Chief's Installer Pro for Windows

Chief's Installer Pro for Windows

Chief's Installer Pro for Windows is an **AWARD WINNING***** shareware **INSTALLER** and **UNINSTALLER** for Microsoft's Windows 3.1x, and IBM's Win-OS/2. In respect of **Windows 95** this is an "enhanced" version of Chief's Installer Pro (i.e., although it is a 16-bit program, **it works correctly under Windows 95, and also Windows NT 3.5, and Windows NT 3.51**).

PC PLUS MAGAZINE GOLD AWARD ***

Registration costs **£39.99** (U.K. Sterling) or **$59.99** (US dollars). Please refer to the **registration** and **registration sites** sections below, for fuller information.

Please read this documentation and the **DISCLAIMER** section carefully before using the program.

See also;

INTRODUCTION

Chief's Installer Pro (**enhanced for Windows 95**) is a program for the SETUP, INSTALLATION, and "unINSTALLATION" of Windows applications. It basically is an "off-the-shelf" installation suite. The program will copy files from floppy disks (or a directory on a hard disk, or a cd-rom disk) to the destination directory, and do everything else that a Windows Installer is supposed to do. If the files are compressed with Microsoft's **COMPRESS.EXE** they will be decompressed automatically. In this respect, Chief's Installer Pro uses the routines in **LZEXPAND.DLL**. If the files are not compressed, they will simply be copied to the destination directories.

SETUP.EXE

The main executable file for Chief's Installer Pro is **INSTALL.EXE** - and this is the only program that needs to be run. Also supplied is **SETUP.EXE**, an optional loader for INSTALL.EXE. **SETUP.EXE** performs the following functions;

[a] display a "initializing install ..." message

[b] copy the necessary Install files (**INSTALL.EXE, WINSTALL.INF, WINSTALL.HLP**) and **WINSTALL.DLL, WINSTALP.DLL, WINSTALL.TXT, WINSTALL.BMP** (if they exist) to the TEMP directory. **CTL3DV2.DLL** (if found) will be copied to the Windows SYSTEM directory - but only if a copy does not already exist there, **and** no copy is currently loaded in memory.

For these purposes, all these files **may** be compressed on the installation disks - but in such cases, they must retain their real names (as above) - except **INSTALL.EXE**, which if compressed with the **-r** option, can be called **INSTALL.EX_** (the original name will be restored by SETUP.EXE). I would however suggest that the .DLL files and the .BMP file should not be compressed.

SETUP will also optionally copy **USER FILES** to the TEMP directory in the same operation. If you want any user files to be copied (because you are going to do something with them) the file names should be specified in a **$TEMPDIR** line in **SETUPINF.INF**. You can have only one $TEMPDIR line in that file, and this can only contain a maximum of 10 file names (separated with semi-colons). You should remember to delete such files with the $CLEANUP command in WINSTALL.INF.

[c] load the copy of **INSTALL.EXE** from the TEMP directory, with the parameters necessary for it to work properly if run in this way

Is **SETUP.EXE** useful? "Very much so" is the answer. If there is more than one installation diskette, then you **should** use SETUP.EXE. It will save you the embarrassment of Windows trying to read from **INSTALL.EXE** after disk 2 (or whatever), and the disk in the floppy drive does not contain **INSTALL.EXE**. Even if there is only one installation diskette, it may still help keep the user occupied while **INSTALL.EXE** is loading.

I have tried to make this installer as flexible and easy to use as possible. To use it, you only need to take the following steps;

1. Create your installation disk set by placing your application's files on them (up to 64 installation disks are supported).

2. You can compress the files with **COMPRESS.EXE** (it is immaterial whether you do so or not). If files are compressed and an underscore is used in the compressed

file names, these names will only be converted to the original file names if they were compressed with the **-r** option.

3. Create an installation information file in ASCII format, using a text editor. The file should be called **WINSTALL.INF**, and should be in the prescribed format.

4. Run INSTALL.EXE or SETUP.EXE. It is recommended that you always tell your users to run SETUP.EXE, instead of INSTALL.EXE.

See also;
RESERVED WORDS

FEATURES

Below is a summary of the **features and restrictions** in Chief's Installer Pro.

1. You can only install into any combination of the following;
    [a] a target directory, and any number of subdirectories under it's directory tree
    [b] the Windows directory
    [c] the Windows SYSTEM directory
    [d] the TEMP directory
    [e] any other specified directory
2. Only a maximum of **64** installation disks are supported. Chief's Installer Pro will prompt for the disks as they are required.
3. Chief's Installer Pro will optionally offer to put the destination directory into the "PATH" statement in AUTOEXEC.BAT
4. Chief's Installer Pro will optionally create Program Manager group files, and icons for any supplied file(s) - .EXEs, README files, etc.
5. Chief's Installer Pro will optionally run any supplied program(s) during the installation, as part of the installation process.
6. Chief's Installer Pro will optionally run any supplied program(s) immediately after the installation is complete
7. The stipulated format of the WINSTALL.INF file must be followed strictly.
8. Chief's Installer Pro allows you to supply on-line for the installation. You have to create a Windows help file called WINSTALL.HLP. This help file can be accessed by clicking on the "HELP" button. A simple one is supplied. You can either use that, or create your own. If no WINSTALL.HLP file is found in the path, then the "HELP" button is removed from the Install window.
9. Chief's Installer Pro will check whether there is sufficient space on the target drive - based on information which you supply as to how much disk space your application requires. If you need extra temporary disk space for the installation, Chief's Installer Pro can check for this also.
10. Chief's Installer Pro will make entries in any INI (or other) file(s) specified by you. Unlimited entries can be made. Do NOT use this feature to insert an entry which may already exist - the old entry will be deleted and replaced by the new one (e.g., do NOT use it for "DEVICE=" lines in SYSTEM.INI)
11. Chief's Installer Pro provides support for non-English languages.
12. Chief's Installer Pro provides support for **PARTIAL INSTALLATIONS** of programs. In this respect, you can have up to **10** installation options.
13. Chief's Installer Pro provides support for displaying a **banner** in the background, and for specifying the text of the banner, the font to use used for it, the font size, the color of the text, the color of the background, and a brush to paint the background.
14. Chief's Installer Pro provides support for displaying a **bitmap** file, stretched to fill the screen.
15. Chief's Installer Pro will optionally check the disks being inserted by the user to verify that they are the correct ones.
16. Chief's Installer Pro provides an **UNINSTALLER**, which can be used to uninstall any program that was installed with Chief's Installer Pro.
17. Chief's Installer Pro will check the target directory for the existence of any copy of each file being installed, and will optionally prompt the user for overwrite permission.
18. Chief's Installer Pro will check for **version information** in existing copies of shared binary files, and for date stamps in non-shared files.
19. Chief's Installer Pro will use the 3D dialog effects in **CTL3DV2.DLL** if a copy of that file is found. If CTL3DV2.DLL is not found, the program will try to use

CTL3D.DLL. If that is also not found, then the program will use standard Windows dialogs (the absence of these files will NOT cause an error). This feature is disabled under Windows 95, because the 3D effect is built into Windows 95.
20. Chief's Installer Pro provides support for restarting Windows if any active shared file was overwritten. A dialog asking for confirmation appears automatically if any active shared file was overwritten during the installation. The text on this dialog can be changed by the $RESTARTWIN-MESSAGE reserved word.
21. Chief's Installer Pro provides support for displaying a README file to the user when the installer is executed. The README file should be a plain ASCII file not bigger than 16kb. The file should be called **WINSTALL.TXT**. You can cause the contents of the file to be displayed automatically by setting **$AUTO-CLICK-BUTTON** to 4 (otherwise the user will have to click on the "readme" button to display the text. If the file **WINSTALL.TXT** is not found, then the "readme" button will be removed at run time.
22. Chief's Installer Pro provides support for making entries into the the **Registration Database**, by the reserved word **$REG-DATA**.
23. Chief's Installer Pro provides limited support for installing TRUETYPE fonts.
24. Chief's Installer Pro provides a batch language and support for running Chief's Installer Pro batch files.
25. Chief's Installer Pro provides support for UNZIP - the UNZIP command is compatible with PKZIP(tm) 2.x ZIP archives.
26. Chief's Installer Pro provides an Integrated Development Environment (IDE) and a Project Manager for developing and managing installation projects.
27. Chief's Installer Pro's IDE provides an option to convert Visual BASIC Setup Wizard (.VBZ) files to a Chief's Installer Pro project.
28. Chief's Installer Pro provides a compiler for compiling your WINSTALL.INF file and Chief's Installer Pro batch files.
29. Chief's Installer Pro is fully compatible with 32-bit versions of Windows, such as Windows 95 and Windows NT, and with Windows emulators such as OS/2's Win-OS/2.
30. Chief's Installer Pro provides support for displaying messages (up to 10) during the course of the installation. All the messages must be in an ascii file called WINSTALL.MSG. Each message section must begin with a [#<number>] (e.g., [#1]), and cannot be more than 8 lines long (max: 45 characters per line). The messages are displayed in turn, automatically (at the rate of 100 divided by the number of messages - as per the percent meter).
31. Chief's Installer Pro provides (limited) support for installing files from certain subdirectories of the source directory (e.g., on a cd-rom or a network drive). The scheme is to use each such subdirectory as a pseudo floppy disk - i.e., the files in the subdirectory will be specified on a $DISK# line, and then the subdirectory itself will be called $DISK# (e.g., $DISK2 = for the files which will be on the $DISK2 lines). Thus the installer will automatically look for directories names $DISK# under the source directory, to install $DISK# files from. This includes $DISK1 as well.

FOREIGN LANGUAGE SUPPORT

All the string messages presented to the user by the install erare in string tables. There are internal ENGLISH string tables in INSTALL.EXE, UNINSTAL.EXE, and SETUP.EXE. These will normally be used to display all the messages and information. However, Chief's Installer Pro provides 2 methods of changing/translating the string tables for use by the various .EXEs. One method is by compiling the string tables into DLLs (for INSTALL.EXE and UNINSTAL.EXE) and the other is by putting the string tables in an ASCII file (for SETUP.EXE).

INSTALL.EXE will always look for a dynamic link library file called **WINSTALL.DLL** from which to load the string tables. If this file is found, the string tables are read from it by Install at startup time. If you are going to use this DLL, it MUST be in the same directory as INSTALL.EXE. If WINSTALL.DLL is not found, Install will use the default (English) string tables inside INSTALL.EXE itself.

By the same token, **UNINSTAL.EXE** will also look for a DLL called **UNINST.DLL** in order to load the string tables from it. If you create your own UNINST.DLL, it **must** be in the same directory as UNINSTAL.EXE. If UNINST.DLL is not found, then UnInstall will use the default (English) string tables inside UNINSTAL.EXE itself.

Finally, **SETUP.EXE** will also always look for an ASCII file called **SETUPINF.INF** in order to load the string tables from it. Each of the strings must be numbered (by a hash, followed immediately by its numeric ID), and they must all be together in a section called **[SETUP]**. This is because SETUP.EXE will use the **GetPrivateProfileString** API to retrieve them. A sample of this file is provided - it mirrors the English string tables inside SETUP.EXE. If you are using an English language installation, you should delete this file, because you do not need it.

If you create your own SETUPINF.INF, it **must** be in the same directory as SETUP.EXE, it must be uncompressed, and it must be **exactly** in the same format as the sample that I have provided. If SETUPINF.INF is not found, then Setup will use the default (English) string tables inside SETUP.EXE itself.

What all this means is that you can change the language used by the installer by simply producing your own translations of the English string tables, and compiling them into the relevant DLLs (or putting them into SETUPINF.INF in the required form). For these purposes, I have provided copies of my resource scripts, and a sample SETUPINF.INF file. These serve as a guide to the string tables and the numeric IDs of the strings. Please do NOT change the numeric ID of any string.

If anybody produces a non-English translation of the script files, please send me a copy, so that I can package non-English versions of with subsequent releases of the installer. If I package your translation, you name will make it into the "Credits" section for each version that contains your translation.

Please note this;

[a] NO checking is carried out to verify the contents of these string tables, or even that the string tables actually exist. Thus, if you edit the string tables and/or produce your own DLLs or SETUPINF.INF file, you are on your own.

[b] I can only vouch for the accuracy of the English version of any string table - and even then, only the version which was produced by me. If I package any non-English version of any file, please do not stone me if the translation is incorrect - but if you do find errors, please DO send me what you think is a correct translation of the

string table.

THE INF FILE

The configuration file for each installation is called **WINSTALL.INF**. This is an ASCII file that has to be created with a text editor (e.g., the Windows NOTEPAD applet). Every line in the file should end with a carriage return plus a line feed - so please avoid using UNIX editors to create your INF files).

It is essential that the instructions on the format of WINSTALL.INF be followed carefully, otherwise, the installer will not work correctly. The best approach is to edit the sample files which I have provided. They contain sufficient comments for them to be understood. **WINSTALL.INF** is a standard **ASCII** file, in the following format;

1. Any line beginning with a ";", or "[" or "REM" is ignored
2. Empty lines are ignored

   e.g., these lines will be ignored

   [This line will be ignored]

   ; So will this line

   REM so will the one just above me, and me as well!
3. Each line must not contain more than 220 characters
4. There are **RESERVED WORDS** for every valid entry, and these must be followed strictly.

The Chief's Installer Pro IDE (AUTOCALC.EXE)

Considerable assistance can be derived by using the Project Manager in the Chief's Installer Pro development environment (you need to run **AUTOCALC.EXE** for this) in creating INF files. See the help file **AUTOCALC.HLP** for full documentation on the Project Manager.

The IDE is an attempt to provide a facility for managing your installation projects. It does a good job of producing template INF files for each project, but like most other features of Chief's Installer Pro, it is entirely optional. If you do not like it, simply delete it - I will not take it personally!

See also;
RESERVED WORDS

RESERVED WORDS

Everything is done in Install through the use of **RESERVED WORDS**. Each reserved word begins with a dollar sign ($) and determines a certain aspect of Install's behaviour. Below is a list of the reserved words and the methods of using them.

Below is an alphabeticl list of RESERVED WORDS;

$ABORT-MESSAGE
$ABORT-UNINSTAL-QUESTION
$AUTOEXEC.BAT
$AUTO-CLICK-BUTTON
$AUTO-REPLACE
$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-MESSAGE
$BANNER-SHADOW-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-TEXT-COLOR
$BANNER-WINDOW-BRUSH
$BATCH-FILE
$BIG-METER-COLOR
$BITMAP
$CANCEL-BUTTON-TITLE
$CHECK-MY-DLL-VERSIONS
$CLEANUP
$CLOSE-GROUP-BOX
$COPYBUFFER
$DATA-SPACE
$DEST
$DIALOG-ICON
$DISK
$DISKDIR
$EXEC
$FINAL-MESSAGE
$FONT
$FORCE-EXIT-WINDOWS
$FORCE-OVERWRITE-OLDER-FILES
$FORCE-RESTART-WINDOWS
$GROUP
$ICO
$ICON
$INI
$LAN-SYSDIR
$MAKE-UNINSTALL-LOG
$MAX-DUPLICATES
$NO-ABORT-BUTTON
$NO-CTL3D.DLL
$NO-END-DIALOG
$NO-HELP-BUTTON
$NO-PATH-DIALOG
$OK-BUTTON-TITLE
$OPTIONAL
$OPTIONHELP
$PAINTDIALOGS
$PRE-EXEC

$README-BUTTON-TEXT
$README-FONT
$REG-DATA
$RESTARTWIN-MESSAGE
$SETCHECKBOX
$SHOW-FILE-PERCENT
$SMALL-METER-COLOR
$SOURCEDIR
$SPACE
$SWAP-SPACE
$SYSDIR
$SYSDIR-SPACE
$TARGET
$TEMPDIR
$TEXT-BACKGROUND
$TITLE
$UNZIP
$USER-OPTION
$VERIFY-INSTALL-DISKS
$VERSION-COPY-ERROR-MESSAGE
$VERSION-INFO-TITLE
$VERSION-INFO-MESSAGE
$WINDIR
$WINDIR-SPACE
$WINDOW
$WINDOW-BACKGROUND

$TITLE

This is used to specify the name or title of your application. This is what will appear in the title of the installation program's window - You can have only ONE such line. If you don't supply any text for the **$BANNER-MESSAGE** reserved word, this title will be used for the banner, with the words "Welcome to" prepended to it.

The Syntax is;

$TITLE=<program title>

EXAMPLE:

$TITLE=Great Program v1.20

See also;

$BANNER-MESSAGE

$TARGET

This is used to specify the name of the DEFAULT target directory for the installation. The user will be able to change this at run time. If the target directory does not exist, Chief's Installer Pro will create it - and, if necessary, will create directories recursively. You can have only ONE such line.

