Selector

Anders Persson

COLLABORATORS			
	TITLE : Selector		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Anders Persson	August 3, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Selector

1.1 Selector

```
Selector 5.0 Guide

  Introduction~~~~~~~~~~~~~~~   The
  Copyrights~~~~~~~~~~~~~~~~   beginning...
  Installation~~~~~~~~~~~~~~
  Beginners~course~~~~~~~~~~

    Concepts~~~~~~~~~~~~~~~~~~   Command
    Window~Commands~~~~~~~~~~~   Reference
    Startup~commands~~~~~~~~~~   part...
    Panel~defining~commands~~~~
    The~OnExit~command~~~~~~~~~
    The~OnChange~command~~~~~~~
    All~examples~with~source~~~
    Command~Index~~~~~~~~~~~~~
    Search~document~~~~~~~~~~~   (Only if you have SearchGuide installed)

      Getting~input~~~~~~~~~~~~~   Miscellaneous
      Keys~~~~~~~~~~~~~~~~~~~~~~
      SEL:~Searchpath~~~~~~~~~~~
      Making~Resident~~~~~~~~~~~~

        Env~variables~~~~~~~~~~~~~   A little more
        Arguments~&~Other~variables   advanced
        Selector~in~shell~scripts~~   programming
        More~tips~&~trix~~~~~~~~~~~
        Bonus Tools     ~~~~~~~~~~


          Version~History~~~~~~~~~~~~
          Selector Name Conflict~~~~~
          The~End~~~~~~~~~~~~~~~~~~~   Bye!
```

1.2 Concepts

Concepts:

The Selector window is defined by a source file. This file is loaded by
the shell command: Selector "Source file"

The source file is defined by commands in three different categories:

1. Startup~commands
These commands executes every time the file is loaded or the window is
updated by the update command.

2. Window~commands:
These commands specifies the windows apperance. They can only be specified
once in the same sourcefile.

3. Panel~defining~commands:
These command specifies the contetns of the panel, with buttons,
text strings etc.

More about commands:

Many of the commands have default values.
You don't have to use any startup or window commands if you don't
need them. A window is allways opened automatic anyway.

You can only have one command on each line. You might think that it
looks like there is many commands on the same line when you take a
look at the Exec or Button commands. For example the exec command:
It is in fact one command that can execute many external (AmigaDos)
commands. And these must be written on one line.

ITEMS:
The selector panel is built of items. These can be buttons,
textstrings or spaces.

STRING SYNTAX:
Every string (or number) can be stated in 3 different ways:

Case 1: The string can't contain spaces, "-signs or '-signs:
A_String

Case 2: It can contain spaces and '-signs:
"Isn't it good?"

Case 3: It can contain spaces and "-signs:
'That is "cool"'

You can't use a string that contains both "- and '-signs.

```
There is no real difference between <string> and <number>.
All are in fact strings.

------------------------------------------------------------------------

COMMENTS:
You can write comments in your source by setting a ; before it.
; A comment...
```

