Framework Specification for Macintosh Allegro CL

The framework provides an easy-to-use object-oriented Lisp interface to the Macintosh Carbon library, used to create GUI applications for the Macintosh. The primary goal of this design is to make common things easy while still allowing full access to the underlying Carbon technology for those parts of application with less common needs.

Basic views

The basic framework object is a view. Views (windows, menus, controls) are represented by CLOS objects, and are created using `make-instance`.

`view` `() [class]`

An abstract superclass of all views (windows, menus, controls). All views support the following initargs:

- `:view-nick-name` – initializes the view nick-name, see `set-view-nick-name`
- `:view-subviews` – initializes the view subviews, see `add-container`
- `:view-size` – initializes the view size, see `set-view-size`
- `:view-width` – initializes the view `width` (overrides `:view-size`)
- `:view-height` – initializes the view height (overrides `:view-size`)
- `:view-tooltip` – initializes the view tooltip, see `set-view-tooltip`
- `:view-font` – initializes the view font, see `set-view-font`.
- `:view-enabled-p` – (default true), initializes the enabled/disabled state, see `enable-view/disable-view`
- `:view-carbon-ref` – specifies an existing Carbon ControlRef/HIViewRef/MenuRef/WindowRef to base the view on.
- `:view-command-id` – initializes the view command id, see `set-view-command-id`.

`(`**`view-p`**` object) [function]`

True if `object` is a view.

`(`**`set-view-nick-name`**` view name) [method]`

Give the `view` a nick-name `name`, which can be any object.

`(`**`view-nick-name`**` view) [method]`

Returns the `view`'s nick-name.

`(`**`view-named`**` name view) [method]`

Finds a subview of `view` whose nick-name is `eql` to `name`.

`(`**`view-subviews`**` view &key type) [method]`

Returns all subviews of `view`, i.e. all views whose `view-container` is `view`, optionally restricted to those of type `type`.

`(`**`add-subviews`**` view &rest subviews) [method]`

Make `view` be the container for each of the `subviews`.

`(`**`remove-subviews`**` view &rest subviews) [method]`

Remove each of the `subviews` from `view`.

`(`**`set-view-width`**` view new-width) [method]`

Sets the width of view to `new-width`, which must be a non-negative `real`.

`(`**`view-width`**` view) [method]`

Returns the width of `view`.

`(`**`set-view-height`**` view new-height) [method]`
Sets the size of view to `new-height`, which must be a non-negative `real`.

`(`**`view-height`**` view) [method]`
Returns the height of `view`.

`(`**`set-view-size`**` view new-size) [method]`
Sets the size of view to `new-size`, which is the height and width encoded as a `point`.

`(`**`view-size`**` view) [method]`
Returns the size (height and width) of `view` as a `point`.

`(`**`set-view-tooltip`**` view tooltip) [method]`
Sets the tooltip of `view` to `tooltip`, which must be a string or `nil` to remove the view's tooltip. The tooltip is shown when the mouse hovers over the view.

`(`**`view-tooltip`**` view) [method]`
Returns the tooltip of `view`.

`(`**`set-view-font`**` view font-spec) [method]`
Sets the font of `view` to `font-spec`, overriding the view's default font (system or window font). Once you have set a view's font with this function, you can cause the view to revert to its default font by passing `nil` as the font-spec.

`(`**`view-font`**` view) [method]`
Returns the font of `view`.

`(`**`enable-view`**` view) [method]`
Makes view enabled, so it will respond to user input when active.

`(`**`disable-view`**` view) [method]`
Makes view disabled, so it will be drawn grayed out and will not respond to user input.

`(`**`view-enabled-p`**` view) [method]`
returns true if view is enabled.

`(`**`set-view-command-id`**` view command-id) [method]`
Sets the Carbon command id for the view. `command-id` may be an ostype or it may be a keyword defined by `define-carbon-command-event`, q.v.

`(`**`view-command-id`**` view) [method]`
Returns the command id of the view, if any, as an ostype, or `nil` if the view doesn't have a command id associated with it.

`(`**`view-carbon-ref`**` view) [method]`
Returns the ControlRef/HIViewRef/MenuRef/WindowRef of the Carbon object represented by `view`, to allow you to use Carbon directly.

`(`**`view-active-p`**` view) [method]`
True if view is active (able to accept user input).

`(`**`invalidate-view`**` view &optional rect) [method]`

Indicates that the contents of the view has become invalid (typically as a result of changes in the data displayed in the view). The system will arrange for the view to be redrawn. Optionally a rectangle may be specified to indicate that only the portion of the view bound by the rectangle needs to be redrawn.

View mixins

View text mixin

`view-text-mixin` `() [class]`

A mixin for views that display text. Initargs:

- `:view-text` – initializes the view text, see `set-view-text`

`(`**`set-view-text`**` view-text-mixin text) [method]`

Sets the text to display in `view-text-mixin` to `text`, which must be a string or `nil` to remove the view's text.

`(`**`view-text`**` view-text-mixin) [method]`

Returns the text of `view`.

View orientation mixin

`view-orientation-mixin` `() [class]`

A mixin for one-dimensional views that can be oriented horizontally or vertically. Initargs:

- `:view-orientation` – sets the view orientation to one of `:horizontal` or `:vertical`.

`(`**`view-orientation`**` view-orientation-mixin) [method]`

Returns the orientation of `view-orientation-mixin`.