By default, ALL the files will be installed into whatever is the target directory chosen by the user. You can however specify that certain files should be installed into subdirectories UNDER THE TARGET DIRECTORY TREE, or into the WINDOWS DIRECTORY, or into the WINDOWS SYSTEM DIRETORY, or into the TEMP DIRECTORY.

To use this, you use the $DEST, $WINDIR, $SYSDIR, $TEMPDIR reserved words.

This reserved word can take **optional** extra parameters - details of an INI file **in the Windows directory** from which entries from a previous installation of your program can be obtained. The entry here should point to the directory into which the user installed any previous version. If such an entry is found, it will be used to replace the default one in your $TARGET line. If no entry is found, the default will be used.

If the extra parameters are used, then the $TARGET line **MUST** contain 4 entries, in the following format;

$TARGET=Default;INI file name;Section;KeyName

"Section" corresponds to "ApplicationName" in the Windows API speak (i.e., the title of the relevant section in the INI file) and "KeyName" corresponds to it's ordinary meaning with regard to the **GetPrivateProfileString** API call, which is what is used to retrieve the entries from the INI file (see also the **$INI** reserved word). The INI file name should **not** contain any path (just the filename only) - the program will only look for the file in the **Windows** directory.

e.g.,

$TARGET=C:\CHIEPRO;CHIEFPRO.INI;ChiefPro;ChiefDir

Note that you will need to create the requisite entry with the **$INI** reserved word.

e.g.,

$INI=$WINDIR\CHIEFPRO.INI;ChiefPro;ChiefDir;$DEST

The Syntax is;
$TARGET=<default directory>[;INI file name;Section;KeyName]

EXAMPLES:

$TARGET=C:\MYPROG
$TARGET=C:\MYPROG;PROG.INI;ProgPriv;ProgDir

See also;
$DEST
$INI
$SYSDIR
$TEMPDIR
$WINDIR

$SPACE

Use this to specify the amount of disk space needed for the installation. The amount should be in **BYTES** and should only contain whole numbers (no spaces, no letters, and no decimals).

This information is used by Install to warn the users of the amount of space that they need to have free on their disks, and to show the progress of the installation in the "percentage meter". Install will check to see that the specified amount of space exists on the target drive before installation begins. If there is insufficient space, Install will abort with an error message.

There is no need for the number to correspond exactly with the actual required number of bytes - a difference of up to 2% of the size of your application (plus or minus) is allowed, and such differences will be catered for automatically. In fact, it is always good to add about 1% to the actual disk space needed - because of the vagaries of disk cluster sizes, it may be wise to over-estimate the disk space needed (a little bit of trial and error is in order here). You can have only ONE such line.

The amount specified here should also take into account any disk space requirements specified in any $OPTION lines.

NOTE: The Chief Pro IDE (**AUTOCALC.EXE**) can be used to calculate the required space automatically. Please read **AUTOCALC.HLP** for fuller details.

The Syntax is;

$SPACE=<required disk space>

EXAMPLE:

$SPACE=2002003

See also;

$USER-OPTION
$OPTIONAL
$SWAP-SPACE
$SYSDIR-SPACE
$WINDIR-SPACE

$SYSDIR-SPACE

This reserved word is **optional**. It is only useful if you are using the **$SYSDIR** command to install shared files into the Windows SYSTEM directory. Its purpose is to enable the installer to ascertain that there is sufficient space on the drive which holds the Windows SYSTEM directory (in cases where the user is installing the program onto another drive). The line should only contain the total size of the files that will be installed to the Windows SYSTEM directory.

This line does not in any way affect the entry that should be on the **$SPACE** line, because they serve different (but sometimes overlapping) purposes. If the user is installing the program onto the same drive as that on which Windows is installed, this line is ignored at run time.

NOTE: The bonus program **AUTOCALC.EXE** can be used to calculate the required space automatically. Please read **AUTOCALC.TXT** for fuller details.

The Syntax is;

$SYSDIR-SPACE=<space>

See also;

$SPACE
$SYSDIR
$WINDIR-SPACE

$WINDIR-SPACE

This reserved word is **optional**. It is only useful if you are using the **$WINDIR** command to install shared files into the Windows directory. Its purpose is to enable the installer to ascertain that there is sufficient space on the drive which holds the Windows directory (in cases where the user is installing the program onto another drive). The line should only contain the total size of the files that will be installed to the Windows directory.

This line does not in any way affect the entry that should be on the **$SPACE** line, because they serve different (but sometimes overlapping) purposes. If the user is installing the program onto the same drive as that on which Windows is installed, this line is ignored at run time.

NOTE: The bonus program **AUTOCALC.EXE** can be used to calculate the required space automatically. Please read **AUTOCALC.TXT** for fuller details.

The Syntax is;

$WINDIR-SPACE=<space>

See also;

$SPACE
$SYSDIR-SPACE
$WINDIR

$SWAP-SPACE

Use this to specify the amount of any temporary swap disk space needed for the installation. The amount should be in BYTES and should only contain whole numbers (no spaces, no letters, and no decimals).

This information is used by Install to warn the users of the amount of space that they need to have free on their disks - bit it does not show in the "percentage meter". The installer will check on the drive which contains the TEMP directory to ensure that there is sufficient swap space. This obviously presumes that all your scratch and temporary files will be created in the TEMP directory. You can have only ONE such line.

EXAMPLE:

```
$SWAP-SPACE=0
```

See also;
$SPACE
$SYSDIR-SPACE
$WINDIR-SPACE

$DISK

You use this reserved word to specify the disk(s) in the installation set, and the file(s) which should be copied from them. Each file name or file specification should be separated from the next one by a semi-colon.

You can use the wildcard character "*" in this respect.

The Syntax is;
$DISK n = <filenames(s)>

where:

n = numbers from 1 to 64

<filename(s)> = the file specifications

the easiest thing to do would be to just specify "*.*" - to copy all the files - but you can be more specific.

NOTES:

If you specify a file that does not exist on the disk, it will just be ignored.

Note that each line cannot be longer than 220 characters in total. Since this might mean that all the files you want to specify for a disk might not fit on one line, you can either put all the file specifications for each disk on a single $DISK line, or you can split them up into several $DISK lines for better readability. For example, for DISK 1 of the installation set, you can either put all the files on one "$DISK1=" line (if they will fit on one line) or you can have several "$DISK1=" lines, each line listing different file specifications to make up your DISK 1).

In most cases, judicious use of wildcard characters should mean that you only need one line for each disk in your installation set (e.g., if you use something like: **$DISK1=*.EX_;*.HL_;*.TXT**). However, of you wish to be more specific about the files on each disk, the flexibility is available to spread each "$DISK" across many lines.

Chief's Installer Pro will prompt the user for each disk in the installation set. However, unless you turn on disk verification with the **$VERIFY-INSTALL-DISKS** reserved word, no attempt will be made to check that the disk being inserted is the correct one. **Please note this point**.

Please ensure that the $DISK lines are numbered consecutively, otherwise there may be problems - for example, don't jump from "$DISK1=" to "$DISK3=" (the problem here is - where is $DISK 2? - if you put "$DISK2=" after "$DISK3=", then there may be problems numbering the disks.)

Compressed files with underscores in the filename will have the filenames expanded into the name of the original files **ONLY if** the original files were compressed with the **-r** option

VERY IMPORTANT NOTE: Please be very careful with the way wildcard characters are used, **especially** if your program spans more than one disk. For example, it is very convenient to use ***.*** for all the disks in your installation set, and while this will be okay if your program is only going to be installed from floppy disks, imagine the chaos, if somebody copies all the files to a directory on the hard disk before installing, or if a CD-ROM distributor puts your program in a directory on a CD-ROM. You will have the same files ("*.*") being installed over and over again, for each disk on the installation set, and your users will not be impressed.

Thus, unless you are **absolutely certain** that your program will only ever be installed from floppy disks, you need to be selective in your use of wildcard characters - at the least, to make sure that if all your program's files are installed from a single source directory, none of the files which belong to one disk can be confused with files belonging to another disk. In this wise, it may be advisable to place files on each disk according to type and/or extension (e.g., $DISK1=*.EX_;*.HL_ : $DISK2=*.DL_;*.VB_ - etc., etc).

You can also use the $UNZIP command on a $DISK line, to specify that a file on that disk should be UNZIPPED instead of just copied or expanded. In this case, the files will only go to the destination provided in the $UNZIP command.

EXAMPLES:

```
$DISK 1 = *.*
$DISK 2 = *.DLL;*.HLP;*.DRV;WS*.*;*.EX_
$DISK 2 = EXPAND.*;COMPRESS.EXE;FRED.EXE;CHIEF.EXE
$DISK 3 = HELP.DOC;*.FFF

$DISK4=$UNZIP;$SOURCEDIR\BIN.ZIP;$DEST\BIN
```

See also;
$VERIFY-INSTALL-DISKS
$UNZIP

$DEST

[i] Where ever this appears **at the beginning of a line**, the following take place;

(a) "$DEST" is replaced with the target directory selected by the user. For example, entry of "$DEST\BIN=PROG.EXE", if the user installed to "C:\NEWPROG", becomes "C:\NEWPROG\BIN=PROG.EXE".

In this respect, you can also use $DEST to provide for installing files to other drives and directories (i.e., not under the directory tree of the target directory, by puttin two exclamation marks (!!) after the $DEST, and then adding the "=" sign, and the relevant directory, followed be another "=" sign, and then the file(s) which are to go there. You can use other reserved words (e.g., $WINDIR) here.

(b) anything after the "=" sign is taken as the file(s) to be installed into that directory (**instead of into the target directory**). There can be up to 30 file names, each separated by a semi-colon. Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.TXT;*.INI;*.DLL;*.EXE**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible.

The sub directories will be created when necessary - but note that the order in which they are specified may be important - if there are deep levels of nesting, the ones higher up the tree **must** be specified first.

Please **NOTE** that in the case of files compressed with the -r switch, you should use the real (original) names of the **uncompressed** files, and **NOT** the names of the compressed files. For example, if the file **MYPROG.DLL** was compressed to **MYPROG.DL_**, you should put **MYPROG.DLL** on this line. The compressed filenames are only allowed on **$DISK** lines.

You can have up to any number of $DEST lines (i.e., no limit).

[ii] Where ever this appears **elsewhere in a line**, the "$DEST" is replaced with the target directory selected by the user. This use of the $DEST reserved word is only useful in the "$ICON", "$INI", and "$EXEC" lines.

The Syntax is;

$DEST=<filename(s)>

EXAMPLES:

```
$DEST\BIN=*.EXE;*.DLL;WINSTALL.INF
$DEST\HELP=*.HLP;*.TXT;*.WRI
$DEST\SAMPLES=SAMPLE1.INF;SAMPLE2.INF;SAMPLE3.INF
$DEST!!=D:\TEMP=*.INI;*.TXT;*.TKT
$DEST!!=$WINDIR\BAK=*.INI;*.TXT;
```

See also;

$EXEC
$ICON
$INI
$TARGET
$SYSDIR
$TEMPDIR
$WINDIR

$WINDIR

[i] Whenever this appears at the beginning of a line, the files on that line are installed to the Windows directory (instead of the target directory). You can have an unlimited number of $WINDIR lines. Each line can contain a maximum of **30** file names, each separated by a semi-colon. Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.INI;*.EXE**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible. Note that each line cannot be longer than 220 characters.

Please **NOTE** that in the case of files compressed with the -r switch, you should use the real (original) names of the **uncompressed** files, and **NOT** the names of the compressed files. For example, if the file **MYPROG.DLL** was compressed to **MYPROG.DL_**, you should put **MYPROG.DLL** on this line. The compressed filenames are only allowed on **$DISK** lines.

[ii] Where ever this appears elsewhere in a line, the "$WINDIR" is replaced with the Windows directory. This use of the $WINDIR reserved word is only useful in the "$ICON", "$INI", and "$EXEC" lines.

EXAMPLES:

```
$WINDIR=PROG1.EXE;PROG2.EXE;PROG2.EXE;*.INI
$WINDIR=RATTER.EXE;RETTO.EXE;DRAT.EXE
$WINDIR=ROTTO.INI;ROUTER.INI;TROUBLE.INI
```

See also;
$DEST
$SYSDIR
$TARGET
$TEMPDIR

$SYSDIR

[i] Whenever this appears at the beginning of a line, the files on that line are installed to the Windows SYSTEM directory (instead of the target directory). You can have an unlimited number of $SYSDIR lines. Each line can contain a maximum of **30** file names, each separated by a semi-colon.

Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.VBX;*.DRV;*.DLL;*.TTF**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible.

When wildcards are used in this way, **any conflict is resolved in the following order: [1] $SYSDIR, [2] $WINDIR, [3] $TEMPDIR, [4] $DEST**. This means for example that, if you use "*.DLL" in a $SYSDIR line, and then you put something like PROG.DLL in a $DEST line, the file PROG.DLL will still be installed into the Windows SYSTEM directory, because the $DEST entry is resolved last. Please note this point.

Note also that each line cannot be longer than 220 characters.

Please **NOTE** that in the case of files compressed with the -r switch, you should use the real (original) names of the **uncompressed** files, and **NOT** the names of the compressed files. For example, if the file **MYPROG.DLL** was compressed to **MYPROG.DL_**, you should put **MYPROG.DLL** on this line. The compressed filenames are only allowed on **$DISK** lines.

[ii] Where ever this appears elsewhere in a line, the "$SYSDIR" is replaced with the Windows SYSTEM directory. This use of the $SYSDIR reserved word is only useful in the "$ICON", and "$EXEC" lines.

EXAMPLES:

```
$SYSDIR=PROG1.DLL;PROG2.DLL;PROG2.DLL;MYDRV.DRV
$SYSDIR=RATTER.DLL;RETTO.DLL;DRAT.DRV;*.VBX;*.TTF
$SYSDIR=ROTTO.DRV;ROUTER.DRV;TROUBLE.DRV
```

See also;
$DEST
$TARGET
$TEMPDIR
$WINDIR

$TEMPDIR

[i] Whenever this appears at the beginning of a line, the files on that line are installed to the TEMP directory (instead of the target directory). There can be up to 30 file names, each separated by a semi-colon. Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.TMP;*.$$$**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible. You can have only ONE such line.

[ii] Where ever this appears elsewhere in a line, the "$TEMPDIR" is replaced with the TEMP directory. This use of the $TEMPDIR reserved word is only useful in the "$INI", and "$EXEC" lines.

See also;
$DEST
$TARGET
$SYSDIR
$WINDIR

$AUTO-REPLACE

Use this to specify any files that should be replace automatically if they already exist (i.e., without first prompting the user for confirmation).

This reserved word if effective for matching files which exist in the target directory, and which are NOT newer (by their date stamp) than the files being installed. If the existing file has got a more recent date stamp than the one being installed, then the user WILL be prompted before it is overwritten.

You can have an unlimited number of $AUTO-REPLACE lines, and up to 30 file names on each line (separated by semi-colons). Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.TXT;*.INI;*.DLL;*.EXE**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible.

Please **NOTE** that in the case of files compressed with the -r switch, you should use the real (original) names of the **uncompressed** files, and **NOT** the names of the compressed files. For example, if the file **MYPROG.DLL** was compressed to **MYPROG.DL_**, you should put **MYPROG.DLL** on this line. The compressed filenames are only allowed on **$DISK** lines.

The Syntax is;

$AUTO-REPLACE=<filename(s)>

EXAMPLES:

$AUTO-REPLACE=PROG1.EXE;PROG1.DLL;PROG4.DLL;MYPROG.INI
$AUTO-REPLACE=RATTER.RAT;RETTO.RET;DRAT.DRA
$AUTO-REPLACE=ROTTO.ROT;ROUTER.RUT;TROUBLE.HUT

See also;

EXISTING FILES
$FORCE-OVERWRITE-OLDER-FILES
$SKIP-IDENTICAL-FILES

$INI

This is used to specify any **ASCII** files that configuration information should be written into. Normally, these will be INI files of some sort, but they can be any file, as long as any such file is in ASCII format.

You have have up to any number of **$INI** lines (i.e., no limit) - and each line **MUST** be in the format prescribed below;

Each line must contain at least 4 entries - each separated with a semi-colon

[a] the first entry is the NAME of the file to be written to - a full path must be supplied - otherwise, the file is presumed to be in the WINDOWS directory. You can use "$DEST" here, to specify files in the directory tree of the target directory. If the file does not exist, it is created.

[b] the second entry is the title of the section ("Application name" in Windows API speak) that should contain the entry.

[c] the third entry is the name of the entry you wish to make ("Keyname" in Windows API speak)

[d] the fourth entry is the string that you wish to associate with the entry. If you wish to specify an empty string as the value for the entry, just supply " " as the 4th entry.

