Freeman Installer & Uninstaller User Manual Contents

Freeman Installer & Uninstaller is a shareware package to facilitate general software installation and uninstallation tasks under Microsoft Windows.

If you don't know how to use help, press F1 now.

Disclaimer

This software is provided "as is" without representation or warranty of any kind, either expressed or implied, including without limitation, any representations or endorsements regarding the use of, the results of, or performance of the software, its appropriateness, accuracy, reliability, or correctness. The entire risk as to the use of this software is assumed by the user. In no event will Ka Iok Tong be liable for any damages, direct, indirect, incidental or consequential resulting from any defect in the software, even if Ka Iok Tong has been advised of the possibility of such damages. This disclaimer shall supersede any verbal or written statement to the contrary. If you do not accept these terms you must cease and desist using this software immediately.

Copyright & License

Freeman Installer & Uninstaller

version 1.1 beta

Freeman Installer & Uninstaller version 1.1 beta is copyrighted software. It is not public-domain software.
The author hereby grant anyone who accepts the conditions stated in this section the rights to,

- use this software as an install program; and/or
- ship this software with his/her own application(s) as an install program to install those application(s) only; and/or
- distribute this software under the same conditions stated in this section without charging more than the cost of shipping & handling.

No other rights is granted. In particular, it is illegal and prohibited to,

- modify this software without the author's written permission except those explicitly stated in this document; and/or
- distribute any derived work of this software as a competitive product.

If you don't accepts these conditions, it is required that you stop keeping this software.

The directory where install.exe or llatsni.exe resides. For install.exe, it is usually a:\. For llatsni.exe it is always the Windows directory.

Most of your files will be copied to this directory or its subdirectories.

How Are install.inf and llatsni.inf Read In?

Basically the meaning of the entries given in install.inf and llatsni.inf are independent of the order they are read in by install.exe and llatsni.exe with the exception of macros.

The way install.exe deals with install.inf is,

- Immediately after install.exe is started, macros called "s", "w", "y" will be bound to the source directory, Windows directory, Windows system directory respectively. As a result, they can be referenced as $s, $w, $y in the install.inf.
- Execute the macro definitions given in section [macros1]. Usually you define most of your own macros here. You can introduce some shortcuts to save typing and more importantly, youcan retrieve the value of entries in some other INI files such as win.ini. However, you can not reference the target directory ($i) because it has not been defined.

  The macros defined here can be referenced in the rest of install.inf.
- Read in the entries given in section [general]. You can reference the macros $s, $w, $y and/or those defined in [macros1]. One example is basing your default target directory on, say, the Windows directory,

  ```
  [general]
  defDir = $w\mytarget
  ```
- Ask the user for the install target directory and bind the macro called "i" to it. As a result, it can be referenced as $i in the rest of install.inf.
- Execute the macro definitions given in section [macros2]. You can reference the target directory now. Usually you only define here those macros whose value depend on the target directory. One example is saving the target directory in a INI file,

  ```
  [macros2]
  {$i\cshwin.ini, location, homeDir} = $i
  ```

  The macros defined here can be referenced in the rest of install.inf.
- Read in the entries given in section [disks], [files] and [items]. You can reference all macros already defined. One example is referencing $i in section [files],

  ```
  [files]
  cshwin.dll  = $i\lib, 58384, 0, C Shell DLL, 0, 0
  ```

  In this case cshwin.dll will be copied to a subdirectory called "lib" under the install target directory.

The way llatsni.exe deals with llatsni.inf is,

- Immediately after llatsni.exe is started, macros called "s", "w", "y", "i" will be bound to the source directory, Windows directory, Windows system directory, install target directory espectively. Because llatsni.exe is always started from the Windows directory (see Uninstallation Behavior), $s and $w will always be the same.

  As a result, $s, $w, $y, $i can be referenced in the llatsni.inf.
- Execute the macro definitions given in section [macros]. Unlike install.inf, there is only one section, i.e, [macros] where you can define your own macros. One example is setting/clearing entries in INI files,

  ```
  [macros]
  {$w\win.ini, CShell, LastFileOpened} =
  ```

  In this case the entry called "LastFileOpened" in section [CShell] in wini.ini will be set to empty and thus be deleted. If this entry is the only entry (left) in [CShell], [CShell] itself will be deleted as well.

  The macros defined here can be referenced in the rest of llatsni.inf.

- Read in the entries given in section [general]. You can reference all macros already defined.
- Read in the entries given in section [files] and [items]. You can reference all macros already defined. One example is referencing $i in section [files],

```
[files]

cshwin.dll  = $i\lib, 58384, C Shell DLL
```

  In this case cshwin.dll, residing in a subdirectory called "lib" under the install target directory, will be deleted.

See Also

Installation Behavior, Uninstallation Behavior, Macros

Installation Behavior

The following steps will be performed by install.exe during installation in the order shown,

- Create a background window covering the whole screen. The window will be painted in blue colors with gradually varying darkness from top to bottom. A text string,of a certain font (including style) and a certain color, will be displayed at a certain (x, y) coordinates relative to the left-top corner of the screen.

  These parameters are given in section [general] in install.inf. For example,

```
[general]
bkTxt = C Shell for Windows             ;display "C Shell for Windows"
bkTxtX = 10                             ;10pt to the right of the corner
bkTxtY = 20                             ;20pt below the corner
bkTxtH = 60                             ;60pt high
bkTxtColor = 255, 0, 0                  ;red = 255, green = 0, blue = 0
bkTxtFont = Times New Roman Bold Italic ;Times New Roman font (bold & italic)
```

  Note that you can attach styles only to TrueType fonts.

- Pop up a dialog box to display welcome messages. There are 3 lines of message and 2 buttons called "install" and "don't install" respectively in the dialog box. The user can choose "install" to continue the process and "don't install" to abort.

  The 3 lines of message are given in section [general] in install.inf. For example,

```
[general]
welcomeTxt1 = This program will install C shell for Windows
welcomeTxt2 = v1.0 onto your system
welcomeTxt3 = Copyright (c) 1994 John && Jack
```

  Note that if you would like to display an and sign ("&"), you need to write two consecutive and signs as shown in **welcomeTxt3**. This applies to all text strings to be displayed in dialog boxes.

- Pop up a dialog box to ask for an install target directory. The name of the application being installed and the default value for theinstall target directory will be be displayed in the dialog box.

  The name of the application being installed is given in section [general] in install.inf. For example,

```
[general]
appName = C Shell for Windows
```

  The default install target directory is given in section [general] in install.inf. For example,

```
[general]
defDir = c:\cshwin        ;default install target dir is c:\cshwin
```

  The install target directory entered will be stored as a macro called i. As a result, you can reference the install target directory as $i in install.inf.

- Before the install target directory is stored in $i, the free space of the drive where it resides will be checked against the minimum free space required. If it is less than required, install.exe will prompt the user to input another directory.

  The minimum required free space is given in section [general] in install.inf. For example,

```
[general]
minFreeSpace = 450        ;at least 450K free disk space
```

- Pop up an abort dialog box with a progress bar and start copying the files. Before each file is copied, decompression and/or version checking will be performed as required. The user can abort the process at any time by pressing the abort button.