1.3 Window

```
Window commands:
----------------
These commands specifies the windows apperance. They can only be specified
once in the same sourcefile.

------------------------------------------------------------------------

TITLE <string> | OFF
Sets the window title to "string" or turns it off (the whole dragbar)
Default is the name of the file.

------------------------------------------------------------------------

CLOSE  ON | OFF
Specifies if there will be any close gadget on the window. If you
have turned of the title bar, then this command will have no effect.
Default is ON

------------------------------------------------------------------------

X <number> [percent|fromMax] | auto
Sets the x position of the window. Examples:

x 120             ;Sets the x position to 120.

x 50 percent      ;Sets the x position so that the window will be placed
                   in the middle of the screen in x-led.
x auto            ;Same as the last one.

x 100 percent     ;Sets the x position so that the window will be placed
                   as much to the right it can be.
x 0 frommax       ;Same as the last one.

x 3 frommax       ;Sets the x pos, so that the gap between the right
                   border of the screen and the right border of the
                   window is 3 pixels.

Default is auto

------------------------------------------------------------------------

Y <number> [percent|fromMax] | auto
Works in the same way as the X-command.
```

```
---------------------------------------------------------------------------

BACKGROUND <number>
Sets the background color to <number>. Instead of <number> you can use
"dark" or "light" for the colors set by "lightcolor" and "darkcolor".
Default is 0

---------------------------------------------------------------------------

LIGHTCOLOR <number>
Sets the color of the buttons light colored lines
(for the 3d effect) to <number>.
Default is 2

---------------------------------------------------------------------------

DARKCOLOR <number>
Sets the color of the buttons dark colored lines
(for the 3d effect) to <number>.
Default is 1

---------------------------------------------------------------------------

GAP <number>
Sets the gap (space) between two items. So if you set Gap to 0,
there will be no space between the items. If you set it to 2,
there will be a space of two pixels between every button (item).
Note that the lines set by startbox~and~endbox only will be seen if
you have a gap of 2 or more. Also see Gap~Demo.
Default is 2

---------------------------------------------------------------------------

SIZE <number> | auto
Sets the width of the buttons and text (in characters) to <number>.
If set to auto, then the size will be set according to the longest
button name (or text field) specified in the file.
Default is auto

---------------------------------------------------------------------------

ROWS <number> | auto
Specifies how many rows the items will be placed in. If you for example
set rows to 3, then the first 3 items will be placed under each other.
The 4:th item will be placed in right of item 1. The 5:th in right of
item 2 etc. Please try out the Rows~demo supplied and you will understand.

If you have set rows to a value so low that all items not will fit on
screen because the panel is going to be to whide. Then the number of
rows is increased so that everything will fit on the screen. This is
also the case if you specify a value to high.

If you set rows to auto, then it will be set so that the window will
be used optimaly and with as few columns as possible.
Default is auto

---------------------------------------------------------------------------
```

```
NOBORDER
 This specifies that the window will have no border. This can be usefull
if you for example want to make it look like there is buttons directly
on the workbench.

------------------------------------------------------------------------

SCREEN <PubScreenName> [OrDefault] | Default
or  SCREEN <width> <hegth> <planes> <mode>

'SCREEN PubScreenName' will open the window on the public screen with
the name 'PubScreenName'. For example: 'SCREEN Workbench' for the
workbench screen. Please note that this name is case sensitive.

'SCREEN default' uses the default public screen. This is often
the workbench screen.

'SCREEN Name OrDefault' uses the screen 'Name' if it is available,
else the default screen is used.

'SCREEN <width> <hegth> <planes> <mode>'
This will open a screen with the specifications above. Mode 1 for lores,
2 for medres, 3 for interlace, 4 for hires interlace and 5 for common.
The screen opened is NOT a public screen.

Note: The screen will be opened/locked only once. This means that you
can not change the screen parameters with the update command. This is the
same for both public screens and opened screens.

Default is: 'SCREEN default'

------------------------------------------------------------------------

PALETTE <colornumber> <% red> <% green> <% blue>
Sets the palette for colornumber. You MUST open a Screen before you
can change any palette.

Here follows a demo of the SCREEN, PALETTE commands:

        screen 320 255 3 1
        palette 0 .4 .3 .5
        palette 1 .4 .3 .5
        palette 2 .1 0 .4
        palette 3 .6 .7 1
        palette 4 .5 .6 .8
        palette 5 .6 .5 .7
        palette 6 .7 .4 .6
        palette 7 .8 .3 .5

        darkcolor 2
        lightcolor 3

        noborder
        title off

        gap 6
```

```
        color light
        text "Super Neon Crackers"
        text "Proudly Presents:"
        space

        startbox
        color dark 4
        button "Snus Maze" 'run >nil: coolgame1' exit
        color dark 5
        button "Second Game" 'run >nil: coolgame2' exit
        color dark 6
        button "Oh, nomore supermasken" 'run >nil: coolgame3' exit
        color dark 7
        button "Bumper" 'run >nil: coolgame4' exit
        endbox
        space
        color light
        text "NeonBBS (+46)31-1234567"

        Try~it!
```

```
FONT <fontname> <size>

Sets the font for the window.
Ex:
     Font topaz 9

Instead of name or size, you can write "default" for the default font.
Ex:
     Font default 18

You can set selectors default font by starting the program SelectorFont
from workbench, or from shell with the keyword "default".

SelectorFont can also be used to request a font from the user, ex:

   defenv myfont "default"
   font [myfont]

   text "Current font:"
   text "[myfont]"
   button "Change it!" 'setenv myfont `SelectorFont`' update

   unsetenv myfont

Try~it!
```

```
See also: Startup-commands
          Panel~defining~commands
          Concepts
```

1.4 Startup

```
Startup commands:
-----------------
These commands executes every time the file is loaded or the window is
updated by the update command.

------------------------------------------------------------------------

EXEC 'command 1' ['command 2'] ['command 3'] .... etc.
Executes external commands as if you would write them into a shell
window. Note the use of the '-signs. This is made so you can use
"-signs in your command lines. Every command line is stated within
two '-signs. Like this:
Exec 'list >"ram:A file"' 'type "ram:A file"'
It is however possible to use the "-signs instead of the '-signs if
you not are going to use any "-signs in your command. Like this:
Exec "list ram:" is equvivalent to Exec 'list ram:'.
But Exec 'list "ram disk:"' is not equal to Exec "list "ram disk:""
where selector will believe that you want to execute the command
"list " and then the command ram etc. Read more about string syntax
under Concepts.

------------------------------------------------------------------------

SETENV variable "value"
Sets the env variable <variable> to "value". You can also accomplich
this with the external command setenv. Like this:
exec 'setenv variable "value"'

------------------------------------------------------------------------

DEFENV variable "value"
Sets the env variable <variable> to "value", but only if the variable
not previosly is set. This way you can specify default values on env
variables.

------------------------------------------------------------------------

DEFARG argument "value"
(Where argument is arg1....arg9)
Sets the argument to "value" if the argument not was passed from the
command line or button..load command. Please see the defarg~example
under Arguments~&~Other~variables.

------------------------------------------------------------------------

UNSETENV variable
Deletes the env variable <variable>. Please see the unsetenv~technique
example.

------------------------------------------------------------------------

VERSION <number>
This command tests selectors version number. Example:
```

```
   Version 4.3

If selectors version number is lower than 4.3, selector will end and say
that "This script needs Selector 4.3"

-------------------------------------------------------------------------

See also: Window~commands
          Panel~defining~commands
          Concepts
```