View range mixin

`view-range-mixin` `() [class]`

A mixin for views that contain a numerical value in a range, such as a scroll bar or a progress bar. Initargs:

- `:view-value` – initializes the view value, see `set-view-value`
- `:view-minimum-value` – initializes the view minimum value, see `set-view-minimum-value`
- `:view-maximum-value` – initializes the view maximum value, see `set-view-maximum-value`

`(`**`set-view-value`**` view-range-mixin value) [method]`

Sets current value to `value`, which must be a `real`.

`(`**`view-value`**` view-range-mixin) [method]`

Returns the current value.

`(`**`set-view-minimum-value`**` view-range-mixin minimum) [method]`

Sets the minimum value to `minimum`, which must be a `real`.

`(`**`view-minimum-value`**` view-range-mixin) [method]`

Returns the minimum value.

`(`**`set-view-maximum-value`**` view-range-mixin maximum) [method]`

Sets the maximum value to `maximum`, which must be a `real`.

`(`**`view-maximum-value`**` view-range-mixin) [method]`

Returns the maximum value.

View state mixin

`view-state-mixin` `() [class]`

A mixin for views that have one of finite number of states, such as a checkbox. Initargs:

`:view-state` – initializes the view state, see `set-view-state`

`(`**`view-valid-states`**` view-state-mixin) [method]`

Returns the sequence of valid states for the view. Concrete subclasses must provide a method for this function. The default method returns `nil`.

`(`**`set-view-state`**` view-state-mixin state) [method]`

Sets current state to `state`, which must be `eql` to an element of `view-valid-states`.

`(`**`view-state`**` view-state-mixin) [method]`

Returns the current state.

View action mixin

`view-action-mixin` `() [class]`

A mixin for views that invoke a single action when activated, such as a push button. Initargs:

`:view-action` – initializes the view action, see `set-view-action`

`(`**`set-view-action`**` view-action-mixin action) [method]`

Sets the action of `view-action-mixin` to `action`, which must be either `nil` or a funcallable object (`function` or `symbol`). If not `nil`, the action will be called with one argument, the view itself.

`(`**`view-action`**` view-action-mixin) [method]`

Returns the action of `view-action-mixin`.

View mouse tracker mixin

`view-mouse-tracker-mixin` `() [class]`

A mixin for views that support mouse tracking, used to implement live update. Initargs:

`:view-mouse-tracker` – initializes the view mouse tracker, see `set-view-mouse-tracker`

`(`**`set-view-mouse-tracker`**` view-mouse-tracker-mixin tracker) [method]`

Sets the mouse tracking handler for the view `view-mouse-tracker-mixin` to `tracker`, which should be either `nil` or a funcallable object (function or symbol). If not `nil`, `tracker` will be called when the user first clicks in the view and then repeatedly for as long as the user holds down the mouse button. The function will receive two arguments, the view and the part code for the part of the view where the click first occurred, unless the mouse has moved out of the initial part, in which case the second argument will be `nil`.

`(`**`view-mouse-tracker`**` view-mouse-tracker-mixin tracker) [method]`

Returns the mouse tracker of `view-mouse-tracker-mixin`.

View nib mixin

`view-nib-mixin` `() [class]`

A mixin for views that can be loaded from nib files, currently windows and menus. Initargs:

`:view-nib-name` – specifies the name of the view description in a nib file.

`:view-nib-path` – pathname of the nib file containing `view-nib-name`. Default is `"main".` The pathname type defaults to `"nib"` and in fact cannot be anything else.

See Interface Builder Support section below for more information.

Subviews

Subview

`subview` `(view) [class]`

Class of views that appear inside other views inside a window. Additional initargs:

- `:view-container` – initializes the view container, see `set-view-container`
- `:view-position` – initializes the view position, see `set-view-position`

`(`**`set-view-container`** `subview new-container-view) [method]`

Remove `view` from its current container and add it to `new-container-view`. The new container can also be `nil`, to just remove `view` from its container view.

`(`**`view-container`** `subview) [method]`

Returns the view containing `view`, or `nil` if none.

`(`**`view-window`** `subview) [method]`

Returns the window containing `view`, or `nil` if `view` is not contained in any window.

`(`**`set-view-position`** `subview new-position) [method]`

Sets the position of view in its container to `new-position`, which must be a `point`.

`(`**`view-position`** `subview) [method]`

Returns the position of `view` in its container, as a `point`.

Controls

Controls, sometimes known as dialog items or widgets, are subviews that manage redisplay and user interactions in stylized ways. The following controls are predefined:

Separator line view

`separator-line-view` `(subview view-orientation-mixin) [class]`

A line (horizontal or vertical) that can be used to visually separate portions of a dialog. When specifying the view size, one of width or height should be 1 (which is the default, so this can be achieved by only specifying one of width or height).

Progress bar view

`progress-bar-view` `(subview view-orientation-mixin view-range-mixin) [class]`

Shows progress towards completion of a lengthy operation. A progress bar has a `view-minimum`, `view-maximum`, and the current `view-value`, which should be programatically incremented from the minimum to the maximum. As a special case, `view-value` may be `nil` to indicate that the total length of any operation is not known, in which case the minimum and maximum are ignored and the progress bar displays as a spinning spiral cylinder that indicates that something is happening but gives no indication how soon the operation will complete.