  However, if install.exe is in debug mode, the copying and version checking will be skipped as if they were already performed. The usage of this to speed up the installation process when you are debugging your install.inf.

  The debug mode can be turned on in section [general] in install.inf. For example,

  ```
  [general]
  debug = 1
  ```

  Will turn on the debug mode. Debug mode is off by default.

  All files to be copied are given in section [files] in install.inf. For example,

  ```
  [files]
  cshwin.exe  = $i, 58384, 0, C Shell EXE,             0, 0
  bwcc.dl_    = $y, 46837, 2, Borland Custom Controls, 1, 1
  ```

  In this case, cshwin.exe will be copied to the target directory (see Predefined Macros). The original size of it is 58384 bytes (this value is used to calculate the percentage of the progress bar). It is on the first installation disk. The description for it is "C Shell EXE". Version checking will not be performed for it (because it is not available from other sources). It was not compressed.

  In this case, bwcc.dl_ will be copied to the Windows system directory (see Predefined Macros). The original size of it is 46837. It is on the third installation disk. The description for it is "Borland Custom Controls". Version checking will be performed for it (because it is available from other sources). It was already compressed.

  When a file is not found on the install source directory, install.exe will think it is time prompted the user to insert another installation disk. It will pop up a dialog box displaying the description for the disk on which the missing file is supposed to be residing.

  The descriptions for the installation disks are given in section [disks] in install.inf. For example,

  ```
  [disks]
  0 = Installation disk #1  ;disk 0 is called "Installation disk #1"
  1 = Installation disk #2  ;disk 1 is called "Installation disk #2"
  2 = Library disk          ;disk 2 is called "Library disk"
  ```

  In this case, install.exe will display "please insert library disk" when the file bwcc.dl_ can not be found.

  When the time it takes to actually copy a certain file is less the minimum time, install.exe will wait until the minimum time has elapsed. This feature is designed specially for small applications consisting of small files. Without this feature, the installation will be extremely fast and as a result the user will be left behind in the dust without knowing what is going on here. With this feature, the installation will be slowed down and the user will "enjoy" the installation more and probably feel that there is much stuff in the application be installed.

  The minimum time is given in section [general] in install.inf. For example,

  ```
  [general]
  minTime = 500             ;at least take 500ms per file
  ```

  If you specify 0 here or don't specify it at all, this feature will be turned off.
- Pop up a dialog box asking the user if install.exe should create the program items. If yes, install.exe will do so immediately.

  The items to be created are given in section [items] in install.inf. For example,

```
[items]

Manual = C Shell, $w\winhelp.exe $i\cshwin.hlp, $w\winhelp.exe
```

In this case, an item called "Manual" will be created in a group called "C Shell". If there is no such group, one will be created. The command line for the item is something like "c:\windows\winhelp.exe c:\cshwin\cshwin.hlp". The icon path for the item is something like "c:\windows\winhelp.exe", i.e., the first icon in winhelp.exe (the yellow question mark) will be used. You could also use the path to any icon file (.ICO) you have already installed as the icon path.

See Also

install.inf, Macros, Uninstall Behavior

Uninstallation Behavior

The following steps will be performed by llatsni.exe during uninstallation in the order shown,

- Copy itself (llatsni.exe) and llatsni.inf to the Windowsdirectory, under the name of _f_i_c_.exe and _f_i_c_.inf respectively, cutting off its connection with the application being uninstalled. This is necessary because Windows can not delete a file being used. The fact that llatsni.exe itself can not be deleted makes the directory where it resides can not be deleted as well. Copying itself to Windows directory minimizes this effect.
- Run its clone in the Windows directory and pass it as the first command line parameter the install target directory.
- Quit.

The following steps will be performed by the clone of llatsni.exe during uninstallation in the order shown,

- Create a background window covering the whole screen. The window will be painted in blue colors with gradually varying darkness from top to bottom. A text string,of a certain font (including style) and a certain color, will be displayed at a certain (x, y) coordinates relative to the left-top corner of the screen.

  These parameters are given in section [general] in llatsni.inf. For example,

```
[general]
bkTxt = C Shell for Windows             ;display "C Shell for Windows"
bkTxtX = 10                             ;10pt to the right of the corner
bkTxtY = 20                             ;20pt below the corner
bkTxtH = 60                             ;60pt high
bkTxtColor = 255, 0, 0                  ;red = 255, green = 0, blue = 0
bkTxtFont = Times New Roman Bold Italic ;Times New Roman font (bold & italic)
```

  Note that you can attach styles only to TrueType fonts.
- Pop up a dialog boxto display warning messages. There are 3 lines of message and 2 buttons called "don't remove" and "remove" respectively in the dialog box. The user can choose "remove" to continue the process and "don't remove" to abort.

  The 3 lines of message are given in section [general] in llatsni.inf. For example,

```
[general]
warningTxt1 = This program will remove C shell for Windows
warningTxt2 = v1.0 from your system
warningTxt3 = Copyright (c) 1994 John && Jack
```

  Note that if you would like to display an and sign ("&"), you need to write two consecutive and signs as shown in **warningTxt3**. This applies to all text strings to be displayed in dialog boxes.
- Pop up an abort dialog box with a progress bar and start deleting the files. The user can abort the process at any time by pressing the abort button.

  However, if llatsni.exe is in debug mode, the deletion will be skipped as if it were already performed. The usage of this to speed up the uninstallation process when you are debugging your llatsni.inf.

  The debug mode can be turned on in section [general] in llatsni.inf. For example,

```
[general]
debug = 1
```

Will turn on the debug mode. Debug mode is off by default.

All files to be deleted are given in section [files] in llatsni.inf. If a star (*) appears in place of the file name, the directory will be deleted instead. For example,

```
[files]
*           = $i, 58384, All files
bwcc.dll    = $y, 46837, Borland Custom Controls
```

In this case, all files in the target directory (see Predefined Macros) as well as the target directory itself will be deleted. The total size of the files is 58384 bytes (this value is used to calculate the percentage of the progress bar). The description for it is "All files".

In this case, bwcc.dll which resides in the Windows system directory (see Predefined Macros) will be deleted . The original size of it is 46837. The description for it is "Borland Custom Controls".

If a directory is to be deleted, all files and subdirectories in it will also be deleted.

It will work even for those files having a read only, hidden, or system.

If a file or a directory can not be deleted due to any reason (such as there is no such file or the file is in a read only network drive), it will simply be skipped. No error will be signaled.

When the time it takes to actually delete a certain file is less the minimum time, llatsni.exe will wait until the minimum time has elapsed. This feature is designed specially for small applications consisting of small files. Without this feature, the uninstallation will be extremely fast and as a result the user will be left behind in the dust without knowing what is going on here. With this feature, the uninstallation will be slowed down and the user will "enjoy" the uninstallation more and probably feel that everything is under control.