1.5 panel

```
Panel defining commands:
------------------------
These command specifies the contetns of the panel, with buttons,
text strings etc.

-------------------------------------------------------------------------

BUTTON 'Button Name' ['command 1' 'command 2' ... 'command n']
           [ UPDATE | LOAD <file> [arguments] | EXIT [<string>]
           | QUIT [<string>]] | DumpX <variable> | DumpY <variable> ]

This is selectors most important command. It spesifies the button name
and what actions that will happen if the button is pressed. The command
part (['command 1' .... 'command n']) has exactly the same syntax as the
exec command. So read about the exec~command for more details. But
with the button command, you can leave the command part out if you want.
You can for example make a button that does nothing at all:
Button "Nothing".

The EXIT option simply ends the program. You can use the EXIT <string>
command to write out a result to the shell window before quitting. If you
have used the OnExit command, this will be executed before ending.

The  QUIT option works in the same way as the
EXIT option except that the OnExit command (if used) not will be executed
before quitting.

The UPDATE option reloads the source file and updates the panel. This
is usefull if you make use of env~variables. If selector was loaded
with arguments, these will remain the same. If the window parameters
(size, title etc.) not are changed, then the window don't have to be
closed and opened again, and this speeds things upp a lot. It also
prevents you from "jumping" to the workbench every time the window
is updated. So if you like a fast update, then make sure that the
window size is the same. You can do this with the size~command.
NOTE: The X and Y parameters will be ignored on an update. The new window
will remain in the same position on the screen. If you want X and Y not
to be ignored, please use the LOAD [this] option instead.

The LOAD <filename> [arguments] option is similar to the UPDATE option.
But LOAD loads the source <filename> and rebuildes the panel after
that file. The possibility of passing~aguments with the load command was
```

```
introduced in Selector 4.1

The DumpX <variable> option saves the current X position of the window
to the env variable specified. This is useful if used from the OnExit
command for saving a default start position for the window.

The DumpY <variable> option works in the same way as DumpX.


Here is some examles of buttons:

Button Help 'run >nil: multiview help.doc'
 This one loads multiview with the file help.doc if the button
 Help is pressed.

Button Exit exit
 Selector is terminated when the button is pressed

Button Exit exit "Selector is ending"
 Selector writes the string to the shell window before quitting
 (when the button is pressed).

Button "Another Exit" 'delete t:temp' exit
 The file t:temp is deleted before ending

button Reload 'setenv cool Yes' update
 This one sets the env variable cool to Yes and then reloads the
 source and rewrites the window.

button Update Update
 is (almost) eqvivalent to
button Update Load [this]
 or
button Update 'run >nil selector [this]' exit
 (if you have selector in the path.)

Button "View Source" "type [this]"
 Types the source file

Button Other Load Other.sel "One" Two '"Three"'
 Loads selector with the source Other.sel and passes the arguments
 One, Two and "Three"

[this]~is~the~name~of~the~source.

------------------------------------------------------------------------

TEXT <string>
Writes the text <string> as an item in the panel. Example:
Text "This is written"
Text "in the panel."

------------------------------------------------------------------------

SPACE
Inerts a half space (half the heght of an item). But it is
counted as one item.
```

```
------------------------------------------------------------------------

COLOR <foreground> [background]     (or foreground <color>)
Sets the color of the following buttons and text. Example:
color 1
text "This is written in color 1"
color 2 1
text "This is written in color 2 on a background of color 1"

Instead of color numbers, you can use "light" and "dark" for the
colors set by the darkcolor~and~lightcolor commands. Example:

color light
color 3 dark

------------------------------------------------------------------------

CENTER ON | OFF
Specifies if the folowing text and buttons text is going to be
centered or not. If it is turned off, the text will be left justified.
Default is ON

------------------------------------------------------------------------

STARTBOX and ENDBOX
Specifies a inverted 3D box in the panel. Example:
Startbox
  text "This text and the next button"
  button "will be within a box"
endbox

Note 1: You must have a GAP (se~the~gap~command) of 2 or more for the
box to be seen.

Note 2: If you make use if the ROWS command, it is possible for a
box to begin on one column and end on another. Then this will look
like there is two boxes. But the line between them is so thin, that
it looks like the items are grouped together anyway.

Note 3: The boxes can not be nested.

------------------------------------------------------------------------

See also: Window~commands
          Startup-commands
          concepts
```

1.6 env

```
Env variables:
--------------
It is possible to make use of env variables anywhere in the source.
They are stated with []-signs around them. So if you have an env
variable with the name color ($color) then you should write [color]
```

in the selector source. Here is a little example:

```
Defenv g 6
gap [g]

text "The gap is now: [g]"
button "No Gap" 'setenv g 0' update
button exit 'unsetenv g' exit
```

Try~it!

How the variables works.

When the source is loaded (or updated) this is what happens:

1 The next line is loaded.

2 It is seached for any [variables], and they are replaced by the contents of the real env variables (if there is one). The contents of the env variable is actually copied into the text line. So the only time you can change any variables in a selector source is when it is loaded or updated. You can of course change the real env variables, and then update selector. This is what is done in the above example.

3 After this, the line is analyzed, and if it is a window command or startup command, it is executed. If it's an OnChange command, the contents of the variable to be watched is remembered at this point.

4 When all lines has been gone though, then the window is calculated and opened. Then the whole source is gone through again, and the Panel defining commands are analyzed.

This is why you must have the defenv command before the gap [g] command in the the example above.

The Defenv command sets the env variable g to 6 if the variable g not allready exists. So 6 is the default value for g.