Slider view

`slider-view` `(subview view-orientation-mixin view-range-mixin view-mouse-tracker-mixin) [class]`

Lets the user make a selection along a continuous range of allowable values. A slider has a `view-minimum`, `view-maximum`, and the current `view-value`, which ranges between the minimum and the maximum in response to user moving the slider. If the slider has a non-`nil` `view-mouse-tracker`, it will be called as the indicator is dragged, with the `view-value` updated appropriately. Additional initargs:

`:slider-indicator` – sets the shape of the slider indicator, one of `:right`, `:left`, `:up`, `:down`, `:none`
`:slider-tick-count` – sets the number of tick marks to draw on the slider bar, an integer value between `0` and `#xFFFF`, or `nil` (the default) for no tick marks. If non-`nil`, the size of the view must be large enough to accommodate the tick marks.

`(`**`slider-indicator`**` slider-view) [method]`

Returns the shape of the slider indicator, one of `:right`, `:left`, `:up`, `:down` for an indicator that points in the corresponding direction, or `:none` for a round indicator.

`(`**`slider-tick-marks`**` slider-view) [method]`

Returns the number of tick marks in `slider-view`, on `nil` if the slider doesn't have tick marks.

Scroll bar view

`scroll-bar-view` `(subview view-orientation-mixin view-range-mixin view-mouse-tracker-mixin) [class]`

Scroll bars are used to show parts of a document. A scroll bar has a `view-minimum`, `view-maximum` and the current `view-value` which ranges between the minimum and the maximum in response to the user moving the scroller (the movable oblong part), and represents the relative location, in the whole document, of the portion that can be seen. The size of the scroller is proportional to how much of the document is visible. If the slider has a non-`nil` `view-mouse-tracker`, it will be called as the scroller is dragged, with the `view-value` updated appropriately. Additional initargs:
`:scroll-bar-scroller-size` – initializes the scroller size, see `set-scroll-bar-scroller-size`.

`(`**`set-scroll-bar-scroller-size`**` scroll-bar-view size) [method]`

Sets the size of the currently visible part of the document, expressed in terms of the same units as used for minimim and maximum. This determines the size of the scroller.

`(`**`scroll-bar-scroller-size`**` scroll-bar-view) [method]`

Returns the size of the currently visible part of the document, expressed in terms of the same units as used for minimim and maximum.

Text input view

`text-input-view` `(subview view-text-mixin) [class]`

Lets the user enter text. Handles unicode and draws its text using anti-aliasing. Additional initargs:
`:text-input-locked` – initializes the text locked state, see `lock-text-input`
`:text-input-multi-line-p` – if true, allows multi-line input.
`:text-input-selection-start` – initializes the current selection start offset, see `set-text-input-selection`
`:text-input-selection-end` – initializes the current selection end offset, see `set-text-input-selection`

`(`**`lock-text-input`**` editable-text-view) [method]`

Locks the text in the view so it can't be modified (even though it's still shown as enabled).

`(`**`unlock-text-input`**` editable-text-view) [method]`

Unlocks the text in the view so it can be modified again.

`(`**`text-input-locked-p`**` editable-text-view) [method]`

True if text in view is locked.

`(`**`text-input-multi-line-p`**` editable-text-view) [method]`

True if multi-line input is supported.

`(`**`text-input-selection`**` editable-text-view) [method]`

Returns two values, the start and end offset of the selection in the text.

(**`text-input-selection-start`** `editable-text-view) [method]`
Returns the start offset of the selection.

(**`text-input-selection-end`** `editable-text-view) [method]`
Returns the end offset of the selection

(**`set-text-input-selection`** `editable-text-view &key start end)`
Sets the selection to the range specified.

Password text input view

`password-input-view` `(text-input-view) [class]`
Subclass of `text-input-view` for entering passwords. Echos typin as bullets.

Static text view

`static-text-view` `(subview view-text-mixin) [class]`
Displays text. Additional initargs:
`:static-text-truncation` – initializes text truncation style, see `set-static-text-truncation`.(**`set-static-text-truncation`** `static-text-view truncation) [method]`
Sets the truncation style for `static-text-view` to one of `:end`, `:middle` or `nil`. If `nil` (the default), text wraps to multiple lines instead of truncating, otherwise it's drawn on one line, truncating at the end or in the middle.
(**`static-text-truncation`** `static-text-view) [method]`
Returns the truncation style of `static-text-view`, one of `:end`, `:middle` or `nil`.

List box view

`list-box-view` `(subview) [abstract class]`
This is an abstract class of views based on the Carbon ListBox control, a scrollable view that displays data in rows and columns. It provides all the behavior except storing the data to display. To use it, you should subclass it and define a specialized method on the `list-box-cell-text` generic function. Additional initargs:
- `:list-box-dimensions` – initializes the dimensions, see `set-list-box-dimensions`.
- `:list-box-selection-type` – the type of selection handling to use, one of `:single`, `:contiguous`, or `:disjoint`.
- `:list-box-horizontal-scroll-p` – determines whether to have a horizontal scroll bar
- `:list-box-vertical-scroll-p` – determines whether to have a vertical scroll bar.
- `:list-box-cell-size` – initializes cell seize, see `set-list-box-cell-size`.
- `:list-box-cell-width` – width of each cell, overrides `list-box-cell-size`
- `:list-box-cell-height` – height of each cell, overrides `list-box-cell-size`
- `:list-box-grow-space-p` – true to indicate that the conrol is draw so that there is room for a size box.

(**`list-box-cell-text`** `list-box-view cell) [method]`
Returns the text contents of the cell in column `(point-h cell)` and row `(point-v cell)`. The default method returns `nil`. Subclasses should provide a particular implementation.

(**`set-list-box-dimensions`** `list-box-view dimensions) [method]`
Sets the number of columns in `list-box-view` to `(point-h dimensions)` and the number of rows to `(point-v dimensions)`.

