Inno Setup

- **What is Inno Setup?**
- **How to Use**

- **Other Information**

What is Inno Setup?

Inno Setup version 2.0.14

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. Support for Windows NT 3.51 can also be optionally included.
- Full source code is available (Borland Delphi 2.0-5.0).
- Supports creation of a single EXE to install your program for easy online distribution. Disk spanning is also supported.
- Standard wizard interface, including support for the latest Windows 2000/XP wizard style.
- Customizable setup types, e.g. Full, Minimal, Custom.
- Complete uninstall capabilities.
- Copying of files:
  Includes integrated "deflate" file compression (the same compression .zip files use). The installer has the ability to compare file version info, replace in-use files, use shared file counting, and register DLL/OCX's and type libraries.
- Creation of shortcuts anywhere, including in the Start Menu and on the desktop.
- Creation of registry and .INI entries.
- Silent install and uninstall.
- **Y2K compliance.**
- Proven track record. Inno Setup has been around for over 3 years, and currently has many thousands of users worldwide.

Is it really free, even for commercial use?

Yes, it is completely free, even for commercial use. (There are restrictions on modifying and redistributing the source code, however; see the LICENSE.TXT file included in the source code package for details.)

Why was it created?

I was writing an application which came in two versions, one for 16-bit Windows compiled in Delphi 1 and the other for 32-bit Windows compiled in Delphi 2, and soon realized that I had no decent installation program for it. Delphi 1 came with no installation builder whatosever, while Delphi 2 came with the bulky InstallShield Express but it only supported Win32. For my very basic needs, buying an installer just for this one program seemed like overkill. I searched the Internet for free Windows installers, but amazingly very few turned up, and those I did find did not meet my needs or had poorly designed user interfaces (which really defeats the purpose of having an install program in the first place). I had written an installer for a DOS program a year before using good old Turbo Pascal & Turbo Vision (not nearly as sophisticated though), so I had limited experience with this sort of thing. So I figured I'd at least attempt to write a simple Windows installer myself before purchasing a commercial one.

Why is it free?

I had this fairly featureful installer written, and I realized there were plenty of other people with exactly the same predicament I had. In its early stages, it didn't offer enough to compete with other shareware and commercial installers available at the time, so I decided, "Why not just give it away for free?" No use in keeping it to myself. So I wrote documentation to go along with it and created a tiny web site (page) for it. I didn't have anything secretive in the source code, so I went ahead and posted it too. And from there Inno Setup was born.

$Id: isetup.rtf,v 1.71 2001/09/14 21:54:16 jr Exp $

Documentation Conventions

"Windows 95/NT 4+"	This is shorthand for "Windows 95, 98, 2000, XP, NT 4.0, Me, and later."
"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*. (Do not be afraid of the term "script"; it only takes a few minutes to learn the basics!)

Scripts have an ".iss" (meaning Inno Setup Script) extension. The script controls every aspect of the installation. It specifies which files are to be copied 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 an 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
Creating the Script

Creating the Script

An Inno Setup Script is a simple ASCII text file. Its format is very similar to .INI files, so if you have experience with those you should find this easy to learn.

The file is arranged in sections and entries as shown in the example below. Section names are always enclosed in brackets "`[]`", and in the case of the `[Setup]` section, you always put an equal sign "`=`" between directive names and their values. Here is an illustration of how the file is organized:

```
[Section name goes here, i.e. Setup]
This is a directive=this is its value
This is another directive=and this is another value

[Section name goes here, i.e. Files]
Source: "MYPROG.EXE"; DestDir: "{app}"
```

You can also put "comments" in your file (which are ignored by the Setup Compiler itself) by placing a semicolon at the beginning of a line. For example:

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

See also
Parameters in Sections
Constants
[Setup] section
[Types] section
[Components] section
[Tasks] section
[Dirs] section
[Files] section
[Icons] section
[INI] section
[InstallDelete] section
[Messages] section
[LangOptions] section
[Registry] section
[Run] section
[UninstallDelete] section
[UninstallRun] section

Parameters in Sections

All of the sections in a script, with the exception of `[Setup]`, 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 has a name, followed by a colon, and then its 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 a semicolons (and optionally a space also), and can be listed in any order.

