Pashua 0.10.3 Documentation

Table of Contents

Introduction

What is Pashua?

Pashua ist a tool for creating native Aqua dialog windows on Mac OS X. It can be invoked from the shell and therefore is particularly useful for creating GUIs from programming languages that do not offer graphic user interfaces natively, for instance Perl, PHP or Ruby.

Pashua was written by Carsten Blüm and released as donationware (free to use, but donations welcome). Pashua’s website is www.bluem.net/en/mac/pashua/

How to use Pashua

Creating a dialog

Basically, Pashua is used like many command-line programs: you call it from your application and pass the path to a configuration file as argument (or let Pashua read the configuration string from stdin by passing `-` as the filename). Pashua will display a dialog that is created according to the description in the configuration file / string, while typically your application will wait for the dialog to close.

Getting values back from Pashua

Pashua will quit when the user clicks the “OK” button (or enters return), clicks the “Cancel” button or hits Escape (provided there is a cancelbutton in the window). In the first case, for every element defined in the config string, the value is returned. Let’s say, the window contains a textfield named `city` that holds the string “Hamburg”, then the result (written to stdout by Pashua) would contain a line `city=Hamburg`. Your application could read Pashua’s output and parse each of the lines to get the value for the element.

In the latter case (“Cancel” button), no values are returned, except the cancel button itself, which is returned as `1`. This means: if your configuration string contains a definition like `cb.type = cancelbutton` and the button will be pressed, then Pashua’s output will contain `cb=1`.

Using the example scripts

However, you don’t have to deal with calling Pashua and interpreting the result manually—the example scripts included in the distribution do that for you. In most of these scripts, you only have to pass the configuration string to a function or method and that’s it.

Of course, if you have special needs or don’t like the way the example scripts work, you are free to write your own code; it’s completely up to you.

The window configuration

As said above, the dialog window to be created is described in a configuration file or string. For this purpose, Pashua uses a very simple and light-weight configuration syntax which describes the GUI elements to display, their properties (e.g. default values) und their order in the window.

The following part of the documentation will describe the basics of the configuration syntax, the GUI elements available and the attributes that can be set for these elements.

Basic configuration syntax rules

Pashua's configuration syntax is pretty simple. Basic rules are:

- Empty lines are ignored.
- Lines starting with the hash symbol `#` are treated as comments—like in Perl, Shell, PHP, Python, Ruby, Tcl, probably a few other languages and many Unix configuration files.
- Every command must be on its own line.
- Every command consists of: a string (called "identifier" below), followed by an equal sign `=`, followed by another string. The only exception to this rule is: a line consisting of nothing but a hyphen `-` will create a horizontal separator line at the appropriate position in the window.
- Every "identifier" must consist of a unique name for a GUI element (any ASCII string, preferably all lowercase—please note that window attributes must use an asterisk `*` as that name, see window attributes), followed by a period, followed by an attribute name. Each type of GUI element has its own set of allowed attributes, and a large part of this documentation deals with those properties. The name for the GUI element is not only a string that is used internally by Pashua, but it will also be used for returning values to the calling script / application, so you should regard these names as some sort of variable names.
- Anything after the equal sign is considered the value.
- Attributes and the names of the element types are case-sensitive.
- Leading and trailing whitespace is ignored for both the identifier and the value. For instance, for Pashua the following two lines would be the same:
  ```
  name.label=Please enter your name
  name.label   =    Please enter your name
  ```
- The GUI elements are displayed in the order in which they appear in the config string. The only exception to this rule are buttons: the default button is always in the lower right corner of the window, the cancel button (if defined) is located left to the default button and any other buttons will appear in the lower left corner, from left to right, in the order they were defined.
- You don't have to specify attributes in consecutive blocks. You can add as many empty lines as you like, and you could even mix up commands of different elements.
- When an attribute is specified more than once, the last one will be used:
  ```
  x.label = Label A
  x.label = Label B
  x.label = Label C
  ```
  … would result in the element being labeled with "Label C". An exception to this rule is `option`,

which is used to set values for radiobuttons, combo boxes and popups and which may be used in multiple commands.

