WindowKey.doc

COLLABORATORS			
	TITLE : WindowKey.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 20, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

WindowKey.doc

1.1 WindowKey documentation - English

Welcome to WindowKey 1.04 copyright 1994 Frédéric Delacroix.

This is the document that must be spread with all other files of the package. It is to be viewed by AmigaGuide (copyright Commodore), but can also be read by human eyes, with less comfort.

TABLE OF CONTENTS

1.2 1.1 Distribution and disclaimer

WindowKey is Copyright 1994 by Frédéric Delacroix. Permission to copy & distribute it is granted to anyone who follows this conditions (generally known as SHAREWARE):
  - All files (or the whole archive) are distributed together (this

includes the executables, documentation files, catalog files, the catalog transcription file and all the icons). There is one exception though: If you're planning a release for a "mono-linguistic" community (say, for example, France only), you are allowed to distribute only the files that are relative to your language, that is to say: the doc file and the catalog file. All other files must be present.
- All distributed files remain UNMODIFIED IN ANY WAY (no silly "spread by..." message). If you have comments to add, do so in a separate file! Archiving is although allowed, but executable crunching is not recommended as the program is self-detaching (and thus needs to cut its SegList).
- You may NOT ask for money for this program. A small fee for copy & sending is allowed, but you may NOT charge more than Fred Fish does for a single AmigaLib disk.

- This is for all users of WindowKey: as the program is released as SHAREWARE, you must send me a little contribution of $10 (or equivalent) if you continue to use it after a short evaluation period. If you want the source of the program (written for Devpac 3), just add $10 more and I will send it to you. My address can be found at the end of the document.

WindowKey was tested a lot, but I cannot guarantee that it will always work as expected. I will not be held responsible for any direct or indirect damage/ loss of data that might result by the use of this program. Remember: you use it AT YOUR OWN RISK.

Special note: the catalogs and doc files are only available in english and french for now (my school spanish is soooo bad :-). So if you can make a translation of these files into your own language, I would appreciate you send it to me so I can include it in a future release. For the doc file, it's easy: just edit a copy of this one (please, don't change the node names, they are not displayed anyway). For the catalogs, just fill in the .ct files with your own strings and send them to me. I will send you corresponding catalogs in return. Please note that some strings (like the Commodities description message and gadget labels) are limited in length. For some, there is a shortcut key (in upper case please) or menu equivalent right before the label.
I will not distribute the cd files along with the program. If you want a translation, send me the ct file with you registration fee...

The icons I used for the drawers and some files are from IconPack, which is a collection of nice icons drawn and copyrighted by Tom Ekström.

1.3 1.2 Credits and special thanks

WindowKey was written on the wonderful HiSoft Devpac 3, running on an old A500 with OS 2.1. It uses the great reqtools.library which is Copyright Nico François and errormsg.library, which is copyright by myself. ARexx is copyright ©1987 by William S. Hawes.

Thanks to: Nico François for the reqtools.library
Khalid Aldoseri for his kd_freq.library
William S. Hawes for ARexx
Hisoft for Devpac 3
Commodore for the amiga and its OS

CEI for saving the amiga (?)
J.P.Deloffre and C.Peugnet at the ProMedia association
Yves Quiquempois for Enforcer testing and improvement ideas
AmigaNews.

1.4 1.3 System requirements

WindowKey should work on any amiga that matches these requirements:

- You need Kickstart 2.04 or newer (V37+), or WindowKey will refuse to run. The only thing to do if you are still under 1.3 is UPGRADE ! Believe me, it's worth it.

- You need the reqtools.library V38+ (release 2.1) installed in your LIBS: directory. It is not provided in this archive but you can find it nearly everywhere in public domain collections.

- You need the errormsg.library V4.0+ installed in your LIBS: directory. Version 4.01 is provided somewhere in this archive.

- You need the iffparse.library V37+ installed in your LIBS: directory. This came along with Workbench 2.0+. V39 is even better as bug free.

- WindowKey uses the locale.library V38+ if it is available to support multiple languages. This library is normally shipped with OS 2.1+.

- The preferences program makes use of the amigaguide.library to display this file whenever the Help key is pressed, but it is not mandatory.

- ARexx is needed for the ARexx facilities of WindowKey (!). You can find ARexx along with the original distribution of Workbench 2.0 and higher.