The minimum time is given in section [general] in llatsni.inf. For example,

```
[general]
minTime = 500             ;at least take 500ms per file
```

If you specify 0 here or don't specify it at all, this feature will be turned off.

- Delete the program items.

  All items to be deleted are given in section [items] in llatsni.inf. If a star (*) appears in place of the item name, the group will be deleted instead. For example,

  ```
  [items]
  ATM = Accessory
  *   = C Shell
  ```

  In this case, an item called "ATM" will be deleted in a group called "Accessory".

  In this case, all items in the group called "C Shell" and the group itself will be deleted.
- Pop up a dialog box to notify the user that it is going to restart Windows. If the user says no, uninstallation will be aborted.
- Exit Windows and perform a DOS command like "del c:\windows_f_i_c_.*", and then restart Windows. Because I use wildcard in the command, any file with file name _f_i_c_ in the Windows directory will deleted.

In order for llatsni.exe to work properly, you need to,

- Distribute llatsni.exe and llatsni.inf with your application.
- Install llatsni.exe and llatsni.inf to the install target directory. Otherwise, when it is run, llatsni.exe will have no way to determine the install target directory and can not pass the install target directory to its clone.

- Create a program item for llatsni.exe.
- Write llatsni.inf properly. llatsni.inf should as least specify that all files (including llatsni.exe and llatsni.inf) you installed which are not shared by other programs be deleted, all program items you created be deleted, all INI entries which are not shared by other programs be deleted.

See Also

llatsni.inf, Macros, Install Behavior

The name of the application being installed. It will be used in some dialog boxes. It will be evaluated before being used. See String Evaluation.

The first line of welcome text in the welcome dialog box. It will be evaluated before being used. See String Evaluation.

The second line of welcome text in the welcome dialog box. It will be evaluated before being used. See String Evaluation.

The third line of welcome text in the welcome dialog box. It will be evaluated before being used. See String Evaluation.

The first line of warning text in the warning dialog box. It will be evaluated before being used. See String Evaluation.

The second line of warning text in the warning dialog box. It will be evaluated before being used. See String Evaluation.

The third line of warning text in the warning dialog box. It will be evaluated before being used. See String Evaluation.

Minimum free disk space required (in KB).

The default install target directory, i.e., the default value when prompting the user to input the install target directory. It will be evaluated before being used. See String Evaluation.

The text string displayed as a part of the background. It will be evaluated before being used. See String Evaluation.

The height of the font (in points) for bkTxt.

The text font for bkTxt. You can attach styles to the end of the font itself. Usually there are only two styles available, i.e., italic and bold. If you specify both of them, write italic first and then bold. Note that you can attach styles only when the font is a truetype font. It will be evaluated before being used. See String Evaluation.

The color of the font for bkTxt. You specify the color by specifying the intensity of the red, green, and blue components of the color, separated by comma.

The horizontal distance (in points) from the left top corner of the screen to the left top corner of bkTxt.

The vertical distance (in points) from the left top corner of the screen to the left top corner of bkTxt.

The minimum time it takes to copy a file. When the time it takes to actually copy a certain file is less the minimum time, install.exe will wait until the minimum time has elapsed.

The minimum time it takes to delete a file. When the time it takes to actually delete a certain file is less the minimum time, llatsni.exe will wait until the minimum time has elapsed.

The disk number for the disk being described. The disks should be numbered as 0, 1, 2, ... etc.

The description for the disk being described. It will be used when prompting the user to insert the correct disk. It will be evaluated before being used. See String Evaluation.

The name of the macro. See Macro.

The value of the macro. It will be evaluated before being used. See String Evaluation.

The name of the file which will be copied. If the file is a compressed file, use the name of the compressed file rather than the original one.

The name of the file which will be deleted. If the file is a compressed file, use the name of the original file rather than the compressed one. If you put a star (*) here, the all files, subdirectories, and the directory itself will be deleted.

The directory to which the file will be copied to. It will be evaluated before being used. See String Evaluation. Usually you will need to reference to $i whose value is the install target directory.

The directory in which the file will be deleted. It will be evaluated before being used. See String Evaluation. Usually you will need to reference to $i whose value is the install target directory.

The size of the file before compression. It is used to calculate the percentage of bytes already copied or deleted. If you don't care about the accuracy, you can simply use the value 1 for all files and change the meaning to percentage of files already copied or deleted.

The disk number on which the file resides. It is used to retrieve the disk description listed in section [disks] to ask the user to insert the correct disk when a file can not be found.

The description for the file. It will be evaluated before being used. See String Evaluation.

Put non-zero here if version checking is required for the file, otherwise put zero here. Usually only those files which may be installed by different software applications such as bwcc.dll and vbrun300.dll need version checking.

Put non-zero here if the file is a compressed file, otherwise put zero here.

The name of the item that will appear in the program group. If there is already such an item, it will be deleted first.

The name of the item that will be deleted. If you put a star (*) here, all the items in the this group and the group itself will be deleted.

The group in which the item will appear. If there is no such group, one will be created. It will be evaluated before being used. See String Evaluation.

The group in which the item will be deleted. It will be evaluated before being used. See String Evaluation.

The command line for the item. It will be evaluated before being used. See String Evaluation. Usually you will need to reference to $i whose value is the install target directory.

The icon file for the item. It will be evaluated before being used. See String Evaluation. Usually you will need to reference to $i whose value is the install target directory. If the file is an EXE file, the first icon stored in it will be used to represent the icon. If the file is an ICO file, the icon stored in the file will be used.

Install Information File - install.inf

Every time install.exe runs, an installation information file called "install.inf" will be accessed by install.exe and has to be in the same directory of install.exe. This file is in standard Windows INI file format and provides install.exe with the installation specific parameters such as the name of the application being installed, minimum free disk space required and etc.

Before doing anything, please keep in mind that never use tab in INI files (including install.inf and llatsni.inf, of course). Windows will consider tab as end of line.

Here is an example install.inf,

```
[macros1]
csw   = C Shell for Windows
cswv  = $csw v 1.0
group = $csw

[general]
appName = $cswv
welcomeTxt1 = This program will install $cswv
welcomeTxt2 = onto your system
welcomeTxt3 = Copyright (c) 1994 John && Jack
minFreeSpace = 450
defDir = c:\cshwin
bkTxt = $csw
bkTxtH = 60
bkTxtX = 0
bkTxtY = 0
bkTxtColor = 255, 0, 0
bkTxtFont = Times New Romain Bold Italic
minTime = 500

[macros2]
{$w\cshwin.ini, general, homeDir} = $i

[disks]
0 = Executable Disk
1 = Library Disk

[files]
cshwin.exe  = $i,          17000, 0, $fi EXE ,               0, 0
manual.wr_  = $i\doc,       3000, 0, User Manual,            0, 1
bwcc.dl_    = $y,          15000, 1, Borland Custom Control, 1, 1
[items]
C Shell = $group, $i\cshwin.exe,              $i\install.exe
Manual  = $group, $w\write.exe $i\manual.wri, $w\write.exe
```

See Also

Installation behavior, Macro, When are macros bound?