A first example

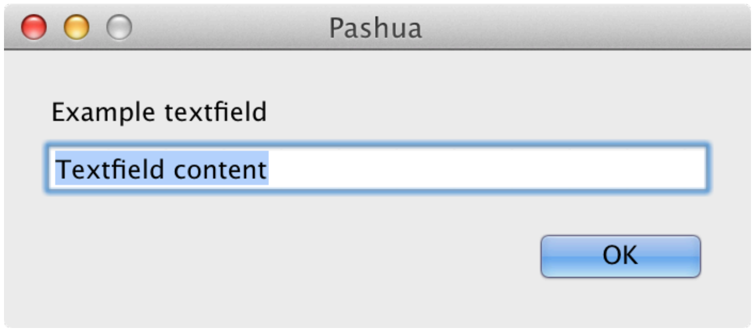
Let's have a look at a simple example:

```
# Add a text field
tx.type = textfield
tx.label = Example textfield
tx.default = Textfield content
tx.width = 310
```

These lines would simply mean:

- Line 1: A comment
- Line 2: The window should contain a textfield that we simply call "tx" to be able to address it in the next lines and to get the returned value later on.
- Line 3: The textfield should be titled "Example textfield".
- Line 4: Set the initial text displayed in the textfield to "Textfield content".
- Line 5: Set the textfield's width to 310 Pixels.

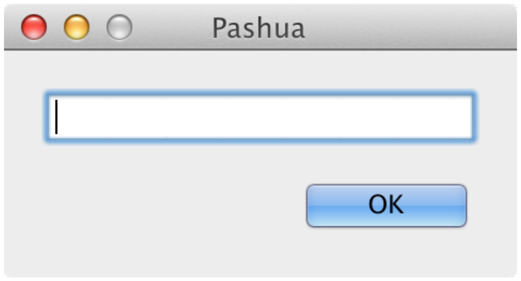
Note that the above lines would suffice to display the following window:



Or, if you like to keep things *really* simple, passing ...

```
tx.type = textfield
```

... to Pashua would result in this window:



Note: Pashua uses the standard UI elements of the Mac OS X version on which it is running. All the screenshots in this documentation were taken on OS X 10.9; if you would run the examples on Mac OS X 10.6, you would therefore see a slightly different result.

Window attributes

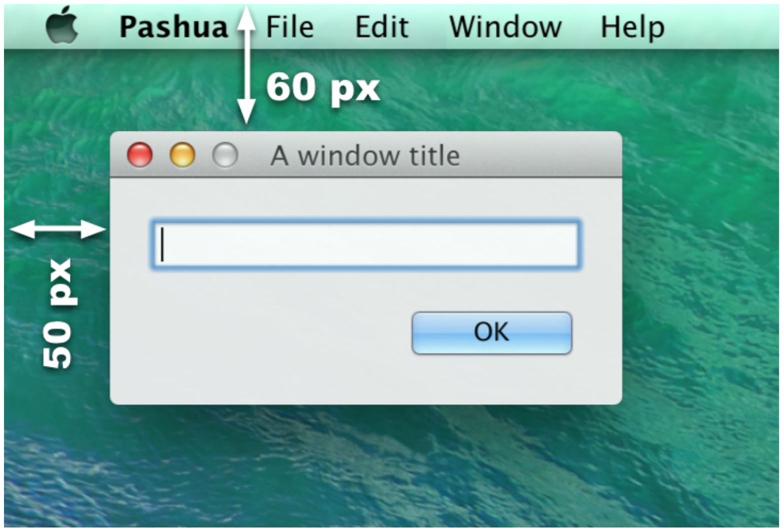
Window attributes are defined similar to element attributes. The only difference (apart from the fact that of course windows have other attributes than, let's say, textfields) is the fact that you don't have to specify an element name before the attribute, but simply an asterisk, for instance `*.title = My windowtitle`

Table: Window attributes

Name	Purpose	Required	Default
`appearance`	Only allowed value is `metal`, which will create a "brushed metal" window	No	-
`autoclosetime`	If set to an integer number larger than 1, the dialog will automatically close after the specified number of seconds have passed. The timer starts in the very moment when Pashua has finished parsing the configuration string and everything is set up.	No	-
`autosavekey`	Can be used to preserve the window position between launches. To let Pashua differentiate between applications, you have to set this to an arbitrary string. I.e.: one application can set this to "abc" and another one to "def", and for both applications, the window position will be saved and restored separately.	No	-
`floating`	Setting `autoclosetime`to `1`will result in the window floating above other windows.	No	0
`title`	Sets the window title	No	Pashua
`transparency`	Sets the window's transparency, decimal value from 0 (invisible) to 1 (opaque)	No	1
`x`	Sets the horizontal position where the window should be opened on the screen (`0` is the left border of the screen)	No	Window will be positioned automatically
`y`	Sets the vertical position where the window should be opened on the screen (`0` is the upper border of the screen)	No	Window will be positioned automatically

Example: Setting window attributes

```
*.title = A window title
*.transparency = 0.95
*.x = 50
*.y = 60
```



Element type: `button`

A button works similar to the default button: when clicked, it closes the window and returns all elements' values, but additionally, the button's own value is returned as `1`. Buttons ("regular" buttons, not the cancel button or the default button) are always positioned in the lower left area of the window, though you can position them absolutely using attributes `x` and `y`

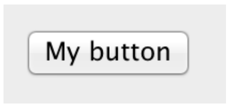
Table: Attributes for elements of type `button`

Name	Purpose	Required	Default
`label`	Sets the button's text	Yes	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–

Return value: Either `1` (button clicked) or `0` (button not clicked)

Example: Using `button`

```
b.type = button
b.label = My button
```



Element type: `cancelbutton`

A `cancelbutton` can be triggered using Escape and closes the window without returning any values, except the `cancelbutton`'s own variable, which will be returned as `1`. The cancel button is always positioned to the left of the default button and can not be moved to any other position.

