aquaterm egg

Bindings to the C API for AquaTerm
Extension for Chicken Scheme
Version 1.0.0

Thomas Chust

Table of Contents

1 About this egg

1.1 Version history

`1.0.0` Initial release

1.2 Usage

Load this egg like so:

```
(require-extension aquaterm)
```

2 Documentation

This documentation page is acutally a rewrite of the documentation for the Objective C AQTAdapter class. The C API this binding is relying on is not documented in its own right.

For in-depth information on the functionality of the routines and general information you should consult the original AquaTerm documentation as well as this manual.

Please not that the whole AquaTerm library is using the unit pt for its plot coordinates and sizes unless stated otherwise.

2.1 Initializing and Deinitializing the Library

`aqt:init` [procedure]

```
(aqt:init) => <boolean>
```

Initializes the library and return whether it was successful. You should call this routine before using any of the other functions.

`aqt:terminate` [procedure]

```
(aqt:terminate) => <void>
```

Deinitializes the library, call it as the last thing before terminating your program.

2.2 Controlling Plots

`aqt:open-plot` [procedure]

```
(aqt:open-plot (id <exact>)) => <void>
```

Creates a new plot with an arbitrary identification number that can be used in calls to `aqt:select-plot` to reference that plot.

Note that the plot is not displayed before you call `aqt:render-plot!`

`aqt:select-plot` [procedure]

```
(aqt:select-plot (id <exact>)) => <void>
```

Selects the plot identified by `id` as the target for subsequent commands.

`aqt:set-plot-size!` [procedure]

```
(aqt:set-plot-size! (width <number>) (height <float>)) => <void>
```

Defines the limits of the current plot area.

This function must be called before any drawing command following an `aqt:open-plot` or `aqt:clear-plot!` command, otherwise behaviour is undefined.

`aqt:set-plot-title!` [procedure]

```
(aqt:set-plot-title! (title <string>)) => <void>
```

Sets a title for the current plot, which will be displayed in the plot window's titlebar and will be used as a default filename when saving the plot from AquaTerm.

`aqt:render-plot!` [procedure]

```
(aqt:render-plot!) => <void>
```

Displays the current plot in an AquaTerm window rendering all drawing operations performed so far.

`aqt:clear-plot!` [procedure]

```
(aqt:clear-plot!) => <void>
```

Clears the current plot and resets all its attributes. Remeber to call `aqt:set-plot-size!` again after using this function.

To clear the plot without resetting it, use the `aqt:erase-rect!` procedure.

`aqt:close-plot!` [procedure]

```
(aqt:close-plot!) => <void>
```

Closes the connection to the current plot and disables event handling.

Note that this function does not cause the plot window to disappear if it was shown before.

2.3 Clipping

`aqt:set-clip-rect!` [procedure]

```
(aqt:set-clip-rect! (x <number>) (y <number>) (width <number>) (height <number>))
```

Set a rectangular clipping region to apply to all subsequent operations, until changed again by `aqt:set-clip-rect!` or `aqt:set-default-clip-rect!`

`aqt:set-default-clip-rect!` [procedure]

```
(aqt:set-default-clip-rect!) => <void>
```

Reset the clipping region to the entire plot for all subsequent operations, until changed again by `aqt:set-clip-rect!` or `aqt:set-default-clip-rect!`

2.4 Managing Colors

`aqt:set-color!` [procedure]

```
(aqt:set-color! (red <number>) (green <number>) (blue <number>)) => <void>
```

Set the drawing color in the active plot to the given `red`, `green` and `blue` intensities between 0 and 1.

`aqt:set-background-color!` [procedure]

```
(aqt:set-background-color! (red <number>) (green <number>) (blue <number>)) => <v
```

Set the background color in the active plot to the given `red`, `green` and `blue` intensities between 0 and 1.

`aqt:get-color` [procedure]

```
(aqt:get-color) => <number>, <number>, <number>
```

Retrieves the drawing color of the active plot and returns three values for its red, green and blue intensities between 0 and 1.