Replacable commands:

Using env variables, you can not only have variable strings and numbers, but you can change the whole funcionality of the panel by using variable commands. Try this:

```
close off
defenv item "button 'Remove button' 'setenv item Text Nothing' update"
[item]
button exit 'unsetenv item' exit
```

Try~it!

Here the command [item] is defined to be a button that replaces itself with the text string 'Nothing'

You can also use env variables to switch some commands off with the
comment sign (;), like this:

```
close off
defenv c ";"
gap 4
rows 3

startbox
[c] text "Just press"
[c] text "the different"
[c] text "buttons"
endbox
button "Help text" 'setenv c " "' update
button "No Help" 'setenv c ";"' update
button exit 'unsetenv c' exit
```

Try~it!

Getting user input to an env variable.

For this, you must use the AmigaDos echo
command and open a new con window:

```
close off
defenv Me "Boray"
button '[Me]' 'echo >nil: <CON:30/100/400/40/Your_Name to env:Me ?' update
button exit 'unsetenv Me' exit
```

Try~it!

Requeststring

Another and a much better way to get user input is to use the external
command Requeststring that also is included in this package. (In the
Tools drawer). This command is made by Adam Dawes and is freeware.
(Requeststring.txt)
Here is the above example using requeststring instead of echo and CON:

```
close off
defenv Me "Boray"
button '[Me]' 'requeststring >env:Me body "Enter Your Name"
               title Request default "[me]"' update
button exit 'unsetenv Me' exit
```

Try~It!

(Please note that the line beginning with "button '[Me]' and the following
line are supposed to be on the same line.)

```
---------------------------------------------------------------------------

In source, but not in env:

If you have an env variable in your source, and that variable doesn't
exists, then the name of the variable will be used.

Example:

   setenv variable1 "This variable exists!"

   text '[variable1]'
   text '[variable2]' ;This doesn't...

Try~it!

---------------------------------------------------------------------------

Also see Other~variables and shell~scripts.
```

1.7 other

```
Other variables:
---------------------------------------------------------------------------

Current source name

[THIS] is the current loaded source file. It is no real env variable.
You can see it as a local variable for each running selector.

---------------------------------------------------------------------------

Arguments

[arg1] [arg2] .... [arg9] is the optional arguments passed to
selector from the command line. (Selector <source> [<arg1> ... <arg9>])
Or with the Button..load command:
(button ButtName load <source> [<arg1> ... <arg9>])

When you pass arguments to Selector, they are treated as normal
Selector strings, for example:

A source called arg.sel:

  text '[arg1]'
  text '[arg2]'
  text '[arg3]'

Run~it~like~this:

Selector arg.sel 'This is arg 1' "This is arg 2" 'This is "arg 3"'


Default arguments
```

```
You can use the DefArg command to specify a default
value for an arg variable. This default value will be used if the
argument wasn't passed from the command line (or button..load command):

Example: (source called "argtest.sel")

   defarg arg1 "First arg not given"
   defarg arg2 "Second arg not given"
   text '[arg1]'
   text '[arg2]'

Try it like this:

   1 selector~argtest.sel~Passing~arguments
   2 selector~argtest.sel
   3 selector~argtest.sel~*~*
   4 selector~argtest.sel~*~'Only~second~arg~given'
   5 selector~argtest.sel~'Only~first~arg~given'~*
   6 selector~argtest.sel~'Here~to'

As you can see, passing "*" will give the default string. You must use
the "*" if you only want to pass the second argument (like in
case 4 above).

------------------------------------------------------------------------

About arguments without default value

If you don't have a default value for an argument that not is passed, then
the name of the argument will be used.

Example: (source called "argtest2.sel")

   text '[arg1]'
   text '[arg2]'

Try it like this:

   selector~argtest2~Hello!

------------------------------------------------------------------------

Also see the use of selector in shell~scripts.
```

1.8 shell

```
Selector in shell scripts
-------------------------
Selector can be used as a powerful tool within shell scripts.
This thanks to:

   * The possibility of using env variables within selector source files.

   * The possibility of passing variables to selector from the
     command line.
```

* The possibility of passing a return string (or variable) from a selector script (with the button option exit <string>).

* The possibility of making selector sources with the AmigaDos command List LFORMAT.

General requester

It is sometimes a good idea to have a little requester to use in shell scripts.

Here is a selector source called req.sel:

```
gap 6
rows 2
title "Request:"

space
color 2
text "[arg1]"

color 1
startbox
  button Yes exit Yes
endbox

button No exit No
```

(The [arg1] variable is the first argument given to selector)

Here is a shell script that uses req.sel:

```
if `selector req.sel "Really delete?"` eq Yes
  delete "file"
endif
```

Try~them~together!

Confirmation tool

In the example above we used a general requester to confirm if a command was to be executed or not. This spesific problem can be solved in a much better way: Pass the command to selector, and let selector execute it if you press Yes.

```
gap 6
close off
title 'Please Confirm'
rows 2
```

```
    color 2
    text "Okay to:"
    text '[arg1]'

    color 1
    startbox
      button Yes '[arg1]' exit
    endbox
    button No Exit
```

Use it like this:

```
  Selector confirm.sel 'delete "ram:the file"'
```

If you press Yes, the command given by [arg1] is executed.

Try~it!