Table: Attributes for elements of type `cancelbutton`

Name	Purpose	Required	Default
`label`	Sets the button title	No	Depends on the localization.
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–

Return value: Either `1` (button clicked / user pressed Escape / user pressed Cmd-W) or `0` (button not clicked)

Example: Using `cancelbutton`

```
cb.type = cancelbutton
cb.label = Close this dialog
cb.tooltip = Closes this window without returning the values entered
```

Element type: checkbox

Displays a checkbox

Table: Attributes for elements of type `checkbox`

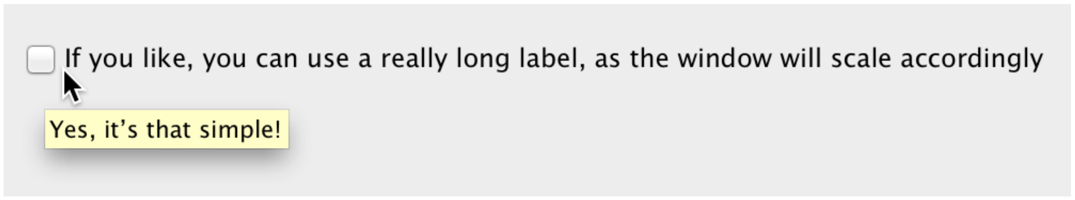
Name	Purpose	Required	Default
`label`	Creates a label / title next to the checkbox	Yes	–
`default`	Set this to `1` if you want the checkbox to be checked by default.	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx`	No	`0`

	value.		
`rely`	Relative vertical distance to the next element below (“relative” means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: Either `1` (checkbox checked) or `0` (checkbox not checked)

Example: Using `checkbox`

```
chk.type = checkbox
chk.label = If you like, you can use a really long label, as the window will scale accordingly
chk.tooltip = Yes, it’s that simple!
```


If you like, you can use a really long label, as the window will scale accordingly
Yes, it’s that simple!

Element type: combobox

A combo box is a combination of a popup menu and a textfield: the user can either choose a value from a list or enter any string.

Table: Attributes for elements of type combobox

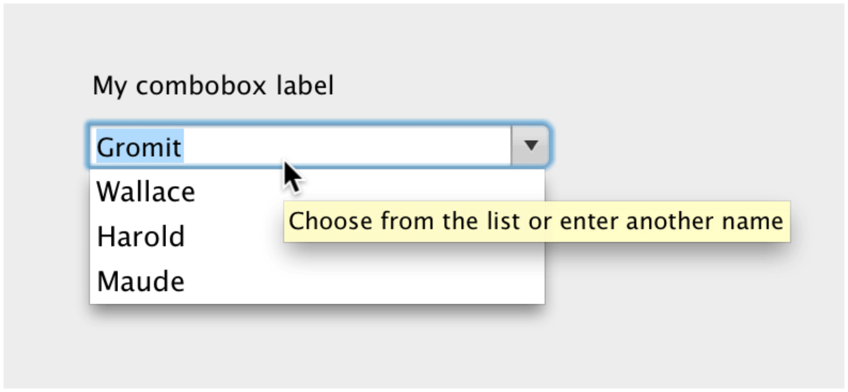
Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`option`	Adds a value to the list of values. Can (usually should) be used more than once.	Yes (one option attribute is required, others are optional)	–
`completion`	Controls if and how autocompletion is performed. Possible values are `0` (no completion), `1` (completes the first item in the completion list which matches the entered string, case-sensitive), or `2` (ditto, but case-insensitive).	No	`1`
`mandatory`	If set to a true value (everything other than 0, “n”, “no”, empty), input is mandatory.	No	No
`placeholder`	If present, this string will be as the field’s placeholder string.	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–

`width`	Width in pixels	No	200
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: String contents (may be an empty string)

Example: Using `combobox`

```
cb.type = combobox
cb.label = My combobox label
cb.default = Gromit
cb.option = Wallace
cb.option = Harold
cb.option = Maude
cb.width = 220
cb.tooltip = Choose from the list or enter another name
```



Element type: `date`

The `date` element lets the user choose a date, a time or both. It can be displayed in textual or graphical style.