`aqt:get-background-color` [procedure]

```
(aqt:get-background-color) => <number>, <number>, <number>
```

Retrieves the background color of the active plot and returns three values for its red, green and blue intensities between 0 and 1.

2.5 Managing the Colormap

`aqt:colormap-size` [procedure]

```
(aqt:colormap-size) => <exact>
```

Returns the number of colormap entries available for the programmer.

`aqt:set-colormap-entry!` [procedure]

```
(aqt:set-colormap-entry! (idx <exact>) (red <number>) (green <number>) (blue <num
```

Set the entry at position `idx` in the colormap to the given `red`, `green` and `blue` intensities between 0 and 1.

`aqt:get-colormap-entry` [procedure]

```
(aqt:get-colormap-entry (idx <exact>)) => <number>, <number>, <number>
```

Retrieves the color at index `idx` in the colormap and returns three values for its red, green and blue intensities between 0 and 1.

`aqt:take-color-from-colormap-entry!` [procedure]

```
(aqt:take-color-from-colormap-entry! (idx <exact>)) => <void>
```

Set the drawing color in the active plot to the value of the colormap entry indicated by `idx`.

`aqt:take-background-color-from-colormap-entry!` [procedure]

```
(aqt:take-background-color-from-colormap-entry! (idx <exact>)) => <void>
```

Set the background color in the active plot to the value of the colormap entry indicated by `idx`.

2.6 Drawing Text

`aqt:set-fontname!` [procedure]

```
(aqt:set-fontname! (name <string>)) => <void>
```

Sets the font to be used for future operations.

The default font is `"Times-Roman"`.

`aqt:set-fontsize!` [procedure]

```
(aqt:set-fontsize! (size <number>)) => <void>
```

Sets the font size in points for future operations.

The default font size are 14 pt.

`aqt:add-label!` [procedure]

```
(aqt:add-label! (text <string>) (x <number>) (y <number>) (angle <number>) #!opti
```

Adds text at position `x, y`, rotated by `angle` degrees and aligned vertically and horizontally (with respect to position and rotation) according to `h-align` and `v-align`.

The following values are allowed for the alignment parameters:

`h-align` – **Horizontal Alignment**

`left`

`center`

`right`

`v-align` – **Vertical Alignment**

`middle`

`baseline`

`bottom`

`top`

`aqt:add-sheared-label!` [procedure]

```
(aqt:add-sheared-label! (text <string>) (x <number>) (y <number>) (angle <number>
```

Works like `aqt:add-label!` but additionally shears the text by `shear-angle` degrees in order to look right in 3D images for example.

2.7 Drawing Lines

`aqt:set-linewidth!` [procedure]

```
(aqt:set-linewidth! (width <number>)) => <void>
```

Sets the current linewidth to `width` points, used for all subsequent lines.

Any line currently being built by `aqt:move-to!` and `aqt:add-line-to!` will be considered finished since any coalesced sequence of line segments must share the same linewidth.

The default linewidth is 1 pt.

`aqt:set-linestyle-pattern!` [procedure]

```
(aqt:set-linestyle-pattern! (phase <number>) . pattern) => <void>
```

Set the current line style to pattern style, used for all subsequent lines. The linestyle is specified as at most 8 numbers, where even positions in the list correspond to dash-lengths and odd positions correspond to gap-lengths.

To produce e.g. a dash-dotted line, use the command `(aqt:set-linestyle-pattern! 0 4 2 1 2)`.

`aqt:set-linestyle-solid!` [procedure]

```
(aqt:set-linestyle-solid!) => <void>
```

Sets the current line style to solid, used for all subsequent lines.

This is the default setting.

`aqt:set-line-cap-style!` [procedure]

```
(aqt:set-line-cap-style! (style <symbol>)) => <void>
```

Sets the current line cap style, used for all subsequent lines.

Any line currently being built by `aqt:move-to!` and `aqt:add-line-to!` will be considered finished since any coalesced sequence of line segments must share the same cap style.