List LFORMAT

The use of the List LFORMAT command and Selector gives a powerful combination as seen in the folowing shell script:

```
.key dir

ECHO >ram:panel "title '<dir>'"
LIST >>ram:panel <dir> files pat ~(#?.info) LFORMAT "button '%n' 'run >nil:
*"%f%n*"'"
RUN >nil: SELECTOR ram:panel
```

Try~it! (The script is executed with the argument "sys:utilities")

TinyMess

Using the OnChange~command and the same principle used in the clock~example, we can make a little message window that is shown for two seconds. We can than use this from a shell script:

```
       title off
       text "[arg1]"

       setenv tinymess "[arg1]"
       onchange tinymess exit
       unsetenv tinymess
```

Try~it:

```
run >nil: selector tinymess 'Please wait!'
```

See also: SEL: and More~trix~&~tips

1.9 keys

```
Keys:
-----
You are able to exit anytime by pressing ESC....
You can also press the keys 1..to 9.. for pressing the first
nine buttons. If you press return, then the last button pressed
will be pressed once more.
```

1.10 intro

```
 Introduction

 Selector was from the begining just a program shipped with the
 blanker "WakeUpII". It was ment to serve as a launching tool for
 the different programs included in that package. But it has grown
 to a highly configurable launching tool for almost any purpose. But
 it is not yet another program like toolsdeamon or launchingpad.
 Selector serves as a complement to these types of programs.

 It's strenght lies in what it once was design to do. To be a
 launching tool for a group of software.

 Selector is a powreful tool to use within shell scripts or to
 make a little GUI for your CLI type program. I have for example
 made a GUI for GMPlay (gmplay_gui13.lha on Aminet) using selector.

 It is really easy to produce a button panel, info window  or a little
 requester with selector.

  -This~I~wan't~to~see!

------------------------------------------------------------------------
```

1.11 start

```
Beginners cource

Now let's see how easy Selector is to program. If there is a command
that you don't understand, look in the command~index.

------------------------------------------------------------------------

We will begin with a little info window:

  text "This is a little info"
  text "window made with"
  text "Selector 4.0"
  space
  text "Close window to continue"

The script above is written to a file called "b1.sel"
```

```
Then it is run with:   Selector b1.sel

Try~it!

-Now, wasn't that easy?

-------------------------------------------------------------------------

Now we will try to make a little button panel instead:

  button "The clock" 'run >nil: sys:utilities/clock'
  button "Startup" 'type >con: s:startup-sequence'
  button exit exit

Try~it!

-------------------------------------------------------------------------

Now, how about a little requester?

  close off
  gap 6
  rows 2

  color 2
  text "Leave from this"
  text "requester now?"

  color 1
  startbox
    button Yes Exit
  endbox
  button No

Try~it!

-------------------------------------------------------------------------

It is possible to execute several external AmigaDos commands on the same
button. In the folowing example, we make one tiny button that uses the
AmigaDos command echo to create another selector source called ram:s
And then it loads it with the button option LOAD.

  title off
  gap 0

  button "Press" 'echo >ram:s "text *"A button that makes*""' 'echo
>>ram:s "text *"Another selector script*""' 'echo >>ram:s "text *"and
runs it*""' load ram:s

(Please note that the 3 last lines above should be on the same line.)
Try~it!

-------------------------------------------------------------------------

Now I think it's time for you to look at one of the folowing sections:
```

```
Concepts
Panel~defining~commands
Window~commands
Startup~commands
```

1.12 tips

```
Creating a switch button

You can simplify making a switch button by writing a little shell script:

  .key var,s1,s2

  if $<var> EQ <s1>
         setenv <var> <s2>
  else
         setenv <var> <s1>
  endif

Name this script "switch", Set the script falg (Protect "switch" +S)
and use it as shown below:

   close off
   gap 4
   defenv s ON

   button exit 'unsetenv s' exit
   space
   startbox
     text "The switch is:"
     button [s] 'switch s ON OFF' update
   endbox

Try~it!

------------------------------------------------------------------------

More about variables...

Remember that the env variables in a selector source file only are
updated when a source is updated~or~loaded. So don't try something
like this:

button "Load source" "setenv name `requestfile`" load [name]

The variable [name] is not changed... You can solve the problem
above like this:

button "Load source" "run >nil: selector `requestfile`" exit

------------------------------------------------------------------------

Changable arguments

Maybe you would like to pass a variable from the command line, that then
```

could be altered. This you can do by passing the arg to an env variable:

Example: (Source called ChArgs.sel)

```
defarg arg1 'Nothing passed'
defenv vary '[arg1]'
close off

button '[vary]' 'setenv vary "Button pressed"' update
button Quit 'unsetenv vary' exit
```

Try it like this:

```
Selector~ChArgs~'Press~the~button!!'
Selector~ChArgs
```

Unsetenv technique

If you wan't some variables in an information window without buttons, you could wonder how to unset them again. This because there is no exit button to unset them on... well, the answer is simple, use the unsetenv command at the end of the source:

Ex: (source called "etips.sel")

```
defarg arg1 c:type
exec 'setenv size `list "[arg1]" lformat "%l"`'

text 'The size of [arg1]'
text 'is [size] bytes'

unsetenv size
```

Try it like this:

```
selector~etips
selector~etips~sys:utilities/more
```

This technique is possible because of the~way~selector~inserts the env variables into the source, line by line. You can use this technique as long as you only use [variables], and no $variables. This because the variables only exists when the panel is loaded, and not when a button is pressed etc. Then they exist in the source, but not in ENV:

This technique is in fact preferable over the alternative to unset the variables on an exit button. It also makes the env variables occupied for a shorter time, which in turn prevents name conflicts. It also in many cases let you skip the exit button and "close off" command. The fact that I not used this technique in many of the examples, is that I didn't think of this possibility until some minutes ago. I also added the unsetenv command to speed the technique up (instead of exec 'unsetenv...)