1.5 2.1 Introduction

WindowKey is yet another input helper program. The job of those programs is well known: they activate, bring to front, push to back, zoom, etc... any window or screen (in the Intuition sense of the word) on user request.

Lots of programs exist that do that, including Commodores'FKey provided on the workbench disk, but none of them had a user interface as powerful and were as flexible as WindowKey. The interface for WindowKey was directly reproduced from that of my other program Injector. The reason why I chose to make two different programs instead of including the features of WindowKey in Injector is that you can use either program without the other, which sounds quite normal since their jobs are completely different (this is one reproach to other input helper programs which tend to do everything and end up doing nothing).

Another reason to the existence of WindowKey is the possibility to control

Intuition only from the keyboard, instead of having to hunt for a mouse hidden under a pile of paper when wanting to change the active window.

To make this work, WindowKey has its own language, made of keywords, with or without arguments enclosed in parentheses. You will have to read the references sections for further informations.

1.6 2.2 Starting WindowKey

WindowKey can be started either from CLI of from Workbench.

If used from the CLI, WindowKey will automatically detach itself, enabling the CLI window to close or the startup-sequence to continue, no need for the Run command. There are no CLI arguments recognized. The config file provides for that. Some more options are descibed in the tooltypes section.

If you use WindowKey from the Workbench, the best place for it to be is the WBStartup drawer. This way, WindowKey will be started each time the system boots up (don't forget to register!). You can of course double-click its icon too.

Even when started from Workbench, WindowKey cuts its SegList so that Workbench can close while WindowKey is active. Plus, there is no need for the DONOTWAIT tooltype.

1.7 2.3 ToolTypes

This section describes the tooltypes that are recognized by WindowKey. Tooltypes are the ONLY way to pass these arguments, even from CLI. They are: PUBSCREEN,CX_PRIORITY,PREFSPATH, AREXXCONSOLE, BREAKKEY and TOOLPRI.

1.8 The PUBSCREEN tooltype

This tooltype tells WindowKey which public screen should be used for the requesters it could produce. The default is '*', which means that the frontmost screen will be used, provided it is public (just as CON: windows). If the screen is not available, the default public screen is used.

1.9 The CX_PRIORITY tooltype

This is used to specify a number that will be used as the priority of WindowKey's Broker in commodities' list. This is useful if you want WindowKey's hotkeys to override another commodity's hotkeys (or the contrary). The higher the priority is, the sooner WindowKey will receive the InputEvents (relatively to other brokers). The default is 0.

1.10 The PREFSPATH tooltype

This option will tell WindowKey where to find the preferences program whenever it is invoked by the EXEC_PREFS command. The default is 'WindowKeyPrefs', which must of course be in your valid path. I do use PREFSPATH=SYS:Prefs/WindowKey .

1.11 The AREXXCONSOLE tooltype

This tooltype describes the console window WindowKey will open when executing an ARexx program. This MUST be an interactive console, for example a CON: window (maybe an AUX: stream ? I haven't tried it).

The default is CON:////WindowKey and ARexx/SCREEN*/AUTO/WAIT/CLOSE, which will provide you with a reasonable console window on the front screen if public, or the default public screen.

1.12 The BREAKKEY tooltype

This tooltype is new as of version 1.01. It is a commodities description string used to tell which Hotkey should stop any action by WindowKey.
This "break key" feature was introduced mainly for consistency reasons with Injector, as the actions performed by WindowKey are quite short and rarely need to be stopped by the user. However, the feature exists.

Contrary to "normal" hotkeys, the event is NOT removed from the chain. This way you can quit WindowKey at the same time, or stop another program like Injector at the same time. This could have disastrous effects if the application owning the active window recognizes this message.

The default value is lcommand rcommand lshift rshift q

1.13 The TOOLPRI tooltype

This tooltype exists as of version 1.01. It is a standard tooltype, as understood by Workbench, to set the priority of WindowKey's task. However, as WindowKey launches its own process, the priority was always -1 in earlier versions. Now you can set it with this tooltype, which is also recognized for CLI.

My advice is a priority of at least 1 (never more than 5).

1.14 2.4 Removing WindowKey

WindowKey can be removed by several ways. You can of course use the Exchange program and select Kill WindowKey. You can also run WindowKey (the master program, not the preferences program) again. The last way of removing WindowKey is to make it execute the CMD_QUIT command.