Table: Attributes for elements of type `date`

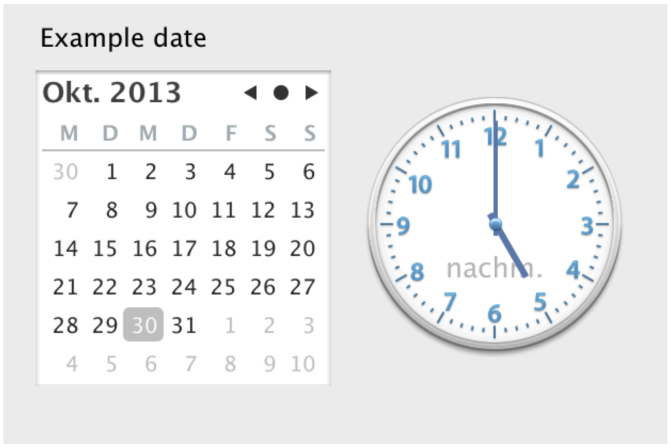
Name	Purpose	Required	Default
`label`	Sets the date picker's label (displayed above)	No	–
	Should the textual display style be used instead of the		

`textual`	graphical style?	No	`0` (No)
`date`	Should the user be able to chose a date?	No	`1` (Yes)
`time`	Should the user be able to chose the time?	No	`0` (No)
`default`	Default date and/or time that should be selected when the dialog is opened. The only string format that is *guaranteed* to work is “yyyy-mm-dd [hh:mm]”, for instance “2011-11-29 12:34” or “2011-11-29”. Other string formats such as “12/24/2004”, “next wednesday” or “tomorrow” *may* work, too.	No	Current date and/or time
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–

Return value: Depends on the values of attributes `date` and `time`. If only a date can be selected, it will be a date string in `yyyy-mm-dd` format, if only a time can be selected, the format will be `hh:mm`. If both are enabled, you will get a date and time string in `yyyy-mm-dd hh:mm` format. If you only need part of this information, you have to extract the desired substrings yourself.

Example: Using `date`

```
d.type = date
d.label = Example date
d.default = 2007-05-30 17:00
d.time = 1
```



Element type: `defaultbutton`

When the default button is pressed, the window is closed and all elements’ values are returned to the calling script. The default button is always located in the lower right corner of the window and can't be moved to any other position.

A default button is added to each window *automatically* – you only have to specify it explicitly, if you want to set the label or a tooltip or need the return value (i.e.: has it been clicked?) of this button.

Table: Attributes for elements of type `defaultbutton`

Name	Purpose	Required	Default
`label`	Sets the button title	No	OK
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–

Return value: Either `1` (button clicked / user pressed Return) or `0` (button not clicked)

Example: Using `defaultbutton`

```
db.type = defaultbutton
db.label = Click here to close the window and save the values
```

Element type: `image`

This element displays an image (or a PDF file), optionally scaling it to a maximum width or height. Pashua can handle any image type that is supported by NSImage. This includes TIFF, GIF, JPEG, PNG, PDF, PICT and EPS.

Table: Attributes for elements of type `image`

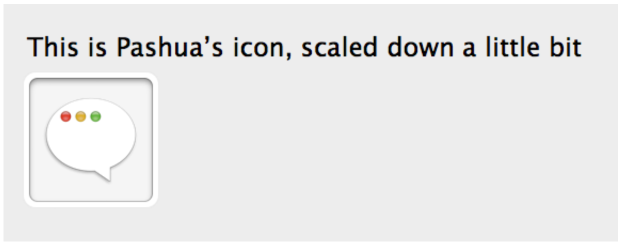
Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`path`	Filesystem path to the image	Yes	–
`border`	Set this to `1` to display a border for the image	No	0
`maxwidth`	If this attribute is set to some integer number, the image will be scaled down to this width (including border), if it's wider	No	–
`maxheight`	If this attribute is set to some integer number, the image will be scaled down to this height (including border), if it's higher	No	–
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
	Absolute vertical position in the window, measured from the		

`y`	*lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: None

Example: Using `image`

```
img.type = image
img.label = This is Pashua's icon, scaled down a little bit
img.path = /Applications/Pashua.app/Contents/Resources/AppIcon.icns
img.maxwidth = 64
img.border = 1
```


This is Pashua's icon, scaled down a little bit

Element type: `openbrowser`

An `openbrowser` is used for choosing a filesystem path. It consists of a textfield, a button and (optionally) a label. The textfield holds the actual element value (the file path), while the button (which is localized) is used to invoke a file selector sheet. Moreover, a file can be dragged & dropped onto the textfield.