Example:

The corresponding source (using this technique) for the

```
dissapearing~button~example would be:

   defenv item "button 'Remove button' 'setenv item Text Nothing' update"
   [item]
   unsetenv item

Try~it!

But, as mentioned, this technique is NOT allways possible to use, for
example in the switch~example.

------------------------------------------------------------------------

Passing variables

If you want to write, for example a general requester that can be called
from other selector scripts, you can make this in several different ways.
I want to show you three different ways to accomplish the same goal.

In the examples, we want to call a color requester, and set the
background to this color. Each of the alternatives uses different
ways of passing variables.

1. The simplest way. One that starts a new copy of selector. This
   takes of cource a little more memory, and is a little bit slower
   than the next two alternatives. It is much more slow if you not
   have a hard disk and selector not resident.

   Calling file:
   background [color]
   button "Select color" "setenv color `selector color.sel`" update

   Called file:
   button "Gray" exit 0
   button "Black" exit 1
   button "White" exit 2
   button "Blue" exit 3

2. One that loads the other file to this copy of selector, and then
   reloads the old source. For this you must pass the name of the calling
   source to the called file.

   Calling file:
   background [arg1]
   button "Select color" load 'color.sel "[this]"'

   Called file:
   button "Gray" load '"[arg1]" 0'
   button "Black" load '"[arg1]" 1'
   button "White" load '"[arg1]" 2'
   button "Blue" load '"[arg1]" 3'

3. Similar to number 2. But here the name of the variable that will
   be set is passed to color.sel, and set in that file.

   Calling file:
   background [color]
```

```
   button "Select color" load 'color.sel "[this]" color'

   Called file:
   button "Gray" 'setenv [arg2] 0' load '"[arg1]"'
   button "Black" 'setenv [arg2] 1' load '"[arg1]"'
   button "White" 'setenv [arg2] 2' load '"[arg1]"'
   button "Blue" 'setenv [arg2] 3' load '"[arg1]"'
```

1.13 cr

```
Copyrights

Selector is Copyrighted (C) 1997 by Anders Persson, but you can
distribute it with your freeware software as long as you also include
the readme file (Selector/Selector50.readme).

You can also include selector with any shareware or commercial
software, as long as you send me a registred version of your
product.

Here is my address:

Anders Persson
Mellangardsv. 9
S-417 29 Goteborg
SWEDEN

The fact that Selector is freeware doesn't mean that I don't like gifts,
like money or perhaps a program that you have made...
```

1.14 input

```
Getting input

If you like to run a CLI/shell program that must have input from the
keyboard, then you MUST open a new con: window for both output
and input. Like this:

button "Check" 'checkmate <>con:0/11//210/CheckMate/Close [cont] b [base]'

button "DiskCopy" 'diskcopy <>con:0/11//180/DiskC/Close from df0: to df0:'
```

```
Filename input

If you have AmigaDos3.0 or better, you can use the C:requestfile
command to get a filename. Here is a little example:

  button "Read text" 'multiview `requestfile`'
```

```
Try~it!

Please note the difference between the ` and ' signs. The ` sign
(placed over the tab button) is used in AmigaDos to pass output from
one command directly into another command. This is how it works:
Ex:

list `echo "S:"`

First the command echo is executed. The output from this command is
  S:
The text `echo S:` is then replaced by the output from echo (S:)
  list S:
This command is then executed.

------------------------------------------------------------------------

Also se: How~to~get~user~input~to~an~env~variable.
```

1.15 res

```
Making Selector Resident

Making Selector resident will speed things up, and save memory if you
have more than one selector running at the same time. If you only have
one selector running, it makes no difference until you start one more...

So the only time you not should make it resident is, if you don't
use selector... very often

You can make Selector resedent by adding this line to your
user-startup (or startup-sequence)

Resident C:Selector pure

(If you have Selector in C: of course)
```

1.16 sel

```
Selectors searchpath

If you want to load Selector with a source file called "Filename", you
probably do this by the folowing command:

   Selector "Filename"

Now, selector starts to look for the file "Filename" in the following
order:

1. "Filename"
```

```
2. "Filename.sel"

3. "SEL:Filename"

4. "SEL:Filename.sel"

This way, you can have a directory where you have all your selector
files (.sel) and then assign SEL: to it. You can then, without bothering
in what directory you stand, allways reach the files. This works in the
same way, as the S: directory works for the AmigaDos execute command.

You should then add something like this to your user-startup:

Assign SEL: work:selector

------------------------------------------------------------------------
```

1.17 why

```
Imagine you have made a software packackage,
including some programs, for example one for listing all tasks of the
system, and another for showing which tasks that requires most CPU
time. And some other programs that has other functions...

How would this be for the user? Well... He could click himself to
the drawer containing these programs every time he needed them. But
when he had done that a couple of times, he would probably put
some (or all) of the programs in his launching tool. For example
as a sub-menu in toolsdeamon.

Let's make it more easy for the user, so that he won't need to include
all the different programs, one by one, in his launching tool.
With Selector, you can ship a little nice button panel with
your software, and the user will only need to add one item to his
launching tool.
```