(**`list-box-dimensions`** `list-box-view) [method]`
Returns the dimensions of `list-box-view` as `(point` column-count row-count`)`.

`(`**`set-list-box-visible-dimensions`**` list-box-view dimensions) [method]`

Resizes `list-box-view` so that `(point-h dimensions)` columns and `(point-v dimensions)` rows are visible.

`(`**`list-box-visible-dimensions`**` list-box-view) [method]`

Returns the number of cells visible in the horizontal and vertical dimensions.

`(`**`set-list-box-cell-size`**` list-box-view size) [method]`

Sets the size of a cell in `list-box-view`. All cells have the same size. Changing the cell size changes the `list-box-visible-dimensions`.

`(`**`list-box-cell-size`**` list-box-view) [method]`

Returns the size of a cell in `list-box-view`. All cells have the same size.

`(`**`list-box-selection-type`**` list-box-view) [method]`

Returns the selection type of `list-box-view`, one of `:single`, `:contiguous`, or `:disjoint.`

`(`**`list-box-horizontal-scroll-p`**` list-box-view) [method]`

Returns true if `list-box-view` has a horizontal scroll bar.

`(`**`list-box-vertical-scroll-p`**` list-box-view) [method]`

Returns true if `list-box-view` has a vertical scroll bar.

`(`**`list-box-scroll-to-cell`**` list-box-view cell) [method]`

Scrolls `list-box-view` so that the cell in column `(point-h cell)` and row `(point-v cell)` is in the upper-left corner.

`(`**`list-box-scroll-position`**` list-box-view) [method]`

Returns the cell in the upper-left corner, represented as a point.

`(`**`list-box-cell-select`**` list-box-view cell) [method]`

Selects the cell in column `(point-h cell)` and row `(point-v cell).`

`(`**`list-box-cell-deselect`**` list-box-view cell) [method]`

Deselect the cell in column `(point-h cell)` and row `(point-v cell).`

`(`**`list-box-cell-selected-p`**` list-box-view cell) [method]`

True if the cell in column `(point-h cell)` and row `(point-v cell)` is selected.

`(`**`list-box-selected-cells`**` list-box-view) [method]`

Returns a list of all currently selected cells in `list-box-view`. Each cell is represented by a point.

Array list box view

`array-view`` (list-box-view) [class]`

A type of list box view that displays the elements of a two dimensional array. Additional initargs:

`:list-box-array` – initializes the view array, see `set-list-box-array`.

`(`**`set-list-box-array`**` array-view array) [method]`

Sets the array displayed in `array-view` to `array`, a 2-dimensional array. It should not be directly modified while being used by an array view.

`(`**`list-box-array`**` array-view) [method]`

Returns the array displayed in `array-view`. It should not be directly modified while being used by an array view.

`(`**`list-box-cell-text`**` array-view cell) [method]`

The default `array-view` method obtains the array element corresponding to `cell` and converts it to a string using `princ-to-string`.

Sequence list box view

`sequence-view`` (list-box-view view-orientation-mixin) [class]`

A type of list box view that displays the elements of a sequence, horizontally or vertically. Additional initargs:

- `:list-box-sequence` – initializes the view sequence, see `set-list-box-sequence`.

`(`**`set-list-box-sequence`**` sequence-view sequence) [method]`

Sets the sequence displayed in `sequence-view` to `sequence`. `sequence` can be any Common Lisp sequence type, i.e. a `list` or a `vector`. It should not be directly modified while being used by a sequence view.

`(`**`list-box-sequence`**` sequence-view) [method]`

Returns the sequence displayed in `sequence-view`. It should not be directly modified while being used by a sequence view.

`(`**`list-box-cell-text`**` sequence-view cell) [method]`

The default `sequence-view` method obtains the sequence element corresponding to `cell` and converts it to a string using `princ-to-string`.

Push button view

`push-button-view`` (subview view-text-mixin view-action-mixin) [class]`

Push buttons are rounded rectangles that display the `view-text`, and when pressed invoke the `view-action`. Additional initargs:

- `:default-button-p` – if true, this is the default button. Default buttons are drawn with an outline and may be activated by hitting Return.
- `:cancel-button-p` – if true, this is the cancel button. This has no visible representation, but does cause the button to play the cancel button theme sound instead of the regular pushbutton theme sound when pressed.

`(`**`default-button-p`**` push-button-view) [method]`

True if this is the default button.

`(`**`cancel-button-p`**` push-button-view) [method]`

True if this is the cancel button.

Radio button view

`radio-button-view`` (subview view-text-mixin view-state-mixin) [class]`

Radio buttons are small circles that contain a black dot when they are selected (pushed). The `view-text` is displayed next to the circle. Radio buttons have a `view-state` which may be `:on`, `:off`, or `:mixed`, indicating that the current setting contains a mix of on and off values. Radio buttons normally occur in groups, i.e. their `view-container` is a `radio-button-group-view`, and only one button in a group may be pushed at a time. If the button is not a subview of a `radio-button-group-view` , pressing it will toggle it on/off.

`(`**`view-valid-states`**` radio-button-view) [method]`

Returns `'(:on :off :mixed).`

Radio button group view

`radio-button-group-view`` (subview) [class]`

An invisible enclosure for radio buttons. The only `view-subviews` allowed are radio-button-view's. The radio buttons are automatically set to be mutually exclusive.

`(`**`selected-group-radio-button`**` radio-button-group-view) [method]`

Returns the currently selected radio button in the group.