Uninstall Information File - llatsni.inf

Every time llatsni.exe runs, an installation information file called "llatsni.inf" will be accessed by llatsni.exe and has to be in the same directory of llatsni.exe. This file is in standard Windows INI file format and provides llatsni.exe with the installation specific parameters such as the name of the application being installed, which files to delete and etc.

Before doing anything, please keep in mind that never use tab in INI files (including install.inf and llatsni.inf, of course). Windows will consider tab as end of line.

Here is an example llatsni.inf,

```
[macros]
csw   = C Shell for Windows
cswv  = $csw v 1.0
group = $csw

[general]
appName = $cswv
warningTxt1 = This program will remove $cswv
warningTxt2 = from your system
warningTxt3 = Copyright (c) 1994 John && Jack
bkTxt = $csw
bkTxtH = 60
bkTxtX = 0
bkTxtY = 0
bkTxtColor = 255, 0, 0
bkTxtFont = Times New Romain Bold Italic
minTime = 500

[files]
*        = $i, 17000, All files in install target directory
bwcc.dll = $y, 15000, Borland Custom Control

[items]
C Shell = Accessory
*       = $group
```

See Also

Uninstallation behavior, Macro, When are macros bound?

String Evaluation

All strings appearing in the right hand side of entries in install.inf and llatsni.inf will be evaluated before actually being used.

There are following rules,

- Characters appearing between double quotes (") or single quotes (') will be coped verbatim. For example,

```
"abc,;$xxx"     ===>    abc,;$xxx
a'b'c",;$x"xx   ===>    abc,;$xxx
```

- To use double quotes in your string, put them in single quotes. For example,

```
'abc"def'       ===>    abc"def
```

- To use single quotes in your string, put them in double quotes. For example,

```
"abc'def"       ===>    abc'def
```

- Preceed a macro name with a dollar sign to reference the value of the macro. For example, suppose that $xxx is "Hello World!",

```
$xxx+def        ===>    Hello World!+def
${xxx}def       ===>    Hello World!def
```

  If the macro being referenced is undefined, an error will be signaled.

- Spaces at the beginning and ending of the string will be ignored. If you need them, you should put them in quotes. For example,

```
  xx            ===>    xx
"  "xx"  "      ===>      xx
```

- Comma (,), curly bracket ({ or }), or dollar sign ($) has special meaning. If you would like to use them as ordinary characters, you should put them in quotes. For example,

```
"${x},$$$x{x"   ===>    ${x},$$$x{x
```

Macro

Freeman Installer supports macros. A macro is like a variable. A macro has a name and a value.

There are two kinds of macros. One is internal macro and the other is external macro.

Internal Macro

An internal macro is stored in RAM. An internal macro is created when it is defined, i.e., a macro definition, with the name of the macro as the left hand side, is executed. It is destroyed when the program (install.exe or llatsni.exe) terminates.

The name of an internal macro is a string consisting of alphabets and/or digits. The value of a macro is a string. The value of a macro is retrieved (i.e., the macro is referenced) by preceding the name of the macro with a dollar sign "$", i.e., $xxx means the value of the macro called "xxx". However, we also use $xxx to mean the macro called "xxx" itself. A pair of curly brackets may be used to delimit the macro name, i.e., ${xxx} is equivalent to $xxx. For example,

```
[macros]

xxx = Hello World!

y12 = $xxx abc

z   = ${xxx}abc
```

After executing the first defintion, a macro called "xxx" is created and its value is "Hello World!". After executing the second definition, a macro called "y12" is created and its value is "Hello World! abc", i.e., "$xxx abc" after being evaluated. After executing the third definition, a macro called "z" is created and its value is "Hello World!abc", i.e., "${xxx}abc" after being evaluated. . Note that in this case the curly brackets are necessary because if there is no curly bracket, "$xxxabc" will lead the program to expand a (non-existing) macro called "xxxabc".

There are 4 predefined internal macros available as shown below,

```
$w  --- Windows directory, usually c:\windows

$y  --- Windows system directory, usually c:\windows\system

$s  --- Source directory, usually a:\

$i  --- Install target directory entered by the user
```

If you use the C++ class library, you can define even more predefined macros to be used in your install.inf and/or llatsni.inf.

External Macro

An external macro represents an entry in an INI file.

The name of an internal macro takes the following form,

```
{file, section, entry}
```

Where **file**, after being evaluated, is the INI file name. **section**, after being evaluated, is the section name. **entry**, after being evaluated, is the entry name. If neither drive nor directory is included in result of the evaulation of **file**, it will be considered residing in the Windows directory. For example,

```
{c:\cshwin\cshwin.ini, locations, homeDir}
```

represents an entry called "homeDir" in section [locations] in the file c:\cshwin\cshwin.ini.

For example,

```
[macros]

yyy = windows
```

```
xxx = ${win.ini, $yyy, programs} vbx

{win.ini, $yyy, programs} = $xxx
```

Will add "vbx" to the original value of the entry called "programs", i.e., if the original win.ini looks like,

```
[windows]

programs = exe bat pif
```

After executing the above three macro definitions, win.ini will be changed to,

```
[windows]

programs = exe bat pif vbx
```

In fact, the same effect can be achieved in a single definition,

```
[macros]

{win.ini, windows, programs} = ${win.ini, windows, programs} vbx
```

Note that there is no dollar sign in the left hand side because it is not a part of the macro name. This applies to internal macros as well.

If you try to retrieve the value of an external macro whose corresponding INI file doesn't exist, an error will be signaled. If you try to retrieve the value of an external macro whose corresponding INI file exists but one of the section or the entry doesn't exist, the string "unknown" will be returned as the value.

If you try to set the value of an external macro one of the corresponding INI file or the section or the entry doesn't exist, what is missing will be created. There is one exception, however, if you set the value to an empty string, the entry (if any) will be deleted. If that entry is the only entry in the section, the section will be deleted as well.

See Also

String Evaluation, When are macros bound?

File version attributes here refer to some of the attributes stated in the VERSION resource statement, consisting of language ID, character set, file type (EXE, DLL, etc.) and subtype (display driver, keyboard driver, etc.).

Version Checking

When the 4th field in an entry in section [files] is a non-zero value, install.exe will perform version checking when trying to overwrite an existing file on the user's system which has the same file name as the distribution file.

Install.exe will decide to overwrite the existing file if and only if at least one of the following conditions holds,

- The distribution file and the existing file share the same file version attributes and the version number of the distribution file is greater than that of the existing file.
- The existing file has no version information.

Install.exe will decide to keep the existing file if and only if at least one of the following conditions holds,

- the distribution file and the existing file share the same file version attributes and the version number of the distribution file is equal to or smaller than that of the existing file.
- The existing file has version information but the distribution file has not.

Install.exe will pop up a dialog box with the file version attributes shown, asking the user to decide to overwrite or not and giving the user a chance to specify another path as the destination path for the distribution file if and only if the distribution file and the existing file have one of more different file version attributes.