[e] you can have an **optional** fifth entry **NO-REPLACE**. Use this to signify that an existing entry should not be replace. By default, an existing entry in an INI file will be replaced by the ones specified in the $INI lines. Using **NO-REPLACE** as the fifth entry in a $INI line will ensure that while an entry will be made if none already exists, old entries will be left intact.

[f] you can have an **optional** sixth entry - **a user option number**. Use this to assign the INI entry to a user option. This is done by specifying the relevant user option (e.g., $USER-OPTION3) as the 6th entry on the $INI line (or as the 5th entry, if NO-REPLACE is not used for the 5th entry). If the specified user option is de-selected by the user at runtime, then the INI entry will not be made.

The Syntax is;

$INI=<Filename>;<Section>;<KeyName>;<String>[;NO-REPLACE][;$USER-OPTION#]

EXAMPLES:

```
$INI=$DEST\MYPROG.INI;CONFIG;STARTUP;PROG.EXE -FE=XDS.XCL
$INI=WIN.INI;EXTENSIONS;GFD;$DEST\BIN\GFD2.EXE ^.GFD
$INI=C:\AUTOEXEC.BAT;MYPROG;SET PROGDIR;$DEST;$USER-OPTION1
```

See also;

$DEST
$SYSDIR
$WINDIR

$GROUP

This is used to specify the DEFAULT name of the Program Manager Group in which the icons will be created. This can be the name of an existing group (e.g., "Accessories", "Main", etc.) in which case, the items will just be added to the ones already in that group. However, you may specify a completely new group. If this does not exist, it will be created.

You can have only ONE such line - but you can specify other group names for different icons in the $ICON reserved word.

$GROUP can also take an extra (and optional) parameter - the word **AUTO**, the word **DISABLE**, or the word **SHOW-COMBO**. If used, this parameter should appear after the group name and should be separated from the group's name by a semi-colon.

1. **AUTO** - means create a group automatically - do not allow the user to uncheck the "Create Program Manager Group" checkbox.

2. **DISABLE** - means do NOT create any group at all - and do not allow the user to specify that a group should be created.

If either of these parameters is used, then the checkbox will not be presented. In none of them is used, then the checkbox will be presented and the user will have a choice. Both of these options deny the user a choice in the matter (i.e., either the group will be created automatically or it will not be created at all, regardless of what the user may want).

3. **SHOW-COMBO** - means show a combo box displaying the names of the available Program Manager groups. The user can then choose any of the listed groups to use as the main group.

The Syntax is;

$GROUP=<groupname>[;parameter]

EXAMPLES:

$GROUP=My Program
$GROUP=My Program;AUTO
$GROUP=My Program;DISABLE
$GROUP=My Program;SHOW-COMBO

See also;

$ICO
$ICON

$ICON

This is used to specify the names of the files for which you want Program Manager icons to be created. There can be a maximum of 128 icons.

Each $ICON line should contain only **ONE entry**. This is the name of the file to create an icon for (this could be a program file plus a parameter or any other file). This should be followed by a semi-colon, and after the semi-colon, the title that Program Manager should give to the icon; and (optionally), preceded by a semi-colon, the name of any other group (i.e., if different from the one in the $GROUP reserved word) that the icon should be created in; and (optionally), the name of the .ICO file to use for the file.

If no group is specified on this line, then the one pointed to by the $GROUP reserved word will be used.

If no external .ICO file is specified, then Program Manager will use the first icon it finds in the specified file, or if the file has no icon, then a default icon will be used.

If you specify the name of an external .ICO file, then the full path name of the icon file must be provided, AND, that path MUST be the same as the path of the file that a Program Manager icon is being created for. What this means is that the full path of that file must be the first thing on the $ICON line (i.e., you cannot specify an executable, and then the file as an argument to that executable).

Secondly, if you specify an external .ICO file, then you MUST also specify the group in which the icon will be created (i.e., there must be 4 entries on the $ICON line in such cases). In this case, you can simply put $GROUP as the group name.

Normally, existing icons will not be duplicated if the installation is run again. To change this behaviour, you can specify ALLOW-DUPLICATES as the LAST parameter to $ICON, in which case, an icon will be created regardless of whether an icon by the same title already exists in the group.

The Syntax is;

$ICON=<filename>;<title>[;<group>;<.ICO file>][;ALLOW-DUPLICATES]

EXAMPLES:

```
$ICON=$DEST\MYMAIN.EXE;Cool Prog v1.20
$ICON=BACKUP.EXE;Backup Applet;Accessories
$ICON=$DEST\MYPROG.HLP;My help file;$GROUP;$DEST\PROG.ICO
$ICON=$DEST\README.TXT;Readme file;$GROUP;$DEST\TEXT.ICO
$ICON=NOTEPAD.EXE REGISTER.TXT;Registration documentation;ALLOW-
DUPLICATES
```

See also;

$DEST
$GROUP
$ICO
$SYSDIR
$WINDIR

$PRE-EXEC

This line is **optional**. It specifies the name(s) of any program(s) that should be run during the installation, as part of the installation process. These programs will be run immediately after the files have been copied from the disks. Install will **try** to wait for these programs to terminate, before continuing. Such attempted waiting will work for Windows programs, but will fail if used to run **DOS programs** under OS/2.

There can be only ONE such line, but it may contain up to 5 programs, each separated with a semi-colon.

The Syntax is;

$PRE-EXEC=<program name> [parameters] [;<other program>]

EXAMPLE:

```
$PRE-EXEC=$WINDIR\EXPAND.EXE $DEST\REE.BI_;$TEMPDIR\GAGOFF.EXE >
NUL
```

See also;

$DEST
$EXEC
$SYSDIR
$TEMPDIR
$WINDIR

$CLEANUP

This is optional. It specifies the name(s) of any temporary files(s) that should be deleted after the installation. Such deletions (if any) will be done immediately after any $PRE-EXEC lines have executed and returned. If there is no $PRE-EXEC line, then the deletions will be done after the $INI lines have been processed. If there are no $INI lines, then the deletions will be immediately after the $DISK lines have been processed. If the specified files do not exist, they are simply ignored.

You can have an unlimited number of $CLEANUP lines. Each line should contain only ONE entry. You can use wildcard characters here, but note that the program will NOT accept "*.*".

Please use this reserved word with care. I accept no responsibility for any problems caused by using it.

The $TEMPDIR can be used here with the Install program files in cases where you choose to use SETUP.EXE as a loader. This way, Install can cleanup the files which have been copied by SETUP.EXE to the TEMP directory.

The Syntax is;

$CLEANUP=<filespecs>

EXAMPLES:

$CLEANUP=$TEMPDIR\TMP*.*
$CLEANUP=$DEST\TEMPFIL.INI
$CLEANUP=$TEMPDIR\INSTALL.EXE
$CLEANUP=$TEMPDIR\WINSTAL*.*

See also;

$DEST
$TEMPDIR

$EXEC

This line is **optional**. It specifies the name(s) of any program(s) that should be run immediately after the installation is completed (with any optional parameters to be passed to the programs). You can have only ONE such line, but you can put as many as 5 programs on this line, each separated by a semi-colon.

The Syntax is;

$EXEC=<program name> [parameters] [;<other program>]

EXAMPLE:

$EXEC=CONFIG.EXE -DIR=C:\TEMP; MYPROG1.EXE; MYPROG2.EXE -NEW

See also;

$DEST
$PRE-EXEC
$SYSDIR
$TEMPDIR
$WINDIR

$WINDOW

This line is **optional**. It should specify whether you want Install to start up maximized or not. If the entry here is **MAXIMIZE** then Install will start maximized - otherwise it will just start normally. You can have only ONE such line.

The Syntax is;

$WINDOW=MAXIMIZE

$COPYBUFFER

This line is **optional**. It sets the size of the buffer used by Install to copy the files. The buffer size should be a whole number, representing the number of BYTES to be used. This number MUST be between 2048 and 32760. If it is set lower than 2048, then Install will replace the supplied value with 2048 - and if it is set higher than 32760, then Install will use 32760.

The higher the buffer, the faster the files are copied. However, the buffer size also dictates;

[a] the frequency with which the "percent" meter is updated

[b] the frequency with which Install will "yield" the CPU and allow Windows to do other things (each time COPYBUFFER bytes are copied, Install "yields" for 128 milliseconds.

Therefore, if the number is set too high, the percent meter will not be updated frequently enough, and the display might look odd. If, on the other hand, the setting is very low, then the percent meter will be updated frequently, but the file copying will become much slower.

The DEFAULT value is 8190, and this will be used if this setting is left empty. I suggest a setting of 16384 (i.e., 16kb) as a good setting which adequately compromises between speed of copying, and the frequency of the progress bar's being updated.

The Syntax is;

$COPYBUFFER=<buffersize>

EXAMPLE:

$COPYBUFFER=4095

$WINDOW-BACKGROUND

This line is **optional**. It can be used to set the background color of the main window of Chief's Installer Pro. The default is to have a light gray background for the main dialog, and a white background for other dialogs (the light gray will also be used for other dialogs if you use the $PAINTDIALOGS command). Because the background is a Windows brush handle, the only valid values for this setting are 0, 1, 2, 3, or 4.

0 = **White Brush**
1 = **Light gray Brush**
2 = **Gray Brush**
3 = **Dark gray Brush**
4 = **Black Brush**

If you use this to change the window background, be sure to also set the $TEXT-BACKGROUND (below) to an appropriate setting. For example, if this setting is 2 (dark gray) then the text background should be set to 128,128,128 (so that the window and text backgrounds should match).

The Syntax is;
$WINDOW-BACKGROUND=<value>

EXAMPLE:

$WINDOW-BACKGROUND=1

See also;
$TEXT-BACKGROUND

$TEXT-BACKGROUND

This line is **optional**. It can be used to set the background color of the **text** in the main window of Chief's Installer Pro. The default is to have a light gray background. Unless there is a pressing need to use another color, the color used here should be the same as that used for the $WINDOW-BACKGROUND.

The color used here can either be either;

[a] one long integer value (you can use hexadecimal values in Pascal notation) - see below for explanation

or

[b] three values represent RGB (red, green, blue) values.

If using RGB values, they should be separated by commas, or semi-colons (e.g: 128,128,128 - for a dark gray background)

If using a hexadecimal value (those that begin with $00 and then are followed by **SIX** values). The SIX values here represent Blue, Green, Red - or reversed RGB. In this respect, **FF** turns the color to full intensity, **00** turns it off, and any other value varies the intensity.

Note that the color that results from any value depends on the display driver of the user (particularly the number of colors). For a 256 color setup, you can use the following **EXAMPLE** values;

1. **White** : **$00FFFFFF**
2. **White** : **255,255,255**
3. **Black** : **$00000000**
4. **Black** : **0,0,0**
5. **Dark Gray** : **$00808080**
6. **Dark Gray** : **128,128,128**
7. **Red** : **$000000FF**
8. **Red** : **$255,0,0**
9. **Blue** : **$00FF0000**
10. **Blue** : **0,0,255**
11. **Light Cyan** : **$00FFFF00**
12. **Green** : **$0000FF00**
13. **Yellow** : **$0000FFFF**
14. **Magenta** : **$00FF00FF**
15. **Light Gray** : **$006F9FFF**
17. **Light Gray** : **192,192,192**
16. **Gray** : **$00C0C0C0**

The Syntax is;

$TEXT-BACKGROUND=<color value>

EXAMPLES:

$TEXT-BACKGROUND=192,192,192
$TEXT-BACKGROUND=128,128,128
$TEXT-BACKGROUND=$00FFFFFF

See also;

$WINDOW-BACKGROUND
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-WINDOW-BRUSH

$PAINTDIALOGS

This line is **optional**. It is for use in those cases when you want the status dialogs to be painted with the same text and background colors as the main Install window (the default is that the status "percent" dialogs have a white background). This line takes no parameter.

The Syntax is;

$PAINTDIALOGS

$SETCHECKBOX

This line is **optional**. It automatically checks the checkbox titled "create Program Manager item". This line takes no parameter.

The Syntax is;

$SETCHECKBOX

$NO-END-DIALOG

This line is **optional**, and is not very useful. All it does is to suppress the final dialogs which inform the user about whether the installation was successful or not, and that the installation is completed. The default behaviour of Install is to present these dialogs to the user. Use this reserved word to disable that feature.

If this feature is used, the warning dialog that comes up if the size of the files actually installed is less than 98% of the size stated in the $SPACE reserved word is also disabled. This line takes no parameter.

The Syntax is;

$NO-END-DIALOG

$NO-PATH-DIALOG

This line is **optional**. It disables the dialog box which asks the user whether the target directory should be added to the PATH statement in AUTOEXEC.BAT. The default behaviour is to present this dialog. Use this reserved word to disable that feature. This line takes no parameter.

$SHOW-FILE-PERCENT

This line is **optional**. What it does is to show a small percent meter for the progress of each individual file being installed (i.e., in addition to the large percent meter which shows the progress of the whole installation process). This reserved word takes no parameter.

The Syntax is;

$SHOW-FILE-PERCENT

$MAKE-UNINSTALL-LOG

This line is **optional**. What it does is to cause Chief's Installer Pro to create a log file of all the changes it is making to the system. This file is a binary file (to prevent tampering with it) and is called **UNINSTAL.LOG**. It is created in the target directory, and should be left there. This file will be used by the **UNINSTALLER** to uninstall the program, if the user so wishes.

If the user is installing over an existing installation and a copy of UNINSTAL.LOG already exists, Chief's Installer Pro will just add the new information to the end of the existing one. This may result in some information being duplicated in the file, but will not lead to any strange result. The uninstaller is smart enough to handle any duplicated information.

This reserved word can take as an **optional** parameter the name of the file to use as the LOG file for **UNINSTAL.EXE**. The parameter should be separated with a semi-colon, and should contain a filename only (**no path**). If no filename is provided, the default name UNINSTAL.LOG will be used. If a filename is used here, it **MUST** be supplied as a **SECOND** parameter to UNINSTAL.EXE

This reserved word can also take another **optional** parameter - the word **OVERWRITE**. When this is used, the installer marks the LOG file so that UNINSTAL.EXE will overwrite every file and directory which it has deleted, so that they cannot be undeleted. This parameter should be used with **great care**.

The Syntax is;

$MAKE-UNINSTALL-LOG[;logfilename][;OVERWRITE]

EXAMPLES:

$MAKE-UNINSTALL-LOG

$MAKE-UNINSTALL-LOG;OVERWRITE

$MAKE-UNINSTALL-LOG;VER2.LOG

$MAKE-UNINSTALL-LOG;VER2.LOG;OVERWRITE

See also;

THE UNINSTALLER

$USER-OPTION

The user option lines are **optional**. By default, Chief's Installer Pro will install **all** the files which are specified in the **$DISK** lines. However, sometimes, the user will only want to install the binaries, or the documentation, or the libraries, or any other partial installation.

The **$USER-OPTION** reserved word gives you the means of providing user-selectable installation options, for various parts of your program. So you can split your program's installation into program files, help files, libraries, dictionaries, bitmaps, etc., etc., and the user will be given a dialog with check boxes which allows him to choose, or just to install everything. This means that your users will now have the facility for incremental installation of different parts of your program.

There can be up to **10** $USER-OPTION lines, each of them specifying a different optional part of your program. If you specify a user option, you **must** also use the **$OPTIONAL** reserved word (see below) to specify the files which make up that user option. In such cases, appropriate check boxes will appear.

Each $USER-OPTION line must contain the title of the option (this is the text that will appear beside it's check box), followed by a semi-colon, and then the amount of disk space (in bytes) which the option will require. This amount will be added to the amount specified in the **$SPACE** reserved word, such that if you make all the different parts of your installation optional, then the $SPACE line must specify 0 (zero) as the required disk space.

NOTE: The bonus program **AUTOCALC.EXE** can be used to calculate the required space automatically. Please read **AUTOCALC.TXT** for fuller details.

$USER-OPTION lines can also take an extra (and optional) parameter - the word **UNCHECKED**. If used, it must be put last, separated from the size of the option's files by a semi-colon. If it is used, the checkbox for the option is not checked when the installer starts. The user can check it afterwards.

The Syntax is;

$USER-OPTION n = <title>;<disk space needed>[;UNCHECKED]

where:

n = any number from 1 to 10

<title> = the text to show beside the option's check box

<disk space needed> = the amount of disk space required by the option (in bytes)

EXAMPLES:

$USER-OPTION1=Program files;171000
$USER-OPTION2=Optional DLL files;10000
$USER-OPTION3=Optional executables;104000
$USER-OPTION4=Readme files;14384;UNCHECKED

See also;

$OPTIONAL
$OPTIONHELP
$SPACE

$OPTIONAL

This reserved word is used to specify the files that make up any user-selectable installation options specified with the **$USER-OPTION** reserved word. Each line should specify a list the files that make up the particular option number, each file name separated from the next one by a semi-colon. Some **limited** use of wildcards is allowed here - if you want to use wildcards, then it must be an asterix, followed by a dot, and then the full extension of the files - e.g., ***.TXT;*.INI;*.DLL;*.EXE**. Great care must be taken not to confuse the program when using wildcards in this way. Careless use of wildcards might lead to files going where they were not meant to go. It is better to name individual files whenever possible. A maximum of 30 file specifications is allowed on each line - but note that each line cannot be longer than 220 characters in total.