In all cases, WindowKey cannot be removed while an ARexx program is still running, as WindowKey must wait for the reply message from ARexx (don't want to wake up the guru, do you ? :-). Future versions might implement a "delayed quit" feature (is it really useful?).

1.15 3.1 Starting the preferences program

To modify its configuration, WindowKey makes use of a separate program, called the preferences program. This way, memory is not used to store unused data while WindowKey is active: the preferences program is loaded only when necessary.

The preferences program can be started either from CLI, Workbench, or WindowKey. It ALWAYS recognizes these tooltypes (even from CLI):

PUBSCREEN=<public screen name>. This defines the screen that should be used to open the window. The default is *, meaning that the front screen should be used if it is public.

CREATEICONS=<YES or NO>. This tells the preferences program of the original state of the "Create icons ?" menu item.

ACTION=<USE,SAVE or EDIT>. This is valid only for project icons that have their default tool set to the prefs program, or given via multi-selection. Such project icons are configuration files that are to be loaded immediately by the prefs program. This ACTION tooltype tells the prefs program what to do with this congfiguration file.

USE will save the given file as the "current" configuration, that is to say into the file "ENV:WindowKey.Prefs".

SAVE will save the given file as the "permanent" configuration, that is to say into both "ENV:WindowKey.Prefs" and "ENVARC:WindowKey.Prefs".

EDIT is the default, it won't save anything (yet), but rather let you edit the file as if you had started the prefs program then selected the Open... menu item.

Moreover, there are some options that can be used on the command line, that OVERRIDE the above-mentionned tooltypes. The template is:
FILE,PUBSCREEN/K,USE/S,SAVE/S,EDIT/S,NCI=NOCREATEICONS/S,CI=CREATEICONS/S

FILE is of course the the name of the file to load, PUBSCREEN has the same meaning as the tooltype of the same name, USE,SAVE and EDIT are switches that are equivalent to the corresponding values of the ACTION tooltype, and NOCREATEICONS is equivalent to the tooltype CREATEICONS=NO. CREATEICONS is equivalent to the tooltype CREATEICONS=YES. The last two keywords may be used to override defaults, I guess they won't be used very much.

1.16 3.2 The Graphics User Interface

When started in EDIT mode, the preferences program opens a window with some gadgets and menues. If you're used to normal preferences programs, you should not have problems using it. Yet here are some explanations.

The main display is a list. It contains all currently defined hotkeys. You can select a name in that list by clicking on it, or using the shortcut key (h for the english version) with or without SHIFT to scroll through the list without using the mouse. You can also use the up and down arrows to move by one entry in the corresponding direction, by one page when shifted, or go to the top or bottom of the list with control.

On the right, there are 5 gadgets that are used to manage the appearance of the list. They do not change the way WindowKey will treat it (except maybe that if two hotkeys conflict, the first one will be used). Top will send the selected entry to the top of the list, Bottom will send it to the bottom, Up will move it one entry upwards, Down one entry downwards, and Sort will sort the list in alphabetical order.

Below are some control gadgets. There are three string gadgets that are used to edit hotkeys. For WindowKey, a hotkey is a combination of three things: a name, a key descriptor, and a command line. The 3 string gadgets let you edit those fields.

There is a checkbox on the right of the key descriptor gadget. If it is checked, then the hotkey is enabled. If it is not checked, then the hotkey will not be available. However, the hotkey is still available via the EXEC_NAMED command.

The Name field is used for displaying the list and by the EXEC_NAMED command.

The Key string gadget lets you edit the field that will tell commodities.library which key combination should trigger the action of WindowKey. This must be a valid commodities descriptor string. See Defining hotkeys for further information. After typing in that string gadget, the preferences program asks commodities if the string is OK, and will alert you if it is not. If the description is correct, then the checkbox is automatically checked, and unchecked if it is not.

The Command string gadget is there to hold the command stream WindowKey should execute when receiving the event corresponding to that hotkey. WindowKey uses its own "language" for defining such actions. See the Language reference section for details.

On the right of the string gadgets are three boolean gadgets named Create, Copy and Delete. They are used respectively for creating a new hotkey (that appears at the bottom of the list), Copying an existing entry (appearing right after the currently selected one) and Deleting an existing Hotkey.

