackslashUnlessRoot(const S: String): String;
```

Description:

Removes the trailing backslash from the string, if one exists and does not specify a root directory of a drive (i.e. "C:\").

Remarks:

none

Support function: RemoveBackslash

Prototype:

```
function RemoveBackslash(const S: String): String;
```

Description:

Removes the trailing backslash from the string, if one exists

Remarks:

none

Support function: AddPeriod

Prototype:
```
function AddPeriod(const S: String): String;
```

Description:
Adds a trailing period to the string, if one wasn't there already.

Remarks:
none

Support function: ExtractFileExt

Prototype:

```
function ExtractFileExt(const FileName: string): String;
```

Description:

Extracts the extension part of the given file name. The resulting string includes the period character that separates the name and extension parts. The resulting string is empty if the given filename has no extension.

Remarks:

none

Support function: ExtractFileDir

Prototype:

```
function ExtractFileDir(const FileName: string): String;
```

Description:
Extracts the drive and directory parts of the given file name. The resulting string is empty if FileName contains no drive and directory parts.

Remarks:
none

Support function: ExtractFilePath

Prototype:

```
function ExtractFilePath(const FileName: string): String;
```

Description:

Extracts the drive and directory parts of the given file name. The resulting string is the rightmost characters of FileName, up to and including the colon or backslash that separates the path information from the name and extension. The resulting string is empty if FileName contains no drive and directory parts.

Remarks:

none

Support function: ExtractFileName

Prototype:

```
function ExtractFileName(const FileName: string): String;
```

Description:

Extracts the name and extension parts of the given file name. The resulting string is the leftmost characters of FileName, starting with the first character after the colon or backslash that separates the path information from the name and extension. The resulting string is equal to FileName if FileName contains no drive and directory parts.

Remarks:

none

Support function: ExtractFileDrive

Prototype:

```
function ExtractFileDrive(const FileName: string): String;
```

Description:

Returns a string containing the 'drive' portion of a fully qualified path name for the file passed in the FileName. For file names with drive letters, the resulting string is in the form '<drive>:'. For file names with a UNC path the resulting string is in the form '\\<servername>\<sharename>'. If the given path contains neither style of path prefix, the result is an empty string.

Remarks:

none

Support function: ExtractRelativePath

Prototype:

```
function ExtractRelativePath(const BaseName, DestName: String): String;
```

Description:

Convert a fully qualified path name into a relative path name. The DestName parameter specifies file name (including path) to be converted. BaseName is the fully qualified name of the base directory to which the returned path name should be relative.

ExtractRelativePath strips out common path directories and inserts '..\' for each level up from the BaseName.

Remarks:

none

Support function: ExpandFileName

Prototype:

```
function ExpandFileName(const FileName: string): String;
```

Description:

Returns a string containing a fully qualified path name for the file passed in the FileName. A fully qualified path name includes the drive letter and any directory and subdirectories in addition to the file name and extension.

Remarks:

none

Support function: ExpandUNCFileName

Prototype:

```
function ExpandUNCFileName(const FileName: string): String;
```

Description:

Returns a string containing a fully qualified path name for the file passed in the FileName. A fully qualified path name includes the drive portion of the filename in the UNC format '\\<servername>\<sharename>' if the drive letter is mapped to a network resource instead of a local drive and any directory and subdirectories in addition to the file name and extension.

Remarks:

none

Support function: GetTimeString

Prototype:

```
function GetTimeString: String;
```

Description:
Returns the current time as a string.

Remarks:
none

Support function: GetDateString

Prototype:

```
function GetDateString: String;
```

Description:
Returns the current date as a string.

Remarks:
none

Support function: GetDateTimeString

Prototype:

```
function GetDateTimeString: String;
```

Description:
Returns the current date and time as a string.

Remarks:
none

Support function: SetLength

Prototype:
```
procedure SetLength(var S: String; L: Longint);
```

Description:
Sets the length of a string.

Remarks:
none

Support function: CharToOemBuff

Prototype:
```
procedure CharToOemBuff(var S: String);
```

Description:
Translates an ANSI string to a string with characters from the OEM-defined character set.

Remarks:
none

Support function: OemToCharBuff

Prototype:
```
procedure OemToCharBuff(var S: String);
```

Description:
Translates a string with characters from the OEM-defined character set into an ANSI string.

Remarks:
none

Support function: SysErrorMessage

Prototype:

```
function SysErrorMessage(ErrorCode: Integer): String;
```

Description:
Returns an error message string that corresponds to the given operating system error code.

Remarks:
none

Support function: GetArrayLength

Prototype:
```
function GetArrayLength(var Arr: Array): Longint;
```

Description:
Gets the length of an array.

Remarks:
none

Support function: SetArrayLength

Prototype:
```
procedure SetArrayLength(var Arr: Array; I: Longint);
```

Description:
Sets the length of an array. Always call SetArrayLength before accessing the elements in an array.

Remarks:
none

Support function: Low

Prototype:

```
function Low(var U: Array): Longint;
```

Description:
Returns the lowest value within the range of the index type of the array

Remarks:
none

Support function: High

Prototype:

```
function High(var U: Array): Longint;
```

Description:

Returns the highest value within the range of the index type of the array

Remarks:

none

Support function: DirExists

Prototype:

```
function DirExists(const Name: String): Boolean;
```

Description:

Returns True if the specified directory name exists. The specified name may include a trailing backslash.

Remarks:

none

Support function: FileExists

Prototype:

```
function FileExists(const Name: String): Boolean;
```

Description:
Returns True if the specified file exists.

Remarks:
none

Support function: FileOrDirExists

Prototype:

```
function FileOrDirExists(const Name: String): Boolean;
```

Description:
Returns True if the specified directory or file name exists. The specified name may include a trailing backslash.

Remarks:
none

Support function: FileSize

Prototype:

```
function FileSize(const Name: String; var Size: Integer): Boolean;
```

Description:

Sets Size to the size of the specified file in bytes. Returns True if the file size was set successfully and False otherwise.

Remarks:

none

Support function: DiskFree

Prototype:

```
function DiskFree(Drive: Char): Integer;
```

Description:
Returns the number of free bytes on the specified drive. DiskFree returns -1 if the drive is invalid.

Does not support large drives.

Remarks:
none

Support function: DiskSize

Prototype:

```
function DiskSize(Drive: Char): Integer;
```

Description:

Returns the size in bytes on the specified drive. DiskFree returns -1 if the drive is invalid.

Does not support large drives.

Remarks:

none

Support function: FileSearch

Prototype:

```
function FileSearch(const Name, DirList: string): String;
```

Description:

Searches through the directories passed in DirList for a file named Name. DirList should be directory names separated by semicolons. If FileSearch locates a file matching Name, it returns a string containing a fully-qualified path name for that file. If no matching file exists, FileSearch returns an empty string.

Remarks:

none

Support function: FindFirst

Prototype:

```
function FindFirst(const FileName: String): String;
```

Description:
Returns the first file found in the directory specified by FileName. FileName may include wildcards.

Remarks:
none

Support function: FindNext

Prototype:

