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

Copyright & license

How the classes work together

If you don't need bells and whistles, you don't need to read this topic.

Class installer

This class provides the common routines pertaining to installation and uninstallation. For example, it has a member function to copy a file, to create a program group, to create a program item. But it doesn't know any logic of the whole installation. For example, it doesn't know the files will be copied one after one, nor does it know the program items will be created after all the files have been copied. This kind of control flow is determined by the class installcontroller.

The routines provided by this class are more than those actually used by the installer. For example, it has a member function to execute an EXE file and wait until it terminates, the installer doesn't call this at all. When you need to perform an extra operation, the installer is the best place to look into.

See member functions of the class installer.

Class installcontroller

This class provides the most control flow of the whole program. It asks the class installinfo for the install info such as how many files to copy, what are their names, and it dispatches the class installer to do the real job. For example, it has a member function called copyfiles() which looks like this,

```
int installcontroller::copyfiles()
{
    n = ask installinfo how many files are there;
    for (i = 0; i < n; i++)
    {
        ask installinfo what the name of the i'th file is;
        ask installinfo what the source dir of the i'th file is;
        ask installinfo what the target dir of the i'th file is;
        call the installer to copy the file;
    }
}
```

There is a hole is this class. It leaves a virtual function called run() to be defined by its derived class. run() is supposed to control the top level control flow of the whole program.

The followings are the data members you might want to access,

```
saint ics;                                    /* is the component selected? */
saint ifs;                                         /* is the file selected? */
saint iis;                                         /* is the item selected? */
saint ies;                                        /* is the entry selected? */
saint irs;                                      /* is the reg key selected? */
installer ir;                                                  /* installer */
installinfo *ii;                                 /* custom install info obj */
```

Note that,

1. ir is the installer obj. The typical use is to call the common routines such as creating a program item,

   ```
   ir.createitem(...);
   ```

2. ii points the installinfo obj. The typical use is to typecast it to a pointer to the actual

installinfo obj, i.e., an instance of class fiuserinfo (a derived class of installinfo), to access the extra variables,

```
((fiuserinfo*)ii)->extravariable
```

3. ics, ifs, iis, ies and irs are integer arrays to store the selection state of the components, files, program items, INI entries and registration keys respectively. See determine install or not according to criteria other than version info and user selection for the typical use of them.

The followings are the function members you might want to access,

```
char *getdiri();       // returns the install target dir
char *getdirs();       // returns the source dir
char *getdirw();       // returns the Windows dir
char *getdiry();       // returns the shared dir
```

Class installinfo

This is an abstract class. It prescribes that any derived class must answer some questions (asked by the class installcontroller). For example, it has a pure virtual function getnofiles() which is supposed to return the total number of files to copy,

```
class installinfo
{
    ......
    virtual int getnofiles() = 0;
    ......
};
```

It has only one data member, i.e., a reference to the installcontroller obj it is working with,

```
class installinfo
{
    ......
    installcontroller &ic;       /* the controller we are working with */
    ......
};
```

The typical use of ic is to access the controller, from which we can access its data members and the installer,

```
char *p;
p = ic.getdiri();
ic.ir.createitem(...);
```

Class fiusercontroller

This is a user class generated by Freeman Constructor. It is a derived class of installcontroller. It defines its only member function run() to fill out the hole left by the class installcontroller.

```
class fiusercontroller:public installcontroller
{
   public:
   fiusercontroller(...):installcontroller(...)
   {


   }
   int run();
};
```

run() is generated according to the install info you told Freeman Constructor. Here is a typical definition of run(),

```
// return: true --- ok   false --- error or cancel
int fiusercontroller::run()
{
   if (!welcome())          // show welcome message
   {
      return 0;
   }
   if (!selectcomps())      // select components
   {
      return 0;
   }
   if (!askdir())           // ask the user to input install target dir ($i)
   {
      return 0;
   }
   logmover lm(*this);      // when destructed, move log file to $i
   if (!copyfiles())        // copy files
   {
      return 0;
   }
   switch (createitems())   // create program items
   {
      case 0:
         break;             // ok
      case 1:
         break;             // don't create
      case 2:
         return 0;          // error
   }
   if (!setinientries())    // create/set ini entries
   {
      return 0;
   }
   if (!setregkeys())       // create/set registration keys
   {
      return 0;
   }
   lm.move();               // restarting windows won't activate lm's destructor
   done();                  // display installation complete message
   return 1;
}
```

It performs the following tasks in the order shown,

1. Displays welcome dialog box.
2. Displays the component selection dialog box, if any.
3. Prompts the user for the install target directory.
4. Copies files, if any.
5. Creates program items, if any.
6. Sets INI entries, if any.
7. Sets registration keys, if any.
8. Displays installation completion dialog box or restart Windows dialog box.

For most customizations, run() is the best place to modify (usually add several lines of code).

Class fiuserinfo

This is a user class generated by Freeman Constructor. It is a derived class of installinfo. It defines all the member functions prescribed by the class installinfo, to actually answer the questions asked by the class installcontroller. For example, it may define getnofiles() like this,

```
class fiuserinfo::installinfo
{
    ......
    int getnofiles()
    {
        return 10;
    }
    ......
};
```

These member functions are generated according to the install info you told Freeman Constructor. Usually you don't need to modify them at all unless you need to determine the install info dynamically.

installer::copy(char srcpath[], char dstpath[], int ischkver, int isshared, int iscompressed);

Copies a file. Optionally performs version checking and decompression.

In:

```
srcpath          ---     path of the source file
dstpath      --- path of the target file
ischkver     --- need version checking or not
isshared     --- is a shared file or not. It will just be saved in the log
iscompressed --- is a compressed file or not
```

Return:

```
0 --- copied
1 --- skipped
2 --- error
```

Member functions of class installer

Here we only list the member functions you are most likely need to call. If you really need to know other member functions, you can check the header file installr.h or even ask me.

If not stated otherwise, all member functions returning an integer will return true on success and return false on failure (error or user cancel).

If not stated otherwise, all member functions will report error before returning if it encountered an error.

If not stated otherwise, all changes made by all member functions will be recorded in the install log.