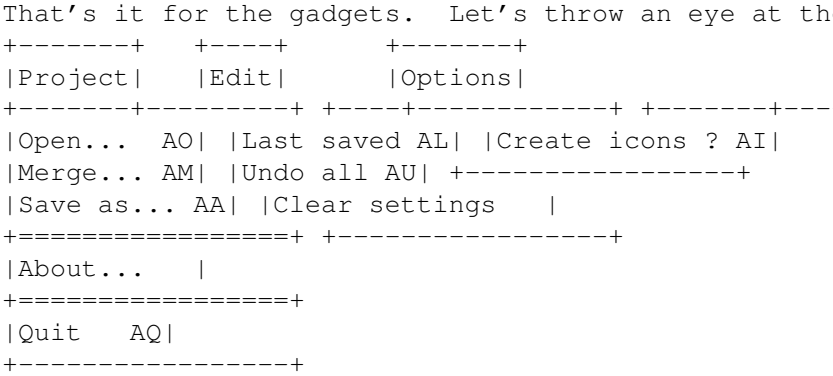
Farther below are 5 gadgets. They control the behaviour of the program.
Save will store the edited file as the permanent configuration. Use will save it as the current configuration. Both of them will make the preferences program quit (unless an error occurs while saving).

Test will change the current configuration but won't end the prefs program, so that you can test if the newly created configuration really suit your needs, and change it if it doesn't.

Help will display this file. You need amigaguide.library V34+ for that.

Cancel will revert all you have done. If you had changed the current configuration (via test), then it will be put back as it was before the program was called. Then the prefs program ends.

That's it for the gadgets. Let's throw an eye at the menues:



Open will load a new configuration file. The currently edited one will be lost. Merge will load a new file and merge it with the configuration you are currently editing. Save as will write the currently edited file to a new file. For these three actions, you will get an ASL file requester.

About will inform you about the version of WindowKey and my address. Quit will end the program (Warning: no "Cancel" nor "Undo" is performed, a tested configuration will remain active).

Last saved will get the last saved version of the permanent configuration file (the configuration will be loaded from ENVARC:WindowKey.Prefs). Undo All will cancel all the changes you have done. Clear settings will of course clear the file you are currently editing. Use with care !

At last, Create icons is an on/off menuitem that tells the preferences program whether it should create an icon for a file that is saved to disk by the "Save as..." function. The icon will be a project, whose image will be gotten from ENV:sys/def_prefs.info (or the default project icon if inexisting), the default tool is set to the prefs program, and ACTION=USE will be set as a tooltype. This way, double-clicking on that icon will change the current configuration of WindowKey without really entering the prefs program (does not mean it is not loaded though).

As of version 1.01, starting the prefs program again will activate, bring to front and unzip it if necessary. Sending it a CTRL-F has the same effect. Sending a CTRL-C will make it quit.

As of version 1.04, the prefs program is font-sensitive. The checkbox might look a bit strange under OS2.0 with a font other than topaz/8. This is because gadtools does not scale the imagery to the gadget size automatically. This was fixed for OS3.0.

1.17 3.3 Language reference

This section will teach you how to write commands for WindowKey. These must be entered in the Command string gadget of the preferences program's window when a hotkey is selected, or sent via ARexx to the port named WindowKey.

A command is made of a keyword and an optionnal argument. Commands that take arguments must have their keyword followed immediately by the argument enclosed in parentheses. Thus it may be necessary for ARexx programs to surround the parentheses by quotes, as the former are treated in a special way by ARexx. But the parentheses must remain. Check the example programs provided. Several commands are separated by spaces.

Before I list all commands available, I must tell you of the way WindowKey names windows and screens, as such names will be taken as argument by most of these commands. Everything is described in detail in the sections titled Window names and Screen names.