```
function FindNext: String;
```

Description:
Returns the next file found in the directory after a call to FindFirst.

Remarks:
none

Support function: GetCurrentDir

Prototype:

```
function GetCurrentDir: String;
```

Description:
Returns a string containing the name of the current directory.

Remarks:
none

Support function: SetCurrentDir

Prototype:

```
function SetCurrentDir(const Dir: string): Boolean;
```

Description:
Sets the current directory. The return value is True if the current directory was successfully changed, or False if an error occurred.

Remarks:
none

Support function: GetWinDir

Prototype:

```
function GetWinDir: String;
```

Description:
Returns fully qualified path of the Windows directory. Only includes a trailing backslash if the Windows directory is the root directory.

Remarks:
none

Support function: GetSystemDir

Prototype:

```
function GetSystemDir: String;
```

Description:
Returns fully qualified path of the Windows System directory. Only includes a trailing backslash if the Windows System directory is the root directory.

Remarks:
none

Support function: GetTempDir

Prototype:

```
function GetTempDir: String;
```

Description:

Returns fully qualified path of the temporary directory, with trailing backslash. This does not use the Win32 function GetTempPath, due to platform differences.
Gets the temporary file path as follows:
1. The path specified by the TMP environment variable.
2. The path specified by the TEMP environment variable, if TMP is not defined or if TMP specifies a directory that does not exist.
3. The Windows directory, if both TMP and TEMP are not defined or specify nonexistent directories.

Remarks:

none

Support function: GetShellFolder

Prototype:

```
function GetShellFolder(Common: Boolean; const ID: TShellFolderID): String;
```

Description:

Gets the location of the specified shell folder. Returns the 'common' version of the shell folder location if Common is True and the user has administrative privileges.

Remarks:

none

Support function: GetShortName

Prototype:

```
function GetShortName(const LongName: String): String;
```

Description:
Returns the short version of the specified long filename. If the short version of the long filename is not found, the long filename is returned.

Remarks:
none

Support function: GenerateUniqueName

Prototype:

```
function GenerateUniqueName(Path: String; const Extension: String): String;
```

Description:
Generates a unique filename for a file in the specified path with the specified extension.

Remarks:
none

Support function: GetVersionNumbers

Prototype:

```
function GetVersionNumbers(const Filename: String; var VersionMS, VersionLS:
Cardinal): Boolean;
```

Description:

Gets the file version numbers of the specified file.

Remarks:

none

Support function: GetVersionNumbersString

Prototype:
```
function GetVersionNumbersString(const Filename: String; var Version: String): Boolean;
```

Description:
Gets the file version numbers of the specified file, as a string.

Remarks:
none

Support function: InstExec

Prototype:

```
function InstExec(const Filename, Params: String; WorkingDir: String; const WaitUntilTerminated, WaitUntilIdle: Boolean; const ShowCmd: Integer; var ResultCode: Integer): Boolean;
```

Description:

Executes the specified executable file. Use WaitUntilTerminated and WaitUntilIdle to specify whether InstExec should return immediately or should wait until the executed file is termimated or is idle. Returns True if the specified file was executed successfully, False otherwise. If True is returned and WaitUntilTerminated is True then ResultCode return the exit code of the executed file. If False is returned then ResultCode specifies the error that occurred. Use SysErrorMessage(ResultCode) to get a description of the error.

Remarks:

none

Support function: InstShellExec

Prototype:

```
function InstShellExec(const Filename, Params: String; WorkingDir: String;
const ShowCmd: Integer; var ErrorCode: Integer): Boolean;
```

Description:

Opens the specified file, for example an executable file or a document file. Returns True if the specified file was opened successfully, False otherwise. If False is returned then ErrorCode specifies the error that occurred. Use SysErrorMessage(ErrorCode) to get a description of the error.

Remarks:

none

Support function: RenameFile

Prototype:

```
function RenameFile(const OldName, NewName: string): Boolean;
```

Description:

Attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns True. If it cannot rename the file (for example, if a file called NewName already exists), it returns False.

Remarks:

none

Support function: ChangeFileExt

Prototype:

```
function ChangeFileExt(const FileName, Extension: string): String;
```

Description:

Takes the file name passed in FileName and changes the extension of the file name to the extension passed in Extension.

Remarks:

none

Support function: FileCopy

Prototype:

```
function FileCopy(const ExistingFile, NewFile: String; const FailIfExists:
Boolean): Boolean;
```

Description:

Copies ExistingFile to NewFile, preserving time stamp and file attributes.
If FailIfExists is True it will fail if NewFile already exists, otherwise it will overwrite it.
Returns True if successful; False if not.

Remarks:

none

Support function: DeleteFile

Prototype:

```
function DeleteFile(const FileName: string): Boolean;
```

Description:

Erases the file named by FileName from the disk.
If the file cannot be deleted or does not exist, the function returns False.

Remarks:

none

Support function: DelayDeleteFile

Prototype:

```
procedure DelayDeleteFile(const Filename: String; const Tries: Integer);
```

Description:

Attempts to delete Filename, retrying up to Tries times if the file is in use. It delays 250 msec between tries.

Remarks:

none

Support function: LoadStringFromFile

Prototype:

```
function LoadStringFromFile(const FileName: String; var S: String): Boolean;
```

Description:

Loads the specified binary or text file into the specified string. Returns True if successful, False otherwise.

Remarks:

none

Support function: LoadStringsFromFile

Prototype:
```
function LoadStringsFromFile(const FileName: String; var S: TArrayOfString): Boolean;
```

Description:
Loads the specified text file into the specified string array. Returns True if successful, False otherwise.

Remarks:
none

Support function: SaveStringToFile

Prototype:

```
function SaveStringToFile(const FileName, S: String; const Append: Boolean): Boolean;
```

Description:

Saves or appends the specified string to the specified file. Returns True if successful, False otherwise.

Remarks:

If Append is True and the specified file does not exist, a new file is created.

Support function: SaveStringsToFile

Prototype:

```
function SaveStringsToFile(const FileName: String; const S: TArrayOfString;
const Append: Boolean): Boolean;
```

Description:

Saves or appends the specified string array to the specified file. Returns True if successful, False otherwise.

Remarks:

If Append is True and the specified file does not exist, a new file is created.

Support function: CreateDir

Prototype:

```
function CreateDir(const Dir: string): Boolean;
```

Description:
Creates a new directory. The return value is True if a new directory was successfully created, or False if an error occurred.

Remarks:
none

Support function: ForceDirectories

Prototype:

```
procedure ForceDirectories(Dir: string);
```

Description:

Creates all the directories along the specified directory path all at once. If the first directories in the path do exist, but the latter ones don't, ForceDirectories creates just the ones that don't exist.

Remarks:

none

Support function: RemoveDir

Prototype:

```
function RemoveDir(const Dir: string): Boolean;
```

Description:
Deletes an existing empty directory. The return value is True if a new directory was successfully deleted, or False if an error occurred.

Remarks:
none

Support function: DelTree

Prototype:

```
function DelTree(const Path: String; const IsDir, DeleteFiles,
DeleteSubdirsAlso: Boolean): Boolean;
```

Description:

Deletes the specified directory including all files and subdirectories in it (including those with hidden, system, and read-only attributes). Returns True if it was able to successfully remove everything.

Remarks:

none

Support function: CreateShellLink

Prototype:

```
function CreateShellLink(const Filename, Description, ShortcutTo, Parameters,
WorkingDir, IconFilename: String; const IconIndex, ShowCmd: Integer): Boolean;
```

Description:

Creates a lnk file named Filename, with a description of Description, with a HotKey hotkey, which points to ShortcutTo.

Remarks:

none

Support function: RegisterServer

Prototype:
```
procedure RegisterServer(const Filename: String; const FailCriticalErrors:
Boolean);
```

Description:
Registers the OLE server (a.k.a. ActiveX control) with the specified filename. If FailCriticalErrors is True, the system is allowed to display error messages. Throws an exception if not successful.

Remarks:
none

Support function: UnregisterServer

Prototype:
```
function UnregisterServer(const Filename: String; const FailCriticalErrors:
Boolean): Boolean;
```

Description:
Unregisters the OLE server (a.k.a. ActiveX control) with the specified filename. If FailCriticalErrors is True, the system is allowed to display error messages. Returns True if successful, False otherwise.

Remarks:
none

Support function: RegisterTypeLibrary

Prototype:

```
procedure RegisterTypeLibrary(const Filename: String);
```

Description:
Registers the type library with the specified filename. Throws an exception if not successful.

Remarks:
none

Support function: UnregisterTypeLibrary

Prototype:
```
function UnregisterTypeLibrary(const Filename: String): Boolean
```

Description:
Unregisters the type library with the specified filename. Returns True if successful, False otherwise.

Remarks:
none

Support function: IncrementSharedCount

Prototype:
```
procedure IncrementSharedCount(const Filename: String; const AlreadyExisted:
Boolean);
```

Description:
Increments the shared count (a.k.a. the reference counter) of the specified file.

Remarks:
none

Support function: DecrementSharedCount

Prototype:

```
function DecrementSharedCount(const Filename: String): Boolean;
```

Description:
Decrements the shared count (a.k.a. the reference counter) of the specified file.

Remarks:
none

Support function: RestartReplace

Prototype:

```
procedure RestartReplace(const TempFile, DestFile: String);
```

Description:
Renames TempFile to DestFile the next time Windows is started. If DestFile already existed, it will be overwritten. If DestFile is '' then TempFile will be deleted.

Remarks:
none

Support function: UnregisterFont

Prototype:
```
procedure UnregisterFont(const FontName, FontFilename: String);
```

Description:
Unregisters the font with the specified face and filename.

Remarks:
none

Support function: ModifyPifFile

Prototype:
```
function ModifyPifFile(const Filename: String; const CloseOnExit: Boolean):
Boolean;
```

Description:
Changes the "Close on exit" setting of a .pif file. Returns True if it was able to make the change.

Remarks:
none

Support function: RegKeyExists

Prototype:

```
function RegKeyExists(const RootKey: Integer; const SubKeyName: String):
Boolean;
```

Description:
Returns True if the specified key exists.

Remarks:
none

Support function: RegValueExists

Prototype:
```
function RegValueExists(const RootKey: Integer; const SubKeyName, ValueName:
String): Boolean;
```

Description:
Returns True if the specified key and value exists.

Remarks:
none

Support function: RegGetSubkeyNames

Prototype:

```
function RegGetSubkeyNames(const RootKey: Integer; const SubKeyName: String;
var Names: TArrayOfString): Boolean;
```

Description:

Opens the specified registry key and reads the names of its subkeys into the array Names (which is first cleared). Returns True if successful.

Remarks:

none

Support function: RegQueryStringValue

Prototype:

```
function RegQueryStringValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;
```

Description:

Queries the specified REG_SZ or REG_EXPAND_SZ registry key/value, and returns the value in ResultStr. Returns True if successful. When False is returned, ResultStr is unmodified.

Remarks:

none

Support function: RegQueryMultiStringValue

Prototype:

```
function RegQueryMultiStringValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;
```

Description:

Queries the specified REG_MULTISZ registry key/value, and returns the value in ResultStr. Returns True if successful. When False is returned, ResultStr is unmodified.

Remarks:

none

Support function: RegQueryDWordValue

Prototype:

```
function RegQueryDWordValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultDWord: Cardinal): Boolean;
```

Description:

Queries the specified REG_DWORD registry key/value, and returns the value in ResultDWord. Returns True if successful. When False is returned, ResultDWord is unmodified.

Remarks:

none

Support function: RegQueryBinaryValue

Prototype:

```
function RegQueryBinaryValue(const RootKey: Integer; const SubKeyName,
ValueName: String; var ResultStr: String): Boolean;
```

Description:

Queries the specified REG_BINARY registry key/value, and returns the value in ResultStr. Returns True if successful. When False is returned, ResultStr is unmodified.

Remarks:

none

Support function: RegWriteStringValue

Prototype:

```
function RegWriteStringValue (const RootKey: Integer; const SubKeyName,
ValueName, Value: String): Boolean;
```

Description:

Writes the specified REG_SZ registry key/value. Returns True if successful, False otherwise.

Remarks:

none

Support function: RegWriteMultiStringValue

Prototype:
```
function RegWriteMultiStringValue (const RootKey: Integer; const SubKeyName,
ValueName, Value: String): Boolean;
```

Description:
Writes the specified REG_MULTISZ registry key/value. Returns True if successful, False otherwise.

Remarks:
none

Support function: RegWriteDWordValue

Prototype:

```
function RegWriteDWordValue (const RootKey: Integer; const SubKeyName,
ValueName: String; const Value: Cardinal): Boolean;
```

Description:

Writes the specified REG_DWORD registry key/value. Returns True if successful, False otherwise.

Remarks:

none

Support function: RegWriteBinaryValue

Prototype:

```
function RegWriteBinaryValue (const RootKey: Integer; const SubKeyName,
ValueName, Value: String): Boolean;
```

Description:

Writes the specified REG_BINARY registry key/value. Returns True if successful, False otherwise.

Remarks:

none

Support function: IniKeyExists

Prototype:

```
function IniKeyExists(const Section, Key, Filename: String): Boolean;
```

Description:
Returns True if the specified INI key exists.

Remarks:
none

Support function: IsIniSectionEmpty

Prototype:
```
function IsIniSectionEmpty(const Section, Filename: String): Boolean;
```

Description:
Returns True if the specified INI section is empty.

Remarks:
none

Support function: GetIniBool

Prototype:
```
function GetIniBool(const Section, Key: String; const Default: Boolean; const Filename: String): Boolean
```

Description:
Reads a Boolean from an INI file.

Remarks:
none

Support function: GetIniInt

Prototype:

```
function GetIniInt(const Section, Key: String; const Default, Min, Max: Longint; const Filename: String): Longint;
```

Description:
Reads a Longint from an INI file. If the Longint read is not between Min/Max then it returns Default. If Min=Max then Min/Max are ignored.

Remarks:
none

Support function: GetIniString

Prototype:
```
function GetIniString(const Section, Key, Default, Filename: String): String;
```

Description:
Reads a String from an INI file.

Remarks:
none

Support function: SetIniBool

Prototype:
```
function SetIniBool(const Section, Key: String; const Value: Boolean; const Filename: String): Boolean;
```

Description:
Writes a Boolean to an INI file.

Remarks:
none

Support function: SetIniInt

Prototype:
```
function SetIniInt(const Section, Key: String; const Value: Longint; const
Filename: String): Boolean;
```

Description:
Writes a Longint to an INI file.

Remarks:
none

Support function: SetIniString

Prototype:
```
function SetIniString(const Section, Key, Value, Filename: String): Boolean;
```

Description:
Writes a string to an INI file.

Remarks:
none

Support function: DeleteIniSection

Prototype:

```
procedure DeleteIniSection(const Section, Filename: String);
```

Description:
Deletes the specified section from an INI file.

Remarks:
none

Support function: DeleteIniEntry

Prototype:
```
procedure DeleteIniEntry(const Section, Key, Filename: String);
```

Description:
Deletes the specified key from an INI file.

Remarks:
none

Support function: ScriptDlgPageSetCaption

Prototype:

```
procedure ScriptDlgPageSetCaption(const Caption: String);
```

Description:

Sets the caption of the custom wizard page.

Remarks:

none

Support function: ScriptDlgPageSetSubCaption1

Prototype:

```
procedure ScriptDlgPageSetSubCaption1(const SubCaption1: String);
```

Description:

Sets the first subcaption of the custom wizard page. Ignored by the classic style wizard.

Remarks:

none

Support function: ScriptDlgPageSetSubCaption2

Prototype:

```
procedure ScriptDlgPageSetSubCaption2(const SubCaption2: String);
```

Description:

Sets the seconds subcaption of the custom wizard page.

Remarks:

none

Support function: ScriptDlgPageShowBackButton

Prototype:

```
procedure ScriptDlgPageShowBackButton(const Show: Boolean);
```

Description:

Hides or shows the Back button on the custom wizard page.

Remarks:

none

Support function: ScriptDlgPageOpen

Prototype:
```
procedure ScriptDlgPageOpen;
```

Description:
Opens the custom wizard page.

Remarks:
none

Support function: InputDir

Prototype:

```
function InputDir(const AppendDir: String; var Value: String): Boolean;
```

Description:

Displays a custom wizard page to select a directory.
When the AppendDir string is not empty, Setup will automatically append it to directories the user double-clicks on.

Remarks:

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputFile

Prototype:
```
function InputFile(const Prompt, Filter, DefaultExtension: string; var Value: string): Boolean;
```

Description:
Displays a custom wizard page to select a file.

Remarks:
An example Filter: 'Text files (*.txt)|*.txt|All files (*.*)|*.*'

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputFileArray

Prototype:

```
function InputFileArray(const Prompts, Filters, DefaultExtensions:
TArrayOfString; var Values: TArrayOfString): Boolean;
```

Description:

Displays a custom wizard page to select multiple files.

Remarks:

An example Filter: 'Text files (*.txt)|*.txt|All files (*.*)|*.*'

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputOptionArray

Prototype:

```
function InputOptionArray(const Prompts: TArrayOfString; var Values:
TArrayOfString; const Exclusive, ListBox: Boolean): Boolean;
```

Description:

Displays multiple checkboxes on a custom wizard page. If Exclusive is True, radiobuttons are displayed. If ListBox is True, the checkboxes or radiobuttons are placed inside a scrollable listbox. Returns True if the user clicked Next, False otherwise.

Remarks:

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputOption

Prototype:

```
function InputOption(const Prompt: String; var Value: String): Boolean;
```

Description:
Displays a checkbox on a custom wizard page. Returns True if the user clicked Next, False otherwise.

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputQuery

Prototype:

```
function InputQuery(const Prompt: String; var Value: String): Boolean;
```

Description:
Displays an input box on a custom wizard page. Returns True if the user clicked Next, False otherwise.

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputQueryArray

Prototype:

```
function InputQueryArray(const Prompts: TArrayOfString; var Values:
TArrayOfString): Boolean;
```

Description:

Displays multiple input boxes on a custom wizard page. Returns True if the user clicked Next, False otherwise.

Remarks:

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: InputQueryArrayEx

Prototype:

```
function InputQueryArrayEx(const Prompts: TArrayOfString; const PasswordChars:
TArrayOfChar; var Values: TArrayOfString): Boolean;
```

Description:
Displays multiple input boxes on a custom wizard page. Returns True if the user clicked Next, False otherwise.

Use PasswordChars to Indicate the character, if any, to display in place of the actual characters typed in the control.

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: OutputMsg

Prototype:

```
function OutputMsg(const Msg: String; const WaitUntilClick: Boolean): Boolean;
```

Description:

Displays a message on a custom wizard page. If WaitUntilClick is True: waits until the user clicks a button and returns True if the user clicked Next, False otherwise. If WaitUntilClick is False: hides the Next and Back buttons and returns immediately.

Remarks:

Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: OutputMsgMemo

Prototype:

```
function OutputMsgMemo(const Prompt, Msg: String): Boolean;
```

Description:
Displays a message in a memo on a custom wizard page. Supports Rich Text (RTF).

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: OutputProgress

Prototype:

```
procedure OutputProgress(const Msg1, Msg2: String; const Progress,
MaxProgress: Longint);
```

Description:
Displays a progress bar on a custom wizard page, hides the Next and Back buttons and returns immediately.

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: ScriptDlgPageProcessCustom

Prototype:

```
function ScriptDlgPageProcessCustom(): Boolean;
```

Description:
Displays a custom wizard page with custom VCL controls on it. Returns True if the user clicked Next, False otherwise.

Remarks:
Use Terminated() to check whether to user canceled Setup instead of clicking the Next or Back button.

Support function: ScriptDlgPageClearCustom

Prototype:

```
procedure ScriptDlgPageClearCustom();
```

Description:
not yet available

Remarks:
none

Support function: ScriptDlgPageClose

Prototype:

```
procedure ScriptDlgPageClose(const FullRestore: Boolean);
```

Description:
Closes the custom wizard page. Set FullRestore to True if the user selected to go back to the non custom wizard page that was displayed before the custom wizard page was openend. Set FullRestore to False otherwise.

If you're not sure what to set FullRestore to, set it to True.

Remarks:
none

Support function: MsgBox

Prototype:

```
function MsgBox(const Text: String; const Typ: TMsgBoxType; const Buttons:
Integer): Integer;
```

Description:
Displays a message box.

Remarks:
none

Support function: GetOpenFileName

Prototype:

```
function GetOpenFileName(const Prompt: String; var FileName: String; const
InitialDirectory, Filter, DefaultExtension: String): Boolean;
```

Description:

Displays a dialog box that enables the user to select a file. Returns True if the user selected a file, False otherwise. The name of the selected file is returned in the FileName string.

Remarks:

An example Filter: 'Text files (*.txt)|*.txt|All files (*.*)|*.*'

Support function: BrowseForFolder

Prototype:

```
function BrowseForFolder(const Prompt: String; var Directory: String):
Boolean;
```

Description:

Displays a dialog box that enables the user to select a directory using the current value of Directory as the initially selected directory. Returns True if the user selected a directory, False otherwise. The selected directory is returned in the Directory string.

Remarks:

none

Support function: ExitSetupMsgBox

Prototype:

```
function ExitSetupMsgBox: Boolean;
```

Description:
Displays the 'Exit Setup?' message box. Does not terminate Setup.

Remarks:
none

Support function: LoadDLL

Prototype:

```
function LoadDLL(const DLLName: String; var ErrorCode: Integer): Longint;
```

Description:

Loads the specified DLL. Returns the DLL handle if the DLL was loaded successfully, zero otherwise. If zero is returned then ErrorCode specifies the error that occurred. Use SysErrorMessage(ErrorCode) to get a description of the error.

Remarks:

none

Support function: CallDLLProc

Prototype:

```
function CallDLLProc(const DLLHandle: Longint; const ProcName: String; const
Param1, Param2: Longint; var Result: Longint): Boolean;
```

Description:

Calls the specified function in a DLL specified using the DLL handle returned by LoadDLL. Returns True is the procedure was called successfully, False otherwise.
The function must use the standard calling convention, accept two 4 byte integer parameters and return a 4 byte integer result.

Remarks:

none

Support function: FreeDLL

Prototype:

```
function FreeDLL(const DLLHandle: Longint): Boolean;
```

Description:
Unloads a DLL specified using the DLL handle returned by LoadDLL.

Remarks:
none

Support function: CastStringToInteger

Prototype:

```
function CastStringToInteger(var S: String): Longint;
```

Description:
Casts a string to an integer so that a string can be passed to a DLL using CallDllProc.

Remarks:
none

Support function: CastIntegerToString

Prototype:

```
function CastIntegerToString(const L: Longint): String;
```

Description:
Casts an integer to a string so that a string can be received from a DLL using CallDllProc.

Remarks:
none

Support function: Sleep

Prototype:

```
procedure Sleep(const Milliseconds: LongInt);
```

Description:
Suspends the execution of Setup for a specified interval.

Remarks:
none

Support function: Random

Prototype:

```
function Random(const Range: Integer): Integer;
```

Description:
Returns a random number within the range 0 <= X < Range.

Remarks:
none

Support function: Beep

Prototype:
```
procedure Beep;
```

Description:
Beeps.

Remarks:
none

Support function: BringToFrontAndRestore

Prototype:

```
procedure BringToFrontAndRestore;
```

Description:
Makes sure that Setup is visible and the foreground window.

Remarks:
none

Pascal Scripting: Support Classes Reference

Below is the list of support classes that can be used from within the Pascal script. There's also one support object available: `WizardForm` of type `TWizardForm`.

```
TObject = class
  constructor Create;
  procedure Free;
end;

TPersistent = class(TObject)
  procedure Assign(Source: TPersistent);
end;

TComponent = class(TPersistent)
  function FindComponent(AName: string): TComponent;
  constructor Create(AOwner: TComponent);

  property Owner: TComponent; read write;
  procedure DESTROYCOMPONENTS;
  procedure DESTROYING;
  procedure FREENOTIFICATION(ACOMPONENT:TCOMPONENT);
  procedure INSERTCOMPONENT(ACOMPONENT:TCOMPONENT);
  procedure REMOVECOMPONENT(ACOMPONENT:TCOMPONENT);
  property COMPONENTS: TCOMPONENT INTEGER; read;
  property COMPONENTCOUNT: INTEGER; read;
  property COMPONENTINDEX: INTEGER; read write;
  property COMPONENTSTATE: Byte; read;
  property DESIGNINFO: LONGINT; read write;
  property NAME: STRING; read write;
  property TAG: LONGINT; read write;
end;

TStrings = class(TPersistent)
  function Add(S: string): Integer;
  procedure Append(S: string);
  procedure AddStrings(Strings: TStrings);
  procedure Clear;
  procedure Delete(Index: Integer);
  function IndexOf(const S: string): Integer;
  procedure Insert(Index: Integer; S: string);
  property Count: Integer; read;
  property Text: String; read write;
  property CommaText: String; read write;
  procedure LoadFromFile(FileName: string);
  procedure SaveToFile(FileName: string);
  property Strings: String Integer; read write;
  property Objects: TObject Integer; read write;
end;

TStringList = class(TStrings)
  function FIND(S:STRING;var INDEX:INTEGER):BOOLEAN;
  procedure SORT;
  property DUPLICATES: TDUPLICATES; read write;
  property SORTED: BOOLEAN; read write;
  property ONCHANGE: TNOTIFYEVENT; read write;
  property ONCHANGING: TNOTIFYEVENT; read write;
end;

TGraphicsObject = class(TPersistent)
```

```
  property ONCHANGE: TNOTIFYEVENT; read write;
end;

TFont = class(TGraphicsObject)
  constructor Create;
  property Handle: Integer; read;
  property Color: Integer; read write;
  property Height: Integer; read write;
  property Name: string; read write;
  property Pitch: Byte; read write;
  property Size: Integer; read write;
  property PixelsPerInch: Integer; read write;
  property Style: TFontStyles; read write;
end;

TCanvas = class(TPersistent)
  procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
  procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
  procedure Draw(X, Y: Integer; Graphic: TGraphic);
  procedure Ellipse(X1, Y1, X2, Y2: Integer);
  procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: Byte);
  procedure LineTo(X, Y: Integer);
  procedure MoveTo(X, Y: Integer);
  procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
  procedure Rectangle(X1, Y1, X2, Y2: Integer);
  procedure Refresh;
  procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
  function TextHeight(Text: string): Integer;
  procedure TextOut(X, Y: Integer; Text: string);
  function TextWidth(Text: string): Integer;
  property Handle: Integer; read write;
  property Pixels: Integer Integer Integer; read write;
  property Brush: TBrush; read;
  property CopyMode: Byte; read write;
  property Font: TFont; read;
  property Pen: TPen; read;
end;

TPen = class(TGraphicsObject)
  constructor CREATE;
  property COLOR: TCOLOR; read write;
  property MODE: TPENMODE; read write;
  property STYLE: TPENSTYLE; read write;
  property WIDTH: INTEGER; read write;
end;

TBrush = class(TGraphicsObject)
  constructor CREATE;
  property COLOR: TCOLOR; read write;
  property STYLE: TBRUSHSTYLE; read write;
end;

TComponent = class(TControl)
  constructor Create(AOwner: TComponent);
  procedure BringToFront;
  procedure Hide;
```