Table: Attributes for elements of type `openbrowser`

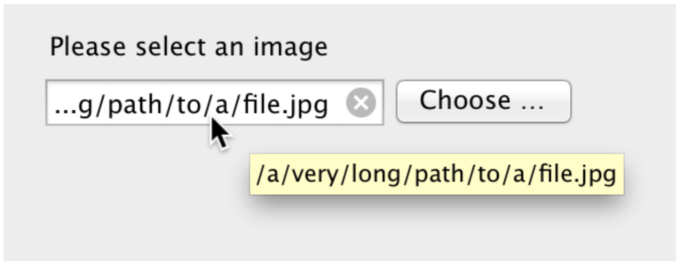
Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`width`	Sets the width (overall width of texfield and button)	No	300
`default`	Default path	No	–
`filetype`	File types that can be selected in the open dialog or dropped onto the textfield; takes a space-delimited list of filename extensions (such as `jpg gif tif` etc.). In addition to filename extensions, you can use `directory` to let the user choose directories. If only `directory` is specified, the user won't be able to choose any files. If only filename extensions are specified, the user won't be able to choose	No	–

	directories. If you don't specify filetype at all, the user will be able to choose anything in the open dialog box.		
`mandatory`	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
`placeholder`	If present, this string will be as the field's placeholder string.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: Full filesystem path for the selected item (may be an empty string)

Example: Using `openbrowser`

```
opb.type = openbrowser
opb.label = Please select an image
opb.default = /a/very/long/path/to/a/file.jpg
opb.filetype = jpg tiff tif gif png psd
opb.width = 250
```



Element type: `password`

This element is identical to a `textfield`, except that it hides whatever is typed into it. Moreover, you can't copy or drag text from a `password` element.

Table: Attributes for elements of type `password`

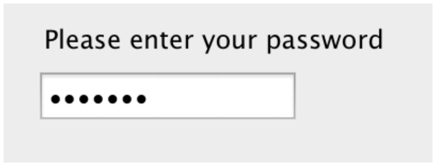
Name	Purpose	Required	Default
`label`	Creates a label/title above this element	No	–
`width`	The textfield's width in pixels	No	200

`default`	The textfield's initial contents	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`mandatory`	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: String contents (may be an empty string)

Example: Using `password`

```
pw.type = password
pw.label = Please enter your password
pw.default = Secret!
pw.width = 120
```



Element type: popup

A popup is an element that lets the user choose one value from a list of possible values

Table: Attributes for elements of type popup

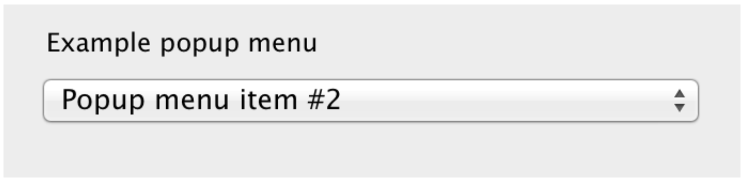
Name	Purpose	Required	Default
`option`	Any string that should appear as an entry in the popup, can (and probably should) be used more than once	Yes (one option is required, others are (sic!) optional)	–
`default`	Default value (should match one of the	No	–

	optionattributes		
label	Creates a label above this element	No	–
disabled	If set to 1, the element will be disabled, so that the default value cannot be changed.	No	0
tooltip	String to use as tooltip for the button. Use \n to insert a linebreak.	No	–
mandatory	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
width	Width in pixels	No	200
x	Absolute horizontal position in the window, measured from the left border of the content area	No	–
y	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
relx	Horizontal offset, relative to the position the element would have if relx was not used (e.g.: relx specifies the distance from the left window border). Any integer can be used as relx value.	No	0
rely	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as rely value.	No	0

Return value: Selected item as string (may be an empty string)

Example: Using popup

```
p.type = popup
p.label = Example popup menu
p.width = 310
p.option = Popup menu item #1
p.option = Popup menu item #2
p.option = Popup menu item #3
p.default = Popup menu item #2
```



Element type: radiobutton

A radiobutton element lets the user choose a value from a pre-defined list of values.

Table: Attributes for elements of type radiobutton

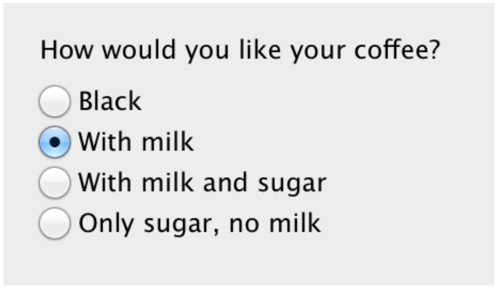
Name	Purpose	Required	Default

`label`	Creates a label above this element	No	–
`option`	Any string that should be used as a selectable value. Should be used more than once.	Yes (at least one `option` is required)	–
`default`	The value that should be selected by default. Of course, this must be one of the `option` values to work.	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`mandatory`	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: Selected radiobutton's label as string (may be an empty string)

Example: Using `radiobutton`

```
radio.type = radiobutton
radio.label = How would you like your coffee?
radio.option = Black
radio.option = With milk
radio.option = With milk and sugar
radio.option = Only sugar, no milk
radio.default = With milk
```


How would you like your coffee?