You can have an unlimited number of $OPTIONAL lines for **each** option specified by a $USER-OPTION line. This facility is to allow for situations where all the file names will not fit on one line. The fact that you can have multiple lines for each option number, and that each line can contain up to 30 file names, means that you can in theory have a large number of files making up each option. However, please do not go overboard with this, because **each** file name on **each** $OPTIONAL line has to be checked against **every** file being installed, to see whether it should be installed or not. Therefore, if there are too many files in the $OPTIONAL lines, the installation process will be slowed down (this might not be a problem on machines with fast CPUs).

The Syntax is;

$OPTIONAL n = <filenames>

where:

n = any number from 1 to 10 (corresponding to the relevant $USER-OPTION)

<filenames> = the files which make up the option - each separated by a semi-colon

EXAMPLES:

$OPTIONAL1=INSTALL.EXE;WINSTALL.HLP;INSTALL.TXT;WINSTALL.INF
$OPTIONAL1=SAMPLE1.INF;SAMPLE2.INF;SAMPLE3.INF;SAMPLE4.INF
$OPTIONAL2=ENGLISH.dll;dansk.dll;deutsch.dll
$OPTIONAL3=UNINSTAL.EXE;SETUP.EXE;
$OPTIONAL4=*.WRI;*.TXT;*.DOC;*.PS;READ.ME

See also;
$OPTIONHELP
$USER-OPTION

$BANNER-FONT

This reserved word specifies the font to use for the **banner** text that will be displayed in the background of Chief's Installer Pro's dialog window (on the Windows desktop). Most of the Windows **TRUETYPE** fonts can be used here. The font will be in bold faced characters, and will be italicised.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If the named font does not exist on the system, then Windows will try to use a substitute font, or at the least, a COURIER font. If this line is empty, then Chief's Installer Pro will default to using the TrueType **TIMES NEW ROMAN** font.

The Syntax is;

$BANNER-FONT=<font name>

EXAMPLE:

$BANNER-FONT=ARIAL

See also;

$BANNER-FONT-SIZE
$BANNER-MESSAGE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$WINDOW

$BANNER-FONT-SIZE

This reserved word specifies the "point" size of the font used to display the banner. The size should be a whole number.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If this line is empty, Chief's Installer Pro defaults to using **35 point**. Note that you should be careful to cater for the smallest display resolutions (practically, 640x480 displays). Therefore the font size should be small enough for the banner message to fit in a standard VGA screen).

The Syntax is;

$BANNER-FONT-SIZE=<font size>

EXAMPLE:

$BANNER-FONT-SIZE=45

See also;

$BANNER-FONT
$BANNER-MESSAGE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$WINDOW

$BANNER-MESSAGE

This line specifies the message to be displayed as the banner for your installation. This message is displayed in the banner window, on the Windows desktop. The message should be short enough to fit on one line, taking into account the font being used, and its size.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If this line is empty, Chief's Installer Pro will default using the title of your program as specified on the **$TITLE** line, and the words "Welcome to" will be prepended to that title.

The Syntax is;

$BANNER-MESSAGE=<banner message> [;CODE]

In this syntax, "CODE" is optional. Possible values are **CENTERED** (centre the banner message); or **VERTICAL** (display the text vertically). If VERTICAL is used, you also neet to provide an "x" coordinate for the text

EXAMPLES:

$BANNER-MESSAGE=This is a Great Program!
$BANNER-MESSAGE=This is a Great Program;CENTERED
$BANNER-MESSAGE=Chief Pro;VERTICAL;5

See also;

$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$TITLE
$WINDOW

$BANNER-TEXT-COLOR

This reserved word specifies the color to be used for the banner text. This color can be either one long integer value (TColorRef in Windows) or three RGB values. The values that can be used here are the same as those that can be used in the **$TEXT-BACKGROUND** reserved word. Please see the documentation on it for further details.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If this line is empty, install defaults to using a white text color (**$00FFFFFF**).

The Syntax is;

$BANNER-TEXT-COLOR=<color value>

EXAMPLE:

$BANNER-TEXT-COLOR=$00C0C0C0

See also;

$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-MESSAGE
$BANNER-TEXT-BACKGROUND
$BANNER-WINDOW-BRUSH
$BANNER-SHADOW-COLOR
$TEXT-BACKGROUND
$WINDOW

$BANNER-TEXT-BACKGROUND

This reserved word specifies the color to be used for the banner text background. This color can be either one long integer value (TColorRef in Windows) or three RGB values. The values that can be used here are the same as those that can be used in the **$TEXT-BACKGROUND** reserved word. Please see the documentation on it for further details.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If this line is empty, install defaults to using a blue text background (**$00800000**).

The Syntax is;

$BANNER-TEXT-BACKGROUND=<color value>

EXAMPLE:

$BANNER-TEXT-BACKGROUND=$00800000

See also;

$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-MESSAGE
$BANNER-TEXT-COLOR
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$TEXT-BACKGROUND
$WINDOW

$BANNER-SHADOW-COLOR

This reserved word is **optional**. It is used to give the banner text a "shadow". The color value here is in the same format as **$BANNER-TEXT-COLOR** and **$BANNER-TEXT-BACKGROUND**. The shadow is disabled by default (by giving the shadow color a default value less than 0). If the value is 0 or higher, then the shadow becomes enabled, and the **$BANNER-TEXT-BACKGROUND** line becomes disabled.

The Syntax is;

$BANNER-SHADOW-COLOR=<color value>

EXAMPLE:

$BANNER-SHADOW-COLOR=255,0,0

See also;

$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$TITLE
$WINDOW

$BANNER-WINDOW-BRUSH

This reserved word specifies the color to be used to paint the background of the banner window. The painting is not done as a straight color. Rather, it **starts** as the color you specify (at the top of the screen), and gradually changes, until it reaches **black** (at the bottom of the screen). This presents a pleasant visual effect. This is even more so if you use the same or nearly the same color as the one used in the **$BANNER-TEXT-BACKGROUND** line.

This color can be either one long integer value (TColorRef in Windows) or three RGB values. The values that can be used here are the same as those that can be used in the **$TEXT-BACKGROUND** reserved word. Please see the documentation on it for further details.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used. This is because when Chief's Installer Pro's main window is maximized, the banner is not displayed at all (for obvious reasons).

If this line is empty, install defaults to using a blue color (**$00800000**).

The Syntax is;

$BANNER-WINDOW-BRUSH=<color value>

EXAMPLE:

$BANNER-WINDOW-BRUSH=100010

See also;

$BANNER-FONT
$BANNER-FONT-SIZE
$BANNER-MESSAGE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$TEXT-BACKGROUND
$WINDOW

$BITMAP

This reserved word specifies a Windows bitmap file to display in the banner window. The bitmap will be stretched to fill the screen, and therefore the painting will often be slow. The banner text will then be displayed on top of the bitmap. If a bitmap file is specified the painting of the banner window background, as specified in the **$BANNER-WINDOW-BRUSH** line will not take place, since the bitmap will be occupying the whole screen. Only standard Windows .BMP files are supported.

This line is **optional**. It also depends on the file **WINSTALP.DLL**. That file contains all the **banner** functionality, and it's presence is not needed for Chief's Installer Pro to function (you just won't get any banner). If the file is not found by Chief's Installer Pro, this line will have no effect. It is also ineffective if the **$WINDOW=MAXIMIZE** reserved word is used.

$BITMAP can take an extra optional parameter **NORMAL**. If used, this should appear **AFTER** the name of the bitmap file, separated by a semi-colon (e.g., $BITMAP=winstall.bmp;normal). This parameter disables the stretching of the bitmap, and the bitmap will be displayed in its normal size, centered on the screen. In such cases also, the main dialog will be hidden once the "Start Install" button is clicked - so that more of the bitmap will be visible.

This bitmap line is there only as an added extra. The "bitblitting" is often very slow, has problems with large bitmap files, and so many not be ideal in many cases. But it is there anyway. You may simply ignore it. If the line is empty, then Chief's Installer Pro will by default look for a file called **WINSTALL.BMP** to use for the background bitmap. If **WINSTALL.BMP** is not found, then the program simply use the banner window brush value to paint the background of the banner window.

The Syntax is;

$BITMAP=<bitmap file name>[;NORMAL]

EXAMPLES:

$BITMAP=MYPROG.BMP
$BITMAP=WINSTALL.BMP;NORMAL

See also;

$BANNER-MESSAGE
$BANNER-TEXT-COLOR
$BANNER-TEXT-BACKGROUND
$BANNER-SHADOW-COLOR
$BANNER-WINDOW-BRUSH
$WINDOW

$VERIFY-INSTALL-DISKS

This line is **optional**. By default, when the user is prompted to insert a particular numbered disk (e.g., "Please insert disk 4 in drive"), Chief's Installer Pro does not perform any check to verify that the disk being inserted is actually the correct one. You can however use this reserved word to force Chief's Installer Pro to perform these checks.

When this reserved word is used, Chief's Installer Pro will check each installation disk (from disk 1 onwards) to verify that it is the correct disk. This check is performed by looking for a file on the disk, which corresponds to the **$DISK#** being installed, but with the extension **.DSK**. Thus for example, if Chief's Installer Pro asked for **disk 4** to be inserted in the drive, it will check for the existence of a file called **$DISK4.DSK** on any disk that is inserted. If the file exists, then this is taken as the correct disk, and the installation continues. If the file is not found on the disk, then the user is prompted to insert the disk again, and this will go on until either the correct disk is inserted, or the user clicks on "Abort".

The contents of the **.DSK** file are normally irrelevant. It can be an empty file - but the file must exist. However, you can optionally provide for the contents of the .DSK file to be checked. To do this, you need to supply 'READ-FILES' as a parameter. If this feature is turned on, then the installer will treat the .DSK file for every disk as a Windows .INI file, and check in the **[disk-id]** section for the keyname 'disk-id'. The entry here must match the name of the .DSK file being examined - e.g., a .DSK file called $DISK1.DSK must have the following entries;

```
[disk-id]
disk-id=$DISK1.DSK
```

Note that if the entry does not match, the installer will assume that the disk is the wrong one and will keep prompting for the disk. Thus, you should use this feature very carefully.

EXAMPLES:

$VERIFY-INSTALL-DISKS
$VERIFY-INSTALL-DISKS;READ-FILES

See also;
$DISK

$AUTO-CLICK-BUTTON

The main window of Chief's Installer Pro has got four push buttons, labelled (in English) "Start Install", "Abort", "Help", and "View Read Me". You can use this reserved word to send a mouse click to any one of these buttons. When you send a mouse click in this way, the effect is exactly as if the user had clicked on that push button with the left mouse button. This will activate whatever the push button is supposed to do.

The line takes one parameter - the ID of the button to send the mouse click to. For this purpose, **1=Start Install**, **2=Abort** , **3=Help**, and **4=View Read Me**. This line is useful for example, for starting the installation without giving the user any opportunity to make any selections or choices, or for clicking on the "Help" or "Readme" button so that your help file or your readme file (WINSTALL.TXT) will be loaded automatically (i.e., to force your users to read your documentation).

This line is optional.

The Syntax is;

$AUTO-CLICK-BUTTON=<button ID>

EXAMPLE:

$AUTO-CLICK-BUTTON=1

See also;

COMMAND LINE OPERATION

$NO-CTL3D.DLL

This line is **optional**, and is probably not very useful. It is for the purpose of giving people the option of dispensing with the use of CTL3DV2.DLL. If this line is found, the Chief's Installer Pro will not use the 3D dialog effects in CTL3DV2.DLL. The question is "why would anyone want to do this?". The answer is that some Windows video drivers are buggy and might not necessarily want to co-exist peacefully with CTL3DV2.DLL in all circumstances.

Not mentioning any names, but I know of one company which produces buggy Windows drivers for their display cards which sometimes fall into the category described above. I personally do not use this reserved word, and it may indeed be unnecessary to use it. However, I think that it is good to have the option.

This line takes no parameter.

$RESTARTWIN-MESSAGE

Chief's Installer Pro provides support for restarting Windows if any active shared file was overwritten. A dialog asking for confirmation appears automatically if any active shared file was overwritten during the installation. The text on this dialog can be changed by this reserved word. If this reserved word is not used, a default message is used, which tells the user that at least once active DLL has been replaced, and that the user should restart Windows immediately.

The message on this line can be up to 200 characters. Obviously, that is too wide for a dialog box. Therefore I have decided to support **one** formatting control here. You can insert carriage returns at any point in the message by using the "newline" code (i.e., **\n**). If a literal "\n" is desired, an exclamation mark should precede the "n" (i.e., "\!n"). Furthermore, the "\n" is case sensitive - so, for example, "\N" will not be converted.

This line can be used to DISABLE the dialog that asks if the user wants to restart Windows. To disable the feature, use **$RESTARTWIN-MESSAGE=DISABLE**.

Please note that if you disable this feature in this way, it is up to you to inform your user that Windows must be restarted when an active DLL has been overwritten. How you will ascertain this fact is beyond me. I have only included this feature because a user asked for it. If you use it, you are on your own.

EXISTING-FILES

Chief's Installer Pro will check in the target directories for existing copies of every file being installed. If no copy of the file exists, then the installation will proceed. If the file exists, the existing copy and the copy being installed will both be checked to see which one is newer.

Chief's Installer Pro uses two methods of deciding whether or not a file is older than another. In the case of **shared binary files** (i.e., the ones that go into the Windows and/or the Windows SYSTEM directory), the version information in the files will first be compared. If there is no version information in the files, then their date stamps of the file will be compared. In the case of other files, only the date stamps will be compared (except in the case of proprietary DLLs if you use the **$CHECK-MY-DLL-VERSIONS** reserved word).

When Chief's Installer Pro has ascertained which of the two copies of a file is newer, what happens next depends on the choices you made in your INF file, and/or the choices made by your user. Normally, Chief's Installer Pro will simply display a dialog box informing the user that a copy (or a newer copy) of the file already exists in the target directory, and then show the user the details of the two copies.

You can decide in advance that certain files specified by you should be over-written automatically (**$AUTO-REPLACE**) or that all **older** versions of files should be over-written automatically (**$FORCE-OVERWRITE-OLDER-FILES**). In this case, a file is regarded as "older" if it is **not newer**. In the case of files which both have version information, if the version number is the same (e.g., they are both 1.1) then the date stamps will be used to decide which is "older". If the date stamps are the same, then the one that already exists in the target directory is treated as "older" than the one being installed. In the case of files without version information, if both files have the same date stamp, then the existing copy is still regarded as "older".

SHARED FILES;

For the purposes of the installation, Chief's Installer Pro will regard a file as a **shared** file only if the following conditions apply;

[1] the file is being installed into the Windows or Windows SYSTEM directory, and
[2] the file's extension is either; [a] **.DLL** or [b] **.EXE** or [c] **.VBX** or [d] **.OCX** or [e] **.DRV** or [f] **.CPL**

See also;

$AUTO-REPLACE
$CHECK-MY-DLL-VERSIONS
$FORCE-OVERWRITE-OLDER-FILES
$SKIP-IDENTICAL-FILES
$VERSION-INFO-TITLE
$VERSION-INFO-MESSAGE
$VERSION-COPY-ERROR-MESSAGE

$CHECK-MY-DLL-VERSIONS

This line is **optional**. By default, when another copy of a **.DLL** file already exists in the target directory, Chief's Installer Pro will check for version information in that DLL only if the target directory is the Windows directory or the Windows SYSTEM directory. That is, only **SHARED** DLLs will be normally checked for version information. DLLs which are going into your application's directory for example will only be checked for their date stamps.

If you want **ALL** DLL files to be checked for their version information (i.e., regardless of their destination directory), then you should use this reserved word. This line takes no parameter.

See also;
EXISTING FILES

$VERSION-INFO-TITLE

This line is **optional**. By default, when Chief's Installer Pro is reporting the version information on an existing copy of a shared file, the version number is reported under the heading "File Version". You can use this reserved word to change that string to something else. This is really useful only for those who want to display that string in a language other than English. If you change this string, please try to make the replacement as short as possible.