```
  procedure Invalidate;
  procedure refresh;
  procedure Repaint;
  procedure SendToBack;
  procedure Show;
  procedure Update;
  procedure SetBounds(x,y,w,h: Integer);
  property Left: Integer; read write;
  property Top: Integer; read write;
  property Width: Integer; read write;
  property Height: Integer; read write;
  property Hint: String; read write;
  property Align: TAlign; read write;
  property ClientHeight: Longint; read write;
  property ClientWidth: Longint; read write;
  property ShowHint: Boolean; read write;
  property Visible: Boolean; read write;
  property ENABLED: BOOLEAN; read write;
  property HINT: STRING; read write;
end;

TWinControl = class(TControl)
  property Parent: TWinControl; read write;
  property Handle: Longint; read write;
  property Showing: Boolean; read;
  property TabOrder: Integer; read write;
  property TabStop: Boolean; read write;
  function CANFOCUS:BOOLEAN;
  function FOCUSED:BOOLEAN;
  property CONTROLS: TCONTROL INTEGER; read;
  property CONTROLCOUNT: INTEGER; read;
end;

TGraphicControl = class(TControl)
end;

TCustomControl = class(TWinControl)
end;

TScrollingWinControl = class(TWinControl)
  procedure SCROLLINVIEW(ACONTROL:TCONTROL);
  property HORZSCROLLBAR: TCONTROLSCROLLBAR; read write;
  property VERTSCROLLBAR: TCONTROLSCROLLBAR; read write;
end;

TForm = class(TScrollingWinControl)
  constructor CREATENEW(AOWNER:TCOMPONENT; Dummy: Longint);
  procedure CLOSE;
  procedure HIDE;
  procedure SHOW;
  function SHOWMODAL:INTEGER;
  procedure RELEASE;
  property ACTIVE: BOOLEAN; read;
  property ACTIVECONTROL: TWINCONTROL; read write;
  property BORDERICONS: Longint; read write;
  property BORDERSTYLE: TFORMBORDERSTYLE; read write;
```

```
  property CAPTION: STRING; read write;
  property AUTOSCROLL: BOOLEAN; read write;
  property COLOR: TCOLOR; read write;
  property FONT: TFONT; read write;
  property FORMSTYLE: TFORMSTYLE; read write;
  property KEYPREVIEW: BOOLEAN; read write;
  property POSITION: TPOSITION; read write;
  property ONACTIVATE: TNOTIFYEVENT; read write;
  property ONCLICK: TNOTIFYEVENT; read write;
  property ONDBLCLICK: TNOTIFYEVENT; read write;
  property ONCLOSE: TCLOSEEVENT; read write;
  property ONCLOSEQUERY: TCLOSEQUERYEVENT; read write;
  property ONCREATE: TNOTIFYEVENT; read write;
  property ONDESTROY: TNOTIFYEVENT; read write;
  property ONDEACTIVATE: TNOTIFYEVENT; read write;
  property ONHIDE: TNOTIFYEVENT; read write;
  property ONKEYDOWN: TKEYEVENT; read write;
  property ONKEYPRESS: TKEYPRESSEVENT; read write;
  property ONKEYUP: TKEYEVENT; read write;
  property ONRESIZE: TNOTIFYEVENT; read write;
  property ONSHOW: TNOTIFYEVENT; read write;
end;

TCustomLabel = class(TGraphicControl)
end;

TLabel = class(TCustomLabel)
  property ALIGNMENT: TAlignment; read write;
  property AUTOSIZE: Boolean; read write;
  property CAPTION: String; read write;
  property COLOR: Longint; read write;
  property FOCUSCONTROL: TWinControl; read write;
  property FONT: TFont; read write;
  property WORDWRAP: Boolean; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
end;

TCustomEdit = class(TWinControl)
  procedure CLEAR;
  procedure CLEARSELECTION;
  procedure SELECTALL;
  property MODIFIED: BOOLEAN; read write;
  property SELLENGTH: INTEGER; read write;
  property SELSTART: INTEGER; read write;
  property SELTEXT: STRING; read write;
  property TEXT: string; read write;
end;

TEdit = class(TCustomEdit)
  property AUTOSELECT: Boolean; read write;
  property AUTOSIZE: Boolean; read write;
  property BORDERSTYLE: BorderStyle; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property HIDESELECTION: Boolean; read write;
```

```
  property MAXLENGTH: Integer; read write;
  property PASSWORDCHAR: Char; read write;
  property READONLY: Boolean; read write;
  property TEXT: string; read write;
  property ONCHANGE: TNotifyEvent; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;
end;

TCustomMemo = class(TCustomEdit)
  property LINES: TSTRINGS; read write;
end;

TMemo = class(TMemo)
  property LINES: TSTRINGS; read write;
  property ALIGNMENT: TAlignment; read write;
  property BORDERSTYLE: BorderStyle; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property HIDESELECTION: Boolean; read write;
  property MAXLENGTH: Integer; read write;
  property READONLY: Boolean; read write;
  property SCROLLBARS: TScrollStyle; read write;
  property WANTRETURNS: Boolean; read write;
  property WANTTABS: Boolean; read write;
  property WORDWRAP: Boolean; read write;
  property ONCHANGE: TNotifyEvent; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;
end;

TCustomComboBox = class(TWinControl)
  property DROPPEDDOWN: BOOLEAN; read write;
  property ITEMS: TSTRINGS; read write;
  property ITEMINDEX: INTEGER; read write;
end;

TComboBox = class(TCustomComboBox)
  property STYLE: TComboBoxStyle; read write;
  property COLOR: Longint; read write;
  property DROPDOWNCOUNT: Integer; read write;
  property FONT: TFont; read write;
  property MAXLENGTH: Integer; read write;
  property SORTED: Boolean; read write;
  property TEXT: string; read write;
  property ONCHANGE: TNotifyEvent; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
```

```
  property ONKEYUP: TKeyEvent; read write;
end;

TButtonControl = class(TWinControl)
end;

TButton = class(TButtonControl)
  property CANCEL: BOOLEAN; read write;
  property CAPTION: String; read write;
  property DEFAULT: BOOLEAN; read write;
  property FONT: TFont; read write;
  property MODALRESULT: LONGINT; read write;
  property ONCLICK: TNotifyEvent; read write;
end;

TCustomCheckBox = class(TButtonControl)
end;

TCheckBox = class(TCustomCheckBox)
  property ALIGNMENT: TAlignment; read write;
  property ALLOWGRAYED: Boolean; read write;
  property CAPTION: String; read write;
  property CHECKED: Boolean; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property STATE: TCheckBoxState; read write;
  property ONCLICK: TNotifyEvent; read write;
end;

TRadioButton = class(TButtonControl)
  property ALIGNMENT: TALIGNMENT; read write;
  property CAPTION: String; read write;
  property CHECKED: BOOLEAN; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
end;

TCustomListBox = class(TWinControl)
  property ITEMS: TSTRINGS; read write;
  property ITEMINDEX: INTEGER; read write;
  property SELCOUNT: INTEGER; read;
  property SELECTED: BOOLEAN INTEGER; read write;
end;

TListBox = class(TCustomListBox)
  property BORDERSTYLE: TBorderStyle; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property MULTISELECT: Boolean; read write;
  property SORTED: Boolean; read write;
  property STYLE: TListBoxStyle; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
```