File operations

```
int copy(char src[], char dst[], int ischkver, int isshared, int iscompressed);
int deldir(char dir[]);
// deletes a dir. it will fail if the directory is not empty
int multideldir(char dir[]);
// deletes a dir. all the files and sub-dirs will be deleted as well
int multimkdir(char dir[]);
// create a dir. all dirs involved will be created if necessary
int delfile(char path[]);
// deletes a file
int renfile(char oldpath[], char newpath[]);
// renames a file from oldpath to newpath
```

Program item operations

```
int begddepm();
// begins DDE'ing with Program Manager. runs Program Manager if necessary
// before calling any functions dealing with program items, this function
// must be called
void endddepm(int isendpm);
// finishes DDE'ing with Program Manager
int addgrup(char grup[]);
// creates a program group
int delgrup(char grup[]);
// deletes a program group
int additem(char grup[], char item[], char cmdl[], char icon[], int iidx = 0);
// creates a program item
int delitem(char grup[], char item[]);
// creates a program item
```

INI entry operations

```
int delinisect(char file[], char sect[]);
// deletes an INI section and all entries in it
int addinisect(char file[], char sect[]);
// creates an INI section
int insinientry(char file[], char sect[], char entry[], char value[], int seqno);
```

```
// inserts an INI entry at the given index (0 is the first, 1 is the second)
int addinientry(char file[], char sect[], char entry[], char value[]);
// appends an INI entry
int delinientry(char file[], char sect[], char entry[], int seqno);
// deletes an INI entry at the given index (0 is the first, 1 is the second)
int setinivalue(char file[], char sect[], char entry[], char value[], int seqno);
// sets an INI entry at the given index (0 is the first, 1 is the second)
```

Registration key operations

```
int setregkey(char kpath[], char value[]);
// sets the value of the registration key. creates the key if doesn't exist
int addregkey(char kpath[]);
// creates a registration key
int delregkey(char kpath[]);
// deletes a registration key
int getregkey(char kpath[], char value[]);
// get the value of a registration key
int getnosubkeys(char kpath[]);
// get the number of sub-keys of a registration key
```

Misc operations

```
int execwait(char path[], int showcmd);
// like WinExec except that it will wait until the child process terminates
void mergedf(char path[], char d[], char f[]);
// merges the dir and file to get a full path
void getdfrompath(char path[], char d[]);
// merges the dir from a full path
void getffrompath(char path[], char f[]);
// merges the file from a full path
```

Edit>General Information

Application name

The name used to refer to your application. It is used in a couple of dialog boxes.

Default installation target directory

The default value of the installation target directory. It is used to initialize the edit control when prompting for the installation target directory.

Welcome texts

They will be displayed in the welcome dialog box.

Minimum time to copy a file

Suppose that it is 1000 ms. If it only takes 600 ms to copy a file readme.txt, the installer will wait 400 ms before copying the next file. It is used to slow down the copying process to let the user know what is going on. It is particularly useful for small applications.

Putting 0 here will instruct the installer not to wait at all.

Divide files into blocks of this big

Suppose that it is 1024 bytes. If a file manual.hlp is 2050 bytes, the installer will copy it at three times, one block at a time. The block sizes are 1024, 1024, 2 bytes respectively.

Putting 0 here will instruct the installer copy the whole file without dividing it into blocks.

Here blocks are used as a kind of time unit to control how often to check the abort button and how often to update the progress bar.

Check abort button when ?? blocks have been copied

Suppose that it is 2. The installer will check if the abort button has been pressed when 2 blocks have been copied. The smaller this value is, the quicker the response to the abort button is.

Setting it to 1 is usually the best. Checking the abort button virtually doesn't slow down anything.

Never put 0 here. It will cause the installer stop checking the abort button.

Update progress bar when ?? blocks have been copied

Suppose that it is 2. The installer will update the progress bar when 2 blocks have been copied. The smaller this value is, the more frequently the progress bar is updated.

Setting it to a appropriate value according to the total size of the files in your application. Suppose that the total size is 40960 bytes and the block size is 1024 bytes, if you want the progress bar be updated at an interval of 5%, you should put 2 here, i.e., update every 2 * 1024 bytes = 2048 bytes = 5% * 40960 bytes.

Never put 0 here. It will cause the installer stop updating the progress bar.

Is in debug mode

If the installer is in debug mode, the files will not be actually copied. This is used to speed up the testing of the install info you told Freeman Constructor.

Kill Program Manager if we started it to DDE with

If the shell does not support the shell DDE commands, the installer will load Program Manager to create the program items. This controls whether the installer should kill Program Manager or not when finishes creating the program items.

If you don't kill it, the user can have a chance to pick up those items he/she thinks useful and migrate them to his/her shell. But this might also give the user an impression that the install program doesn't clean up its own mess.

Edit>Disk Descriptions

Overview

A disk description is just a string description about a disk. When you are editing the set of distribution files, for each file you can choose a disk description for it. When the installer can not find a particular file, it will use the corresponding disk description as a (part of the) prompt to ask the user to insert the disk. You must enter the disk descriptions here before you can refer to them.

Insert

Insert a new disk description at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new disk description as the last disk description.

Delete

Delete the selected disk description.

Edit>Files

Overview

In here we specify the set of files to be distributed and installed. The order of the files appearing in the list box is important. The first file will be copied first and then the second file and so on. The fastest way to specify the files is selecting them in File Manager and then dragging them into the list box. The files dragged from File Manager will be inserted at the currently selected position. If nothing is selected, they will be inserted at the end. If you want to rearrange the order, you can use the cut and paste method.

Show full path

Show the full path rather than file name of the files in the list box.

Path

The path of the selected file. Note that this is the path on your own computer and has nothing to do with installation. In the current version of Freeman Installer, the purposes of the path are,

1. We need the file name.
2. We need the path to renew the file sizes.

In the future version, the path may be used to compress the files as well.

Source directory

The source directory of the file. If source directory is a: and the file name is readme.txt, the installer will read from the file a:\readme.txt.

If the file resides in the same directory as the installer, you should put $s here. If the file resides in a sub directory of that called doc, you should put $s\doc here. If the file is not in the same directory nor in a sub directory, you must define your own variable.

See variables.

Target directory

The target directory of the file. If target directory is c:\csh\doc and the file name is readme.txt, the installer will write to the file c:\csh\doc\readme.txt. You don't need to bother about whether c:\csh and c:\csh\doc exist or not. If necessary, the installer will create them automatically.

If the file should reside in the install target directory, you should put $i here. If the file should reside in a sub directory of that called doc, you should put $i\doc here. If the file should reside in the Windows directory, you should put $w here. If the file is a shared file like bwcc.dll and ctrl3d.dll, you should put $y here. If the system variables are not enough, you must define your own variable.

See variables.

Description

The description for the file. It will be displayed in the dialog box while the file is being copied.

File size

The size (in bytes) of the file. If the file will be compressed, you should still used the original size here. You are NOT encouraged to enter the size yourself. Rather, you are encouraged to drag the files from File Manager into the list box. This way Freeman Constructor will be able to get the size of the files automatically. If one of more files are modified and as a result their sizes change later, you should issue the menu command Make>Renew File Sizes to update the sizes. Unlike previous versions, the file size must be accurate in order to set the progress bar properly.