Checkbox view

`checkbox-view` `(subview view-text-mixin view-state-mixin) [class]`

Checkboxes are small squares that toggle an X mark on and off when clicked. The `view-text` is displayed next to the square. Checkboxes have a `view-state` which may be `:on`, `:off`, or `:mixed`, indicating that the current setting contains a mix of on and off values.

`(`**`view-valid-states`**` checkbox-view) [method]`

Returns `'(:on :off :mixed)`, the valid values for `view-state`.

Custom view

`custom-view` `(subview) [class]`

Custom views allow you to implement the view's appearance and behavior. Initiargs:

- `:view-mouse-down-handler` – initializes the mouse-down handler, see `set-view-mouse-down-handler`.
- `:view-mouse-up-handler` – initializes the mouse-up handler, see `set-view-mouse-up-handler`.
- `:view-key-handler` – initializes the keyboard input handler, see `set-view-key-handler`.
- `:view-draw-handler` – initializes the handler for drawing the view, see `set-view-draw-handler`.

`(`**`set-view-mouse-down-handler`**` custom-view handler) [method]`

Sets the function to be called when the user clicks inside the value. `handler` should be a funcallable object (a `function` or `symbol`), and it will be called with two arguments, the view and a point indicating the mouse position within the view, in the view's coordinate system (i.e. relative to the top-left corner of the view)

`(`**`view-mouse-down-handler`**` custom-view) [method]`

Returns the view's mouse-down handler.

`(`**`set-view-mouse-up-handler`**` custom-view handler) [method]`

Sets the function to be called when the user releases the mouse. `handler` should be a funcallable object (a `function` or `symbol`), and it will be called with two arguments, the view and a point indicating the mouse position in the view's coordinate system (i.e. relative to the top-left corner of the view). Note that the mouse might be outside the view on mouse-up.

`(`**`view-mouse-up-handler`**` custom-view) [method]`

Returns the view's mouse-up handler.

`(`**`set-view-key-handler`**` custom-view handler) [method]`

Sets the function to be called when the types on the keyboard. `handler` should be a funcallable object (a `function` or `symbol`), and it will be called with two arguments, the view and the key that was entered.

`(`**`view-key-handler`**` custom-view) [method]`

Returns the view's key handler.

`(`**`set-view-draw-handler`**` custom-view handler) [method]`

Sets the function to be called when the view needs to be drawn. `handler` should be a funcallable object (a `function` or `symbol`), and it will be called with one two arguments, the view and a graphics context. See the Graphics section for information about drawing text and graphics in a view. Note that you should never call this function directly, it is only meant to be invoked in the callback context. To programmatically cause your view to be redrawn, call `invalidate-view`.

`(`**`view-draw-handler`**` custom-view) [method]`
Returns the view's draw handler.

Windows

Windows represent an area on the screen that allows the user to enter and view information. Windows are top-level views – they have no container.

Window

`window` `(view view-nib-mixin) [class]`

A class representing windows. Additional initargs:

- `:window-kind` – the window kind determines the appearance of a window. It should be one of the following keywords: `:alert`, `:modal`, `:floating`, `:document`, `:utility`, `:help`, `:sheet`, `:toolbar`, `:overlay`, `:sheet-alert`, `:drawer`. It can also be an integer, in which case it is assumed to be a valid Carbon WindowClass value. The default is `:document`.
- `:window-position` – initializes the window position, see `set-window-position`.
- `:window-position-reference` – the reference location for `window-position`, see optional argument to `set-window-position`.
- `:window-title` – initializes the window title, `see set-window-title`.
- `:window-hide` – leaves the window hidden, see `window-show`.
- `:window-close-p` – creates a window with a close button (true by default for standard windows).
- `:window-minimize-p` – creates a window with a minimize button (true by default for standard windows).
- `:window-zoom-p` – creates a window with a zoom button (true by default for standard windows).
- `:window-resize-p` – creates a window with a resize control (true by default for standard windows).

`(`**`window-kind`**` window) [method]`

Returns the kind of window as a keyword (or, in case of an unrecognizable kind, an integer Carbon WindowClass value).

`(`**`set-window-position`**` window position &optional reference) [method]`

Sets the position of `window`. `position` can be a `point`, representing a position in global coordinates in which the top left corner of the main screen is `0,0`, or it can be one of `:center`, `:cascade`, `:cascade-start`, `:alert`, in which case the window will be centered, cascaded, or put in alert position relative to `reference`. `reference` can be a window, or the keyword `:main-screen`, or a list `` `(:screen ``,some-window`)`. Finally, `position` can also be an integer, in which case it's assumed to be a valid Carbon WindowPositionMethod value.

`(`**`window-position`**` window position &optional reference) [method]`

Returns the position of window as a point in global coordinates in which the top left corner of the mian screen is `0,0`.

`(`**`set-window-title`**` window title) [method]`

Sets the title of `window` to `title`.

`(`**`window-title`**` window) [method]`

Returns the `window`'s title.

`(`**`window-hide`**` window) [method]`

Makes the window invisible (not shown on screen).

`(`**`window-show`**` window) [method]`

Makes the window visible and selects it.

`(`**`window-hidden-p`**` window) [method]`
True if window is hidden.

`(`**`window-close-p`**` window) [method]`
Returns true if the window was created with a close button.

`(`**`window-minimize-p`**` window) [method]`
Returns true if the window was created with a minimize button.

`(`**`window-zoom-p`**` window) [method]`
Returns true if the window was created with a zoom button.