1.18 ci

```
Command index

Background
Button
Center
Close
Color
Darkcolor
Defarg
Defenv
DumpX
DumpY
Endbox
Exec
```

```
Exit
Font
Foreground
Gap
Lightcolor
Load
NoBorder
OnExit
OnChange
Palette
Quit
Rows
Setenv
Size
Screen
Space
Startbox
Text
Title
Unsetenv
Update
Version
X
Y
```

1.19 slut

```
The End

I would very much appreciate if you sent me a mail and told me how
you are using selector and what you think about it.

Send an e-mail to:
  di3andpe@ida.his.se

If you don't get any answer, I have probably left scool, so send
an ordinary mail to this address:

  Anders Persson
  Mellangardsv. 9
  S-417 29 Goteborg
  SWEDEN

If you don't have email, you can of cause use this address right away.

Regards/
Anders Persson
```

1.20 vers

```
Version History
------------------------------------------------------------------------
```

```
Selector 1.08 (940713)

  -Static button panel for launching the different programs
   in the WakeUpII screenblanker package.


Selector 2.06 (940717)

  -The button panel was made configurable, with the commands:
   "Button", "X", "Y", "Size" and "Center"


Selector 3.20 (940729)

  -More commands:
   "Title", "Gap", "BackGround", "ForeGround", "LightColor" and "DarkColor"

  -{this} a variable for current source

  -Support programs for making panel design more easy:
   "AddButton 1.03 (940720)", and "NewPanel 1.01 (940721)"


Selector 4.0ö1 (960425)

  -More commands:
   "text", "space", "Title off", "Close", "rows", "exec", "setenv",
   "defenv", "color", "startbox", "endbox", "button...uppdate"
   and "buton...load"

  -Panels can now be more than one row of buttons. They can also
   contain text strings and inverted boxes.

  -Env variables support

  -Possible to pass variables from command line ([arg1]...[arg9])

  -Possible to pass return data with the "button...exit" command

  -{this} altered to [this]

  -Selector can not be started from Workbench any longer, this because
   it didn't get the whole search-path when it did. And this would get
   incompatible scripts.

  -The support programs (AddButton and NewPanel) became unmodern and
   was removed.

  -This guide document was written.


Selector 4.14 (960529)

  -Bugfix:
   Selector did enter an infinite loop when there was a text
   string or button name that was wider than the screen.
```

-Bugfix:
 It wasn't possible to exit by pressing escape when there was
 no buttons in the panel.

-Bugfix:
 Now possible to use arg0, arga, argb, argc etc. as env variables.

-Possible to pass arguments with the button...load command

-The DefArg command was introduced.

-Unstated~arguments that not have a default value (given with defarg)
 will now contain the name of the argument rather than nothing. This
 was made so that all variables (env, arguments and [this]) will
 work in the same way.

-Env variables and arguments can now contain nothing ("").

-Selector builds the panels on the screen faster.

-Source file name given in error messages.

-The unsetenv~command was introduced and the unsetenv~technique was
 invented.

-More examples in this document

Selector 4.30 (960804)

The documentations for version 4.2 and 4.3 was lost in my second
HardDisk death. I have now (970101) updated this amigaguide document
with all the changes (from 4.14 to 4.30) that I have found.

-New commands: OnChange, NoBorder, Version, Screen~and~Palette.

-The color and background commands were extended.

-The update~of~the~window is made much faster if the window
 parameters not are changed (Size, title etc.).

-Some more examples and a new 'All~Examples' script
 were added to this document.

Selector 5.0 (970319)

-Bug on graphics cards and super hires fixed.

-Fonts support. New command: Font

-New support program: SelectorFont, to set the default font for
 selector, or request a font in your scripts. This program
 needs asl.library.

-Now possible to open the window on any public screen

```
   using the extended SCREEN~command.

  -The X~and~Y~commands were extended.

  -New command: OnExit

  -New button/onchange/onexit options: Quit,~DumpX,~DumpY

  -The button/onchange/onexit option "Update" will now ignore the X and
   Y commands. If you want selector to behave as before: Use Load [this]
   instead of update.

  -AmigaDos1.3 no longer supported. At least AmigaDos2.0 is needed.
   Some of the included examples need AmigaOS3.

  -Added Bonus Tools

  -The copyrights is changed.

          Greetings to Henrik Thurén, "KnC" McWhirter, Jörn Hansson
            and Stefan Nilsson! Thanks for your help and support!

------------------------------------------------------------------------
```

1.21 The OnChange command

```
The OnChange Command

This is a new powerfull command in selector 4.3. It specifies what
will happend when an env variable is changed.

OnChange <variable> [to value] ['command 1' 'command 2' ... 'command n']
        [ UPDATE | LOAD <file> [arguments] | EXIT [<string>] ]
        | QUIT [<string>]] | DumpX <variable> | DumpY <variable> ]

The first part (OnChange <variable> [to value]) specifies the
conditions for the the rest of the commands to execute. <variable> is
the name of the env variable to be watched for any change. If you specify
the [to value] option, then the commands are only executed if the variable
is changed to this specified value. The last part (['command 1' ....
... 'command n'] [ UPDATE | LOAD <file>  etc..)
specifies what will happend, and is stated in exactly the same
way as in the BUTTON~command.

Example:

       DefEnv test "Change Me!"

       Onchange test to end exit
       Onchange test update

       color light
       text "Current value of test is:"
       color dark light
       text "[test]"
```

```
       text
       color dark
       button 'Set it to "Hello!"' 'setenv test "Hello!"'
       button 'Set it to "That is cool!"' 'setenv test "That is cool!"'
       button 'Set it to "end"' 'setenv test end'
```

This example opens a window with the contents of variable "test".
Every time the variable is changed, the window is updated. If it
is changed to "end", the window is closed. You can change the
variable by pressing the three buttons, or with the setenv command
from a shell window. The update is not performed right away because
selector only checks the variables for change every two seconds.