The Syntax is;

$VERSION-INFO-TITLE=<title>

EXAMPLE:

$VERSION-INFO-TITLE=Product Version Number

See also;

EXISTING FILES
$VERSION-INFO-MESSAGE

$VERSION-INFO-MESSAGE

This line is **optional**. When Chief's Installer Pro has retrieved the version information on an already existing copy of a shared file, a dialog informs the user that a copy of the file already exists and then asks for over-write permission. By default, this dialog will not contain any further explanation of the situation, and will not make any recommendation as to the course of action to be taken.

You can use this reserved word to provide some explanation and/or a recommended course of action. If anything appears on this line, it will be added to the dialog.

The message on this line can be up to 200 characters. Obviously, that is too wide for a dialog box. Therefore I have decided to support **one** formatting control here. You can insert carriage returns at any point in the message by using the "newline" code (i.e., **\n**). If a literal "\n" is desired, an exclamation mark should precede the "n" (i.e., "\!n"). Furthermore, the "\n" is case sensitive - so, for example, "\N" will not be converted.

The Syntax is;

$VERSION-INFO-MESSAGE=<recommendation/explanation>

EXAMPLE:

$VERSION-INFO-MESSAGE=You should click on "NO" \n if the target file is NEWER.

See also;

EXISTING FILES
$VERSION-INFO-TITLE

$FINAL-MESSAGE

If you want to give your user any final message (after the installation is complete) then you can put that message on this line. The message will be displayed in a dialog box at the tail end of the installation - just after the **$EXEC** line is executed.

The message on this line can be up to 200 characters. Obviously, that is too wide for a dialog box. Therefore I have decided to support **one** formatting control here. You can insert carriage returns at any point in the message by using the "newline" code (i.e., **\n**). If a literal "\n" is desired, an exclamation mark should precede the "n" (i.e., "\!n"). Furthermore, the "\n" is case sensitive - so, for example, "\N" will not be converted.

The Syntax is;

$FINAL-MESSAGE=<message>

EXAMPLE:

$FINAL-MESSAGE=Please shut down all applications and restart Windows.

$VERSION-COPY-ERROR-MESSAGE

This line is **optional**. By default, when Chief's Installer Pro is unable to successfully install a file, all that the user will get is an error message that there was an error writing to the file, and then the installation will proceed with the other files.

In the case of **shared files**, the problem may be that the file is currently in use and therefore cannot be over-written. With **shared files**, a temporary copy will normally exist in the TEMP directory (and will **NOT** have been deleted by the installer if the attempt to install it was unsuccessful). Chief's Installer Pro will therefore by default display a message advising the user to copy the file manually after closing Windows.

Note that the situation described above will only be exist, if [a] the file is a shared file, and, [b] a copy of it already exists in the Windows or Windows SYSTEM directory, and, [c] an attempt to install over the existing copy was unsuccessful.

You may want to change this message described above to something that suits you, and this reserved word allows you to do that. A carriage return will automatically be added at the end of this message, followed by the full pathname of the temporary copy of the file.

Note that because the last thing that appears in the dialog box is the full path name of the temporary file in the TEMP directory, if you use this line to change the error message, you need to express it in such a way that it leads up to the file name.

The message on this line can be up to 200 characters. Obviously, that is too wide for a dialog box. Therefore I have decided to support **one** formatting control here. You can insert carriage returns at any point in the message by using the "newline" code (i.e., **\n**). If a literal "\n" is desired, an exclamation mark should precede the "n" (i.e., "\!n"). Furthermore, the "\n" is case sensitive - so, for example, "\N" will not be converted.

The Syntax is;

$VERSION-COPY-ERROR-MESSAGE<message>

EXAMPLE:

```
$VERSION-COPY-ERROR-MESSAGE=Please copy it from the TEMP directory
later \n. A temporary copy exists as:
```

See also;

EXISTING FILES

$FORCE-OVERWRITE-OLDER-FILES

This line is **optional**. By default, when a copy of the file being installed already exists in the target directory, Chief's Installer Pro will ask the user whether the existing copy should be over-written or not. This will be the case even when the existing file is an older version which really ought to be replaced. This could be a bit of a nuisance sometimes, and so you might want older versions of files to be replaced automatically. You use this reserved word to achieve that.

Note that this reserved word is different from the **$AUTO-REPLACE** reserved word, in that, this one applies to all files, while the former applies only to selected files.

This line takes no parameter.

See also;
EXISTING FILES
$AUTO-REPLACE
$CHECK-MY-DLL-VERSIONS
$SKIP-IDENTICAL-FILES

$SKIP-IDENTICAL-FILES

This reserved word is **optional**. It causes a file to be skipped if a copy of it already exists in the target directory, and that copy is exactly the same version as the copy on the installation disk. In order to decide whether two files are exactly the same version, their date/time stamps, file sizes, and version information (if the files are shared DLLs) are compared. If there is any discrepancy in any of these, the files are treated as not being the same, and will not be skipped.

The comparisons work correctly in all my tests - but if you are going to use this feature, **please test it thoroughly** with your particular set of files.

See also;
$AUTO-REPLACE
$FORCE-OVERWRITE-OLDER-FILES

$README-BUTTON-TEXT

Chief's Installer Pro provides support for displaying a README file to the user before the installation begins. The readme file should be a plain ASCII file, should not be larger than 8192 bytes, and should be called **WINSTALL.TXT**. A button with the caption "View Read Me" is presented for this purpose. You can change the caption (text) on the "readme" button with this reserved word. The text used here must not be longer than 20 characters.

You can cause the contents of the file to be displayed automatically by setting **$AUTO-CLICK-BUTTON** to 4 (otherwise the user will have to click on the "readme" button to display the text). If the file WINSTALL.TXT is not found, then the "readme" button will be removed at run time.

The Syntax is;

$README-BUTTON-TEXT=<button caption>

EXAMPLE:

$README-BUTTON-TEXT=&Installation Notes

See also;

$README-FONT

$README-FONT

This reserved word is **optional**. It is used to change the font in which the text in the "Readme" dialog is displayed, from a proportional font (MS Sans Serif, 9 point) to a FIXED or MONO spaced font (Courier 8 point).

The Syntax is;

$README-FONT=FIXED

See also;

$README-BUTTON-TEXT

$REG-DATA

This reserved word is **optional.** It provides support for making entries into the Registration Database. You can have an unlimited number of **$REG-DATA** lines, and each line can contain only a single entry. The lines can contain only the keys/sub-keys that you want to create (e.g., associating a file extension with your program, etc). In this case, the installer will prepend **HKEY_CLASSES_ROOT** to each of your entries, but you may wish to provide the full entry, including the root key yourself (NOTE: ** This is a 16-bit program, so please do NOT use any root key other than HKEY_CLASSES_ROOT under Win95 or Windows NT). Full support for the additional Win95 and Windows NT roots will be provided when a Win32 version of Chief's Installer Pro is released.

Entries made in the registration database in this way willl be removed by the uninstaller if and when the user chooses to uninstall the program.

The Syntax is;

$REG-DATA=[ROOTKEY\]<subkey>=<value>

EXAMPLES:

$REG-DATA=HKEY_CLASSES_ROOT\ChiefPro = Chief's Installer Pro
$REG-DATA=.inf = ChiefPro
$REG-DATA=.chf = ChiefPro
$REG-DATA=ChiefPro\shell\open\command = $DEST\install.exe %1
$REG-DATA=ChiefPro\shell\print\command = $DEST\install.exe /p %1
$REG-DATA=ChiefPro\protocol\StdFileEditing\verb\0 = Edit
$REG-DATA=ChiefPro\protocol\StdFileEditing\server = $DEST\install.exe

$DIALOG-ICON

This reserved word is **optional**. It can be used to change the icon that is displayed on the installer's dialogs. The reserved word takes one parameter - a number which corresponds to that of the required icon. The icons must be in a DLL called **WINSTALC.DLL**, and must be given given **numeric names from 3 onwards** (e.g., 3, 4, 5, 6, etc.). Icons 1 and 2 (5.25 and 3.5 inch icons) are inside INSTALL.EXE itself. There can be up to 250 icons in **WINSTALC.DLL**, but only one can be used. If this reserved word is not used, the the default will be the 3.5" icon.

The Syntax is;

$DIALOG-ICON=<icon number>

EXAMPLE:

$DIALOG-ICON=4

$FONT

This reserved word is **optional**. It is used to install TRUETYPE fonts. The reserved word takes 2 parameters. The first is the name of the font file (xxx.TTF) which will be used, and the second is the name or description of the font. Note that the description of the font must be accurate - exactly as it appears in the Windows Control Panel. The program will try to create a **.FOT** file in the Windows SYSTEM directory, and make the necessary entries in the Windows INI files. If the font is already installed, it will simply be reinstalled again.

You can have an unlimited number of **$FONT** lines. Note that you need to direct the font files to the Windows SYSTEM directory with the **$SYSDIR** command.

The Syntax is;

$FONT=<fontfile.TTF>;<font name>

EXAMPLES:

$FONT=ARIALBD.TTF;Arial Bold (True Type)
$FONT=CHIEFBD.TTF;Bold Chief (True Type)

$SOURCEDIR

This reserved word is **optional**. It points to the directory from which the program was actually installed, but can also be used to change the source directory from the default (useful for internal corporate customisations).

EXAMPLES:

```
$INI=$DEST\PROG.INI;History;SourceDir;$SOURCEDIR
$SOURCEDIR=F:\USR\LOCAL\BIN\NEWPROG
```

See also;

$ABORT-MESSAGE

This is the message that will appear to the user when the user does something to terminate the installation (e.g., clicking on the desktop "Abort" button) after the installation has commenced.

The Syntax is;
$ABORT-MESSAGE=<message>

EXAMPLE:

$ABORT-MESSAGE=Do you really want to stop?

See also;
$NO-ABORT-BUTTON

$ABORT-UNINSTAL-QUESTION

This is the question that the user will be asked, if while running the uninstaller, the user clicks on the 'Close' option in the system menu of the uninstall dialog.

The Syntax is;

$ABORT-UNINSTAL-QUESTION=<question>

EXAMPLE:

$ABORT-UNINSTAL-QUESTION=Sure you want to stop the uninstall?

$AUTOEXEC.BAT

The purpose of this (obsolecent in view of Windows 95) is to make entries into the user's AUTOEXEC.BAT file. The installer willl search for the AUTOEXEC.BAT file, first, in the root directory of drive C:, and then in the directories in the "PATH". If it is not found in any of these places, then nothing will happen. There is **NO LIMIT** to the number of $AUTOEXEC.BAT lines that you can have in your WINSTALL.INF file. You can use the $DEST reserved word here.

The Syntax is;

$AUTOEXEC.BAT=<entry>

EXAMPLES:

$AUTOEXEC.BAT=SET PATH=%PATH%;$DEST\BIN
$AUTOEXEC.BAT=SET TROOK=C:\TROOK\TROG

$BATCH-FILE

This allows you to specify a Chief's Installer Pro **batch file** which the installer will execute during the course of the installation. Only one file name is allowed on each $BATCH-FILE line. In **WINSTALL.INF**, you can have an unlimited number of $BATCH-FILE lines. However, in **SETUPINF.INF**, you can only have one $BATCH-FILE line.

The Syntax is;

$BATCH-FILE=<filename>

EXAMPLE:

```
$BATCH-FILE=$SOURCEDIR\FIRST.CHF
$BATCH-FILE=$DEST\SECOND.TXT
```

See also;

BATCH COMMANDS
BATCH FILES

$BIG-METER-COLOR

This changes the colour of the big percent meter. It takes the same syntax as $TEXT-BACKGROUND

The Syntax is;
$BIG-METER-COLOR=<value>

EXAMPLE:

$BIG-METER-COLOR=$00808080

See also;
$TEXT-BACKGROUND

$CANCEL-BUTTON-TITLE

This can be used to change the caption of the "Cancel" buttons on the text entry dialog boxes. This is obsolescent will often be overridden by the new string resource #543.

The Syntax is;
$CANCEL-BUTTON-TITLE=<caption>

EXAMPLE:

$CANCEL-BUTTON-TITLE=&Forget it

See also;
$OK-BUTTON-TITLE

$CLOSE-GROUP-BOX

Group boxes (with internal numeric IDs of 1 to 9) exist around different controls in the main dialog window. This reserved word allows you to close any or all of these group boxes, by supplying their numeric IDs as parameters (separated by semi-colons);

The Syntax is;

$CLOSE-GROUP-BOX=<id number[s]>

EXAMPLE:

$CLOSE-GROUP-BOX=1;5;6

$DATA-SPACE

This is the space requirement for any non-optional data that your app might need to create at installation time. It will not form part of the calculations for the percent meter - but it will be added internally to the required free space when the installer is checking whether there is sufficient space on the target drive.

The Syntax is;
$DATA-SPACE=<value>

EXAMPLE:

$DATA-SPACE=2048000

See also;
$SPACE

$DISKDIR#

A source directory path can (optionally) be specified for the files on each disk (one path only for each disk). This is specified by using the $DISKDIR reserved word. If no source directory is specified, it is presumed that the disk's files are to be installed from the general source directory.

The Syntax is;

$DISKDIR#=<dir name>;[CODE]

In this sceranio, "#" stands for the number of the disk. "CODE" is optional. When used, it specifies whether (for disk 2 to the end) the installer should prompt the user to insert a disk. It defaults to **NOT** prompting (to simplify installing from cd-rom or network drives). If you want the user to be prompted for a particular disk, put **ASK**, or **PROMPT**, or just **1** as the code. Note that "CODE" is not valid for disk 1.

NOTES:

1. If the specified directory is NOT found, the installer will prompt the user for a disk.

2. The installer will still automatically look for $DISK# directories under the specified directory

3. This feature is optional, and is not needed at all. It should be used sparingly, since it has not been tested in all possible scenarios. It is **NOT** advised to use this feature when installing from FLOPPY disks (i.e., it is added mainly for the convenience of those who wish to install from cd-rom, or hard disk). Using it in respect of floppy disks will **NOT** be supported by me.

EXAMPLES:

```
$DISKDIR1=F:\USER\LOCAL\INSTALL
$DISKDIR2=Z:\;ASK
```

See also;

$DISK

$FORCE-EXIT-WINDOWS

This takes no parameter. If it is used, then Windows will be closed down at the end of the installation, without giving the user any say in the matter. Note that this feature uses the **EXITWINDOWS()** API call. Thus it will fail if any program refuses to terminate (e.g., if the user has a DOS session open). Note also that if you use this command, it is advisable to inform your user of what will happen, with the **$FINAL-MESSAGE** command.

See also;
$FINAL-MESSAGE
$FORCE-RESTART-WINDOWS

$FORCE-RESTART-WINDOWS

This takes no parameter. If it is used, then Windows will be closed down and restarted at the end of the installation, without giving the user any say in the matter. Note that this feature uses the **EXITWINDOWSEXEC()** API call. Thus it will fail if any program refuses to terminate (e.g., if the user has a DOS session open). Note also that if you use this command, it is advisable to inform your user of what will happen, with the **$FINAL-MESSAGE** command.

See also;
$FINAL-MESSAGE
$FORCE-EXIT-WINDOWS

$ICO

This is an alternative to the **$ICON** command. This one gives you more control over the icon creation process, and also has the advantage that there is **no limit** to the number of icons you can create with it (cf9**$ICON** has a limit of 128 entries), but this way of creating icons is a more low level approach. Note that icon placement in existing groups is sometimes misaligned when using the $ICO command.

The $ICO command take a number of parameters (up to 11, each separated with a semi-colon, or a comma) to control the details of the icons being created.

The Syntax is;
$ICO=<parameters>

Parameters

1 = the group name (or $GROUP for the default)
2 = allow duplicate icons? (0 for NO; 1 for YES)
3 = command line
4 = icon's title
5 = file to load the icon from (optional)
6 = icon index in the icon file (optional)
7 = x position of the icon in group (optional)
8 = y position of the icon in group (optional)
9 = working directory (optional)
10= hotkey (optional)
11= minimize (0 for NO; 1 for YES) (optional; defaults to NO)

Any parameter marked as optional does not have to be supplied. Just supply a comma or a semi-colon instead. "Hotkey" consists of a number, made up of a code for a system key (CTRL, ALT, or SHIFT) plus the ascii code of the letter to be used with it.

Hotkey Codes:

Alt = 1024
Ctrl = 512
Shift = 256

So, if for example you want to use **Alt+S** as the hotkey, then: 1024+83 (i.e., 1107) would be the number.

EXAMPLES:

```
$ICO=$GROUP;0;$DEST\TROOK.HLP;Trook Help;$DEST.ICO;0;;;$DEST;1107;1
$ICO=$GROUP;0;$DEST\TROOKCFG.HLP;Trook CFG Help
$ICO=Trook Trog;0;$DEST\TROOKTRG.EXE;The Great Trook;;;;;;1108;
```

See also;
$ICON

$LAN-SYSDIR

This was introduced by request. Normally, if the Windows **SYSTEM** directory is not a subdirectory of the Windows directory, this is an indication of a networked Windows environment, and the shared files that are to go to the Windows SYSTEM directory will in such cases normally be installed to the user's Windows directory (in many networks the SYSTEM directory will be read-only).