When the distribution file is a compressed file, what is going on is actual more complicated. Because install.exe can not retrieve the version information out of a compressed file, it has to first decompress the file to create a temporary file in the target directory and then perform version checking against the temporary file and the existing file. If, according to the logic in mentioned above, the existing file should be overwritten, it will actually be deleted and the temporary file will be renamed to the correct name. If the existing file should not be overwritten, the temporary file will be deleted.

See Also

Installation behavior

Preparing Compressed Files

In order to distribution your files in compressed format, your need to,

1. Compress your files with the Microsoft File Compression Utility called "compress.exe" which comes with every copy of Windows SDK. Make sure that you use the -r option to instruct compress.exe to automatically generate the name of the compressed file and more importantly to store the original name in the compressed file. For example,

   ```
   compress -r *.doc c:\temp
   ```

   In this case, all files with extension DOC in the current directory will be compressed and the compressed files will be put into c:\temp.

   If you don't use -r option, Freeman Installer will be unable to know the original file name. Even worse, there is no way for Freeman Installer to detect if -r option has been used or not.
2. Copy those compressed files to floppy disks. All files must be in the root directory of the floppy disks.
3. Mark those files as compressed in install.inf. Remember to use the file name of the compressed file rather than the original file name in section [files] in install.inf.

Introduction

Freeman Installer & Uninstaller (or Freeman Installer in short) is a shareware package to facilitate general software installation and uninstallation tasks under Microsoft Windows.

Here is a quick introduction,

- Installs. Installs your application smoothly.
- Uninstalls. Removes your application completely.
- Works with files compressed by Microsoft compress.exe.
- Supports version checking. Won't ruin your users' ctrl3d.dll or bwcc.dll.
- Supports macro concept. Uses this simple mechanism to save typing, access Windows directory, install target directory, and INI files.
- Optionally slows down the install process to let small applications appear bigger.
- Source code included (registered version only) to support virtually unlimited customizations.

Freeman Installer can be used in two ways,

- For users whose installation and/or uninstallation task is relatively simple and doesn't need much customization, Freeman Installer includes a ready-to-run install program called "install.exe" and a ready-to-run uninstall program called "llatsni.exe". The only thing to do is the user filling out the task-specific information in an install information file called "install.inf" and/or an uninstall information file called "llatsni.inf".

  The advantage of this approach is that you can get an install and/or uninstall program for your own product in several minutes without the need to program.
- For programmer users who need to customize/take greater control of the installation and/or uninstallation, Freeman Installer can be used as a C++ class library to perform the chores common to all installations and uninstallations, including file decompression, file copying and removing (including directories), file version comparison, program group and/or item creation and deletion, INI file entry access and modification.

  In fact, the ready-to-run install.exe and llatsni.exe themselves were built on top of this same library. The source code of install.exe and llatsni.exe is also included in the hope that it can be used as a basis for further possible customizations/enhancements such as having multiple target directories or checking the existence of a dongle.

  The advantage of this approach is that the programmer still has the freedom to do whatsoever he/she likes and works in his/her most favorite programming environment without the need to learn another script/macro language.

  Note that the .LIB files of the class library come with the registered version only.

The fastest way to create an install and uninstall program for your own application is,

1. Compress your files with the Microsoft file compression utility with the -r option presented. Also keep track of which file goes to which disk.
2. Modify an existing install.inf. One is provided with this release of Freeman Installer. It instructs install.exe to install Freeman Installer itself onto your system.
3. Modify an existing llatsni.inf. One is provided with this release of Freeman Installer. It instructs llatsni.exe to remove Freeman Installer itself from your system.
4. Copy install.exe, install.inf, llatsni.exe, llatsni.inf and all those compressed files to floppy disks.

See Also

Installation behavior, Uninstallation behavior

Peek a message & possibly give CPU away to Windows and/or other applications. If a message is retrieved, dispatch it to the progress dialog procedure by **IsDialogMessage** or to other windows by **TranslateMessage** & **DispatchMessage**. You should check if (*__flag__) has been set during the processing of this message after calling this function.

If you have other modeless dialog boxes in addition to the progress dialog box, instead of calling this function, you should write your own message loop, because this function only call **IsDialogMessage** for its own progress dialog box.

Constructor for dlgprog. It creates the progress dialog box and initializes its variables. **percent** is set to 0.

out:

```
r    --  true if ok, false if unable to create the dialog box
```

Destructor for dlgprog. It destroys the progress dialog box.

Set the first static text.

in:

```
txt --  set to this text string
```

Set the second static text.

in:

```
txt --  set to this text string
```

Set the third static text.

in:

```
txt --  set to this text string
```

Show the progress dialog box if it is not already shown. If **iscenter** is non-zero, the progress dialog box will be aligned to the center of the screen.

in:

```
    iscenter  -- aligns the dialog box to the center of the screen if it's non-zero
```

Set the percentage of the progress. If the progress dialog box is shown, the progress bar will be redrawn.

in:

```
    percent --  the new percentage. must be between 0 to 100
```

Draw the progress bar. You should not call this function. It is intended for the dialog procedure of the progress dialog box when the progress bar is requested to be drawn.

Class dlgprog

An instance of class **dlgprog** is a C++ object representing a percentage (0-100) which is shown in a progress dialog box consisting of a progress bar, an abort button, and three static text controls. The progress bar is actually an owner-drawn button.

```
class dlgprog
{
    int percent;                /* the percentage of work already done */
    int *flag;                  /* set this flag when abort button is pressed */
    HWND dlg;                   /* handle to the progress dialog box */
    HWND owner;                 /* handle to owner of the progress dialog box */
    FARPROC p;                  /* procedure instance for the progress dialog box */
    COLORREF metercolor;        /* the color of the progress bar */

    public:
    dlgprog(int *r, HINSTANCE inst, HWND owner, int *flag, COLORREF metercolor);
   ~dlgprog();

    int chkdlgmsg(MSG FAR *msg)  { return IsDialogMessage(dlg, msg); }
    HWND getdlg()                { return dlg;                       }
    void cancel()                { *flag = 1;                        }
    void settitle(char title[])  { SetWindowText(dlg, title);        }

    int peekdispatchmsg();
    void settxt1(char txt[]);
    void settxt2(char txt[]);
    void settxt3(char txt[]);
    void show(int iscenter);
    void setpercent(int p);
    void drawmeter(DRAWITEMSTRUCT FAR *q);
};
```

Initialize **owner** & **inst**, create macro binding for $w (Windows dir), $y (Windows system dir), $s (Source dir), locate the information file and store its full path in **inffile**.

in:

```
owner          --  owner of subsequent dialog boxes
inst--         instance to create dialog boxes
inffile        --  file name of the information file. No drive or dir should be
                   included
```

returns:

```
true--  ok
false   --  out of memory or unable to locate install.inf
```

In the case of error, a fatal error message will be displayed before it returns.