Source disk

The disk description for the file. See Edit>Disk Descriptions.

Will be distributed in compressed format

The file will be compressed.

Need version checking

The file carries version info and needs version checking. This is mainly for DLLs.

Don't remove this file when uninstalling

Don't remove this file when uninstalling. This is mainly for DLLs.

Insert

Insert a new file at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new file as the last file.

Cut

Cut the selected file (and the info about it) and put it onto the clipboard.

Copy

Copy the selected file (and the info about it) onto the clipboard.

Paste

Insert the file (and the info about it) on the clipboard at the currently selected position. If nothing is selected, it is equivalent to Paste End.

Paste End

Insert the file (and the info about it) on the clipboard as the last file.

Edit>Program Items

Overview

In here we specify the set of program items to be created. The order of the items appearing in the list box is important. The first item will be created first and then the second item and so on.

Item

The name of the item.

Group

The name of the program group the item will belong to. If the group does not exist, it will be created automatically.

Command line

The command line of the item. Here are some examples,

```
$i\csh.exe                     run csh.exe in the install target dir
$i\csh.exe -e $i\script        same as above but passing two parameters
$w\winhelp.exe $i\manual.hlp   run help on the manual help file
$i\manual.hlp                  same as above
```

Icon path

The path pointing to the icon file for the item. Here are some examples,

```
$i\csh.exe           use an icon in csh.exe in the install target dir
$w\winfile.exe       use File Manager's icon
```

If you set it to empty, Program Manager will get the icon from the command line.

Icon index

0 means using the first icon in the icon file. 1 means the second icon and so on.

Insert

Insert a new item at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new item as the last item.

Delete

Delete the selected item.

Edit>INI Entries

Overview

In here we specify the set of INI entries to be created, set, or added.

Reboot to take effect

It determines if the installer should ask the user to restart Windows or not upon installation completion. It can take 3 different values,

1. Don't reboot	Never ask the user to restart
2. Reboot	Must ask the user to restart
3. Reboot if system.ini changed	Ask the user to restart if system.ini was changed

File

The INI file (path). If it doesn't exist, it will be created (including all directories involved) automatically. Here are some examples,

```
$i\csh.ini          File csh.ini in install target dir
$w\win.ini          win.ini
$w\system.ini       system.ini
```

Section

The section of the entry. If it doesn't exist, it will be created automatically.

Entry

The name (key) of the entry.

Value

The value of the entry.

Will be appended rather than set

The entry will be appended after all entries with the same name (key). Usually this is used to add device drivers to system.ini because all device driver entries use the same name (key) "driver".

Edit>Registration Keys

Overview

In here we specify the set of registration keys to be created or set.

Key path

The path of the registration key. Here are some examples,

```
PBrush

PBrush\shell

PBrush\shell\open\command
```

Suppose that the path is PBrush\shell\open\command and there is an existing key PBrush but not PBrush\shell, the key PBrush\shell, PBrush\shell\open, and PBrush\shell\command will be created automatically, with their values set to empty.

Value

The value of the registration key. This can be empty. Here are some examples,

Key path	Value
PBrush	Paintbrush Picture
PBrush\shell	
PBrush\shell\open\command	pbrush.exe %1

Insert

Insert a new registration key at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new registration key as the last key.

Delete

Delete the selected registration key.

Edit>Software Components

Overview

A leaf component contains some files, program items, INI entries, and registration keys. If a component is selected by the user, all the files, program items, INI entries, and registration keys contained in it will be installed. If a component is not selected, the files, program items, INI entries, and registration keys might still be installed because they might be contained in another leaf component, i.e., different leaf components can share some or even all their contents.

A branch component contains some other branch components and leaf components. If a branch component is selected by the user, all the components (branch or leaf) contained in it will be selected. If a component is not selected, the components might still be selected because they might be contained in another branch component, i.e., different branch components can share some or even all their contents.

The whole structure must have exactly one component which has no parent, i.e., not contained in any other component. This is the root component.

This structure will be displayed in a DAG (directed acyclic graph). A DAG is like a tree except that sub trees can share their offsprings. Each component is represented as a node in the DAG. A node is in turn visualized as a check box, and with a push button at its right if it is a branch component. The check box lets the user select/deselect the corresponding component. The text of the check box is the name of the component. The push button lets the user expand/collapse the corresponding branch component, i.e., show/hide its sub components. A "+" appearing in the push button means that the branch node can be expanded. A "-" appearing in the push button means that the branch node can be collapsed.

There is a progress bar in the component selection dialog box indicating the maximum disk space needed and the disk space needed by the current selection. When the user select/deselect a component, the progress bar will be updated accordingly.

Maximum installation and minimum installation are simply implemented as trying to select the root component and trying to deselect the root component respectively. If all the components can be deselected, the minimum installation will install NOTHING. Therefore, for the minimum installation to really install something, you should pre-select the necessary components as the minimum installation and disable them (see below for how to do this), making it impossible to deselect them.

Text width

The width of the check box (including the text) in dialog units. Because all nodes use the same size, you should set this value big enough to hold the component with the longest name.

Button width

The width of the push button in dialog units.

Height

The height of the check box and the push button in dialog units (they use the same height).

Name

The name of the component. This will be used as the text of the check box.

Sub-components

All potential sub-components of the currently selected component are listed here. The actual sub components are selected.

If the currently selected component is a leaf component, the potential sub-components are all files, program items, INI entries, registration keys.

If the currently selected component is a branch component, the potential sub-components are all components. Although all components are listed there, you MUST NOT select its ancestor or itself as its sub-component.

Is leaf component

Determine if the currently selected component is a leaf component or a branch component.

Is check box enabled

If the check box is enabled, the user can select or deselect it. Otherwise, the user can not change the state of the check box. This is useful when you need to force the user to install something (such as in the case of minimum installation) or prohibit the user from installing something (such as in the case of crippled software). See below for how to achieve this effect.

Is checked at startup

Determine if the check box is checked or not initially. If the check box is disabled, checking the check box initially will force the user to install it, unchecking the check box initially will prohibit the user from installing it.

Show sub-components at startup

Determine if the sub-components of the currently selected component should be shown or not initially.

Insert

Insert a new component at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new component as the last component.

Cut

Cut the selected component (and the info about it) and put it onto the clipboard.

Copy

Copy the selected component (and the info about it) onto the clipboard.

Paste

Insert the component (and the info about it) on the clipboard at the currently selected position. If nothing is selected, it is equivalent to Paste End.