If you want all these precautions (which after all are only following Microsoft documentation) to be circumvented and for the shared files to still to the Windows SYSTEM directory on the network server, then use this reserved word.

Note that **if you use this command, you are on entirely your own, and I will answer no questions if something goes wrong**. If the SYSTEM directory on the network server cannot be written to for some reason, your installation will fail, and that, probably woefully. **You have been warned!!!**

The Syntax is;

$LAN-SYSDIR=$SYSDIR

That is the only way in which this reserved word will work.

$MAX-DUPLICATES

This command was introduced by request. It allows the installation of the same file into more than one destination directory. The command takes one parameter - the maximum number of times a single file can be duplicated in this way. By default, this value is set at 1. You can increase it to any number up to 30.

Note that increasing this number will mean the installer looping through each $DEST\xx directory the specified number of times, for EACH file being installed. This is guaranteed to slow down the installation considerably, if the specified number is too high. Using it can also lead to multiple copies of files which you do not really want to be duplicated - this is especially so if you use wilcards at all, in your $DEST\xx lines.

Thus, if you are only going to install a few files into more than one directory, it is better to avoid this command altogether, and **use the COPY command in a batch file.** In fact, it is **not** recommended to use this $MAX-DUPLICATES command at all. Nothing can be achieved by it that cannot be better achieved by using the COPY command in a batch file.

The Syntax is;
$MAX-DUPLICATES=<number>

EXAMPLE:

$MAX-DUPLICATES=4

See also;
$BATCH-FILE
$DEST

$NO-ABORT-BUTTON

This specifies that the "Abort" button (on the desktop) should not be displayed. This command takes no parameter.

See also;
$NO-HELP-BUTTON

$NO-HELP-BUTTON

This specifies that the "Help" button (on the desktop) should not be displayed. This command takes no parameter. Note that if the file **WINSTALL.HLP** is not found, then "Help" button will not be displayed anyway.

See also;
$NO-ABORT-BUTTON

$OK-BUTTON-TITLE

This can be used to change the caption of the "OK" buttons on the text entry dialog boxes. This is obsolescent will often be overridden by the new string resource #542.

The Syntax is;
$OK-BUTTON-TITLE=<caption>

EXAMPLE:

$OK-BUTTON-TITLE=&Go on!

See also;
$CANCEL-BUTTON-TITLE

$OPTIONHELP#

This is to present a brief explanation of the user-options to the user. Each user-option will have a small button next to it, if there is an OPTIONHELP for that option. Clicking on that button will display a message box with your brief explanation.

Each user option may have up to 10 $OPTIONHELP# lines - but an absolute maximum limit of 1024 bytes applies to each user option. Realistically, this maximum should be 512 bytes, because of limits in the MessageBox() API when using CTL3D.

The Syntax is;

$OPTIONHELP#=<help text>

EXAMPLES:

$OPTIONHELP1=These files are absolutely necessary for \nthe program to work.
$OPTIONHELP2=These are optional bitmap and icon files.

See also;

$OPTIONAL
$USER-OPTION

$SMALL-METER-COLOR

This changes the colour of the small percent meter. It takes the same syntax as $TEXT-BACKGROUND

The Syntax is;
$SMALL-METER-COLOR=<value>

EXAMPLE:

$SMALL-METER-COLOR=$00808080

See also;
$TEXT-BACKGROUND

$UNZIP

Chief's Installer Pro features support for unzipping files. The UNZIP support is compatible with **PKZIP** (tm) 2.x ZIP archives. This command is used to unzip a file during the installation. You have have an unlimited number of $UNZIP lines in your INF file, each of them specifying a single file. For the $UNZIP command to work, **the file WINSTALZ.DLL must be present on your DISK #1**, and if there is more than one disk in your installation set, **your users MUST run SETUP.EXE, and not INSTALL.EXE** - otherwise, the unzip **will** most certainly fail.

The $UNZIP command can also be used in the $DISK lines, to specify that a ZIP archive should be unzipped from a particular installation disk. In this case, the files in the ZIP archive will only go to the directory supplied as a parameter to $UNZIP.

Please note that if the ZIP archive contains sub-directories, the directory structure inside the ZIP archive will be created/restored in the target directory. This behaviour cannot be changed.

Please note also that this command **cannot** handle ZIP archives which are split across more than one disk.

The Syntax is;

$UNZIP=<zipfile>;<target directory>;<CODE>

or, if used on a $DISK# line;

$DISK#=$UNZIP;<zipfile>;<target directory>;<CODE>

"Code" specifies how to deal with files which already exist in the target directory. Possible values are;

[a] **OVERWRITE-ALL** (overwrite all existing files without warning)

[b] **OVERWRITE-OLDER** (overwrite only older files [by date-stamp])

[c] **SKIP** (skip existing files)

[d] **CONFIRM** (ask for confirmation before overwriting existing files)

EXAMPLES:

```
$UNZIP=$SOURCEDIR\BIN.ZIP;$DEST\BIN;SKIP
$UNZIP=$TEMPDIR\TROOK.001;$DEST\TROOK;OVERWRITE-OLDER
$DISK3=$UNZIP;$TEMPDIR\TROOKCFG.002;$DEST\TROOKCFG;CONFIRM
```

See also;
$DISK

Batch Files

Chief's Installer Pro provides support for running commands via batch files. For this purpose, Chief's Installer Pro implements a batch language and batch commands which are roughly similar to the DOS commands and batch language, but which are different in some respects.

Batch commands provide a very effective means of extending the functionality of Chief's Installer Pro. Since it is impossible to envisage in advance all the possible scenarios in which the installer will be used, this type of functionality presents the best approach to increasing the programs's flexibility.

The batch features are activated by the **$BATCH-FILE** reserved word. This specifies the name of a batch file. Each batch file can contain an **unlimited** number of lines - but note that the bigger a batch file is, the longer it will take to read it. **Each line in a batch file can be up to a maximum of 200 characters**.

NOTES:

1. Whatever is done by these batch commands will NOT be undone by the uninstaller.
2. Batch files are executed in the order in which they appear, and are processed immediately after the files have been installed (after $ini, $fonts, $reg-data, and $autoexec.bat, but before $pre-exec, and $exec)
3. Certain reserved words are invalid when used in a batch file that is run from SETUPINF.INF - e.g., **$DEST**, and **$TARGET**, since they might be changed by the user after the main install dialog is loaded. The same thing goes for **$SOURCEDIR**, which might sometimes be changed by the user.

See also;
BATCH COMMANDS
$BATCH-FILE

BATCH COMMANDS

Chief's Installer Pro batch files support sundry commands. Many of them operate like their DOS, OS/2, or Windows NT counterparts, but most are different in various respects. The main point is that if you are familiar with the DOS internal commands, many of these commands will also be familiar to you. Below is a list of batch commands, and a summary of their syntax, and what they do.

WRITEBAT
WRITEINF
WRITEINI
WRITETEXT

See also;
BATCH FILES

#CONST or $CONST

This command is used to define some **GLOBAL constants** in a Chief's Installer Pro batch file. The defined constants then apply throughout the batch file. Wherever Chief's Installer Pro encounters the defined constant in the batch file, it is replaced by the value which you assigned to it. Such constants should be defined **at the beginning of the batch files**. You can have an unlimited number of #CONST lines.

This command is similar to the #DEFINE command, but is different in a very important respect - with #DEFINE, parts of any word that matches will be replaced - however, with #CONST, **only whole words will be replaced**.

The changes are done in memory - so the physical contents of the batch file are unaltered.

Restrictions and features;

1. #CONST can only be used **ONCE** for any particular constant in a batch file.

2. Each #CONST entry must be on a line by itself.

3. You cannot use one #CONST constant in the definition of another one.

4. If you use the constant's name (or any part of its name) as part of its value, it will be taken as a literal value.

5. A value assigned to a constant with $DEFINE can be used in defining the value of a $CONST constant. **This is one important advantage over using $DEFINE.**

The syntax is:

#CONST <constant> = <value>

Examples;

#CONST OLDDIR=$WINDIR\PROG\OLD

#CONST MYCOMMAND=$DEST\BIN\MYPROG.EXE

See also;
#DEFINE

#DEFINE or $DEFINE

This command is used to define some **GLOBAL constants** in a Chief's Installer Pro batch file. The defined constants then apply throughout the batch file. Wherever Chief's Installer Pro encounters the defined constant in the batch file, it is replaced by the value which you assigned to it. Such constants should be defined **at the beginning of the batch files**. You can have an unlimited number of #DEFINE lines.

The changes are done in memory - so the physical contents of the batch file are unaltered.

Restrictions;

1. $DEFINE can only be used **ONCE** for any particular constant in a batch file.

2. Each #DEFINE must be on a line by itself.

3. **You cannot use one defined constant in the definition of another one.**

4. **You CANNOT use the constant's name (or any part of its name) as part of its value. However, you can use the (already defined) constant's name in a constant that you are defining with the #CONST command**.

5. Each constant defined in this way **MUST be entirely unique**. This is because partial matches will be changed as well. Thus, for example, you cannot have one constant called **DIR** and another one called **DIREC**, if you are going to use #DEFINE with **DIR**. This is because when occurences of **DIR** are being changed, the places where those letters occur in **DIREC** (i.e., the first 3 letters of **DIREC**) will be changed as well - and this is probably not what you want. Also, defining a constant as **COMMAND** and another one as **COMMAND2** or **MYCOMMAND** is not advisable, because they have the string **"COMMAND"** common to them all. **Please note this point**.

GOOD **TIP:** If you must use similar names, the best thing to do is to interpose another character after the FIRST character of the common parts. For example, you could have a constant called **COMMAND**, and others called **C1OMMAND, C2OMMAND, C3OMMAND, etc.**

ANOTHER **TIP:** Most of these restrictions do not exist when you use the **#CONST** command. That command is identical to this one, except that it only changes **whole words**.

The syntax is:

#DEFINE <constant> = <value>

Examples;

#DEFINE OLDDIR=$DEST\BIN\OLD

#DEFINE COMMAND=C:\4DOS\4DOS.EXE

The following examples are NOT allowed;

#DEFINE COPY=COPY *.* A:

#DEFINE COMMAND=COMMAND.COM

#DEFINE COMM=C:\DOS\COMMAND.COM

This is because they represent an attempt to use the constants which are being defined (or part of the constants' names) in the values assigned to the constants. **This will either lead to an almighty CRASH, or to unpredictable and random results**.

See also;
#CONST

#INCLUDE or $INCLUDE

This allows you to import the contents of another Chief's Installer Pro batch file into the currently running batch file, at the place where the command was used. This is the **only** way in which you can call a Chief's Installer Pro batch file from another one.

Note that while nested includes may sometimes work, they are **NOT** supported (i.e., please do not #INCLUDE a file into another one which was itself called with the #INCLUDE command).

The Syntax is;
#INCLUDE <filename>

EXAMPLE:

#INCLUDE $TEMPDIR\PROG2.CHF

APPENDFILE

This is one of the text file commands. It adds a string to text to the end of a text (ASCII) file. If the specified file does not exist, a new one is created.

The Syntax is;
APPENDFILE <filename>;<string>

EXAMPLE:

APPENDFILE C:\AUTOEXEC.BAT;SET TROOKTROG=C:\TROOKT

See also;
WRITETEXT

ATTRIB

Change and/or set the attributes bits of a file. Attribute bits can be concatenated (separated with a semi-colon). The plus sign ("+") turns on an attribute bit, and the minus sign ("-") turns it off. In this respect, "R" = read only; "S" = system file; "A" = archive; "H" = hidden.

The Syntax is;

ATTRIB <filename>;<attr>[;other attrs]

EXAMPLE:

ATTRIB $DEST\TROOK.CNF;+R;-H;-A;+S

BEEP

Make a beeping noise on the PC's speaker. This calls the Windows **MessageBeep()** API function.

CD or CHDIR

Change to another directory.

The Syntax is;
CD <new directory>

EXAMPLE:

CD $DEST\BIN\SOURCE

See also;
MD
RD

COPY or CP

Copy a file (or a group of files) from one location to another. This command simply copies the files (compressed files are not expanded).

This command takes two parameters - the source file and the target file/directory. If the source file is a single file name (i.e., no wildcards) then the second parameter can be a file name or a directory. If the source filename contains wildcards, the second parameter **MUST** be a directory. Error codes are returned in **ERRORCODE**.

The Syntax is;

COPY <source-file> [;] <dest-file>

EXAMPLE:

```
COPY   $DEST\TROOK.*   $DEST\TROOK\BACKUP
```

See also;

EXPANDFILE

CREATEFILE

This creates a new file (with zero bytes). If any file of the same name already exists, the existing file is overwritten. Please note this point. The only purpose of using this is to ensure that a log of the file is kept in UNINSTAL.LOG so that the uninstaller can delete it if necessary (i.e., if you are creating a new file on the system with batch commands, you can call CREATEFILE first, and then use any other command (e.g., APPENDFILE, or WRITETEXT).

The Syntax is;

CREATEFILE <filename>

EXAMPLE:

```
CREATEFILE $DEST\TROOK\TROOK.CFG
```

See also;

APPENDFILE
WRITETEXT

DEL or RM

Delete a file or a group of files. Note that while wildcards are allowed, the wildcard "*.*" will **NOT** be accepted. The user will get a hard-coded error message in English.

The Syntax is;

DEL <filename>

EXAMPLES:

DEL $TEMPDIR*.$$$
DEL $DEST\MYFILE.BAK

DELAY

Pause for some time. This command takes one parameter - the number of **SECONDS** to wait for. If no parameter is supplied, there will be a pause for one second.

The Syntax is;

DELAY <seconds>

EXAMPLE:

DELAY 5

DISPLAY

Shows a modal dialog box in which you can display some text (e.g., while doing some other stuff in the background). This command takes one parameter - either the text to be displayed in the dialog, or **OFF** (to turn off the display dialog). Note that the display dialog does not have any buttons, and has to be removed by **"DISPLAY OFF"**.

The Syntax is;

DISPLAY <message>

or

DISPLAY OFF

EXAMPLE:

```
DISPLAY I am now processing files \n \n Please wait ...
```

EXEC or RUN

Run a program. Processing of the batch file continues as soon as the program is executed.

The Syntax is;
EXEC <program> [program parameters]

EXAMPLE:

EXEC NOTEPAD.EXE $DEST\README.TXT

See also;
EXECHIDDEN
EXECWAIT

EXECHIDDEN or RUNHIDDEN

Run a program, with it's main window hidden. Processing of the batch file continues as soon as the program is executed. Note that the program to be executed must be self-terminating, since your user will have no way of terminating it. It will **NOT** show up in the task list. This command is useful for running something (like a time-stamping program) behind the scenes.

The Syntax is;

EXECHIDDEN <program> [parameters]

EXAMPLE:

EXECHIDDEN $DEST\CONFIG.EXE /NewInstall

See also;

EXEC
EXECWAIT

EXECWAIT

Run a program. Processing of the batch file will stop until the program is closed. This will fail if used to run DOS sessions under **OS/2** (i.e., processing will continue immediately - just like the EXEC command).

The Syntax is;
EXECWAIT <program> [parameters]

EXAMPLE:

EXECWAIT NOTEPAD.EXE $TEMPDIR\README.NOW

See also;
EXEC
EXECHIDDEN

EXIT or RETURN

Exit from the currently running batch file. The installer goes to the next stage. This command takes no parameter.

See also;
HALT

EXITWINDOWS

This command takes no parameter. If it is used, then Windows will be closed down immediately, without giving the user any say in the matter. Note that this command uses the **EXITWINDOWS()** API call. Thus it will fail if any program refuses to terminate (e.g., if the user has a DOS session open).

See also;
EXITWINDOWSEXEC

EXITWINDOWSEXEC

Shut down Windows, run a DOS program, and then restart Windows again. This command takes one parameter - the name of the DOS program to be executed. It is **NOT** recommended to use this command at all, since if it is used that is really the end of your installation - Windows will restart, and just return the user to whatever happens to be the Windows shell.

Note that this command uses the **EXITWINDOWSEXEC()** API call. Thus it will fail if any program refuses to terminate (e.g., if the user has a DOS session open).

See also;
EXITWINDOWS

EXPANDFILE or LZEXPAND

Copy a file (or a group of files) from one location to another. If a file is compressed (with Microsoft's COMPRESS.EXE) the file will be expanded (using the functions in LZEXPAND.DLL).