Black
[✓ With milk]
With milk and sugar
Only sugar, no milk

Element type: `savebrowser`

A `savebrowser` can be used for setting a path and name for a new file. It consists of a textfield, a button and (optionally) a label. The textfield holds the actual element value (the file path and name), while the button (which is localized) is used to invoke a file selector sheet.

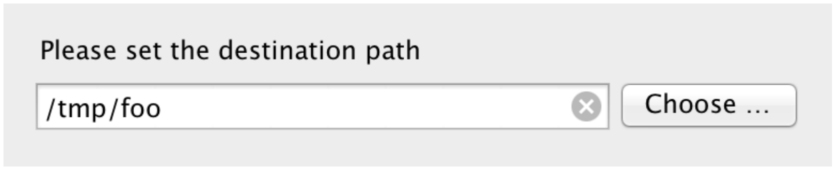
Table: Attributes for elements of type `savebrowser`

Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`width`	Sets the width (overall width of texfield and button)	No	300
`default`	Default path	No	–
`filetype`	File extension to use for the save dialog box; if this attribute is used, the user will be forced to use that extension for the name	No	–
`mandatory`	If set to a true value (everything other than 0, “n”, “no”, empty), input is mandatory.	No	No
`placeholder`	If present, this string will be as the field’s placeholder string.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below (“relative” means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: Full filesystem path (may be an empty string, if no path chosen by user)

Example: Using `savebrowser`

```
svb.type = savebrowser
svb.label = Please set the destination path
svb.default = /tmp/foo
svb.filetype = jpg
svb.width = 360
```



Element type: `text`

This element displays multi-line, wrapping text. The width of the element does *not* adapt automatically to the content, but uses the given width (or the default width, if no width is specified.) On the other hand, the height is automatically set to the minimum height needed to display the text using the given (or default) width.

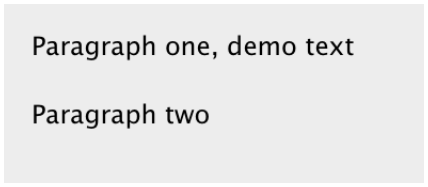
Table: Attributes for elements of type `text`

Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`width`	Sets the width of the text (in pixels)	No	280
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`default`	The text to display (alias for `text`). You can use the string `[return]` to insert a linebreak.	Yes (either `default` or `text` must be set)	–
`text`	The text to display (synonym for `default`). You can use the string `[return]` to insert a linebreak.	Yes (either `default` or `text` must be set)	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: None

Example: Using `text`

```
txt.type = text
txt.default = Paragraph one, demo text[return][return]Paragraph two
```



Paragraph one, demo text

Paragraph two

Element type: `textbox`

A textbox is a scrollable, multi-line text container. The scrollbar will appear automatically if needed. *Note:* If you have changed the system's scrollbar behaviour to display both arrows at both ends (e.g. using TinkerTool), the scrollbar might not appear for small heights.

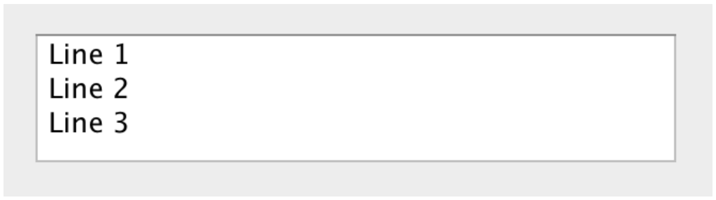
Table: Attributes for elements of type `textbox`

Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`width`	Sets the width of the textbox (in pixels)	No	250
`height`	Sets the height of the text (in pixels)	No	52
`default`	Sets the initial contents. You can use the string `[return]` to insert a linebreak.	No	–
`fontsize`	Size of the text inside the textbox. There are three sizes available, which conform to the system's standard sizes: `regular`, `small`, or `mini`.	No	`regular`
`fonttype`	Set this to `fixed` if the text should be displayed using a monospace font.	No	-
`mandatory`	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: String contents (may be an empty string)

Example: Using `textbox`

```
tb.type = textbox
tb.default = Line 1[return]Line 2[return]Line 3
tb.width = 300
tb.height = 60
```



```
Line 1
Line 2
Line 3
```

Element type: `textfield`

A textfield is a simple, one-line text input field with an optional label displayed above the textfield.