The value of a parameter is typically surrounded in quotes (`"`), but using quotes is optional. However by using quotes you can use leading and trailing spaces in a value, as well as use embedded semicolons and quote characters. To embed a quote, use two consecutive quote characeters. For example, the Setup Compiler would see the following lines

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

""""
```

as

```
This " contains " embedded " quotes

"
```

Constants

The majority of the script entries can have *constants* embedded in them. These are always 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, 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 correctly 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 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 use "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 use "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. Windows 95/NT 4+ have a dedicated directory for fonts (normally named "FONTS" under the Windows directory); this constant points to that directory. On Windows NT 3.51, this constant is equivalent to `{sys}`, since it has no fonts directory.

{dao}
DAO directory. When the installation is run on Windows 95/NT 4+, this is equivalent to `{cf}\Microsoft Shared\DAO`. When run on Windows NT 3.51, this is equivalent to `{win}\MSAPPS\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. However, if support for NT 3.51 enabled by placing a `MinVersion=4,3.51` line in the script's `[Setup]` section, only the `{group}` shell folder constant can be used, and only in the `[Icons]` section.

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.

* = 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.

{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*}
Embeds an environment variable, where *NAME* is the name of the variable to use. If the specified variable does not exist on the user's system, the constant will be replaced with an empty string.

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

{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}
```

{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".

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

Common Parameters

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

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.

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.

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 its Tasks parameter (see below) says it shouldn't.

A Components parameter is ignored if the [Components] section is empty.

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 its Components parameter (see above) says it shouldn't.

A Tasks parameter is ignored if the [Tasks] section is empty.

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

[Setup] section

This section contains global settings used by the installer and uninstaller, and 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

- CompressLevel
- DiskClusterSize
- DiskSize
- DiskSpanning
- DontMergeDuplicateFiles
- OutputBaseFilename
- OutputDir
- ReserveBytes
- SourceDir
- UseSetupLdr

Installer-related

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

- AdminPrivilegesRequired
- AllowNoIcons
- AllowRootDirectory
- AlwaysCreateUninstallIcon
- AlwaysRestart
- AlwaysShowComponentsList
- AlwaysShowDirOnReadyPage
- AlwaysShowGroupOnReadyPage
- AlwaysUsePersonalGroup
- **AppName**
- AppId
- AppMutex
- AppPublisher
- AppPublisherURL
- AppSupportURL
- AppUpdatesURL
- AppVersion
- **AppVerName**
- ChangesAssociations
- CreateAppDir
- CreateUninstallRegKey
- **DefaultDirName**

- DefaultGroupName
- DirExistsWarning
- DisableAppendDir
- DisableDirPage
- DisableFinishedPage
- DisableProgramGroupPage
- DisableReadyMemo
- DisableReadyPage
- DisableStartupPrompt
- EnableDirDoesntExistWarning
- ExtraDiskSpaceRequired
- InfoAfterFile
- InfoBeforeFile
- LicenseFile
- MessagesFile
- MinVersion
- OnlyBelowVersion
- Password
- Uninstallable
- UninstallDisplayIcon
- UninstallDisplayName
- UninstallFilesDir
- UninstallIconName
- UninstallLogMode
- UpdateUninstallLogAppName
- UsePreviousAppDir
- UsePreviousGroup
- UsePreviousSetupType
- UsePreviousTasks

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

- AppCopyright
- BackColor
- BackColor2
- BackColorDirection
- BackSolid
- FlatComponentsList
- ShowComponentSizes
- UninstallStyle
- WindowShowCaption
- WindowStartMaximized
- WindowResizable
- WindowVisible
- WizardImageBackColor
- WizardImageFile
- WizardSmallImageFile
- WizardStyle

Obsolete

- Bits
- DisableDirExistsWarning
- OverwriteUninstRegEntries

[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. 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.

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

```
[Components]
Name: "main"; Description: "Main Files"; Types: full compact custom; Flags:
fixed
Name: "help"; Description: "Help Files"; Types: full
```

The example above generates two components: A "main" component which gets installed if the end user selects a type with name "full", "compact" or "custom" and a "help" component which only gets installed if the end user selects the "full" type. Since the "custom" type is customizable the "help" component will also be installed if the user wants to, even though "custom" isn't in the types list of the "help" component.

The following is a list of the supported parameters:

Name *(Required)*

Description:
The internal name of the component.

Example:
`Name: "help"`

Description *(Required)*

Description:
The description of the component. This description is shown to the end user during installation.