Copy a source file to a destination file. Performs version checking and/or decompression if required. The user will be asked to decide overwrite the destination file or not if the version information is not sufficient enough. Before calling this function, you should call **LZStart**. When you no longer need to call this function, you should call **LZDone**.

in:

```
srcpath        --  source file to be copied
dstpath        --  destination file to be created/overwritten
ischeck        --  perform version checking if it's non-zero
iscompressed   --  perform decompression if it's non-zero
```

returns:

```
0       --  the file has been copied successfully
1       --  the file has been skipped
2       --  error
```

In the case of error, a fatal error message will be displayed before it returns.

See Also

Version checking

Delete a directory. All files and subdirectories in the directory will be deleted as well. It will work even for those files having read-only, hidden, or system attribute.

in:

```
dir     --  the directory to be deleted
```

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Delete a file. It will work even for a file having read-only, hidden, or system attribute.
in:

```
dir     --  the directory where the file resides
file--  the file name
```

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Pop up a dialog box asking the user to enter a directory.

in:

```
    dir     --  default value for the directory
    prompt  --  text string used in the dialog box as a prompt may contain 3 lines
```

out:

```
    dir     --  the directory specified by the user
```

returns:

```
    true--  ok
    false   --  cancel
```

Ask if the user really wants to abort the installation.
returns:

```
true--  yes
false   --  no
```

Ask if the user really wants to abort the uninstallation.
returns:

```
true--  yes
false   --  no
```

Ask if the user to choose between retry and cancel.

in:

```
    prompt  --  text string used in the dialog box as a prompt
```

returns:

```
    true--  retry
    false   --  cancel
```

Ask the user to choose a drive from a list box. All drives recognizable to Windows are listed, including floppy disk drives, remote drives.

in:

```
prompt   --  text string used in the dialog box as a prompt
```

out:

```
driveno --  drive number for the drive, e.g., 0 for a:, 1 for b:, 2 for c:
```

returns:

```
true--  ok
false   --  cancel
```

Ask the user to insert the disk whose description will be displayed in the dialog box.

in:

```
    diskdesc--  the description for the disk
```

returns:

```
    true    --  ok
    false       --  cancel
```

Pop up a dialog box displaying the version information of the source file and the destination file and asks if the user wants to overwrite the destination file by the source file. The user is also allowed to specify another path as the destination file.

In:

```
dstpath    --    destination file to be overwritten
srcpath    --    source file to be copied
dstv       --    version information for dstpath
srcv       --    version information for srcpath
```

out**:**

```
dstpath    --    the new destination path specified by the user if overwrite is chosen
```

returns:

```
0       --  overwrite
1       --  skip
2       --  error
```

In the case of error, a fatal error message will be displayed before it returns.

See Also

Version checking

Get the full path for a source file and the corresponding destination file.

In:

```
    diskno   --  the disk number on which the source file resides. used to ask the
                 user to insert disk if the source file is not found
    srcfile  --  source file name
```

out:

```
    srcpath  --  full path of the source file, i.e., $s\srcfile
    dstdir   --  directory of the destination file
    dstfile  --  destination file name. It is the same as srcfile is srcfile is
                 uncompressed, otherwise it is the original name of srcfile
    dstpath  --  full path of the destination file, i.e., dstdir\dstfile
```

returns:

```
    true--  ok
    false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Read an internal macro name from the string.

in:

```
s   --  the string from which the name will be read
```

out:

```
n   --  number of character in s which have actually been used
r   --  macro name (null terminated)
```

returns:

```
0   --  ok
1   --  syntax error
```

Read an external macro name from the string.

in:

```
    s       --  the string from which the name will be read
```

out:

```
    n       --  number of character in s which have actually been used
    file--  INI file (null terminated)
    sect--  section name (null terminated)
    item--  entry name (null terminated)
```

returns:

```
    0   --  ok
    1   --  syntax error
```

It will also work for internal macro names enclosed in curly brackets. In this case, file and sect will be set to empty.

Initialize everything need to DDE with the program manager. You should call this function before calling **installer::createitem**, **installer::delgrup**, or **installer::delitem**. When finish, you should call **installer::endddepm** to free the resource allocated here.

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Free the resources allocated in **installer::begddepm**.

Display a message box. The message box is similar to the standard Windows message box except that it uses 9 point helv text font and smaller buttons to make it prettier. However, it doesn't provide the functionality specified by MB_SYSTEMMODEL. As a result, you shouldn't use this function display system wide critical errors such as out of memory error.

returns:

```
-1  --  error
0   --  the 1st button has been clicked
1   --  the 2nd button has been clicked
2   --  the 3rd button has been clicked
```

In the case of error, a fatal error message will be displayed before it returns.

Delete a program group. by DDE'ing with the program manager. Before calling this function, **installer::begddepm** should be called. When you no longer need to DDE with the program manager, you should call **installer::endddepm**.

in:

```
grup--  name of the group which is to be deleted
```

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Delete a program item. by DDE'ing with the program manager. Before calling this function, **installer::begddepm** should be called. When you no longer need to DDE with the program manager, you should call **installer::endddepm**.

in:

```
grup--  group name
item--  name of the item which is to be deleted
```

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Create a program item by DDE'ing with the program manager. The program group where the program item resides will be created if it doesn't exist yet. If there is an existing item with the same name residing in the same group, the existing item will be deleted first. Before calling this function, **installer::begddepm** should be called. When you no longer need to DDE with the program manager, you should call **installer::endddepm**.

in:

```
grup--  group name
item--  item name
cmdl--  command line for the item
icon--  path to the icon for the item
```

returns:

```
true--  ok
false   --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Get the version information of the file.

in:

```
    path--  file whose version information is to be retrieved
```

out:

```
    v       --  the version information will be stored here
```

returns:

```
    0       --  ok
    1       --  no version information available
    2       --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Compare the version information of the source file and that of the destination file to determine if the destination file should be overwritten by the source file. User interference may be introduced. If you use installer::copy to copy files, you don't need to call this function yourself.

in:

```
srcpath    -- source file to be copied
dstpath    -- destination file to be overwritten
```

returns:

```
0       --  overwrite
1       --  skip, i.e., don't overwrite
2       --  error
```

In the case of error, a fatal error message will be displayed before it returns.

See Also

Version checking

Check if the free space of the given drive is greater than the given minimum value.
in:

```
drive    --  the drive to be tested
minf--  the minimum required free space in Kb
```

returns:

```
0        --  free space is enough
1        --  too less free space
2        --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Get the free space of the given drive.

in:

```
    drive    --  the drive to be tested
```

out:

```
    f        --  the free space in Kb
```

returns:

```
    true--  ok
    false    --  error
```

In the case of error, a fatal error message will be displayed before it returns.

Evaluate a macro whose name is at the beginning of a string.

in:

```
s   --  the string from which the macro name is to be read
```

out:

```
n   --  the number of characters in s which have actually been used
r   --  the value of the macro
m   --  the number of characters which have been put in r
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read a INI file during the evaluation
```

In the case of error, a fatal error message will be displayed before it returns.

Read in some comma-separated fields (strings, integers, or doubles) from a string. For a string field, it will be evaluated.

in:

```
s           --   the string from which the fields are to be read
specifiers  --   each character represents a field's type. For example, "ssid"
                 means four fields are to be read, the first two are strings, the
                 third is an integer, the last is a double
...         --   address for each field
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read a INI file during the evaluation
```

In the case of error, a fatal error message will be displayed before it returns.