```
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;
end;

TPaintBox = class(TPaintBox)
  property CANVAS: TCANVAS; read;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONPAINT: TNotifyEvent; read write;
end;

TBevel = class(TGraphicControl)
  property SHAPE: TBEVELSHAPE; read write;
  property STYLE: TBEVELSTYLE; read write;
end;

TCustomPanel = class(TCustomControl)
end;

TPanel = class(TCustomPanel)
  property ALIGNMENT: TAlignment; read write;
  property BEVELINNER: TPanelBevel; read write;
  property BEVELOUTER: TPanelBevel; read write;
  property BEVELWIDTH: TBevelWidth; read write;
  property BORDERWIDTH: TBorderWidth; read write;
  property BORDERSTYLE: TBorderStyle; read write;
  property CAPTION: String; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
end;

TPage = class(TCustomControl)
  property CAPTION: String; read write;
end;

TNotebook = class(TCustomControl)
  property ACTIVEPAGE: STRING; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property PAGEINDEX: INTEGER; read write;
  property PAGES: TSTRINGS; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONPAGECHANGED: TNOTIFYEVENT; read write;
end;

TNewStaticText = class(TWinControl)
  property AUTOSIZE: BOOLEAN; read write;
  property CAPTION: String; read write;
  property COLOR: Longint; read write;
```

```
  property FOCUSCONTROL: TWinControl; read write;
  property FONT: TFont; read write;
  property SHOWACCELCHAR: Boolean; read write;
  property WORDWRAP: Boolean; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;

  property DRAGCURSOR: Longint; read write;
  property DRAGMODE: TDragMode; read write;
  property PARENTCOLOR: Boolean; read write;
  property PARENTFONT: Boolean; read write;
  property PARENTSHOWHINT: Boolean; read write;
  property POPUPMENU: TPopupMenu; read write;
  property ONDRAGDROP: TDragDropEvent; read write;
  property ONDRAGOVER: TDragOverEvent; read write;
  property ONENDDRAG: TEndDragEvent; read write;
  property ONMOUSEDOWN: TMouseEvent; read write;
  property ONMOUSEMOVE: TMouseMoveEvent; read write;
  property ONMOUSEUP: TMouseEvent; read write;
  property ONSTARTDRAG: TStartDragEvent; read write;
end;

TNewCheckListBox = class(TCustomListBox)
  function
ADDCHECKBOX(ACAPTION,ASUBITEM:STRING;ALEVEL:BYTE;ACHECKED,AENABLED,AHASINTE
RNALCHILDREN:BOOLEAN;AOBJECT:TOBJECT):INTEGER;
  function
ADDGROUP(ACAPTION,ASUBITEM:STRING;ALEVEL:BYTE;AOBJECT:TOBJECT):INTEGER;
  function
ADDRADIOBUTTON(ACAPTION,ASUBITEM:STRING;ALEVEL,AGROUP:BYTE;ACHECKED,AENABLE
D:BOOLEAN;AOBJECT:TOBJECT):INTEGER;
  property CHECKED: BOOLEAN INTEGER; read write;
  property STATE: TCHECKBOXSTATE INTEGER; read write;
  property ITEMENABLED: BOOLEAN INTEGER; read write;
  property ITEMOBJECT: TOBJECT INTEGER; read write;
  property ITEMSUBITEM: STRING INTEGER; read write;
  property ALLOWGRAYED: BOOLEAN; read write;
  property FLAT: BOOLEAN; read write;
  property MINITEMHEIGHT: INTEGER; read write;
  property OFFSET: INTEGER; read write;
  property MULTISELECT: BOOLEAN; read write;
  property ONCLICKCHECK: TNOTIFYEVENT; read write;
  property BORDERSTYLE: TBORDERSTYLE; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property SORTED: Boolean; read write;
  property STYLE: TListBoxStyle; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;
  property SHOWLINES: BOOLEAN; read write;
  property WANTTABS: BOOLEAN; read write;

  property COLUMNS: Integer; read;
```

```
  property CTL3D: Boolean; read write;
  property DRAGCURSOR: Longint; read write;
  property DRAGMODE: TDragMode; read write;
  property EXTENDEDSELECT: Boolean; read write;
  property INTEGRALHEIGHT: Boolean; read write;
  property PARENTCOLOR: Boolean; read write;
  property PARENTCTL3D: Boolean; read write;
  property PARENTFONT: Boolean; read write;
  property PARENTSHOWHINT: Boolean; read write;
  property POPUPMENU: TPopupMenu; read write;
  property TABWIDTH: Integer; read write;
  property ONDRAGDROP: TDragDropEvent; read write;
  property ONDRAGOVER: TDragOverEvent; read write;
  property ONDRAWITEM: TDrawItemEvent; read write;
  property ONENDDRAG: TEndDragEvent; read write;
  property ONENTER: TNotifyEvent; read write;
  property ONEXIT: TNotifyEvent; read write;
  property ONMEASUREITEM: TMeasureItemEvent; read write;
  property ONMOUSEDOWN: TMouseEvent; read write;
  property ONMOUSEMOVE: TMouseMoveEvent; read write;
  property ONMOUSEUP: TMouseEvent; read write;
  property ONSTARTDRAG: TStartDragEvent; read write;
end;

TNewDirectoryListBox = class(TCustomListBox)
  property DRIVE: CHAR; read write;
  property DIRECTORY: STRING; read write;
  property COLOR: Longint; read write;
  property DIRLABEL: TLABEL; read write;
  property FONT: TFont; read write;
  property ONCHANGE: TNOTIFYEVENT; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;

  property COLUMNS: Integer; read write;
  property CTL3D: Boolean; read write;
  property DRAGCURSOR: Longint; read write;
  property DRAGMODE: TDragMode; read write;
  property INTEGRALHEIGHT: Boolean; read write;
  property PARENTCOLOR: Boolean; read write;
  property PARENTCTL3D: Boolean; read write;
  property PARENTFONT: Boolean; read write;
  property PARENTSHOWHINT: Boolean; read write;
  property POPUPMENU: TPopupMenu; read write;
  property ONDRAGDROP: TDragDropEvent; read write;
  property ONDRAGOVER: TDragOverEvent; read write;
  property ONENDDRAG: TEndDragEvent; read write;
  property ONENTER: TNotifyEvent; read write;
  property ONEXIT: TNotifyEvent; read write;
  property ONMOUSEDOWN: TMouseEvent; read write;
  property ONMOUSEMOVE: TMouseMoveEvent; read write;
  property ONMOUSEUP: TMouseEvent; read write;
  property ONSTARTDRAG: TStartDragEvent; read write;
```

```
end;