This command takes two parameters - the source file and the target file/directory. If the source file is a single file name (i.e., no wildcards) then the second parameter can be a file name or a directory. If the source filename contains wildcards, the second parameter **MUST** be a directory. Error codes are returned in **ERRORCODE**.

The Syntax is;

EXPANDFILE <source-file> [;] <dest-file>

EXAMPLE:

```
EXPANDFILE $DEST\WORK\TROOK.C?? $DEST\TROOK
```

See also;

COPY

FOR

This is the **FOR loop**. It runs a specified command for each file in a set of files.

The Syntax is;

FOR %variable IN (set) DO command [command-parameters]

EXAMPLE:

FOR %i IN ($DEST*.TXT) DO NOTEPAD.EXE %i

GOTO

Jump to a pre-defined label in the batch file. Labels, when defined, must begin with a colon (":") followed immediately with the label's name. When using GOTO, you must not include the colon. Processing continues after coming to the end of the label, unless the end of the label contains a jump to another label (with GOTO).

The Syntax is;
GOTO <label>

EXAMPLE:

```
GOTO END
```

HALT

Terminate the installation. This not only exits the batch file currently running, it also closes down the installer itself. Use with care!!!

See also;
EXIT

IF CHOICE

This command allows the user to choose any one out of up to 10 predefined options (a radio button will be presented for each option). The options are a series of strings, separated by semi-colons (and numbered automatically), the last of which will be the text prompting for the choices. If any of the buttons is checked, and the user clicks on "OK" the condition returns TRUE, the number of the selected option will be returned in CHOICE, and the command attached to the condition will be executed (see the sample batch file **SETUP.CHF** for an example of this being used to ask for the user's language).

The Syntax is;

IF CHOICE "<choices;prompt>" <command> [CHOICE]

EXAMPLE:

```
IF CHOICE "Mono;Colour;Please choose your monitor type" GOTO CHOICE
IF CHOICE "AMD;Intel;Cyrix;Select your CPU vendor" COPY CHOICE.CPU $DEST
```

IF CONFIRM

This allows you to prompt the user for a YES or NO. It presents a message box with a question (posed by you) and if the user clicks on YES, then the condition returns TRUE and the attached command is executed. If the user clicks on NO, then the condition returns FALSE and the attached command is ignored.

The Syntax is;
IF CONFIRM "<question>" <command>

EXAMPLE:

```
IF CONFIRM "Should I abort the install?" HALT
```

See also;
IF NOT-CONFIRM

IF CPU

This allows you to test for the microprocessor (CPU) inside the user's PC, without any input from the user. If the user's CPU is the one specified by you, then the condition returns TRUE. Possible CPU values are **80386**, **80486**, or **P5**

The Syntax is;
IF CPU <cpu-value> <command>

EXAMPLE:

IF CPU P5 Say you have a Pentium processor!

See also;
IF HAS-FPU

IF DISKFREE

This allows you to test for the amount of free disk space on the current drive. This takes an operator as the first parameter, and the size you are testing for, as the second parameter. For the operator, you can use either the "greater than" (">") or the "less than" ("<") symbols. Whether the condition returns TRUE or not depends on the operator used. The "size" parameter should be a whole number, in bytes.

The Syntax is;

IF DISKFREE <operator> <size> <command>

EXAMPLES:

IF DISKFREE > 1024 SAY Free Space is Greater than 1kb!
IF DISKFREE < 2048000 GOTO ABORT

IF ERRORCODE

The result of every batch operation is returned in an internal variable called **ERRORCODE**. This is so that you can receive some (rudimentary) feedback on each command (since you will not receive any error message if a command fails). You can test for the value of ERRORCODE after every command.

Possible ERRORCODE values;

[a] **0** = operation successful; no error

[b] **1** = syntax error; the command was not executed at all

[c] **-1** = some processing error or the other; this is what to watch for!

The Syntax is;

IF ERRORCODE <code> <command>

EXAMPLE:

IF ERRORCODE -1 SAY The last command failed!

See also;

IF NOT-ERRORCODE

IF EXIST

Test for whether a file exists. If the file exists, the condition is TRUE and the attached command is executed.

The Syntax is;
IF EXIST <filename> <command>

EXAMPLE:

IF EXIST C:\AUTOEXEC.BAT APPENDFILE C:\AUTOEXEC.BAT LOADHIGH $DEST\ TROOK.SYS

See also;
IF NOT-EXIST

IF FSIZE

Tests whether the size of a file is exactly the same as the size you specify. If so, the condition returns TRUE. The "size" parameter should be a whole number, in bytes.

The Syntax is;
IF FSIZE <filename> <size> <command>

EXAMPLE:

IF FSIZE $DEST\PROG.EXE 23494 GOTO SAFE

See also;
IF NOT-FSIZE

IF HAS-FPU

This does not require any input from the user. If the user's computer has got a Maths Co-processor (a floating point chip), the condition returns TRUE, and the attached command is executed. It returns FALSE if there is no maths co-processor.

The Syntax is;
IF HAS-FPU <command>

EXAMPLE:

```
IF HAS-FPU SAY You have a Maths Chip!
```

See also;
IF CPU

IF INPUT

This presents an input dialog to the user, where the user can enter some text, in answer to a prompt. If text is entered and the user clicks on "OK" then the condition returns TRUE. The text entered by the user is returned in a variable called **INPUT**, which you can use on the same line only.

In another permutation, you can also check for the text that was entered, by using the "==" operator.

The Syntax is;

IF INPUT "<prompt>" <command> [INPUT]

or

IF INPUT "<prompt>" == <string> <command> [INPUT]

EXAMPLES:

```
IF INPUT "Please enter your TROOK filename" NOTEPAD.EXE INPUT
IF INPUT "Your ID., please:" == FRED GOTO CONTINUE
```

IF ISDIRECTORY

Tests for the existence of a directory.

The Syntax is;
IF ISDIRECTORY <directory-name> <command>

EXAMPLE:

IF ISDIRECTORY C:\DRAG CD C:\DRAG

See also;
IF NOT-ISDIRECTORY

IF NOT-CONFIRM

This allows you to prompt the user for a YES or NO. It presents a message box with a question (posed by you) and if the user clicks on NO, then the condition returns TRUE and the attached command is executed. If the user clicks on YES, then the condition returns FALSE and the attached command is ignored.

The Syntax is;
IF NOT-CONFIRM "<question>" <command>

EXAMPLE:

```
IF NOT-CONFIRM "Should I abort the install?" GOTO CONTINUE
```

See also;
IF CONFIRM

IF NOT-ERRORCODE

Allows you to test for the value of ERRORCODE after each batch operation. See the **IF ERRORCODE** command for full description.

The Syntax is;
IF NOT-ERRORCODE <code> <command>

EXAMPLE:

IF NOT-ERRORCODE 0 SAY An error has occured!

See also;
IF ERRORCODE

IF NOT-EXIST

Tests for the existence of a file. It returns TRUE if the specified file does NOT exist.

The Syntax is;
IF NOT-EXIST <filename> <command>

EXAMPLE:

IF NOT-EXIST TRAGG.CNF CREATEFILE TRAGG.CNF

See also;
IF EXIST

IF NOT-FSIZE

Tests for the size of a file. If the file's size does not exactly match the specified size, the condition returns TRUE.

The Syntax is;
IF NOT-FSIZE <filename> <size> <command>

EXAMPLE:

IF NOT-FSIZE PROG.EXE 53333 SAY This file may have a VIRUS!!!

See also;
IF NOT-FSIZE

IF NOT-ISDIRECTORY

Tests for the existence of a directory. Returns TRUE if the directory does NOT exist.

The Syntax is;

IF NOT-ISDIRECTORY <directory-name> <command>

EXAMPLE:

```
IF NOT-ISDIRECTORY $DEST\BAK MD $DEST\BAK
```

See also;

IF ISDIRECTORY

IF SLANGUAGE

Tests for the value of the "sLANGUAGE" setting in the "INTL" section of the WIN.INI file. If there is an entry, the condition returns TRUE and the value in that setting is returned in an internal variable called **SLANGUAGE** which can be used on the same line.

The Syntax is;

IF SLANGUAGE [== STRING] <command> [SLANGUAGE]

EXAMPLES:

IF SLANGUAGE GOTO SLANGUAGE
IF SLANGUAGE == ENG SAY Your Windows speaks English!

IF VMODE

Allows you to test for the user's display driver mode, without any input from the user. Possible values are **CGA**, **EGA**, **VGA**, **SVGA**, or **SSVGA**. The first three speak for themselves. "SVGA" stands for 800*600 and "SSVGA" stands for 1024*768, or higher screen resolutions.

The Syntax is;

IF VMODE <value> <command>

EXAMPLE:

IF VMODE SVGA SAY Your display is 800 * 600

IF WINVER

Allows you to test for the version of Windows which the user is running. There are a number of ways to test for Windows versions (see the sample batch file **CHIEFPRO.CHF** for details), but the numeric method (1 to 11 - starting from Windows 3.0 to Windows NT) may be the easiest.

1 = Windows Version 3.00
2 = Windows Version 3.10 (no network)
3 = Windows for Workgroups 3.10 (networked)
4 = Windows Version 3.10 (with Win32s, no network)
5 = Windows for Workgroups 3.10 (with Win32s, networked)
6 = Windows Version 3.11 (no network)
7 = Windows for Workgroups Version 3.11 (networked)
8 = Windows Version 3.11 with Win32s
9 = Windows for Workgroups 3.11 (with Win32s, networked)
10 = Windows 95
11 = Windows NT

The Syntax is;

IF WINVER <code> <command>

EXAMPLE:

IF WINVER 10 SAY You are running Windows 95

LOADCTL3D

Loads CTL3DV2.DLL or CTL3D.DLL (in that order of preference), if either of them is found on the system. For each call to this command, there must be a corresponding call to **UNLOADCTL3D**.

See also;
UNLOADCTL3D

MD or MKDIR

Creates a directory.

The Syntax is;
MD <directory name>

EXAMPLE:

MD $DEST\TRRUTT

See also;
RD
RD

RD or RMDIR

Removes a directory.

The Syntax is;
RD <directory name>

EXAMPLE:

RD $DEST\TRRUTT

See also;
CD
MD

REN or MV

Renames a file. This command takes two parameters - the old name of the file, and the new name of the file. Please note that you cannot rename a file across drives - and that it is preferable to supply the full pathnames for the files.

The Syntax is;

REN <old-name> <new-name>

EXAMPLE:

REN $DEST\FRED.TAR $DEST\FRED.GZ

SAY or ECHO

Displays a message in a standard Windows message box.

The Syntax is;
SAY <message>

EXAMPLE:

SAY Hello World!

UNLOADCTL3D

Unloads CTL3DV2.DLL or CTL3D.DLL (if it had been loaded with the LOADCTL3D command). Please do not use this command unless you have previously used the LOADCTL3D command. This command takes no parameter.

See also;
LOADCTL3D

UNZIP

UNZIPs a ZIP archive. The parameter are the same as those taken by the **$UNZIP** reserved word. Please check the documentation on $UNZIP.

The Syntax is;

UNZIP <zip-file>;<dest directory>;<code>

See also;

$UNZIP

WRITEBAT

Writes an entry into a Chief's Installer Pro batch file (at run-time) This takes 2 parameters, separated by a semi-colon - the name of the batch file to write into, and the string to write into the batch file. The file will be written to (transparently) in the appropriate format (i.e., compiled or ASCII), so it does not matter whether the file has been compiled or not.

The Syntax is;
WRITEBAT;<batch file>;<string>

EXAMPLE:

WRITEBAT $DEST\REG.CHF;SAY You are not registered!

See also;
WRITEINF

WRITEINI

Writes an entry into an INI file. This takes the same parameters as the **$INI** reserved word, except that (unlike $INI) it cannot take a USER-OPTION as a parameter. Please see the documentation on that reserved word.

See also;
$INI

WRITEINF

Writes an entry into a Chief's Installer Pro INF file (at run-time) This takes 2 parameters, separated by a semi-colon - the name of the INF file to write into, and the string to write into the INF file. The file will be written to (transparently) in the appropriate format (i.e., compiled or ASCII), so it does not matter whether the file has been compiled or not.

The Syntax is;
WRITEINF;<INF file>;<string>

EXAMPLE:

```
WRITEINF $TEMPDIR\WINSTALL.INF;$CLEANUP=$DEST\*.CHF
```

See also;
WRITEBAT

WRITETEXT

Writes a string into a text (ASCII) file. This command takes three parameters - the name of the text file to write to; the line number to write to (or "last" to append to the file); and the string to write into the file.

The Syntax is;
WRITETEXT <filename>;<line number>;<string>

EXAMPLES:

```
WRITETEXT $DEST\TTT.TXT;1;This is the first line
WRITETEXT $DEST\TTT.TXT;LAST;This is the last line
```

See also;
APPENDFILE

COMMAND LINE OPERATION

Chief's Installer Pro normally operates in an interactive way. When the program **INSTALL.EXE** is run, a dialog box will be presented to the user, from where the user can select options, click on a button to start the installation, etc. While this is sufficient in most cases, there are situations in which you might want to use your own "pre-installer" (e.g., instead of my own **SETUP.EXE** or for any other purpose). Note that this option is **not** open to you if you are using **SETUP.EXE**.

For such situations, Chief's Installer Pro provides you with the flexibility of running **INSTALL.EXE** with command line parameters. There are three types of parameters that the program can take, and you can use one, or all, or some, or none of them, in any combination.

The first is **/$TARGET=<target directory>**. When this parameter is used, Chief's Installer Pro will assume that all the options which you have enabled in your INF file have been accepted, and will by-pass the first dialog - the installation will start straight away, without the user having the opportunity to select or unselect any of the options manually. This is useful if you want to ensure that the installation is carried out in a particular way (e.g., to ensure a standard setup on all computers in your company). If you are using this parameter, it should be the **first** one that is supplied.

Another parameter you can supply is **the name of the INF file** to use for the installation. Chief's Installer Pro defaults to **WINSTALL.INF**. You can however specify another file name for this purpose. If this is used, it should be the first parameter (if the **/$TARGET=** parameter is not used) or the second parameter (if **/$TARGET=** is used). This parameter will be taken as **paramstr(1)** or **argv[1]**, because INSTALL.EXE does not include the **/$TARGET=** switch in the count of command line parameters.

The final parameter which you can supply is the **source directory**. You cannot use this parameter without using the one which specifies the name of the INF file. If this parameter is used, it should be the last one. If you specify an INF file, it is advisable to also use this parameter to specify the source directory for the installation. This parameter will be taken as **paramstr(2)** or **argv[2]**.

Note that when you choose to run Chief's Installer Pro in this way, the program will faithfully do whatever you say, and will not necessarily verify any of these parameters. This option is provided for added flexibility - but if you use it, you are on your own, and it us up to you to make sure that your program does all the necessary authentification of the parameters you are passing.

In my opinion, it is far better to run Chief's Installer Pro in the normal way, but to allow the installer to click on the "START INSTALL" button automatically, by using the **$AUTO-CLICK-BUTTON** reserved word (with a parameter of **1**). That way, all the normal internal checks would have been carried out, and you would be able to use SETUP.EXE.

The Syntax is;

INSTALL.EXE [/$target=<target dir>] [<INF filename> <source dir>]

EXAMPLES:

INSTALL.EXE /$target=C:\CHIEFPRO
INSTALL.EXE /$target=C:\CHIEFPRO C:\TEMP\CHIEF.INF
INSTALL.EXE /$target=C:\CHIEFPRO C:\TEMP\CHIEF.INF A:
INSTALL.EXE A:\CHIEF2.INF
INSTALL.EXE B:\CHIEF2.INF B:

See also;
$AUTO-CLICK-BUTTON

THE UNINSTALLER

Many Windows programs are easy to install, but most are not so easy tp remove, because of INI files and DLLs thrown all over the place. Users who wish to uninstall their programs face a hazardous task in which they may delete the wrong files, or remove the wrong entries in INI files. This may then make Windows unusable, necessitating an expensive re-install of Windows.

What this means is that many Windows users are reluctant to try out new programs on their systems because of hassles of removing the programs if they don't want them any more. **This may mean that people will never ever get to see your wonderful program :)**. Some other people have to spend a lot of money on commercial uninstallers, which attempt to snoop round the system. The fact that they are trying to undo someone else's work means that this is often a hit-or-miss affair - sometimes leading to an expensive program not getting the job done properly. **Enter the UNINSTALLER!**.

Chief's Installer Pro includes an "uninstall" program (**UNINSTAL.EXE**). Which will undo anything that Chief's Installer Pro did. If you used the **$MAKE-UNINSTALL-LOG** reserved word in your WINSTALL.INF file, Chief's Installer Pro will create a log file called **UNINSTAL.LOG** in the target directory. This file contains details of every change made to the system by Chief's Installer Pro.