Evaluate a string. The whole string must be used exhaustively, otherwise it is considered an error.

in:

```
s   --  the string to be evaluated
```

out:

```
r   --  the value of the evaluation (null terminated)
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read a INI file during the evaluation
```

In the case of error, a fatal error message will be displayed before it returns.

Evaluate a string.

in:

```
s   --  the string to be evaluated
```

out:

```
n   --  the number of characters in s which have actually been used
r   --  the value of the evaluation
m   --  the number of characters which have been put into r
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read a INI file during the evaluation
```

In the case of error, a fatal error message will be displayed before it returns.

Get the value of a generalized macro, i.e., it might be an external macro or an internal macro. All the characters in the string must be used exhaustively to express the name of a macro.

in:

```
gn  --  macro name
```

out:

```
v   --  value of the macro
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read a INI file during the evaluation
```

In the case of error, a fatal error message will be displayed before it returns.

Get the value of an external macro.

in:

```
file--  INI file name. If no drive and no directory is included, it is
    considered to be residing in the Windows directory
sect--  section name
item--  entry name
```

out:

```
v   --  value of the macro
```

returns:

```
0   --  ok
1   --  syntax error
2   --  an undefined macro is referenced during the evaluation
3   --  unable to read the INI
```

If either **file** or **sect** is empty, it will be considered an internal macro whose name is given in **item**.

In the case of error, a fatal error message will be displayed before it returns.

Get the value of an internal macro.

in:

```
n   --  macro name
```

out:

```
v   --  value of the macro
```

returns:

```
0   --  ok
2   --  the macro is undefined
```

In the case of error, a fatal error message will be displayed before it returns.

Set the value of a generalized macro, i.e., it might be an external macro or an internal macro. All the characters in the string must be used exhaustively to express the name of a macro.

in:

```
gn   --  macro name
v    --  macro value
```

returns:

```
0    --  ok
1    --  syntax error
2    --  an undefined macro is referenced during the evaluation
3    --  unable to write to the INI file
4    --  out of memory
```

In the case of error, a fatal error message will be displayed before it returns.

Set the value of an external macro.

in:

```
file--  INI file name. If no drive and no directory is included, it is
    considered to be residing in the Windows directory
sect--  section name
item--  entry name
v       --  macro value. If this is an empty string, the entry will be deleted.
            If the entry is the only entry in the section, the section itself
            will be deleted as well.
```

returns:

```
0   --  ok
3   --  unable to write to the INI file
4   --  out of memory
```

If either **file** or **sect** is empty, it will be considered an internal macro whose name is given in **item**.

In the case of error, a fatal error message will be displayed before it returns.

Set the value of an internal macro. If such a macro doesn't exist yet, it will be created.
in:

```
n   --  macro name
v   --  macro value
```

returns:

```
0   --  ok
4   --  out of memory
```

In the case of error, a fatal error message will be displayed before it returns.

Get the number of entries in a section of a INI file.

in:

```
sect    --  section name
inifile --  INI file. If this is 0, the data member inffile will be used
```

returns:

```
the number of entries in the section
```

Merge a directory and a file name to get the path.

in:

```
    dir     --  directory which might optionally end with a backslash
    file--  file name which might optionally include an extension
```

out:

```
    path--  the resulting path
```

Pop up a standard Windows message box with MB_SYSTEMMODEL on, displaying an "out of memory" message.

See Also

installer::msgbox

Pop up a message box by calling **installer::msgbox**, displaying a fatal error message. The title of the message box is "Fatal Error".

in:

```
    msg     --  the message displayed will be "Fatal error: " + msg
```

Pop up a message box by calling **installer::msgbox**, displaying an error message. The title of the message box is "Error".

in:

```
    msg     --  the message displayed will be "error: " + msg
```

Paint a rectangle with gradually varying colors. the mapping mode of the DC should be MM_TEXT and all coordinates and dimensions should be in pixel. The color of any pixel in the rectangle is calculated by linear interpolation.

in:

```
dc      --  display context
r1      --  the rectangle used to define the color variation
r2      --  the rectangle to be painted
colors  --  colors[0] is the color of the left top corner of r1
            colors[1] is the color of the right top corner of r1
            colors[2] is the color of the left bottom corner of r1
blkw--  r2 will be painted block by block. blkw is the width of the block
blkh--  r2 will be painted block by block. blkh is the height of the block
```

Get the value of an internal macro.

in:

```
n       --  macro name
```

returns:

```
char*   --  the macro is defined
0       --  the macro is undefined
```

In the case of undefined macro, a fatal error message will be displayed before it returns.

Get the entries in a section of a INI file.

in:

```
    sect    --  section name
    inifile --  INI file. If this is 0, the data member inffile will be used
```

returns:

```
    char*   --  the entries in the section terminated by double 0. When you are
                finished with it, you should delete this buffer
    0       --  out of memory
```

Get the binding of an internal macro.

in:

```
n            --  macro name
```

returns:

```
binding* --  if the macro is defined
0            --  if the macro is undefined
```