`(`**`window-resize-p`**` window) [method]`
Returns true if the window was created with a resize control.

`(`**`window-close`**` window) [method]`
Close the window.

`(`**`window-select`**` window) [method]`
Brings window to the front, actives it, and shows it if it is hidden. The previously active window is deactivated.

`(`**`windows`**` &key class) [function]`
Returns all windows of class `class` (default `t`).

Menus

Menus allow the user to view or choose from a list of choices and commands. Each menu item has one or more menu items associated with it. A menu item may also be a submenu, sometimes called a hierarchical menu.

`menu-item`` (view view-action-mixin view-text-mixin) [class]`
An abstract class representing menu items. A menu item can have a `view-action` which is invoked when the item is selected, and a `view-text` which is the name of the menu item. Additional initargs:

- `:menu-item-updater` – intializes the function call to update the menu item, see `set-menu-item-updater`.
- `:menu-item-key` – initializes the menu items command key equivalent, see `set-menu-item-key`.
- `:menu-item-check` – initializes the check mark of the menu item, see `set-menu-item-check`.
- `:menu-item-style` – initializes the font style in which the item text appears, see `set-menu-item-style`.

`(`**`set-menu-item-updater`**` menu-item updater) [method]`
Sets the menu item's updater function to `updater`, which must be either `nil` or a funcallable object (`function` or `symbol`). If non-`nil`, it is invoked with one argument, the menu item, to allow it to be adjusted in the current program context (e.g. enabled/disabled) before it is displayed.

`(`**`menu-item-updater`**` menu-item updater) [method]`
Returns the menu item's updater function.

`(`**`set-menu-item-key`**` menu-item key) [method]`
Sets the keyboard equivalent for the menu item to `key` (which may be `nil` to remove the keyboard equivalent).

`(`**`menu-item-key`**` menu-item) [method]`

Returns the keyboard equivalent of the menu item.

`(`**`set-menu-item-check`**` menu-item check) [method]`

Sets the check mark character that appears next to the menu item text. If check is `nil`, no check mark appears. If check is `t`, then a standard check-mark symbol is used. Otherwise it should be a character to be used as the check mark.

`(`**`menu-item-check`**` menu-item) [method]`

Returns the check mark character that appears next to the menu item text, or `t` if the standard check-mark symbol is used, or `nil` if no check mark appears.

`(`**`set-menu-item-style`**` menu-item style) [method]`

Sets the style in which the item text appears, one of `:normal or :plain`, `:bold`, `:italic`, `:shadow`, `:outline`, `:underline`, `:condense`, and `:extend`.

`(`**`menu-item-style`**` menu-item) [method]`

Returns the style in which the item text appears, one of - `:normal`, `:bold`, `:italic`, `:shadow`, `:outline`, `:underline`, `:condense`, and `:extend`.

`separator-menu-item`` (menu-item) [class]`

A permanently disabled menu item consisting of a long line, used as a separator.

`menu`` (menu-item view-nib-mixin) [class]`

The class of menus. Additional initargs:

`:menu-items` – initializes the menu's menu items, see `set-menu-items`.

`(`**`set-menu-items`**` menu items) [method]`

Sets the menu items associated with `menu` to `items`, which must be a `sequence` of `menu-item`'s.

`(`**`menu-items`**` menu) [method]`

Returns a freshly consed list of `menu-item`'s associated with `menu`.

`(`**`set-menubar`**` menu) [method]`

Sets the root menu, also known as the menubar, to `menu`.

`(`**`menubar`**`) [method]`

Returns the `menu` object that is the current menubar.

Interface Builder Support

You can create a window or menu by specifying all of its properties directly, or you can use the Apple interface builder to create an object description and load it from a nib file. You specify the nib file with the `:view-nib-path` initarg and the particular description with the `:view-nib-name` initarg. Once the view is created from the nib file, any additional initargs specified in the `make-instance` are used to override the nib file settings.

When you load a view from a nib file, `subview` and `menu-item` objects are created automatically for subviews and menu items. After a subview or menu item is automatically created, the generic function `initialize-view-from-nib` is called to give you a chance to update its settings (such as event handlers).

`(`**`initialize-from-nib`**` window subview) [method]`

Called when a window is being created based on a template from a nib file. The default method does nothing.

(**initialize-from-nib** menu menu-item) [method]

Called when a menu is being created based on a template from a nib file. The default method does nothing.

Controls and menu items in nib files will typically have a command id. To associate a handler with the command, you have two options. One is to associate the handler with the item that generates the command, by using `set-view-action` inside the item's `initialize-view-from-nib`. The other option is to allow the comand to be sent to whichever view is the user focus at the time, which you do by **not** setting the item's `view-action`. The command will then be propagated to the outermost view that has a handler for it. The handler can be established for a view either when it is first created , by using the initarg associated with the command id by `define-carbon-command-event`, or at any time by calling `set-carbon-event-handler`.

Carbon Events

Carbon supports a vast number of events which can be used to customize view behavior. We provide an interface that allows you to specify Carbon event callbacks through initargs when creating a view.

(**define-carbon-event** event-keyword event-class event-kind) [macro]

This macro establishes `event-keyword` as the keyword for the Carbon event identified in Carbon by the two values `event-class` and `event-kind`. It allows `event-keyword` to be used as an initarg in creating views to specify a callback for the event (or equivalently the callback can be set later with `set-carbon-event-handler`). Example:

```
(define-carbon-event :click-grow-box-event kEventClassWindow kEventWindowClickResizeRgn)
(make-instance 'my-window :click-grow-box-event #'(lambda (window) …) …)
```

It is acceptable to define multiple event keywords for the same Carbon signature. Doing so allows specifying multiple handlers for the event (which will be executed in LIFO order).