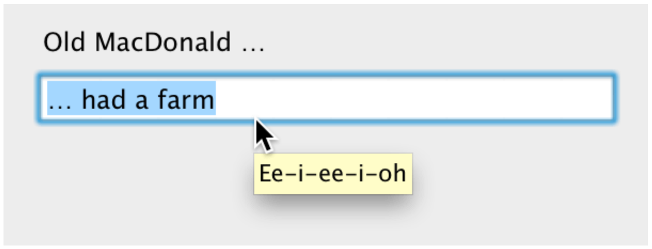
Table: Attributes for elements of type `textfield`

Name	Purpose	Required	Default
`label`	Creates a label above this element	No	–
`width`	The textfield's width in pixels	No	200
`default`	The textfield's initial contents	No	–
`disabled`	If set to `1`, the element will be disabled, so that the default value cannot be changed.	No	`0`
`mandatory`	If set to a true value (everything other than 0, "n", "no", empty), input is mandatory.	No	No
`placeholder`	If present, this string will be as the field's placeholder string.	No	–
`tooltip`	String to use as tooltip for the button. Use `\n` to insert a linebreak.	No	–
`x`	Absolute horizontal position in the window, measured from the left border of the content area	No	–
`y`	Absolute vertical position in the window, measured from the *lower* border of the content area	No	–
`relx`	Horizontal offset, relative to the position the element would have if `relx` was not used (e.g.: `relx` specifies the distance from the left window border). Any integer can be used as `relx` value.	No	`0`
`rely`	Relative vertical distance to the next element below ("relative" means that the value is added to the default distance). Any integer larger than -20 can be used as `rely` value.	No	`0`

Return value: String contents (may be an empty string)

Example: Using `textfield`

```
txf.type = textfield
txf.label = Old MacDonald …
txf.default = … had a farm
txf.width = 270
txf.tooltip = Ee-i-ee-i-oh
```



Old MacDonald …

… had a farm

Ee-i-ee-i-oh

Topics

Unicode and text encodings

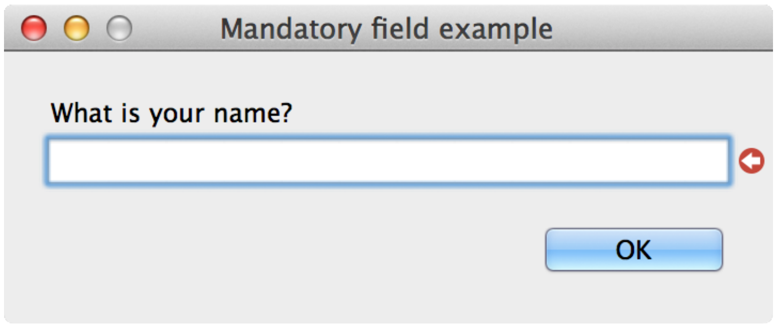
For some time, Pashua had offered specifying the text encoding of the configuration string, as there were languages which had limited or no support for UTF-8. Today, UTF-8 is ubiquitous, so this option was removed. Pashua will auto-detect the encoding of a configuration file and return the values in the same encoding. This implies that command-line option `-e`, which was used to set the encoding, is now completely ignored.

Note: Guessing the encoding only works with configuration files. If you pass the configuration via STDIN, Pashua will expect it to be UTF-8.

Mandatory input

Some of the element types (those for which it makes sense) can be defined as mandatory. This means that the dialog cannot be closed as long as the element is empty. Here, "empty" means that the element does not have a string value or that the string value contains only whitespace; the digit zero (0) is not regarded as empty. Radio buttons are regarded as empty as long as none of the buttons has been clicked.

This screenshot shows visual marker which is displayed next to a mandatory element when the "OK" button is clicked:



Localization

Pashua is localized in German, French and English. The localizations are used for the menu bar, the application about box, button titles (defaultbutton, cancelbutton, savebrowser and openbrowser buttons) and of course for error messages. It's your application's responsibility to provide localization by passing an appropriate configuration string to Pashua for anything else that should be localized.

Dynamic configuration strings

Obviously, configuration strings are nothing but strings, and you can use them in any way you can use a string variable in the programming language of your choice. If you like, you can get field values from an Oracle database with Perl, fetch them from a Web Service with AppleScript, extract it from some textfile on your disk, combine them, split them, replace characters in them or anything else you can think of.