Paste End

Insert the component (and the info about it) on the clipboard as the last component.

Edit>Background Texts

Overview

The background the installer is a gradient blue screen on top which there are some texts. In here you specify how many there are, their positions, colors, fonts, sizes, the order they are drawn. The order they are drawn is the order they appear in the list box. The first one is drawn first and then the second one and so on. If you are going to add 3D effect to the text, you should draw a white text first and then the original text. If you are going to add shadow effect to the text, you should a black text first and then the original text.

Text

The text string to be drawn.

x

The x coordinate of the left top corner of the text. The coordinate is in points, starting from the left top corner of the screen.

y

The y coordinate of the left top corner of the text. The coordinate is in points, starting from the left top corner of the screen.

Choose Font

The change the font of the currently selected text.

Test

Draw the background (including the texts) immediately to let you see the outcome. Clicking the mouse will end the test.

Insert

Insert a new text at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new text as the last text.

Delete

Delete the selected text.

Edit>Advertisement Dialogs

Overview

If you specify some dialog boxes (or rather, their ascii id's), the installer will displayed them one by one while copying files. The order in which they are listed in the list box is important. The first one will be displayed first and then the second and so on. Suppose that the total size to be copied (after user selection) is 3000 bytes and there are 3 advertisement dialog boxes, the first one will be displayed while copying 0--1000 bytes. The second one will be displayed while copying 1001--2000 bytes and so on. An advertisement dialog box is basically a usual dialog box except that it has the built-in capacity to display bitmaps. The capacity is there, you can make use of it or not at your own discretion.

Dialog resource name

The ascii id of the advertisement dialog box. You must use ascii id in the resource file to identify the dialog box resource, i.e., integer id is not supported.

Insert

Insert a new dialog box at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new dialog box as the last dialog box.

Delete

Delete the selected dialog box.

Edit>Variables

Overview

In here you specify the extra variables you need. For most applications, you don't need any extra variables. Freeman Constructor will add one data member to the class fiuserinfo for each extra variable you specify here. For the extra variables to be usable, you need to initialize/set their values yourself. See defining extra variables.

Variable

The name of the extra variable.

Insert

Insert a new variable at the currently selected position. If nothing is selected, it is equivalent to Insert End.

Insert End

Insert a new variable as the last variable.

Delete

Delete the selected variable.

Make>install.inf

Save the install info you told Freeman Constructor into a file called install.inf in the current directory. This contents of this file are 100% identical to that of the one generated by issuing the menu command File>Save. But they are of different purposes. install.inf is intended to be read/interpreted by install.exe. The file generated by issuing File>Save is intended to be kept by yourself.

Make>fiuser.cpp

Compile the install info you told Freeman Constructor into a C++ source file called fiuser.cpp in the current directory. According to the install info, Freeman Constructor defines two tailor-made C++ classes, fiusercontroller and fiuserinfo in fiuser.cpp. This file is ready to compile and link with class library to generate an integrated and compact EXE file. See compilation approach.

Note that if, for customizations, you have modified an existing fiuser.cpp, issuing this command will OVERWRITE your customized version without a single warning.

Make>Renew File Sizes

After specifying the files to be distributed, if you didn't use drag and drop method to specify the files, or some of the files have changed their sizes since then, you should issue this command to force Freeman Constructor to re-collect the sizes of the files. If, in this process, a file can not be found, it will ask you if you want to remove the file from distribution or not. This is a fast way to remove files which no longer exist on your system. It is suggested that you at least issue this command once before making the final distribution disks, because in this version of Freeman Installer, the sizes of the files must be accurate.

Freeman Constructor

Overview

Freeman Constructor is a front end to let you specify the install info, e.g., how many files you want to copy, what are their names and so on. Using Freeman Constructor is the fastest and recommended way to specify your install info.

It can save the install info in INI format as its native file format. Although the INI file is very similar to the INF file supported in the previous versions, the INI file is not supposed to be modified manually. In fact, doing this will not provide you with any extra freedom, i.e., anything you can specify in that INI file can also be specified in Freeman Constructor.

Freeman Constructor can, according to the info you told it, generate an INF file called install.inf. This file is 100% identical to the INI file it may generate. The INF can be interpreted by the file install.exe to do the installation. See translation approach.

Freeman Constructor can compile the info you told it into a C++ source file called fiuser.cpp. This file is ready to compile and link with the class library file to get an integrated and tailored EXE file. See compilation approach.

Practical Hints on Using Freeman Constructor

1. You can use cut, copy and paste in ALL edit controls. When you are entering data in those dialog boxes, using them will save you much time.

```
Cut   --- Ctrl-X
Copy  --- Ctrl-C
Paste --- Ctrl-V
```

2. Looking for an example (typical values)? Load the file install.inf and then go to the dialog boxes to inspect the values.

Command explanations

Edit>General Information

Edit>Disk Descriptions

Edit>Files

Edit>Program Items

Edit>INI Entries

Edit>Registration Keys

Edit>Software Components

Edit>Background Texts

Edit>Advertisement Dialogs

Edit>Variables

Make>install.inf

Make>fiuser.cpp

Make>Renew File Sizes

Variables

Freeman Installer supports variables. The name of a variable is a string containing alphabets and/or digits (but the first character must be alphabet). The value of a variable is a string. You can reference a variable (i.e., retrieve its value) by preceding the name of the variable with a dollar sign "$", i.e., $xxx means the value of the variable called "xxx". However, you can do this only in Freeman Constructor when specifying string parameters. Also, for some groundless reason, component names should not contain variable references.

There are four system variables,

1. $i. Its value is the install target directory entered by the user.
2. $s. Its value is the source directory, i.e., the directory where the installer resides. Usually it is "a:".
3. $w. Its value is the Windows directory. Usually it is "c:\windows".
4. $y. Its value is the directory where shared files such as bwcc.dll and ctrl3d.dll should resides. Usually it is the Windows system directory like c:\windows\system But it is the Windows directory under a network shared copy of Windows.

For examples, you may use

`$i\lib`	as	the target dir of a library file
`$s\doc`	as	the source dir of a document file
`$w\notepad.exe`	as	path of notepad
`$y`	as	the target dir of DLL

These four variables are enough for most occasions. Otherwise, you need to define your own.

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

Version Checking

What is version checking?

Version statement is a kind of resource whose purpose is to attach version info (version number, among some other things) to a file.

Version checking means when trying to copy a file onto the user's system, if there is already an existing file with the same name in the intended target directory, the version statement of two files are retrieved and compared against each other to determine which one is newer.

What is the implementation in Freeman Installer?

The installer will decide to overwrite the existing file if and only if at least one of the following conditions holds,

1. 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.
2. The existing file has no version info.