Example:
`Name: "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:

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.

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

```
[Tasks]
Name: desktopicon; Description: "Create a &desktop icon"; GroupDescription:
"Additional icons:"; Components: main
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.

Example:
`Name: "desktopicon"`

Description *(Required)*

Description:
The description of the task. 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. 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 other tasks that have the same (possibly empty) group description and an exclusive flag. This means Setup will show radio buttons instead of check boxes.

You can't have only one exclusive task within a group. Setup will automatically turn it into a non exclusive task (a checkbox instead of a radiobutton).

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 section is optional, and is not usually needed for most simple installations. This section is used to create additional directories *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. You aren't required to create directories first before copying files to them in 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 section is optional, but is necessary for most installations. It defines all of the files Setup will copy to the user's system.

During Setup, the uninstaller program and data is automatically copied to the application directory, so you do not need to manually add it to the [Files] section.

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

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

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

Default:
normal

Description:
Specifies the copy rule for the file. This must be one of the following:

normal
The most common option. If the file already exists on the user's system and is the same version as the file being copied (determined by the file's version info), the existing file will not be replaced. If the existing file is a newer version than the file being copied, the user will be asked whether the file should replaced or not. If the existing file does not have version info but the file being copied does, the existing file will be overwritten without the user's confirmation.

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

alwaysoverwrite
Always overwrites the existing file, even if the existing file is a newer version than the file being installed. *Never use this on shared system files!*

alwaysskipifsameorolder
Same as `normal`, with the exception that the user is not prompted to replace the existing file. When this mode is used, newer existing files are never replaced.

Example:
```
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 you use the copy mode `onlyifdoesntexist` and the flag `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"; CopyMode: onlyifdoesntexist; Flags: 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.

comparetimestampalso
(Special-purpose) Instructs Setup to proceed to comparing time stamps if a file being copied already exists on the user's system and the existing file has the same version info as the file being installed. Normally in this circumstance Setup would not try to overwrite the existing file, but when this flag is used the existing file is overwritten if it has an older time stamp than the version of the file Setup is trying to install. This flag has no effect if the copy mode isn't either `normal` or `alwaysskipifsameorolder`.

confirmoverwrite
Always ask the user to confirm before copying the file if it already existed.

deleteafterinstall
Instructs Setup to copy 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.

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.

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 copy 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 copied that the user didn't already have.

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

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 "AdminPrivilegesRequired=1" in the [Setup] section.

sharedfile
(Windows 95/NT 4+ only) 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 `onlyifdoesntexist` copy mode should be used, in conjunction with the `uninsneveruninstall` flag. Or if installing the latest version of a particular MFC DLL is an issue, use the copy mode `alwaysskipifsameorolder`, along with the flags `uninsneveruninstall` and `restartreplace`.

When an installation is run on an NT 3.51 system, files installed with this flag are never deleted.

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.

uninsneveruninstall
Never uninstall this file. This flag should be used sparingly, and is usually used in combination with the `onlyifdoesntexist` copy mode. 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

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 section is optional, but is necessary for most installations. It defines the shortcuts Setup is to create in the user's Start Menu (or Program Manager) and/or other locations such as the desktop.

The Uninstall icon is internally created by Setup, so you do not need to manually add it to the `[Icons]` section. By default, an uninstall icon is not created on installations running on Windows 95/NT 4+ -- only installations running on Windows NT 3.51. To force creation of an uninstall icon, use the `AlwaysCreateUninstallIcon [Setup]` section directive.

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

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

The following is a list of the supported parameters:

Name *(Required)*

Description:
The name and location of the shortcut (or Program Manager icon) 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"
```

Filename *(Required)*

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

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

Parameters

Description:
Optional command line parameters for the shortcut, which can include constants. Quotes can only be included on Windows 95/NT 4+.

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. This flag has no effect when installing on Windows NT 3.51.

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 section is optional. It defines any .INI file entries you would like Setup to set on the user's system after it has copied the files.

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

[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 the MessagesFile [Setup] section directive).

However, particular messages can be overriden 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, simply create a MessagesFile [Setup] section directive pointing to this file.

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 the MessagesFile [Setup] section directive).

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. Currently this is not used for anything, but will be once support for multi-language installations is implemented.

`LanguageID` is the numeric "language identifer" of the language. See http://msdn.microsoft.com/library/psdk/winbase/nls_8xo3.htm for a list of valid language identifiers. As with `LanguageName`, this is currently not used, but will be once support for multi-language installations is implemented. Specifically, `LanguageID` will be used for the purpose of auto-detecting the user's preferred language.

`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, *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 `WizardStyle=classic` or `WindowVisible=yes`). If the specified font does not exist on the user's system, *Arial* will be substituted.

`WelcomeFontName` and `WelcomeFontSize` specify the font name and point size to use at the top of Welcome page in the Modern wizard style. If the specified font does not exist on the user's system, *Arial* will be substituted.

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

[Registry] section

This section is optional. It defines any registry entries you would like Setup to create on the user's system after it has copied the files.

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.

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.

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`.

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.

Each program is executed in the order they appear in your script. While processing [Run]/[UninstallRun] entries, Setup waits until the current program has terminated before proceeding to the next one, unless the `nowait` flag is used.

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 .exe or .com file, 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. 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 doesn't change to any specific 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.

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`.

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. Also note that folders can only be launched on Windows 95/NT 4+, since it is a feature new to the Explorer shell.

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, Setup won't display an error message if `Filename` doesn't exist. This is only applicable to the [Run] section, since the uninstaller never displays an error message if an [UninstallRun] entry fails to execute.

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.

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`.

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`.

The `isreadme` flags for entries in the [Files] section is now obsolete. If the compiler detects a entry with a `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`.

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.

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

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

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

Components and Tasks Parameters

Common Parameters

[UninstallDelete] section

This section is optional. This section is used to define additional files or directories for the uninstaller to delete, besides those that were installed with your application. 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.htm

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 (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 commandline:

  ```
  /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 messagebox 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?* messagebox. If it's very silent it will reboot without asking.

/NORESTART
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.

/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 disable 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.

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.

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.)

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/msdownload/ieplatform/ie/comctrlx86.asp

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.)

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 (see http://www.gzip.org/zlib/).

?: Most of the disk spanning code (1.09). (Sorry, I somehow managed to lose your name! Please tell me and I'll update this.)

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).

Martijn Laan: Code for silent uninstallation (1.3.25).

Martijn Laan: Code for system image list support in drive and directory lists (1.3.25).

Martijn Laan: Code for silent installation (2.0.0).

Martijn Laan: Code for the [Types], [Components] and [Tasks] sections (2.0.0).

Martijn Laan: Code for the postinstall flag (2.0.0).

If I have left anyone out, please don't hesitate to let me know. This Credits section is new, and I didn't keep very good records in the past of who contributed what. :(

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/
-or-
http://www.jordanr.cjb.net/

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

[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 SETUP.EXE (which can be renamed to anything you want). If this is `no`, it compiles the setup data into at least four files: SETUP.EXE, SETUP.MSG, 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 similiar 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 and C 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 Kornel Pal):

```
Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Long
End Type

Declare Function CreateMutex Lib "kernel32" Alias "CreateMutexA" _
(lpMutexAttributes As SECURITY_ATTRIBUTES, ByVal bInitialOwner As Long, _
ByVal lpName As String) As Long

Sub Main()
    Dim lpMutexAttributes As SECURITY_ATTRIBUTES

    lpMutexAttributes.nLength = Len(lpMutexAttributes)
    Call CreateMutex(lpMutexAttributes, False, "MyProgramsMutexName")
End Sub
```

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 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. The default is "4,4", although it can be set to "4,3.51" to support Windows NT 3.51 also. 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:
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]: 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 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

Valid values: `yes` or `no`

Default value: `no`

Description:
If this is `yes`, Setup will create an "Uninstall ..." icon in addition to an entry in the *Add/Remove Programs* Control Panel applet.

An uninstall icon is always created on NT 3.51 regardless of this setting since it does not include the *Add/Remove Programs* applet.

[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:
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. Higher numbers are slower. Using a number higher than the default 7 doesn't improve compression very much.

[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 you 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]: 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. If this directive is not specified or is blank, Setup will use the value of `[Setup]` section directive `AppVerName` for the name (as in previous Inno Setup versions).

Due to limitations of Windows 9x's *Add/Remove Programs* Control Panel applet, `UninstallDisplayName` cannot exceed 63 characters.