To make a long story short: *Pashua does not care how the configuration string was created—it only has to fulfill the syntax rules.*

The static strings I wrote for the various example files are nothing but the most simple usage example. For instance, you could build a configuration string for Perl this way:

```
my $conf="user.type=textfield
user.label=Field label
user.width=160";
```

... or with Heredoc syntax:

```
my $conf = <<EOCONF;
user.type = textfield
user.label = Field label
user.width = 160
EOCONF
```

... or this way:

```
my $conf;
my %hash = ( 'user',
    {
        'type'=>'textfield',
        'label'=>'Field label',
        'width'=>160
    }
);

while (my($element, $specs) = each(%hash)) {
    my %attributes = %$specs;
    foreach (sort keys %attributes) {
        $conf .= "${element}.$_ = $attributes{$_}\n";
    }
}
```

The result would be *exactly the same* in any of these cases.

If you take a look at the example scripts, you will see that they make use of dynamic configuration strings: Pashua's icon is added to the example dialog, and this is done by locating the path to Pashua.app at runtime and deriving the icon's path from that location.

Building "real" applications with Pashua

Native OS X applications are nothing like folders with a name that ends with ".app" (invisible in the Finder by default) and that have a specific directory structure in them. You can examine such a folder (called an "application bundle") by context-clicking on the name and choosing the appropriate option from the contextual menu (in English it is "Show Package Contents", in

German it is “Paketinhalt zeigen”).

As a starting point, an example for a doubleclickable, “real” application with a Pashua GUI is provided. When you play around with the application bundle, there are a few things to keep in mind:

- The script that is the “engine” inside the application bundle must have the same name as the bundle itself (without the “.app” extension). Example: If your application should have the name “Hello world”, then the script in `Hello world.app/Contents/MacOS/` must be named “Hello world”.
- The script must have the executable bit set, i.e.: it must be executable at the shell. If it is not yet, do a `chmod +x /path/to/the/script`
- Modify file `Contents/Info.plist` inside the application bundle to match your needs. You will at least have to change the “Doubleclickable Example” string to your applications’name (in the example above, it would have to be “Hello world”). Moreover, you should set the `CFBundleIdentifier` value (currently “com.example.pashua-doubleclickable-example”) to an appropriate value.
- Modify the application author and copyright info in `Contents/Resources/English.lproj/InfoPlist.strings` inside the application bundle.
- When using a Perl or Python script, you should put the corresponding module inside the same folder (`*.app/Contents/MacOS/`) as the script.
- I haven’t managed to get AppleScript to work inside application bundles. You can use a wrapper shell script containing something like

  ```
  #!/bin/sh
  scriptpath=`dirname "$0"`
  osascript "$scriptpath/my_applescript.app"
  ```

  …, but though this will work, the application won’t finish launching (the Dock icon won’t stop hopping).

FAQ

How do I display 2 or more windows one after another?

You can call Pashua multiple times, but this will result in successive application launches (with the icon moving in and out of the Dock), which is annoying. Hiding the Dock icon helps a lot in this case, but it's only a remedy – not a real solution.

When will Pashua support multiple checkboxes/radiobuttons…?

You can use *any* GUI element as often as you like; the only exceptions are the cancelbutton and the defaultbutton. You only have to make sure that the names are unique:

```
chk1.type = checkbox
chk1.label = Checkbox 1

chk2.type = checkbox
chk2.label = A second checkbox

chk3.type = checkbox
chk3.label = Checkbox #3
```

May I distribute applications based on Pashua?

You are allowed to re-distribute Pashua (as an external application or by shipping a modified Pashua application bundle similar to the "Doubleclickable Example") as long as your application is freeware or Open Source Software and, additionally, is not merely a free "add-on" to some commercial service you offer.

Of course, I do not give any guarantee that Pashua works as expected and refuse responsibility for anything that might be caused by an application which makes use of Pashua.

How can I hide the dock icon?

Most Cocoa applications can be modified so that the Dock icon and the application-specific menu bar are hidden—and Pashua is no exception to this rule. You can achieve this by opening the file `Info.plist` inside Pashua's application bundle and changing the line below the one containing `LSUIElement` from `<false/>` to `<true/>`. If you don't notice any change in Pashua's behaviour, you should log out and back in.

What does "Pashua" mean?

I always find it difficult to find names for applications I write, and Pashua was no exeception. As usual, at some point I started to play around with acronyms from "Perl", "and", "shell" (actually, at the beginning I only thought of Perl and shell scripts), "GUI" / "UI" came up with "Pashua", which turned out to be ambiguous enough to interpret it as " *P*erl *A*nd *SH*ell *U*i- *A*ddon" or "*P*ython *A*nd *SH*ell *U*i-*A*ddon" or "*P*hp *A*pplescript *S*imple, *H*omemade *U*ser *i*nterf*A*ce" or whatever ;-)

Version information

Pashua 0.10.3 was released on 01/28/2016.

For information on changes and the complete version history, please take a look at the website.