Now here is the list of the commands WindowKey understands (those that take an argument have () after them):

```
WIN_ACTIVATE()    EXEC_NAMED()
WIN_TOFRONT()   EXEC_REXX()
WIN_TOBACK()    EXEC_PREFS
WIN_ZIP()
WIN_SETREMEMBER() CMD_QUIT
WIN_MAKEVISIBLE() CMD_UPDATE
WIN_SIZE()    CMD_NEWPREFS()
WIN_MOVE()
WIN_CLOSE
WIN_MOVEABS
WIN_SIZEABS

SCR_TOFRONT()
SCR_TOBACK()
SCR_SETREMEMBER()
SCR_MOVE()
SCR_STACK()
```

Word of warning: some of these functions are potentially dangerous. All input helper programs don't tell about it, but it is impossible to be absolutely safe. Indeed, this program uses other programs' windows and screens, which can be closed at any time. Of course, WindowKey examines all Intuition's structures in LockIBase() mode, but cannot, for example, bring a window to the front in this mode. That's why - although the transition is done in Forbid() mode - there could still be problems if you invoke WindowKey's hotkeys while in deep window or screen activity. Such problems are very unlikely, but the risk exists (even Commodore's FKey program does like that). You have been warned.

1.18 Conventions for naming windows

So as to find windows chosen by the user in WIN_ commands, WindowKey uses names, understood in different ways. This window name is

either the current title of the window, ex:

Shell 8 in DH0{42MB}:Progs/WindowKey

or one of the following special names (case sensitive!):

  *A Active window
  *N Next (to the active) window
  *P Previous (to the active) window
  *F Frontmost window (on frontmost screen)
  *B Back window (on frontmost screen)
  *M window under Mouse (all screens are searched)
  *R Remembered window

The following escaped characters are available (they are the same as the ones available in Injector), although they're scarcely used in window titles:

  \a Bell (ASCII 7)
  \b Backspace (ASCII 8)
  \c Control sequence introducer (CSI, ASCII 155)
  \e Escape (ASCII 27)
  \f Form feed (ASCII 12)
  \n New Line (ASCII 10)
  \r Return (ASCII 13)
  \t Horizontal tabulator (ASCII 9)
  \v Vertical tabulator (ASCII 11)
  \xNN Character with NN as hexadecimal ASCII code
  \NNN Character with NNN as octal ASCII code
  \ Backslash
  \) Close parenthesis (not end of arguments)

  The remember feature works as follows: each time you act on a window via one of the WIN_ commands, WindowKey stores the adress of that window so that it can be used again with the *R name. Of course, checks are made to see if that window is still open before using it again. I added this feature to solve this little problem: I wanted to make a hotkey that brought the back window to the front and activated it. Unfortunately,

  WIN_TOFRONT(*B) WIN_ACTIVATE(*F)

  didn't work because intuition defers WindowToFront() and ActivateWindow() to its input handler, which means that the window is not yet the frontmost when WIN_ACTIVATE() is executed. Instead, this works perfectly:

  WIN_TOFRONT(*B) WIN_ACTIVATE(*R)

The remember value can also be set by the WIN_SETREMEMBER command.

1.19 Conventions for naming screens

Naming screens differs slightly from naming windows, since there is already a family of named screens: public screens. WindowKey allows you to specify a public screen by its name (ex: DEVPAC.1 or Workbench, ...) or

its title (which works also for non public screens). If you choose to reference a screen by its title, you must precede that title by a @ sign.

You can also specify a screen by a window name: the first screen containing a window with the specified title will be returned. To use this feature, just use the window title preceded by a !.

Last note: the title of the screen refers to the default title of the screen, not the current title of the screen. The active window can set another title for the screen, but it is not that one that is checked (although it's the one displayed). ex, for the Workbench screen:

"Workbench Screen" is the default title
"Amiga Workbench 334 688 RAM graphique 892 184 autre RAM"
is the current screen title when the workbench window is active.

SCR_TOFRONT(@Workbench Screen) will work, whereas
SCR_TOFRONT(Amiga Work...) will not.

These special (case sensitive) screen names are also recognized:

*A Active screen
*N Next (to the active) screen
*P Previous (to the active) screen
*F Frontmost screen
*B Back screen
*M screen under Mouse pointer
*R Remembered screen.

You can of course also use all escaped (\a,\b,...) characters listed in the Window naming section.

The remember feature is similar to the one working for windows, except that it works for screens (!). Of course, the remembered screen pointer is always checked for validity before being used again.

1.20 The WIN_ACTIVATE() command

This command activates the selected window. Needless to say that WIN_ACTIVATE(*A) is useless...

1.21 The WIN_TOFRONT() command

This command brings the selected window to the front. The screen the window is on is _NOT_ brought to the front.

Small peculiarity: BACKDROP windows cannot be brought to the front (neither can they be pushed to the back); they always stay at the back. That's why in the case of WIN_TOFRONT() and WIN_TOBACK(), BACKDROP windows are ignored, even if the argument is *B. This behaviour does not affect other WIN_ commands.