The installer will decide to keep the existing file if and only if at least one of the following conditions holds,

1. 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.
2. The existing file has version info but the distribution file has not.

If none of these conditions holds, the installer 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.

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.
3. Tell Freeman Constructor that these files are already compressed.

Using Freeman Installer

If you are taking the translation approach,

1. Run Freeman Constructor to specify what files you are going to install onto your users' system, what program items you are going to create and etc.
2. Issue the menu command Make>install.inf to generate the file install.inf.
3. Copy install.exe and install.inf onto your first distribution disk. Before that you should remove all unwanted resources in install.exe. For example, there is a large bitmap which is displayed in an advertisement dialog box while copying files. If you don't need them, you should remove them to save space.
4. Compress those files you want to distribute in compress format. You must use the Microsoft's compress.exe with the -r option, e.g.,

```
compress -r myfile.exe
```

5. Copy all files comprising your application onto your distribution disks.

You need to do something more If you are going to make your application uninstallable.

If you are taking the compilation approach,

1. Run Freeman Constructor to specify what files you are going to install onto your users' system, what program items you are going to create and etc.
2. Issue the menu command Make>fiuser.cpp to generate the file fiuser.cpp.
3. Go to your favorite C++ compiler, open the appropriate project/make file (e.g., b31fins.prj if BC 3.1 is used), make the project to generate an EXE file (e.g., b31fins.exe if BC 3.1 is used).
4. Copy the EXE file (e.g., b31fins.exe) onto your first distribution disk. Before that you should remove the debug info (if any) in the EXE file.
5. Compress those files you want to distribute in compress format. You must use the Microsoft's compress.exe with the -r option, e.g.,

```
compress -r myfile.exe
```

6. Copy all files comprising your application onto your distribution disks.

You need to do something more If you are going to make your application uninstallable.

Making your application uninstallable

In order to make your application uninstallable, you need to,

1. Copy llatsni.exe onto your distribution disk. It is recommended that you compress it.
2. Install it onto your users' Windows directory. Because starting from this version of Freeman Installer, the uninstaller has become an application shared by different applications from different sources (just like bwcc or ctrl3d), it contains a version statement in it. In cooperation, you should tell Freeman Constructor that it needs version checking and it should NOT be removed on uninstallation.
3. Create a program item for it. Use the second icon in llatsni.exe, i.e., icon index should be 1 rather than 0. Because there may be more than one applications to get uninstalled on the user's system but there can be only one copy of llatsni.exe, you must tell llatsni.exe which application is being uninstalled by providing the path to the installation log as the only command line parameter for the uninstaller, i.e., the command line of the program item should be like,

```
$w\llatsni.exe $i\filog.ini
```

An INI file called filog.ini. This file will be created by Freeman Installer automatically during the installation to log down what changes have been made to the user's system. Upon completion, this file will be moved to the install target directory (where it initially was is not important). The uninstaller, upon request, will try to undo these changes recorded in this file.

Translation Approach

In this approach, you use two files to provide the functionality of an installer, i.e., the file install.exe coming with Freeman Installer plus the file install.inf generated by Freeman Constructor according to what you told it. The role played by install.exe is exactly that of an interpreter. It reads the file install.inf and perform the appropriate operations required.

You should take this approach when,

1. You are testing/debugging to see if the info you told Freeman Constructor is really what you want.
2. You don't have a copy of a C++ compiler or it is not one of BC 3.1, BC 4.0, VC 1.0, VC 1.5.
3. You don't care about the size or the speed of the install program.

Usually, the compilation approach is better than the translation approach when you are actually making your final distribution disks. At that time the compilation approach should be used as far as possible.

See Also

Using Freeman Installer

Compilation Approach

In this approach, you use a single file to provide the functionality of an installer, i.e., the EXE file generated by your own C++ compiler, made from the class library (e.g., b31flib.lib for BC 3.1) coming with Freeman Installer plus the file fiuser.cpp generated by Freeman Constructor according to what you told it. The file fiuser.cpp is exactly the compiled image/version of the info you told Freeman Constructor

You should take this approach when,

1. You have finished testing/debugging and you are sure what you told Freeman Constructor is really what you want.
2. You have a copy of BC 3.1, BC 4.0, VC 1.0, or VC 1.5. It does NOT matter whether you know C++ or not. The file fiuser.cpp is ready to compiler and run.
3. When you need bells and whistles. This means what you want to tell Freeman Constructor is more than what it can accept.
4. You want to minimize the size and/or to maximize the speed of the install program.

Usually, the translation approach can give faster feedback than the compilation approach when you are testing/debugging. At that time the translation approach should be used as far as possible.

See Also

Using Freeman Installer

Determine install info dynamically

Suppose that the name of your application is normal "C Shell v2.0". But you want to set it to "C Shell v2.0 upgrade" if a previous version of your application can be found when it is being installed. To do this, you need to locate the member function getappname() of the class fiuserinfo in the file fiuser.cpp. The original getappname() should be like this,

```
char *fiuserinfo::getappname()
{
    wsprintf(sharedbuf, "C Shell v2.0");
    return sharedbuf;
}
```

You need to change it to,

```
char *fiuserinfo::getappname()
{
    if (there is a previous version on this computer)
    {
        return "C Shell v2.0 upgrade";
    }
    wsprintf(sharedbuf, "C Shell v2.0");
    return sharedbuf;
}
```

Because installcontroller does NOT save the info provided by installinfo, getappname() will be called whenever installcontroller needs to know the application name, in this case, more than once. In order to avoid searching for a previous version for more than once, we better off searching for it at the beginning and putting the appropriate application name in a variable. When getappname() is called, we simply return the variable,

```
static char *appname;       // store the application name
int fiusercontroller::run()
{
    // CUSTOMIZATION BEGINS: determine the appropriate application name
    if (there is a previous version on this computer)
    {
        appname = "C Shell v2.0 upgrade";
    }
    else
    {
        appname = "C Shell v2.0";
    }
    // CUSTOMIZATION ENDS
    // show welcome message
    if (!welcome())
    {
        return 0;
    }
    ......
}
char *fiuserinfo::getappname()
{
    return appname;
}
```

Because welcome() will call getappname(), we must set the application name before welcome() is executed, in this case, at the very beginning of the member function run() of the class fiusercontroller.