Try~it!

If you now Try~it~again, you will see that
the variable allready is set to "end" (you did it a little while ago,
remember) and the window is not exiting... why? Well, as the commands
name is "OnChange", selector won't bother because the variable was not
CHANGED since the moment the script was last loaded or updated. You must
first set the variable to something else, wait two seconds, and then set
it to "end" to close the window.

Also try to start more than one window with the example above, and watch
how all the windows are updated when you press the button in just
one of them.

Observe that you (in the example above) can't place the two onchange
statements in the opposite order. Then the second statement would never
be reached. You can look at this as an if....elseif statement.

There are two internal values you can use: *any* and *none*

```
   OnChange test to *none* exit
```

This means that the window will be closed if the env variable "test"
is unset.

```
   OnChange test to *any* exit
```

This is the same as:

```
   OnChange test exit
```

Another OnChange example: "The Clock"

```
       x 2000
       y 11

       exec 'setenv datum `date`'
       onchange datum update
       text "[datum]"
```

```
       unsetenv datum

Try~it!

This script can be quite hard to understand. But it is illustrative for
how the onchange command works. First the variable "datum"
is set to the string returned by the amigaOS command "date". Then follows
the onchange command. It now "remembers" the contents of the variable
"datum". Then as you can see at the bottom of the script, the variable
datum is unset. When the whole window is displayed, selector waits about
two seconds and then compares the old remembered value of "datum" with
the current one... Oh.. It is changed!!! to nothing (*none*).
Now, lets see what to do, I have to update.. yes... And then the
variable is set again by the exec command, and the whole procedure
is starting all over again.

-------------------------------------------------------------------------

You might want to look at how~selector~analyses~a~script.
```

1.22 inst

```
Installing is easy...

Just move the executable file "Selector" to your C: directory.


-------------------------------------------------------------------------
When you have learned a little bit more about Selector, you might
want to look at SEL: or Making~resident.
```

1.23 onexit

```
The OnExit Command

This is a new command in selector 5.0. It specifies what
will happend just before the program ends.

OnExit ['command 1' 'command 2' ... 'command n']
        [ UPDATE | LOAD <file> [arguments] | EXIT [<string>] ]
        | QUIT [<string>]] | DumpX <variable> | DumpY <variable> ]

The syntax is exactly the same as in the BUTTON~command. except that
no button name is given. There can be only one OnExit command in a source.

The command is executed when the window is closed, using the standard
close-gadget, when pressing ESC or when an EXIT command is reached from
a Button/OnChange command. But if the QUIT command is used instead of
the EXIT command, the OnExit command won't get executed before
quitting.

Use this command with care! It's possible to make a window you can't
```

```
get rid of, for example:

   onExit update
   text "Can't get rid of me!"


Better example of OnExit usage:

   defenv xvar auto
   defenv yvar auto

   x [xvar]
   y [yvar]

   onexit dumpx xvar dumpy yvar

   text "Move this window around and"
   text "then close it! Open it again!"
   text "Where will it appear next?"

Try~it!
```

1.24 name

```
Name Conflict

I have now (970309) found out that there is another "Selector" out there...
"Program Selector V3.0" by Nico François 1991, and a compatible replacement
for "Program Selector" called "Chooser V1.0" by Simon Dick 1995. There is
(in my oppinion) no meaning in using these instead of my "Selector" (Sorry
guys...). What you can do with these two programs is basically only what
you can do with the Button and Text commands in my Selector.
```

1.25 tools

```
Bonu Tools

I have here included some programs that I think is useful in shell scripts
or directly from selector. They are either FreWare or PD, so you can (as
I understand) use them however you like. But read the .docs and .readmes
to be sure.


Here they are described:
-------------------------------------------------------------------------
 BreakName, by Kai Iske                                  (BreakName.doc)

       This command is used to break a process with the specified name.


-------------------------------------------------------------------------
```

Change, By myself

This is used to edit a file, to change all (or some) occurences of one text string to another. Try "Change ?" for more information.

Count, By myself

This command counts all occurences of a text string in a file. Try "count ?" for more information.

PSRun, By myself

PSRun <PubScreen> <command>, If the public screen exists, then this screen is moved to front, else "run >nil: <command>" is executed. Try "PSRun ?" for mor info. This is useful to get a button in selector to open a program if it isn't allready open, else to move to the programs screen. Ex:

button Diskmaster 'psrun DM c:DM2 config.dm'

---

psx, by Steve Tibbett (psx.docs)

A public screen manager. You can open new screens, etc.
Try "psx ?". You can also start only "psx" and it will open a GUI.

RequestString, by Adam Dawes (Requeststring.txt)

This command has been described earlier Here
It simply requests a string using the reqtools.library

Unsort, by myself

Unsort <textfile>, The file is "line-randomized". If you want things to happend in random order, you can for example do this:

```
list >ram:batfile mods: lformat "modplay %s"
unsort ram:batfile
execute ram:batfile
```

And don't forget the SelectorFont program. It is also useful in scripts.