1.22 The WIN_TOBACK() command

This command pushes the selected window to the back. The screen the window is on is _NOT_ moved.

Small peculiarity: BACKDROP windows cannot be pushed to the back (neither can they be brought to the front); they always stay at the back. That's why in the case of WIN_TOFRONT() and WIN_TOBACK(), BACKDROP windows are ignored, even if the argument is *B. This behaviour does not affect other WIN_ commands.

1.23 The WIN_ZIP() command

This command "zooms" the window. This command only acts if the selected window has a zoom gadget. By "zoom", I mean switch the dimension and position of the window to alternate values, as they were defined by the owner of the window.

1.24 The WIN_SETREMEMBER() command

This command just sets the window remember feature to the specified window. All other WIN_ commands automatically update the window remember so this command is quite useless (but does not cost anything).

1.25 The SCR_TOFRONT() command

This command brings the selected screen to the front. No other operation is performed.

This hotkey is really useful for cycling screens:

```
SCR_TOFRONT(*B) WIN_ACTIVATE(*F)
```

1.26 The SCR_TOBACK() command

This command pushes the selected screen to the back.

1.27 The SCR_SETREMEMBER() command

This command just sets the screen remember feature, which acts exactly as the window remember, but for screens. All SCR_ commands update the screen remember buffer.

1.28 The EXEC_NAMED() command

This command will execute a hotkey as if the corresponding key had been pressed by the user. This way you can do GOSUB-like processings of hotkeys. The argument is of course the name of the hotkey to be processed. There is a security check in WindowKey: when a hotkey is executing, a special bit in its structure is set, preventing recursive calls to be made (you will just get a message).

1.29 The EXEC_REXX() command

This command tells WindowKey to launch the ARexx program whose name is in argument. The program will have its default host address set to "WindowKey". The default file extension for such programs is .wndk .

Thanks to this command (and ARexx!), you can generate complex macros, with tests in them, etc etc etc...

1.30 The EXEC_PREFS command

This command will ask WindowKey to run the preferences program. This is done via the SystemTagList() call of the dos.library, so the preferences program must be in your path. The default file name for it is "WindowKeyPrefs", but it can be changed by the PREFSPATH tooltype. This can also be acomplished by selecting Show in the commodities exchange program. As of version 1.01, selecting Hide will end the prefs program (by sending it a CTRL-C).

1.31 The CMD_QUIT command

The use of this command is obvious: on receving it, and provided doing so is possible, WindowKey will suicide. This is not possible for example if some ARexx programs are still running.

1.32 The CMD_UPDATE command

This command is not useful in most cases. On receiving it, WindowKey will reload its configuration. This is useless in most cases since, like the IPrefs program, WindowKey uses the notification facilities of AmigaDOS to automatically detect alterations of the preferences file (by the preferences program for example). Generally, ENV: is assigned to the Ram disk, so it is not a problem. But all handlers do not support notification, so, if someone uses an assign over a network filesystem for example, he should use this command for WindowKey to update the configuration.

When this command is received, WindowKey does not continue the processing of the current line, even if there were commands remaining.

1.33 The CMD_NEWPREFS command

This command will switch to another configuration file. The argument must be a valid path to the file in question. You can create such files by selecting the Save as... menuitem in the configuration program.

WindowKey makes the switch by invoking the preferences program with the argument and the USE keyword, so the prefs program must be accessible. You can use the PREFSPATH option for that.

1.34 The WIN_MAKEVISIBLE() command

This command works only on Kickstart 3.0 and newer for now. If you have a screen larger than the monitor can display (probably in autoscroll mode), this command will move the screen in a minimal way so as to make the window given in argument visible.

I will hopefully fix it for Kickstart 2.0 users in the future.

1.35 The SCR_MOVE() command

This command moves a screen. The argument must be formatted as follows:

```
SCR_MOVE(X,Y,Name)
```

Name is a screen name as described before, and X and Y are two signed decimal values, indicating how much (how many pixels) the screen will move on X and Y axis respectively. Of course, there will be restrictions if you try to move the screen too far.

This command works really fine when used in conjunction with the 'repeat' qualifier (see example config file).