Note that we set appname in a member function of fiusercontroller but we reference appname in a member function of fiuserinfo. Therefore appname is required to be visible for both classes. Here we define appname as a static variable. We could also define appname as a data member of fiuserinfo,

```
        class fiuserinfo
        {
            public:
            fiuserinfo(installcontroller &ic):installinfo(ic)
            {

            }
           ~fiuserinfo()
            {

            }
            char *appname;        // store the application name
            char sharedbuf[256];
            ......
        };
    int fiusercontroller::run()
    {
        // CUSTOMIZATION BEGINS: determine the appropriate application name
        if (there is a previous version on this computer)
        {
            ((fiuserinfo*)ii)->appname = "C Shell v2.0 upgrade";
        }
        else
        {
            ((fiuserinfo*)ii)->appname = "C Shell v2.0";
        }
        // CUSTOMIZATION ENDS
        // show welcome message
        if (!welcome())
        {
            return 0;
        }
        ......
```

```
}
char *fiuserinfo::getappname()
{
    return appname;
}
```

Now, you can proceed as normal, i.e., build the project. Also, please keep in mind that if you use Freeman Constructor to generate the file fiuser.cpp again, your customized version will be overwritten.

Disabling built-in operations

Suppose that you don't want to prompt for the install target directory, you need to remove or comment out the function call askdir() in run() of the class fiusercontroller,

```
    int fiusercontroller::run()
    {
        // show welcome message
        if (!welcome())
        {
            return 0;
        }
        // select components
        if (!selectcomps())
        {
            return 0;
        }
// DON'T ask the user to input install target dir ($i)
//      if (!askdir())
//      {
//          return 0;
//      }
        // when destructed, move log file to $i
        logmover lm(*this);
        // copy files
        if (!copyfiles())
        {
            return 0;
        }
        ......
    }
```

Now, you can proceed as normal, i.e., build the project. Also, please keep in mind that if you use Freeman Constructor to generate the file fiuser.cpp again, your customized version will be overwritten.

Performing extra operations

Suppose that you want to execute notepad.exe to load a file readme.txt in the install target directory just after displaying the installation completion message, you need to add a piece of code to run() of the class fiusercontroller,

```
int fiusercontroller::run()
{
    // show welcome message
    if (!welcome())
    {
        return 0;
    }
    ......
    // restarting windows won't activate lm's destructor
    lm.move();
    // display installation complete message
    done();
    // CUSTOMIZATION BEGINS: executing notepad.exe to load readme.txt
    char readmepath[128];   // e.g., c:\cshwin\readme.txt
    char commandline[256];  // e.g., c:\windows\notepad.exe c:\cshwin\readme.txt
    ir.mergedf(readmepath, getdiri(), "readme.txt");   // get readme path
    ir.mergedf(commandline, getdirw(), "notepad.exe"); // get notepad path
    lstrcat(commandline, " ");                         // concat readme path
    lstrcat(commandline, readmepath);
    WinExec(commandline, SW_SHOWNORMAL);               // execute but not wait
    // CUSTOMIZATION ENDS
    return 1;
}
```

In fact, there is no need to specify the full path of notepad because WinExec will search through the Windows directory. The purpose here is to show you how to use,

1. mergedf() to merge a directory and a file to get the full path.
2. getdiri(), getdirw(), getdirs(), getdiry() to get the install target directory, Windows directory, install source directory, and Windows system directory respectively.

If you want to execute notepad and wait until it finishes, you can use the member function execwait() of the class installer. It is similar to WinExec except that it will wait until the child process terminates,

```
int fiusercontroller::run()
{
    // show welcome message
    if (!welcome())
    {
        return 0;
    }
    ......
    // restarting windows won't activate lm's destructor
```

```
        lm.move();
        // display installation complete message
        done();
        // CUSTOMIZATION BEGINS: executing notepad.exe to load readme.txt
        char readmepath[128];   // e.g., c:\cshwin\readme.txt
        char commandline[256];  // e.g., c:\windows\notepad.exe c:\cshwin\readme.txt
        ir.mergedf(readmepath, getdiri(), "readme.txt");   // get readme path
        ir.mergedf(commandline, getdirw(), "notepad.exe"); // get notepad path
        lstrcat(commandline, " ");                         // concat readme path
        lstrcat(commandline, readmepath);
        ir.execwait(commandline, SW_SHOWNORMAL);           // execute and wait
        // CUSTOMIZATION ENDS
        return 1;
    }
```

Now, you can proceed as normal, i.e., build the project. Also, please keep in mind that if you use Freeman Constructor to generate the file fiuser.cpp again, your customized version will be overwritten.

Defining extra variables

Suppose that you want to define an extra variable called cddrive and use it to hold the drive name (e.g., "e:") from which your user is installing your application, you need to,

1. Issue menu command Edit>Variables in Freeman Constructor. Tell it you have an extra variable called cddrive. After that (in fact, even before that) you can use this variable as if it were a system variable, e.g., you can specify the source directory of your files as,

   ```
   $cddrive, or $cddrive\video, or $cddrive\wav
   ```

2. When you generate the file fiuser.cpp later, the class fiuserinfo will have an extra data member called cddrive,

   ```
   class fiuserinfo
   {
       public:
       fiuserinfo(installcontroller &ic):installinfo(ic)
       {

       }
      ~fiuserinfo()
       {

       }
       char cddrive[128];        // we have an extra data member here
       char sharedbuf[256];
       ......
   };
   ```

   This data member corresponds to your variable. You MUST initialize/set this variable properly before it is referenced.

   For simplicity, let's suppose that the drive must be drive e:. We can set it in the member function run() of the class fiusercontroller,

   ```
   int fiusercontroller::run()
   {
       // CUSTOMIZATION BEGINS: setting $cddrive
       fiuserinfo *p;               // point to our info obj
       p = (fiuserinfo*)ii;         // ii is installinfo*, we need fiuserinfo*
       lstrcpy(p->cddrive, "e:");   // initialize $cddrive for future use
       // CUSTOMIZATION ENDS

       // show welcome message
       if (!welcome())
       {
           return 0;
       }
       ......
   }
   ```

   A more realistic approach is that we should prompt the user for the CD drive. The class installer has a member function askdrive() to prompt for a drive. We still put our code in run(),

```
int fiusercontroller::run()
{
    // show welcome message
    if (!welcome())
    {
        return 0;
    }
    // select components
    if (!selectcomps())
    {
        return 0;
    }
    // ask the user to input install target dir ($i)
    if (!askdir())
    {
        return 0;
    }
    // CUSTOMIZATION BEGINS: setting $cddrive
    int driveno;                 // 0 for a:, 1 for b:, 2 for c:
    if (!ir.askdrive(&driveno, "Please select your CD drive"))
    {                            // prompt for the drive
        return 0;                // user cancel
    }
    fiuserinfo *p;                // point to our info obj
    p = (fiuserinfo*)ii;         // ii is installinfo*, we need fiuserinfo*
    p->cddrive[0] = 'a'+driveno; // 0 ==> "a:", 1 ==> "b:", 2 ==> "c:"
    p->cddrive[1] = ':';
    p->cddrive[2] = '\0';
    // CUSTOMIZATION ENDS
    ......
}
```

Here we prompt for the drive after prompting for the install target directory. Of course, you could insert this piece of code somewhere else. But you must put it before you start to copy the files.