The user can subsequently "uninstall" your program by running **UNINSTAL.EXE**. The uninstall program will read the log file **UNINSTAL.LOG** and use its contents to undo everything it did during the installation. This includes deleting the installed files, any directories created by Chief's Installer Pro, any Program Manager groups or icons created by Chief's Installer Pro, any entries made into INI files by Chief's Installer Pro, etc.

Having an uninstaller is one of the requirements for the Windows 95 logo!!!

Note that, unless the INI files are in your program's home directory, the INI files themselves will **NOT** be deleted. Only the entries made into them by Chief's Installer Pro will be deleted. This might result in "orphan" INI files - i.e., INI files with nothing inside them. I believe that this is preferable to deleting all the INI files themselves. This is because entries could have been made into existing INI files, and deleting such files will be disastrous. Therefore, I leave it to users to delete any orphan INI files manually ("better safe than sorry" is the motto here - and, also, "beware Murphy's law").

Shared files (DLL, VBX, and DRV files) installed into the Windows or Windows SYSTEM directory are a special case. If a copy of such files already existed when the installation was running, no log will be made of them, and the uninstaller will NOT delete them (this is because they were obviously not put there by the installer). In cases where no copy of the file was found at install time, the file will be logged, and the uninstaller will delete it - but after asking the user for confirmation. This is because although the file was put there by the installer, it may (after the user has installed other programs) by needed by other programs. This is very often the case with Visual Basic applications, and applications which use BWCC.DLL and/or CTL3Dxx.DLL.

I believe that the uninstaller is a good marketing point for your program. First, there is the perception that a person who provides an uninstaller with his program must be very confident about the program itself. Also, users have nothing to lose by trying your program, since removing it is simply a matter of clicking on the icon for the uninstaller, and then supplying the home directory of the program to be uninstalled. If

the uninstaller does not find the log file in that directory, it aborts with an error message. If the file is found, the user is given ONE opportunity to confirm that he or she does really want to uninstall the program.

The Uninstaller can take optional parameters. The first is the home directory of the program to be uninstalled. This means that you can pass your program's directory (**$DEST**) as a parameter to **UNINSTAL.EXE** when you are creating your program's icons with the **$ICON** command.

e.g: **$ICON=$DEST\UNINSTAL.EXE $DEST;Uninstall my Program!**

The second parameter that the uninstaller can take is the name of the LOG file to use for the uninstall. This parameter is normally optional (the program will default to UNINSTAL.LOG). But note that if a filename was supplied as a parameter to the $MAKE-UNINSTAL-LOG reserved word, then this parameter becomes **MANDATORY** here (i.e., you must supply that filename as a **second** parameter to UNINSTAL.EXE when you create the icons with the **$ICON** reserved word). **Please note this point.**

e.g: **$ICON=$DEST\UNINSTAL.EXE $DEST VER2.LOG;Uninstall my Program!**

Support for non-English languages is provided for the uninstaller by the use of string tables. These can be compiled into a DLL which must be called **UNINST.DLL**. If this file is not found at run time (it must be in the same directory as UNINSTAL.EXE) then the default English string table inside UNINSTAL.EXE will be used.

A copy of the English language version of the resource script (**UNINST.RC**) is provided for you to translate to your chosen language. Please note that if you choose to create your own translations and put them in the DLL, you are on your own.

The uninstaller will optionally uninstall the program in such a way that the deleted files cannot be undeleted. To enable this feature, use the parameter **OVERWRITE** in the **$MAKE-UNINSTALL-LOG** line. Note that when this is used, uninstalled files and directories cannot be undeleted, no matter what is attempted. They are first deleted, then they are overwritten by a 1 byte file, and then that file is deleted. Any such file that can be undeleted (in most cases, no file can be undeleted) will only delete to a file containing 1 byte (of garbage data). Therefore, it is **not** recommended that this feature should be used. It is only there because of a specific request, and if you use it, you are on your own.

See also;
$ICO
$ICON
$MAKE-UNINSTALL-LOG

CREDITS

Many thanks to the following:

1. **Claus Ziegler**, ZieglerSoft, Denmark - a great Windows guru! Thanks for everything, and for the Danish translations of the string tables.

2. **Joachim Rehmet** and **Juergen Kneifel** - for the German translations of the string tables.

3. **Drs. Bob Swart** - for the Dutch translations of the string tables.

4. **Gary W.Rohn** - for sharing the source code to your FI program.

5. **Agustin Cernuda** - for the Spanish translations of the string tables.

6. **Bimmer (Per Bakkendorff)** - for the Norwegian and Swedish translations of the string tables.

7. **Antoine Desir** and **Claude Daneluzzo** - for the French translations of the string tables.

8. **Frederico Berrino** - for the Italian translations of the string tables.

9. **Dr Abimbola Olowofoyeku (The African Chief)** - for the Yoruba translations of the string tables.

DISCLAIMER

I do NOT warantee ANYTHING concerning any of the programs or files which make up "Chief's Installer Pro for Windows". I accept NO RESPONSIBILITY for ANY LOSS OR DAMAGE of ANY kind, including, but not limited to, losses of a physical, mental, social, financial, marital, or of whatever nature, resulting from the use, or the purported use of Chief's Installer Pro for Windows", or any of the files in the package, for any purpose whatsoever. I do not even warantee that the programs will not kill you. You use Chief's Installer Pro for Windows ENTIRELY AT YOUR OWN RISK, and you supply it to your customers, friends, family, acquaintances, or enemies, ENTIRELY AT YOUR OWN RISK.

If these terms are NOT acceptable to you, then you have no licence to use or test Chief's Installer Pro, and you should DELETE all the program's files from all your disks immediately AND PERMANENTLY.

FEEDBACK

Okay. I am keen to obtain feedback, especially from registered users. I also welcome suggestions for features. I cannot promise to implement every suggestion, but at least, I will consider the ideas. If you have any comments, ideas, suggestions, etc., or you just want to tell me how wonderful the program is :) then please feel free to contact me by e-mail. I will try to respond if a response is appropriate.

You can contact me by e-mail at the following internet addresses:

laa12@potter.cc.keele.ac.uk

chief@mep.com

UPDATES

This program is being constantly updated. I will endeavour to release bug fixes as often as I receive bug reports and fix them. However, it is rather difficult to spread the word about new releases and updates. There are a number of internet ftp sites which are not usually busy and to which I can therefore upload new versions. You might want to check these places from time to time.

FTP SITES (and directories)

ftp.demon.co.uk /pub/ibmpc/windows/chief/pro

micros.hensa.ac.uk /micros/ibmpc/win/e/e022

COMPUSERVE

WINSHARE LIB 4

MSBASIC LIB 2

REGISTRATION

Chief's Installer Pro is distributed under the **Shareware** principle. It can be copied and distributed freely, as long as **ALL** the supplied files, including documentation (this file) are included, and **NO ATTEMPT** is made to modify any of the files.

The Shareware principle means that you get a chance to **EVALUATE** the program free of charge for a reasonable period of time (in the case of Chief's Installer Pro, **a maximum of 14 days**). It does not mean that you will NOT have to pay for the program.

This program is NOT crippled in any way, and you only get nagged when you use the (optional) **Chief's Installer Pro compiler**. What this means is that you now have the FULL version of Chief's Installer Pro. Nothing is disabled, there are no extra files, and there is no written manual. All the documentation is in the .HLP file (**CHIEF.HLP**) and the .WRI file (**CHIEF.WRI**).

I felt that releasing the full version in this way was necessary in order to enable people to fully evaluate the program, being that they will be seeing exactly what the program is. It also helps to ensure that when you do register, you do not have to wait for days or weeks to receive your "registered copy".

However, this approach also means that I am relying totally on people's honesty to register. Chief's Installer Pro is a tool to help programmers concentrate on their products by not having to worry about installation routines, thereby improving their productivity. A lot of time and effort has gone into this program, and I am not asking for much, considering that users will be getting a royalty-free license, which covers all their Windows applications! If you find Chief's Installer Pro useful and would like to continue using it, or you would like to use it as the installation routine for your own programs, then I would encourage you to **please REGISTER your copy**.

BENEFITS OF REGISTRATION

1. You will receive a serial number and a registration code which will remove the nags from the compiled INF and batch files, and which will entitle you to a upgrades to future shareware releases of Chief's Installer Pro, up till (**but not including**) the next MAJOR release. MAJOR upgrades will be numbered in whole numbers, and attract an upgrade fee of 50% of whatever is the prevailing registration fee. Minor upgrades will be numbered in .10 increments, and will be free.

2. Freedom to use Chief's Installer Pro as the installation program for an unlimited number of your own applications.

3. A clear conscience.

4. Support (**via e-mail**) for the program.

5. A chance to have an input into the features of future versions (I will **not** accept suggestions for new features from anyone who has not registered).

6. Not having to worry that I will discover that you have used the package as the installation routine for your program without registering <g>

7. You will be very cool indeed.

REGISTRATION FEE:

£39.99 (U.K. STERLING)

$59.99 (U.S.)

$74.99 (Canadian)

$80.00 (Australian)

Kr359.99 (Danish)

Please NOTE that ALL prices are subject to change WITHOUT NOTICE.

Please **NOTE also that the correct fee (including taxes/duties/charges where appropriate) must be sent in all cases**. If a necessary duty/tax is not sent, the order can not be processed.

I would also encourage registrants to please PRINT their registration requests CLEARLY.
We have in the past received several registration requests that were not legible, especially the e-mail addresses. This is especially important for low-grade faxes. If we cannot read your details, then it also follows that we will neither be able to process your registration, nor even to contact you to inform you of the problem.

See also;
REGISTRATION SITES

COMPUSERVE
AUSTRALIA, NEW ZEALAND, ASIA, THE FAR EAST
CANADA, USA
USA
EUROPE
UNITED KINGDOM, IRELAND, EUROPE, EVERYWHERE ELSE
CREDIT CARD ORDERS

REGISTRATION SITES

Below are the registration sites for Chief's Installer Pro. Please send your registration request to the registration site that is most convenient for you. Some of the registration sites cannot accept credit card orders. If you are paying by credit card please check carefully that the registration site you are dealing with can accept payment by credit card.

YOU CAN SEND THE REGISTRATION FEE TO ANY OF THE FOLLOWING REGISTRATION SITES;

COMPUSERVE
AUSTRALIA, NEW ZEALAND, ASIA, THE FAR EAST
CANADA, USA
USA
EUROPE
UNITED KINGDOM, IRELAND, EUROPE, EVERYWHERE ELSE
CREDIT CARD ORDERS

Please fill the REGISTRATION FORM below.

REGISTRATION FORM

COMPUSERVE

On-line registration is available under the SWREG scheme. If you **GO SWREG**, the Registration ID is **7557**.

See also;
CREDIT CARD ORDERS

AUSTRALIA, NEW ZEALAND, ASIA, THE FAR EAST

Please send orders from these areas to:

DAVID PERKOVIC
DP Computing
P.O.Box 712
Noarlunga Center
SA 5168
Australia

Internet: dpc@adam.com.au
dpc@mep.com

Tel: +61 8 326 4364

Mobile: +61 015 973 503

Fee: $80.00 (Australian funds)

NOTES:

1. **Method of payment: Cheques, Money Orders**
2. Make cheques/money orders payable to: "DP Computing".

CANADA, USA

Please send orders from these areas to:

Minds Edge Productions Inc.
P. O. Box 211
3456 Dunbar Street
Vancouver, BC V6S 2C2
Canada

Internet: info@mep.com

Fidonet: 1:153/709

WWW: http://haven.uniserve.com/~shane/mep.html

Fee: $59.99 (US funds)

or: $74.99 (Canadian funds)

NOTES:

1. **Method of payment: Checks, Money Orders (U.S. or Canadian funds)**
2. Make cheques/money orders payable to: "Minds Edge Productions Inc.".
3. **British Columbia residents should add 7% sales Tax.**
4. **Canadian Residents should add 7% GST.**

USA

Please send orders from the USA to:

TODD MERRIMAN
Software Toolz, Inc.
8030 Pooles Mill Dr.
Ball Ground,
GA 30107
U.S.A.

Fax: 770-887-5960

Internet: software@toolz.atl.ga.us

Fee: $59.99 (US funds)

NOTES:

1. **Method of payment: Checks, Money Orders, Visa, Mastercard, American Express.**
2. **Georgia residents should add the appropriate Sales Tax**

See also;
CREDIT CARD ORDERS

EUROPE

Please send orders from EUROPE to:

HENRIK MOERK
Survival BBS
P.O.Box 1538
DK-2700 Bronshoj
Denmark

Tel: +45 3 889 5253

FIDO: 2:231/306

Internet: Eurovga@ibm.net

Fee: Kr 359.99 (Danish funds)

NOTES:

1. **Method of payment: Cheques, Eurocheques, Money Orders, VISA, Mastercard/Eurocard, GIRO, Danish DANKORT, and JCB card**
2. **GIRO: 1-207-4247**
3. Make cheques/money orders payable to: "HENRIK MOERK".
4. **EC residents should add 25% VAT.**

See also;
CREDIT CARD ORDERS

UNITED KINGDOM, IRELAND, EUROPE, EVERYWHERE ELSE

Please send orders from these areas to:

JOHN BARTON
57 Baddeley Green Lane
Baddeley Green
Stoke on Trent
Staffs, ST2 7JL
ENGLAND.

Internet: laa12@keele.ac.uk
chief@mep.com

Compuserve: 100306,1334

Fee: £39.99 (U.K. funds; or equivalent)

NOTES:

1. **Method of payment: Cheques, Eurocheques, Money Orders**
2. Make cheques/money orders payable to: "JOHN BARTON".
3. **ADD**: £5.00 (**if sending a foreign cheque**; note that foreign cheques that do not contain this fee will **not** be processed)

CREDIT CARD ORDERS

Please note that of all these registration sites, **the only ones which can process CREDIT CARD orders are**;

[a] **Compuserve**,
[b] **Todd Merriman**, and
[c] **Henrik Moerk**.

If you will not be registering on-line on Compuserve, please send all credit card orders either to Todd Merriman, or Henrik Moerk.

NOTE: If you want to send your credit card orders to **Todd Merriman** by e-mail, please put your details (as per the registration form) in an ASCII file, and then run the program **TOTOOLZ.EXE** on the file. This will encrypt the contents of the file. You can then safely e-mail the (encrypted) output file to Todd Merriman. This precaution is to protect your credit card details from internet hackers.

See also;
REGISTRATION SITES

To register Chief's Installer Pro, please PRINT and FILL IN the following Registration FORM.

NOTE: **Please specify your CURRENT version of Chief's Installer Pro**.

TO:

_____

_____

_____

_____

I wish to REGISTER my copy of "Chief's Installer Pro".

My current version is _____

I am ordering _____ **copies**

I am paying the REGISTRATION FEE of _____

ADD Tax (if applicable) _____

(See info on the registration sites to see if they collect tax)

Total FEE: _____

I am paying by **Cheque/Money Order/Credit Card (delete as inapproriate)**

NAME _____

ADDRESS _____

_____

_____

POST/ZIP CODE _____

E-MAIL _____

How did you get your copy of Chief's Installer Pro?

_____

IF PAYING BY CREDIT CARD, PLEASE SEND THE FOLLOWING DETAILS;

(NOTE: Not all sites accept credit cards so please refer to the list of REGISTRATION SITES)

CARD ISSUER _____

CARD NUMBER _____

DATE OF ISSUE _____

EXPIRY DATE _____

SIGNATURE _____

DATE _____

PLEASE REMEMBER TO SIGN THE CREDIT CARD ORDER!

TECHNICAL SUPPORT

1. **Technical support can only be provided for people who HAVE registered**. Please note therefore that no support can be provided for anyone who has not registered (not even if you promise me that **"the cheque is in the post"**), and that I can answer no questions from anyone who has not registered. This is so that those who **have** paid can get the support that they deserve and have paid for.

2. **Please note also that NONE of my registration sites can provide technical support**. Therefore, customers are asked to please not telephone or fax any request for technical support to any of my registration sites. Rather, **all** requests for technical support should be sent to **Dr A Olowofoyeku** (address below). The preferred medium of communication is **electronic mail**.

3. **Please read the CHIEF.FAQ file before sending requests for technical support**. The file might very well contain the answer to your query.

4. **Please send requests for technical support by e-mail to**:

   [a] **laa12@keele.ac.uk**

OR,

   [b] **chief@mep.com**

If you do not have e-mail facilities, then please send queries by post to:

Dr A A Olowofoyeku
268 Horwood
Newcastle

STAFFS, ST5 5BQ

ENGLAND.

GOLD AWARD

Chief's Installer Pro was winner of the **PC PLUS Magazine's GOLD AWARD** (U.K. edition, April 1995).