1.36 The WIN_SIZE() command

This command changes the size of window. The argument must be formatted as follows:

```
WIN_SIZE(X,Y,Name)
```

X and Y are decimal (signed) values indicating the change respectively on the width and height of the window. Name is the name of the window to affect. This window will actually be resized if and only if it has a size gadget. The size will remain between the bounds defined by the owner of the window.

This command is not as fast as, for example, SCR_MOVE() because each time the window is resized, the owner may need to refresh it, and maybe newly uncovered windows too. That's why it's a bad idea to use it with the repeat qualifier.

1.37 The WIN_MOVE() command

This command changes the position of a window (this position is of course relative to the screen, and limited by it). The argument must be formatted as follows:

```
WIN_MOVE(X,Y,Name)
```

X and Y are the (signed) decimal values that tell WindowKey how much to move the window on each axis. Name is the name of the window. The window will be moved if and only if it has a drag bar.

This command is not very fast (well, depends on your machine's speed) for the same reasons as those exposed in WIN_SIZE().

1.38 The SCR_STACK() command

This command is one of the most useful in WindowKey. It will arrange all the windows opened on a screen. The argument taken by this function must be formatted like this:

```
SCR_STACK(mode,name)
```

Where mode is a numeric parameter and name a screen name as already defined. The following values are currently defined for mode. Other values are reserved for future expansion and are ignored for now.

1 : Stacks all windows, one under the title bar of another.
2 : Arranges windows diagonally, so that at least the depth gadget of all windows is visible.
3 : Arranges windows horizontally. All windows will get the same height and will be placed on top of each other.
4 : Arranges windows vertically. All windows will get the same width and will be placed side by side.
5 : Full size. All windows will occupy the entire screen.
6 : Grid. WindowKey will try to put windows in an array so that there are as many rows as columns (within the limits of possible). The entire screen will be filled, enlarging the windows of the last row if needed.

As you may notice, the first five arrangements are the same as those avaiable in the Devpac 3 editor. If you have ideas on which new arrangement methods I could add, write !

Of course, none of the windows will be made larger than its maximal size or smaller than its minimal size. The judging is left to Intuition so don't blame me if the result with some windows are a bit strange !

The moving and sizing process may take a while, especially on slow machines, if you have many windows. More than ever, you should have the warning (section Language reference) in mind, do not close windows that have not been moved yet, or you will get the guru. Forwarned is unflatlined.

Last but not least, only the windows that have a size gadget AND a drag bar will actually be moved (ex: Workbench windows, Shells, DME windows...). The others (ex: toolmananger docks) are left untouched.

Idea by Christophe Peugnet/ProMedia.

1.39 The WIN_CLOSE command

This command does not take arguments. Its job is to close the active window. This works on all windows, those that track the closewindow through the IDCMP protocol (most windows) as well as those that use the console.device (Shells).
However, WindowKey does not close anything itself, as the system would crash immediately. Instead, it broadcasts an IECLASS_CLOSEWINDOW event on the input.device's chain. That's why, by the way, you cannot select the window you want to close with the usual window names. If you really want to close a non active window, I suggest you use somthing like that:

WIN_ACTIVATE(Fine window name) WIN_CLOSE WIN_ACTIVATE(*R)

This will work because WIN_CLOSE does not affect the window remember buffer (did you want to keep track of a window that was just closed ?) and because WIN_CLOSE is deferred to the input queue like WIN_ACTIVATE (and thus arrive in the same order to Intuition).

1.40 The WIN_MOVEABS() command

This command is new for V1.03. Its job is similar to WIN_MOVE, but the x and y coordinates are absolute (in fact: relative to the top/left corner of the screen).

WIN_MOVEABS(x,y,name)

See WIN_MOVE for further info.

1.41 The WIN_SIZEABS() command

This command is new for V1.03. Its job is similar to WIN_SIZE, but the new size is absolute: it describes the window's final size and not the change.

WIN_SIZEABS(x,y,name)

See WIN_SIZE for further info.

1.42 4.1 Defining hotkeys

The text that follows does depend on the behaviour of the commodities.library rather than on WindowKey's. This is how commodities will understand the key combination you want an action associated with. If you have problems making yourself understood by commodities, or if you are lazy, Injector 2.12+ might help you...