Now, you can proceed as normal, i.e., build the project. Also, please keep in mind that if you use Freeman Constructor to generate the file fiuser.cpp again, your customized version will be overwritten.

Setting $i is very similar to defining your own variables. The major difference is that the data member corresponding to $i is a (inherited) data member of the class fiusercontroller, rather than that of the class fiuserinfo. We can access this data member directly in run(),

```
    int fiusercontroller::run()
    {
        ......
        lstrcpy(diri, "c:\\cshwin");  // set $i to c:\cshwin
        ......
    }
```

Determine install or not according to criteria other than version info and user selection

There are 5 data members of the class installcontroller related to this purpose,

```
class installcontroller
{
    ......
    saint ics;                          /* is the component selected? */
    saint ifs;                               /* is the file selected? */
    saint iis;                               /* is the item selected? */
    saint ies;                              /* is the entry selected? */
    saint irs;                            /* is the reg key selected? */
    ......
};
```

The class saint can be conceived as int [??] and can be used in the same way.

For example,

```
ics[3] == 1 means that the user has selected the fourth leaf component
ifs[0] == 0 means that the first file is not going to be copied
iis[1] == 1 means that the second program item is going to be created
ies[4] == 0 means that the fifth INI entry is not going to be set
irs[2] == 0 means that the third registration key is not going to be set
```

These data members will be set to true's (i.e., 1's) at the beginning of the program and they will be set again in selectcomps() according to the user selection.

These data members will only be referenced when they are actually needed, e.g., ifs will only be referenced while copying files, i.e., in copyfiles().

Therefore the best time to set ifs according to our own criteria is after selectcomps() returns and before copyfiles() is called.

Suppose that,

1. Your application includes a default phone book usually you will copy onto the user's system.
2. If an existing phone book can be found in the install target directory when the application is being installed, you want to skip (not copy) the default phone book.
3. The phone book is the 10th file you specified in Freeman Constructor.
4. The file name of the phone book is phbook.dat.

To do this, you need to add a piece of code the member function run() of the class fiusercontroller,

```
int fiusercontroller::run()
{
    // show welcome message
    if (!welcome())
    {
        return 0;
    }
    // select components
    if (!selectcomps())
```

```
        {
            return 0;
        }
        // ask the user to input install target dir ($i)
        if (!askdir())
        {
            return 0;
        }
        // CUSTOMIZATION BEGINS: set ifs[9] to false if phbook.dat already exists
        char phbookpath[128];              // hold the path to the phone book
                                           // path = $i + phbook.dat
        ir.mergedf(phbookpath, getdiri(), "phbook.dat");
        if (access(phbookpath, 00) != -1) // it exists
        {
            ifs[9] = 0;                    // don't copy the 10th file (phbook.dat)
        }
        // CUSTOMIZATION ENDS
        ......
    }
```

Also Note that we can use,

1. mergedf() to merge a directory and a file to get the full path.
2. getdiri(), getdirw(), getdirs(), getdiry() to get the install target directory, Windows directory, install source directory, and Windows system directory respectively.

Now, you can proceed as normal, i.e., build the project. Also, please keep in mind that if you use Freeman Constructor to generate the file fiuser.cpp again, your customized version will be overwritten.

Using bitmaps in advertisement dialog boxes

1. Add the bitmap you want to use to the resource file. You must give it an ascii id. Integer id should not be used here.
2. Add the dialog box you want to use to the resource file.
3. Put an owner-draw button in the dialog box. The button should be sized and positioned as if the bitmap were occupying its position.
4. Set the title of the owner-draw button to the ascii id of the bitmap.

When the dialog box is displayed, Freeman Installer will stretch the bitmap and then display it in the place of the owner-draw button.

Modifying the resources of the installer

What resources am I entitled to modify?

You are entitled to modify all resources in the file install.exe, finstall.res, finstall.rc, all BMP files and all ICO files. This includes the icon called iconfinstall which is the logo of Freeman Installer.

However, you are NOT entitled to modify the resources in other places such as in the uninstaller (file llatsni.exe) and Freeman Constructor (file fctor.exe).

Can I add any resource I like?

No. In this version of Freeman Installer, dialog boxes, bitmaps, icons, version statements are the only kinds of resources allowed. You can use them as you would in other Windows programs. Freeman Installer will take care of them automatically.

See Also

Using bitmaps in advertisement dialog boxes

Bells and whistles

Here are some common bells and whistles you might want,

1. Define extra variables. Some examples are,

   a. You will put the install program on a floppy disk but your own application on a CD. So you need to define an extra variable, say, $cddrive and use it in the source directory for the files.

   b. You need two or more install target directories.

   c. Your install target directory is the directory where AutoCAD resides. Therefore you need to search for AutoCAD and set $i to its home directory.

2. Perform extra operations. Some examples are,

   a. Check for a dongle when installing.

   b. Execute external programs during or after the installation.

   c. Your install target directory is the directory where AutoCAD resides. Therefore you need to search for AutoCAD and set $i to its home directory.

   d. Ask your user for his/her name, company, and etc.

   e. Ask your user for a password.

3. Disable built-in operations. Some examples are,

   a. Your install target directory is the directory where AutoCAD resides. Therefore you don't want to ask the user for the install target directory.

4. Determine install info (e.g., which files to copy) dynamically. Some examples are,

   b. Your install target directory is the directory where AutoCAD resides. Therefore you don't want to ask the user for the install target directory.

   c. You need to ask the user to re-enter Windows upon installation completion even though system.ini is not modified at all.

5. Determine to install or not according to criteria other than version info and user selection.
6. Modify the resources of the installer.

Features

New features introduced in v1.5,

1. The installer will create a log file (in INI format) to record down all changes made to the user's system during the installation.
2. The uninstaller will read the log file and display those changes in a structured and easy-to-read format. Upon request, the uninstaller will try to undo those changes. More exactly, the following changes will be undone,

   a. changes (additions, deletions, modifications) of any INI section and entry in any INI files.

   b. changes (additions, deletions, modifications) of any registration key.

   c. changes (additions, deletions, modifications) of any program groups and items.

   d. creations or deletions of any directory.

   e. creations, renaming, moving of any file.

   f. modifications of any file attribute.