(**define-carbon-command-event** event-keyword command-id) [macro]

As a special case, we provide a default handler for `kEventClassCommand/kEventProcessCommand` that does an extra level of dispatch on the command id. Therefore you can define an `event-keyword` for a specific command-id, and it will be treated just like any other kind of event, for example:

```
(define-carbon-command-event :check-spelling-command carbon::kHICommandCheckSpelling)
(make-instance 'my-view :check-spelling-command #'(lambda (view) …) …)
```

(**carbon-event-class** event-keyword) [function]

Returns the event class of `event-keyword`, or `nil` if `event-keyword` is not a defined carbon event.

(**carbon-event-kind** event-keyword) [function]

Returns the event kind of `event-keyword`, or `nil` if `event-keyword` is not a defined carbon event.

(**find-carbon-events** event-class event-kind) [function]

Returns a list of all the event keywords defined for the Carbon event `event-class/event-kind`.

(**set-carbon-event-handler** view event-spec handler) [method]

Sets `handler` as the handler for `event-spec` in `view`. Any previous handler for `event-spec` in `view` is removed. `event-spec` can be either an event keyword as defined by `define-carbon-event` or `define-carbon-command-event`, or it can be an ostype representing a command id. `handler` should be `nil` or a funcallable object (`function` or `symbol`). If non-`nil`, it will be called with one argument, the view. The handler can call `exit-event-handler` to exit. Exiting normally is equivalent to calling `(exit-event-handler t)`.

(**carbon-event-handler** view event-spec) [method]

Returns the handler registered for `event-spec` in `view`, or `nil` if `view` doesn't have a handler registered for `event-spec`.

(**exit-event-handler** &optional error) [method]

An error if called outside the context of an event handler. If called from a handler, does a non–local exit out of the handler. If `error` is `nil` or unspecified, the event is assumed to have been handled and no other handlers will be invoked. Otherwise, `error` should be either `t` (which is equivalent `carbon::eventNotHandledErr`), or a Carbon error code, and event handling will continue normally,

The following event keywords are predefined

```
(define-carbon-event :window-activated-event "wind" kEventWindowActivated)
(define-carbon-event :window-deactivated-event "wind" kEventWindowDeactivated)
(define-carbon-event :window-bounds-changed-event "wind" kEventWindowBoundsChanged)
(define-carbon-event :window-bounds-changing-event "wind" kEventWindowBoundsChanging)
(define-carbon-event :window-draw-content-event "wind" kEventWindowDrawContent)
(define-carbon-event :window-content-click-event "wind" kEventWindowContentClick)
(define-carbon-event :window-close-event "wind" kEventWindowClose)
(define-carbon-event :window-closed-event "wind" kEventWindowClosed)
(define-carbon-event :control-activate-event "cntl" kEventControlActivate)
(define-carbon-event :control-deactivate-event "cntl" kEventControlDeactivate)
(define-carbon-event :control-hit-event "cntl" kEventControlHit)
(define-carbon-event :control-track-event "cntl" kEventControlTrack)
(define-carbon-event :control-draw-event "cntl" kEventControlDraw)
(define-carbon-event :control-bounds-changed-event "cntl" kEventControlBoundsChanged)
(define-carbon-command-event :ok-command-event "ok  ")
(define-carbon-command-event :cancel-command-event "not!")
(define-carbon-command-event :quit-command-event "quit")
(define-carbon-command-event :undo-command-event "undo")
(define-carbon-command-event :redo-command-event "redo")
(define-carbon-command-event :cut-command-event "cut ")
(define-carbon-command-event :copy-command-event "copy")
(define-carbon-command-event :paste-command-event "past")
(define-carbon-command-event :clear-command-event "clea")
(define-carbon-command-event :select-all-command-event "sall")
(define-carbon-command-event :hide-command-event "hide")
(define-carbon-command-event :hide-others-command-event "hido")
(define-carbon-command-event :show-all-command-event "shal")
(define-carbon-command-event :preferences-command-event "pref")
(define-carbon-command-event :zoom-window-command-event "zoom")
(define-carbon-command-event :minimize-window-command-event "mini")
(define-carbon-command-event :minimize-all-command-event "mina")
(define-carbon-command-event :maximize-window-command-event "maxi")
(define-carbon-command-event :maximize-all-command-event "maxa")
(define-carbon-command-event :arrange-in-front-command-event "frnt")
(define-carbon-command-event :bring-all-to-front-command-event "bfrt")
(define-carbon-command-event :window-list-separator-command-event "wldv")
(define-carbon-command-event :window-list-terminator-command-event "wlst")
(define-carbon-command-event :select-window-command-event "swin")
(define-carbon-command-event :rotate-windows-forward-command-event "rotw")
(define-carbon-command-event :rotate-windows-backward-command-event "rotb")
(define-carbon-command-event :rotate-floating-windows-forward-command-event "rtfw")
(define-carbon-command-event :rotate-floating-windows-backward-command-event "rtfb")
(define-carbon-command-event :about-command-event "abou")
(define-carbon-command-event :new-command-event "new ")
(define-carbon-command-event :open-command-event "open")
(define-carbon-command-event :close-command-event "clos")
(define-carbon-command-event :save-command-event "save")
(define-carbon-command-event :save-as-command-event "svas")
(define-carbon-command-event :revert-command-event "rvrt")
(define-carbon-command-event :print-command-event "prnt")
(define-carbon-command-event :page-setup-command-event "page")
```

```
(define-carbon-command-event :app-help-command-event "ahlp")
(define-carbon-command-event :show-character-palette-command-event "chrp")
(define-carbon-command-event :show-spelling-panel-command-event "shsp")
(define-carbon-command-event :check-spelling-command-event "cksp")
(define-carbon-command-event :change-spelling-command-event "chsp")
(define-carbon-command-event :check-spelling-as-you-type-command-event "chsp")
(define-carbon-command-event :ignore-spelling-command-event "igsp")
(define-carbon-command-event :learn-word-command-event "lrwd")
```