A description string is made as follows:

```
     [<class>] {[-][<qualifiers>]} [-][upstroke] [<key code>]
```

All keywords are case-insensitive.

class: set to an InputEvent class. Supported classes are rawkey for keyboard events, rawmouse for mouse events,diskinserted and diskremoved. The default is rawkey, which should generally be used.

qualifiers: this is a set of keywords representing the state of keyboard qualifiers (Shift,Alt etc...). This is a list of known keywords. Those marked with an * are new for commodities.library V38.

```
lshift,left_shift *:        Left shift key.
rshift,right_shift *:       Right shift key.
shift:            Either shift key.
capslock,caps_lock *:       Caps Lock key.
caps:           Either Caps Lock or shift.
control,ctrl *:         Control key.
lalt,left_alt *:        Left Alt key.
ralt,right_alt *:       Right Alt key.
alt:            Either Alt key.
lcommand,lamiga *,left_amiga *,left_command *:  Left amiga key.
rcommand,ramiga *,right_amiga *,right_command *:Right amiga key.
numericpad,numpad *,num_pad *,numeric_pad *:  For numeric pad keys.
leftbutton,lbutton *,left_button *:   Left mouse button.(1)
midbutton,mbutton *,middlebutton *,middle_button *:
            Middle mouse button.(1)
rbutton:,rightbutton *,right_button *:    Right mouse button.(1)
repeat:           Key-repeat active.(2)
```

Notes: (1) commodities.library V37 has a bug which prevents the use of leftbutton,midbutton and rbutton as qualifiers. It was fixed in V38.
(2) for rawkey class only.
(3) if a hotkey is to be insensitive to the state of a qualifier, place a - before the qualifier name.

upstroke: Normally an event is generated only when the key is pressed. You can change this behavior by setting upstroke. This will generate an event only when the key is released. If both press and release are to generate an event, use -upstroke.

key code: These are meaningful only to rawkey and rawmouse classes.
For rawkey, here are the key codes, * are new for V38 of commodities.

```
a to z, 0 to 9:       Normal ASCII characters.
f1 to f10,f11 and f12:      Function keys.
up,cursor_up *:       Up arrow key.
down,cursor_down *:     Down arrow key.
```

```
left,cursor_left *:     Left arrow key.
right,cursor_right *:     right arrow key.
esc,escape *:       Esc key.
backspace       Backspace key.
del         Del key.
help          Help key.
tab         Tab key.
comma         Comma key (,).
return          Return key.
space,spacebar *      Space bar.
enter         Enter key.(4)
insert *        Keypad 0 key.(4)
delete *        Keypad 1 key.(4)
page_up *       Keypad 9 key.(4)
page_down *       Keypad 3 key.(4)
home *          Keypad 7 key.(4)
end *         Keypad 1 key.(4)

Notes:  (4) must be used with the numericpad qualifier.

For rawmouse,valid keycodes are: (only valid for V38 of Commodities):
mouse_leftpress:      Left button pressed.(5)
mouse_middlepress:      Middle button pressed.(5)
mouse_rightpress:     Right button pressed.(5)

Notes:  (5) you must also set the corresponding qualifier.
```

1.43 4.2 History

```
Revision V1.04
--------------
    The preference program is now font-sensitive. Proportionnal
    fonts are not supported, however.

Revision V1.03
--------------
    Added WIN_MOVEABS and WIN_SIZEABS.

Revision V1.02
--------------
    Adapted to version 3.02 of errormsg.library, causing execu-
    tables to shorted, messages to be removed. I will now dis-
    tribute the library together with the program.

Revision V1.01
--------------
    Reproduced the changes made for Injector 2.30: TOOLPRI and
    BREAKKEY tooltypes, support of the Hide command, break si-
    gnals, tab for activation of string gadget...

Revision V1.00
--------------
    --- Initial release ---
```

1.44 4.3 Contacting the author

I remind you that WindowKey is SHAREWARE. I know most of you will not pay me anything after their evaluation period, but please consider I've spent a lot of time writting and debugging this program, and I'd just like a small reward that would enable me to buy a good hardware configuration. Consider that paying $10 will provide you with support for future releases. The source is kept warm for those of you who might be interested in programming the wonderful amiga operating system, for $10 more.

For anything like registration, comments, bug reports, improvements requests, postcards, I can be reached at:

Frédéric DELACROIX
5 rue d'Artres
59269 QUERENAING
FRANCE.