TNewDriveComboBox = class(TCustomComboBox);
  property DRIVE: CHAR; read write;
  property AUTOREFRESH: BOOLEAN; read write;
  property COLOR: Longint; read write;
  property DIRLIST: TNEWDIRECTORYLISTBOX; read write;
  property FONT: TFont; read write;
  property ONCHANGE: TNotifyEvent; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;
  property ONKEYDOWN: TKeyEvent; read write;
  property ONKEYPRESS: TKeyPressEvent; read write;
  property ONKEYUP: TKeyEvent; read write;

  property CTL3D: Boolean; read write;
  property DRAGMODE: TDragMode; read write;
  property DRAGCURSOR: Longint; read write;
  property PARENTCOLOR: Boolean; read write;
  property PARENTCTL3D: Boolean; read write;
  property PARENTFONT: Boolean; read write;
  property PARENTSHOWHINT: Boolean; read write;
  property POPUPMENU: TPopupMenu; read write;
  property TEXTCASE: TNEWTEXTCASE; read write;
  property ONDRAGDROP: TDragDropEvent; read write;
  property ONDRAGOVER: TDragOverEvent; read write;
  property ONDROPDOWN: TNotifyEvent; read write;
  property ONENDDRAG: TEndDragEvent; read write;
  property ONENTER: TNotifyEvent; read write;
  property ONEXIT: TNotifyEvent; read write;
  property ONSTARTDRAG: TStartDragEvent; read write;
end;

TNewPathLabel = class(TCustomLabel)
  property ALIGNMENT: TAlignment; read write;
  property CAPTION: String; read write;
  property COLOR: Longint; read write;
  property FONT: TFont; read write;
  property ONCLICK: TNotifyEvent; read write;
  property ONDBLCLICK: TNotifyEvent; read write;

  property DRAGCURSOR: Longint; read write;
  property DRAGMODE: TDragMode; read write;
  property FOCUSCONTROL: TWinControl; read write;
  property PARENTCOLOR: Boolean; read write;
  property PARENTFONT: Boolean; read write;
  property PARENTSHOWHINT: Boolean; read write;
  property POPUPMENU: TPopupMenu; read write;
  property TRANSPARENT: Boolean; read write;
  property ONDRAGDROP: TDragDropEvent; read write;
  property ONDRAGOVER: TDragOverEvent; read write;
  property ONENDDRAG: TEndDragEvent; read write;
  property ONMOUSEDOWN: TMouseEvent; read write;
  property ONMOUSEMOVE: TMouseMoveEvent; read write;
  property ONMOUSEUP: TMouseEvent; read write;
  property ONSTARTDRAG: TStartDragEvent; read write;
```

```
end;

TNewProgressBar = class(TWinControl)
  property MIN: LONGINT; read write;
  property MAX: LONGINT; read write;
  property POSITION: LONGINT; read write;
end;

TRichEditViewer = class(TMemo)
  property RTFTEXT: STRING'; write;
  property USERICHEDIT: BOOLEAN; read write;
end;

TSetupChildForm = class(TForm)
end;

TWizardForm = class(TSetupChildForm)
  property CANCELBUTTON: TBUTTON; read;
  property NEXTBUTTON: TBUTTON; read;
  property BACKBUTTON: TBUTTON; read;
  property NOTEBOOK1: TNOTEBOOK; read;
  property NOTEBOOK2: TNOTEBOOK; read;
  property DISKSPACELABEL: TNewStaticText; read;
  property DIREDIT: TEDIT; read;
  property DIRLIST: TNEWDIRECTORYLISTBOX; read;
  property DRIVELIST: TNEWDRIVECOMBOBOX; read;
  property GROUPEDIT: TEDIT; read;
  property NOICONSCHECK: TCHECKBOX; read;
  property GROUPLIST: TLISTBOX; read;
  property PASSWORDLABEL: TNewStaticText; read;
  property PASSWORDEDIT: TEDIT; read;
  property PASSWORDEDITLABEL: TNewStaticText; read;
  property READYMEMO: TMEMO; read;
  property TYPESCOMBO: TCOMBOBOX; read;
  property BEVEL: TBEVEL; read;
  property PICTUREPAINTBOX: TPAINTBOX; read;
  property WELCOMELABEL1: TNewStaticText; read;
  property INFOBEFOREMEMO: TRICHEDITVIEWER; read;
  property INFOBEFORECLICKLABEL: TNewStaticText; read;
  property MAINPANEL: TPANEL; read;
  property BEVEL1: TBEVEL; read;
  property PAGENAMELABEL: TNewStaticText; read;
  property PAGEDESCRIPTIONLABEL: TNewStaticText; read;
  property SMALLPICTUREPAINTBOX: TPAINTBOX; read;
  property READYLABEL: TNewStaticText; read;
  property FINISHEDLABEL: TNewStaticText; read;
  property YESRADIO: TRADIOBUTTON; read;
  property NORADIO: TRADIOBUTTON; read;
  property PICTUREPAINTBOX2: TPAINTBOX; read;
  property WELCOMELABEL2: TNewStaticText; read;
  property LICENSELABEL1: TNewStaticText; read;
  property LICENSEMEMO: TRICHEDITVIEWER; read;
  property INFOAFTERMEMO: TRICHEDITVIEWER; read;
  property INFOAFTERCLICKLABEL: TNewStaticText; read;
  property COMPONENTSLIST: TNEWCHECKLISTBOX; read;
  property COMPONENTSDISKSPACELABEL: TNewStaticText; read;
```

```
  property BEVELEDLABEL: TNewStaticText; read;
  property STATUSLABEL: TNewStaticText; read;
  property FILENAMELABEL: TNEWPATHLABEL; read;
  property PROGRESSGAUGE: TNEWPROGRESSBAR; read;
  property SELECTDIRLABEL: TNewStaticText; read;
  property SELECTSTARTMENUFOLDERLABEL: TNewStaticText; read;
  property SELECTCOMPONENTSLABEL: TNewStaticText; read;
  property SELECTTASKSLABEL: TNewStaticText; read;
  property LICENSEACCEPTEDRADIO: TRADIOBUTTON; read;
  property LICENSENOTACCEPTEDRADIO: TRADIOBUTTON; read;
  property USERINFONAMELABEL: TNewStaticText; read;
  property USERINFONAMEEDIT: TEDIT; read;
  property USERINFOORGLABEL: TNewStaticText; read;
  property USERINFOORGEDIT: TEDIT; read;
  property PREPARINGERRORPAINTBOX: TPAINTBOX; read;
  property PREPARINGLABEL: TNewStaticText; read;
  property FINISHEDHEADINGLABEL: TNewStaticText; read;
  property SCRIPTDLGPANEL: TPANEL; read;
  property USERINFOSERIALLABEL: TNewStaticText; read;
  property USERINFOSERIALEDIT: TEDIT; read;
  property TASKSLIST: TNEWCHECKLISTBOX; read;
  property RUNLIST: TNEWCHECKLISTBOX; read;
  property OLDTEXTHEIGHT: INTEGER; read;
  property NEWTEXTHEIGHT: INTEGER; read;
  property CURPAGE: TWIZARDPAGE; read;
  function ADJUSTLABELHEIGHT(ALABEL:TNewStaticText):INTEGER;
  procedure INCTOPDECHEIGHT(ACONTROL:TCONTROL;AMOUNT:INTEGER);
end;
```