Graphics

For drawing in custom views, you use the Macintosh Quartz 2D library directly. You receive a CGContext as the second argument to your `view-draw-handler` function. This is a native Carbon reference (i.e. not a CLOS object) suitable for passing to Quartz functions as a CGContextRef argument. FFI definitions will be provided for all the relevant Quartz 2D functions, types, and constants in the folowing Quartz header files:

```
    CGAffineTransform.h
    CGColor.h
    CGColorSpace.h
    CGContext.h
    CGDataConsumer.h
    CGDataProvider.h
    CGFont.h
    CGFunction.h
    CGGeometry.h
    CGImage.h
    CGImageDestination.h
    CGImageSource.h
    CGLayer.h
    CGPath.h
    CGPattern.h
    CGShading.h
```

Auxiliary Types

Fonts

`(`**`make-font`**` &key family size style justification text-color background-color) [function]`

Creates a font description that can be used in `set-view-font`.

`(`**`font-p`**` object) [function]`

Returns true if object is a font.

`(`**`same-font-p`**` font1 font2) [function]`

True if `font1` and `font2` are the same in all components.

`(`**`font-family`**` font) [function]`

Returns the font family.

`(`**`font-size`**` font) [function]`

Returns the font size.

`(`**`font-style`**` font) [function]`

Returns the font style.

(**font-justification** font) [function]
Returns the font text justification, one of `:default`, `:center`, `:right` or `:left`.

(**font-text-color** font) [function]
Returns the font foreground color.

(**font-background-color** font) [function]
Returns the font background color.

Colors

(**make-rgb-color** &key red green blue alpha) [function]
Creates a color description for the specified values.

(**rgb-color-p** object) [function]
Returns true if object is an `rgb-color`.

(**same-color-p** rgb-color1 rgb-color2) [function]
True if `rgb-color1` and `rgb-color2` are the same in all components.

(**color-red** rgb-color) [function]
Returns the red component of the color.

(**color-green** rgb-color) [function]
Returns the green component of the color.

(**color-blue** rgb-color) [function]
Returns the blue component of the color.

(**color-alpha** rgb-color) [function]
Returns the alpha component of the color.

Points

(**point** h v) [function]
Returns a point with horizontal coordinate h and vertical coordinate v. h and v can be any real numbers.

(**point-p** object) [function]
Returns true if object is a point.

(**point-h** point) [function]
Returns the horizontal coordinate of point.

(**point-v** point) [function]
Returns the vertical coordinate of `point`.

(**point+** point1 point2) [function]
Returns a point with each coordinate being the sum of `point1` and `point2`.

(**point-** point1 point2) [function]
Returns a point with each coordinate being the difference of `point1` and `point2`.

(**point<=** point1 point2) [function]

Returns true if each coordinate of `point1` is <= to corresponding coordinate of `point2`.

(**point<** point1 point2) [function]

Returns true if each coordinate of `point1` is < corresponding coordinate of `point2`.

(**point=** point1 point2) [function]

Returns true if each coordinate of `point1` is = to corresponding coordinate of `point2`.

Rectangles

(**make-rect** &key top left bottom right top-left bottom-right width height size) [function]

Returns a rectangle, specified by either `top/left/bottom/right` coordinates, or `top/left/height/width` coordinates, or the `top-left/bottom-right` points, or `top-left/size` points.

(**rect-p** object) [function]

Returns true if `object` is a rectangle.

(**rect-top** rect) [function]

Returns the top coordinate of `rect`.

(**rect-left** rect) [function]

Returns the left coordinate of `rect`.

(**rect-bottom** rect) [function]

Returns the bottom coordinate of `rect`.

(**rect-right** rect) [function]

Returns the top coordinate of `rect`.

(**rect-width** rect) [function]

Returns the width of `rect`.

(**rect-height** rect) [function]

Returns the height coordinate of `rect`.

(**rect-top-left** rect) [function]

Returns the top and left coordinates of `rect` as a `point`.

(**rect-bottom-right** rect) [function]

Returns the bottom and right coordinates of `rect` as a `point`.

(**rect-size** rect) [function]

Returns the width and height of `rect` as a `point`.

OSTypes

OSTypes are used as identifiers in many Carbon contexts. We do not introduce a special Common Lisp type to represent OSTypes. Instead, all functions that are documented as accepting an OSType will accept either a 32-bit integer, or a 4-character string (of 8-bit characters), or a keyword whose symbol-name is a 4-character string (of 8-bit characters).

(**ostype-value** ostype) [function]

Returns the integer value of `ostype`.

`(`**`ostype-string`**` ostype) [function]`
Returns the string value of `ostype`.

`(`**`ostype-keyword`**` ostype) [function]`
Returns the keyword value of `ostype`.