Examples:
```
UninstallDisplayName=My Program
```

[Setup]: UninstallIconName

Description:
You can use this directive to specify a custom name for the uninstall icon created by Setup (see AlwaysCreateUninstallIcon). If this directive is blank or not specified, Setup names the icon "Uninstall *app-name*", where *"app-name"* is the value of the `AppName [Setup]` section directive.

NOTE: Keep in mind that Windows 95/NT 4+ store icons as literal files so any characters not allowed in normal filenames can't be used in icon names.

Example: `UninstallIconName=Uninstall My Program`

[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]: 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 (or Program Manager group name), which is used in 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: `no`

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:
Determines whether the compiler creates a single SETUP.EXE, or splits it into several files that will fit onto separate floppy disks. When this directive is set to `yes`, disk spanning is enabled. Copy the resulting files SETUP.EXE and SETUP-1.BIN to the first disk, SETUP-2.BIN to the second disk, and so on.

Note that the disk spanning feature has not been tested on CD-ROMs - only floppy disks.

[Setup]: DiskSize

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

Description:
This specifies the total number of bytes available on a disk.

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 diskette 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]: 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]: 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.

When the *Don't create any icons* box is checked on installations running on NT 3.51, the only way the user can "properly" uninstall the application is by executing the unins???.exe file in the application directory. For this reason, it is recommended that you leave AllowNoIcons set to `no`.

[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 Directory* page of the wizard.

[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 (because of `[Files]` section entries with the `restartreplace` flag).

[Setup]: MessagesFile

Default value: `compiler:DEFAULT.ISL`

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:czech.isl
MessagesFile=compiler:default.isl,compiler:mymessages.isl
```

[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.

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.

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.

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 assocations information at the end of the installation, and Uninstall will do the same at the end of uninstallation. This directive has no effect on installations running on Windows NT 3.51.

If your installation creates a file assocation 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]: 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.

[Setup]: WizardImageFile

Default value: `compiler:WIZMODERNIMAGE.BMP` if `WizardStyle` is `modern`, `compiler:WIZCLASSICIMAGE.BMP` if `WizardStyle` is `classic`.

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 if `WizardStyle` is `modern` and 117x263 pixels if `WizardStyle` is `classic`. 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.

[Setup]: WindowStartMaximized

Valid values: `yes` or `no`

Default value: `yes`

Description:
If set to `no`, Setup won't start initially maximized, causing it to hide the taskbar.

[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.

[Setup]: WindowVisible

Valid values: `yes` or `no`

Default value: `no` if `WizardStyle` is `modern`, `yes` if `WizardStyle` is `classic`.

Description:
If set to `no`, the Setup program's background window won't be visible -- only the wizard and the installation progress window.

[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` if `WizardStyle` is `modern`, `clTeal` if `WizardStyle` is `classic`.

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

[Setup]: SourceDir

Description:
Specifies a new source directory for the script.

Example: `SourceDir=c:\files`

[Setup]: OutputDir

Description:
Specifies a new output directory for the script, which is where the Setup Compiler places 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`.

Example: `OutputDir=c:\output`

[Setup]: WizardStyle

Valid values: `modern` or `classic`

Default value: `modern`

Description:
If this is set to `modern`, Setup will use the 'modern' wizard style which is also used by Windows 2000/XP, Windows Installer and other modern installation builders. If this is set to `classic`, Setup will display the 'classic' wizard style which is also used by older versions of Inno Setup and other 'old' installation builders.

[Setup]: UninstallStyle

Valid values: `modern` or `classic`

Default value: `modern` if `WizardStyle` is `modern`, `classic` if `WizardStyle` is `classic`.

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:
Not used if `WizardStyle` is `classic`. Specifies the name of the bitmap file to display in the upperright corner of the 'modern' 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 are not currently supported, since it does not handle palettes. The maximum size of the bitmap is 55x55 pixels.

Example: `WizardImageFile=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` if `WizardStyle` is `modern`, `no` if `WizardStyle` is `classic`.

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.

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`.

Notes on the Program Files directory

The Program Files directory is only natively supported by Windows 95/NT 4+. On NT 3.51 a "simulated" Program Files directory is created on the system drive under the hard-coded name "\Program Files".

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:

3.51.1057	Windows NT 3.51
4.0.1381	Windows NT 4.0
5.0.2195	Windows 2000
5.01.2600	Windows XP

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