   An apparent exception is that the deletion of a file can not be undone, i.e., if a file is deleted (including overwritten), it can not be saved anyway.
3. An easy to use front end to let you specify the install info, e.g., what files to copy, in a GUI environment. According to the info you told it, it can generated an INF file which can be interpreted by a ready-to-run EXE file coming with Freeman Installer. More importantly, it can compile the info you told it into a C++ source file which is ready to compile and link with the class library (provided in the registered version) to get an integrated and tailored EXE file, without the need to add a single line of code and disregarding you know C++ or not. The EXE file will only contain the necessary code you really need, e.g., if you don't set INI entries, it won't contain INI entry setting code.
4. The control flow of the whole program is determined by a function called run() in the file fiuser.cpp. This function, despite its important role, consists of less than 20 lines of statements and is extremely easy to modify to achieve most customizations.
5. Supports selective installation. You can group files, program items, INI entries, registration keys into leaf components, and group leaf or branch components into other branch components. The whole structure will be displayed in a DAG (directed acyclic graph) with a check box in each node to let the user to select. Files, program items, INI entries, registration keys, components can be shared by more than one components (This is why a DAG rather than a tree is used).
6. Displays user defined dialog boxes while copying files. Bitmaps can be used as a control in those dialog boxes simply by specifying the name of the bitmap resources.
7. Loads Program Manager to create program items if the shell doesn't support shell DDE commands. Optionally (depends on my user, you are a potential one) kills Program Manager when done.
8. The definition of $y has been changed to mean Windows system directory when the installer is running under a private copy of Windows and to mean Windows directory when it is running under a network shared copy of Windows. This makes installing shared files (mostly DLLs) onto $y abide by the rules set out by Microsoft (don't install files onto Windows system directory if on a network).

Features inherited from previous versions,

1. Supports version checking. Won't ruin your users' ctrl3d.dll or bwcc.dll.
2. Supports variables. Uses this simple mechanism to access install target directory, install source directory, Windows directory and Windows system directory.

3. Works with files compressed by Microsoft compress.exe.
4. Optionally slows down the install process to let small applications appear bigger.
5. A class library is included (registered version only) to support virtually unlimited customizations.

Project files & class library coming with Freeman Installer

There are four versions of project file coming with Freeman Installer for use with BC 3.1, BC 4.0, VC 1.0, and VC1.5 respectively,

```
b31fins.prj --- Project file for Borland C++ 3.1
b40fins.prj --- Project file for Borland C++ 4.0
v10fins.mak --- Make file for Visual C++ 1.0
v15fins.mak --- Make file for Visual C++ 1.5
```

There are four versions of the class library for use with the four supported compilers,

```
b31flib.lib --- Class library for Borland C++ 3.1
b40flib.lib --- Class library for Borland C++ 4.0
v10flib.lib --- Class library for Visual C++ 1.0
v15flib.lib --- Class library for Visual C++ 1.5
```

All project files and libraries are in large memory model. No other models are supported.

The two major components of a project is the compiler-independent file fiuser.cpp and the compiler-dependent class library. After generating fiuser.cpp in Freeman Constructor and optionally modifying it to add bells and whistles, you can load the appropriate project file for the compiler you are using and build the project. When you are sure the resulting EXE really does what you want, you should remove the debug info (if any) in the EXE.

You can remove the debug info in the EXE by,

```
        tdstrip b31fins.exe       --- for Borland C++ 3.1
        tdstrip b40fins.exe       --- for Borland C++ 4.0 (presumably)
        cvpack -strip v10fins.exe --- for Visual C++ 1.0
        cvpack -strip v15fins.exe --- for Visual C++ 1.5 (presumably)
```

Registering Freeman Installer

When to register?

The copy you are keeping is an evaluation copy only. So, if, fortunately, you like it and you would like to get the full product, you should get registered (Getting registered does NOT necessarily mean paying).

How much is the registration fee? What can I do with the registered version?

There are three kinds of licenses available. They are application license, author license and organization license.

You can choose that kind of license which suits your needs most at your own discretion.

All prices (your or mine) mentioned below don't include shipping & handling.

Application license

The registration fee for an application license is the same as the price of your application at which I can buy/get when I receive your registration form. That is, if you sell it at $20 a copy when I receive your registration form, an application license costs you $20. After getting an application license, you are entitled to distribute Freeman Installer & Uninstaller, including any derived work, with that application to install that application only. Applications with the same name but different version numbers are considered the same application. That is, Turbo C 1.0 and Turbo C 1.5 are considered the same application but Turbo C 1.5 and Borland C 1.0 are considered different applications. How the price of your application varies after you getting the application license does NOT interest me. A particular case is freeware or public domain software, because their prices are $0, the application license fees for them are also $0.

Author license

The registration fee for an author license is US$60. After getting an author license, you are entitled to distribute Freeman Installer & Uninstaller, including any derived work, with all applications you participate in developing and/or you at least own a part of the copyright of which, to install those applications only.

Organization license

The registration fee for an organization license will be determined by negotiation. After getting an organization license, that organization is entitled to distribute Freeman Installer & Uninstaller, including any derived work, with all applications that organization participates in developing and/or it at least owns a part of the copyright of which, to install those applications only.

What can I get by registration?

1. License to let you distribute Freeman Installer with your application(s).
2. The class library.
3. Life time technical support (via email or mail only).
4. Free upgrade to next version.
5. Discounted upgrade to future versions.

How to register?

1. Fill out the order form (double click the order form icon in the Freeman Installer group) and mail/e-mail/fax it to me.
2. Send the registration fee (if any) to me. Payment can be made via CompuServe's SWREG (available only when you are applying for an application license), cheques, money orders, bank drafts, purchase orders, TT (telegraphic transfers), or even cash.
   a. If you pay via SWREG, simply "GO SWREG". My registration ID is 3081.

b. If you send it in cash, you do it at your own risk.
c. If you TT it, please send to this account,

Account name:	ka iok tong
Account number:	012-045167-389
Bank address:	Hongkong Bank of Australia China Town Branch 728 George Street Sydney NSW 2000 Australia

All payments must be made in US$, Australian$, or HK$. If it is in Australian$ or HK$, you can use your local US$--Australian$ or US$--HK$ exchange rate to do the conversion.

What can I do if I regret getting registered?

You can get a 100% refund (excluding shipping & handling) if you destroy the registered version you received before distributing it.

How to contact me?

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

ka iok tong (kent)
Mail Box 20
Wentworth Building
University of Sydney
Sydney NSW 2006
Australia
Internet email: tongk@archsci.arch.su.edu.au
CompuServe email: 100351,3364
ph & fax: (61) (2) 2116314