Class Installer

Class installer is mainly used as a means to provide a separate name space for the procedures/routines common to most installation and uninstallation tasks. As a result, on most occasions, only one instance of this class is created.

```
struct binding                                  /* binding of an internal macro */
{
   dstring s;                                                    /* macro name */
   dstring v;                                                   /* macro value */
};

struct version                                          /* version info of a file */
{
   UINT lang;                                                     /* language id */
   UINT cset;                                                   /* character set */
   DWORD os;                                                              /* os */
   DWORD prmtype;                                 /* primary type (exe, dll, etc) */
   DWORD subtype;                             /* sub type (display, keyboard, etc) */
   DWORD verhi32;                                        /* high 32bit version no */
   DWORD verlo32;                                         /* low 32bit version no */
};

class installer                           /* name space for installation routines */
{
   public:
   installer();                       /* set owner & inst mentioned below to NULL */
  ~installer();

   int init(HINSTANCE inst, HWND owner, char inffile[]);

   int nomacros;                      /* no of intermal macros currently defined */
   int issupressmsg;                  /* should fatal or error msg be supressed? */
   char inffile[_MAX_PATH];                                  /* path of inf file */
   HWND owner;                                               /* owner of dialogs */
   HINSTANCE inst;            /* instance handle used in creating dialog boxes */
   pmconv *c;                           /* dde conversation with program manager */
   binding *macros;                                  /* internal macros bindings */
   simpledde *d;                                                  /* dde wrapper */

   int copy(char srcpath[], char dstpath[], int ischeck, int iscompressed);
   int deldir(char dir[]);
   int delfile(char dir[], char file[]);
   int askdir(char dir[], char prompt[]);
   int askquitinstall();
```

```
   int askquitllatsni();
   int askretry(char msg[]);
   int askdrive(int *driveno, char prompt[]);
   int askinsertdisk(char diskdesc[]);
   int askoverwrite(char srcpath[], version &srcv, char dstpath[], version &dstv);
   int getpaths(int diskno, char srcfile[], char srcpath[], char dstdir[], char dstfile[], char
dstpath[]);
   int readmacroname(char s[], int *n, char r[]);
   int readmacroname(char s[], int *n, char file[], char sect[], char item[]);
   int msgbox(char txt[], char title[], UINT flags);
   int begddepm();
   int getversion(char path[], version *v);
   int delgrup(char grup[]);
   int delitem(char grup[], char item[]);
   int createitem(char grup[], char item[], char cmdl[], char icon[]);
   int getfreespace(int drive, double *f);
   int chkfreespace(int drive, double minf);
   int compareversion(char srcpath[], char dstpath[]);
   int evalmacro(char s[], int *n, char r[], int *m);
   int readfields(char s[], char specifiers[], ...);
   int evalstring(char s[], char r[]);
   int evalstring(char s[], int *n, char r[], int *m);
   int getgmacro(char gn[], char v[]);
   int getemacro(char file[], char sect[], char item[], char v[]);
   int getimacro(char n[], char v[]);
   int setgmacro(char gn[], char v[]);
   int setemacro(char file[], char sect[], char item[], char v[]);
   int setimacro(char n[], char v[]);
   int getnoentries(char sect[], char inifile[] = 0);
   void endddepm();
   void mergedirfile(char path[], char dir[], char file[]);
   void outofmem();
   void fatal(char msg[]);
   void error(char msg[]);
   void errorbadpath(char path[]);                              /* invalid path */
   void errorfreespace(double minfs, double actfs);     /* too less free space */
   void fataldel(char path[]);                          /* can't del a file/dir */
   void fatalren(char oldpath[], char newpath[]);       /* can't rename a file */
   void fatalread(char path[]);                        /* can't read from a file */
   void fatalwrite(char path[]);                       /* can't write to a file */
   void fatalcopy(char srcpath[], char dstpath[]);        /* can't copy a file */
   void fatalshare(char path[]);                        /* file share violation */
   void fatalcreate(char path[]);                       /* can't create a file */
   void fatalaccess(char path[]);                       /* can't access a file */
   void fatalnofile(char path[]);                            /* file not found */
```

```
   void fatalsyntax(char s[]);                                /* syntax error */
   void fataldelitem(char name[]);                 /* can't del a progam item */
   void fataldelgrup(char name[]);                /* can't del a progam group */
   void fatalshowgrup(char name[]);              /* can't show a program group */
   void fatalfileinuse(char path[]);                           /* file in use */
   void fatalundefmacro(char name[]);                      /* undefined macro */
   void fatalcreateitem(char name[]);           /* can't create a program item */
   void fatalcreategrup(char name[]);          /* can't create a program group */
   void fatalreadverinfo();                     /* can't read the version info */
   void changingcolor(HDC dc, RECT r1, RECT r2, COLORREF colors[], int blkw, int blkh);
   char *getimacro(char s[]);
   char *getentries(char sect[], char inifile[] = 0);
   binding *lookup(char n[]);
};
```

Installation C++ Library

A C++ class library, implementing the common installation and uninstallation tasks such as file copying, version checking, program group/item creation, INI file modification and so on, comes with the registered version of Freeman Installer. With this library, a programmer will be able to build his/her own install and/or uninstall program doing what is impossible in the install.exe--install.inf and llatsni.exe-llatsni.inf combination. In fact, install.exe and llatsni.exe themselves were built on top of this library as well, by making commitments such as the assumption of a single install target directory, the assumption of installation = file copying + program group/item creation + INI file modification, and so on.

The C++ class library will be provided in the form of static-link libraries produced by Borland C++ or Visual C++,

```
b31flibs.lib --- small  model library produced by Borland C++ 3.1
b31flibm.lib --- medium model library produced by Borland C++ 3.1
v10flibs.lib --- small  model library produced by Visual C++ 1.0
v10flibm.lib --- medium model library produced by Visual C++ 1.0
v15flibs.lib --- small  model library produced by Visual C++ 1.5
v15flibm.lib --- medium model library produced by Visual C++ 1.5
```

To use the class library, you need to choose one of these LIB file according to the memory model and compiler you use, include it in your project file, and include the header file called "flib.h" in your module making use of the class library.

There are two classes defined in this library which are available to you,

- dlgprog, a dialog box with a progress bar, an abort button, and three static text controls.
- installer, provides routines like file copying, version checking.

The project file and source code for install.exe and llatsni.exe except those for the library will also be included so that you can modify/augment them to get your own customized install program,

```
For Installer:
b31finss.prj --- small  model project file for Borland C++ 3.1
b31finsm.prj --- model  model project file for Borland C++ 3.1
v10finss.mak --- small  model project file for Visual C++ 1.0
v10finsm.mak --- medium model project file for Visual C++ 1.0
v15finss.mak --- small  model project file for Visual C++ 1.5
v15finsm.mak --- medium model project file for Visual C++ 1.5
finstall.cpp --- source code for install.exe (installer)

For Uninstaller:
b31fllas.prj --- small  model project file for Borland C++ 3.1
b31fllam.prj --- model  model project file for Borland C++ 3.1
v10fllas.mak --- small  model project file for Visual C++ 1.0
v10fllam.mak --- medium model project file for Visual C++ 1.0
v15fllas.mak --- small  model project file for Visual C++ 1.5
v15fllam.mak --- medium model project file for Visual C++ 1.5
fllatsni.cpp --- source code for llatsni.exe (uninstaller)
```

In the source code, neither OWL nor MFC is used, i.e., only the native Windows API is used and thus it is not difficult for any of you to modify it.

Registering Freeman Installer

When & why to register?

The version you are keeping is 1.1 beta and it is absolutely free. So, if, fortunately, it suits your needs and you are happy with it, you can use it without any cost.

However, if you would like to,

- help a Ph.D. student getting economically independent of his parents; and/or
- upgrade to version 1.1 and newer; and/or
- be entitled to a life time technical support (mail and e-mail only); and/or
- obtain the C++ class library

You should get registered.

How to register?

The registration fee is US$30. The postage fee is US$5 if you are out of Australia, US$3 if you are in Australia, US$0 if you can and wish to receive the software via Internet e-mail.

All payments must be in US$, Australian$, or HK$ in the form of cash, bank draft, or international money order. Sending cash is usually the cheapest way. However, If you do, you do it at your own risk.

If possible, please fill out the order form (double click the MS Write icon in the Freeman Installer group) and mail/e-mail/fax it to me. Otherwise, please give your name, address, disk format preferred, class library preferred (Borland C++ or Visual C++) in the mail and state that your are ordering Freeman Installer.

How to dig me out?

All queries, bug reports, suggestions, and payments are welcome (especially the last ones) and should be directed to,

Ka Iok Tong
Mail Box 20
Wentworth Building
University of Sydney
Sydney NSW 2006
Australia
e-mail: tongk@archsci.arch.su.edu.au
ph & fax: (61) (2) 2116314