The `style` parameter can have the following values:

Style	Meaning
`butt`	Lines do not extend beyond their endpoint.
`round`	Lines extend into a half-circle beyond their endpoint.
`square`	Lines extend into a half-square beyond their endpoint.

The default line cap style is `round`.

`aqt:move-to!` [procedure]

```
(aqt:move-to! (x <number>) (y <number>)) => <void>
```

Moves the current point to `x, y` in preparation for a new sequence of line segments.

`aqt:add-line-to!` [procedure]

```
(aqt:add-line-to! (x <number>) (y <number>)) => <void>
```

Adds a line segment from the current point (given by a previous `aqt:move-to!` or `aqt:add-line-to!`) to `x, y` and sets the target position as the new current point.

`aqt:add-polyline!` [procedure]

```
(aqt:add-polyline! (points <list>)) => <void>
```

Adds a sequence of line segments specified by a list of start-, join-, and endpoint(s) in `points`, a list of two number lists.

2.8 Drawing Rectangles and Polygons

`aqt:move-to-vertex!` [procedure]

```
(aqt:move-to-vertex! (x <number>) (y <number>)) => <void>
```

Moves the current point to `x, y` in preparation for a new sequence of polygon edges.

`aqt:add-edge-to-vertex!` [procedure]

```
(aqt:add-edge-to-vertex! (x <number>) (y <number>)) => <void>
```

Adds a polygon edge from the current point (given by a previous `aqt:move-to-vertex!` or `aqt:add-edge-to-vertex!`) to `x, y` and sets the target position as the new current point.

`aqt:add-polygon!` [procedure]

```
(aqt:add-polygon! (points <list>)) => <void>
```

Adds a polygon specified by a list of vertices in `points`, a list of two number lists.

`aqt:add-filled-rect!` [procedure]

```
(aqt:add-filled-rect! (x <number>) (y <number>) (width <number>) (height <number>
```

Adds a filled rectangle and attempts to remove any objects that will be covered by it.

`aqt:erase-rect!` [procedure]

```
(aqt:erase-rect! (x <number>) (y <number>) (width <number>) (height <number>)) =>
```

Removes any objects completely inside aRect. Does not force a redraw of the plot.

2.9 Drawing Bitmaps

`aqt:set-image-transform!` [procedure]

```
(aqt:set-image-transform! (M-11 <number>) (M-12 <number>) (M-21 <number>) (M-22 <
```

Sets a transformation matrix for images added by `aqt:add-transformed-image-with-bitmap!`, see the ADC Reference Documentation on Basic Drawing for more information how this works.

`aqt:reset-image-transform!` [procedure]

```
(aqt:reset-image-transform!) => <void>
```

Sets the transformation matrix to unity, i.e. no transform.

`aqt:add-image-with-bitmap!` [procedure]

```
(aqt:add-image-with-bitmap! (data <string|byte-vector>) (data-width <exact>) (dat
```

Adds a bitmap image of size `data-width * data-height` pixels to the plot at position `x, y`, scaling it to `width x height` plot units.

The bitmap must be stored in `data` as 24 bit per pixel, 8 bit per color channel raw RGB data.

Does not apply transform.

`aqt:add-transformed-image-with-bitmap!` [procedure]

```
(aqt:add-transformed-image-with-bitmap! (data <string|byte-vector>) (data-width <
```

Adds a bitmap image of size `data-width * data-height` pixels to the plot at position `x, y`, scaling it to `width x height` plot units and applying the stored transformation matrix to it.

The bitmap must be stored in `data` as 24 bit per pixel, 8 bit per color channel raw RGB data.

2.10 Handling Events

`aqt:set-accepting-events!` [procedure]

```
(aqt:set-accepting-events! (yes <boolean>)) => <void>
```

Inform AquaTerm whether or not events should be passed from the currently selected plot. Deactivates event passing from any plot previously set to pass events.

`aqt:get-last-event` [procedure]

```
(aqt:get-last-event) => <list|boolean>
```

Reads the last event logged by the viewer. Will always return `#f` unless `aqt:set-accepting-events!` was called with a `#t` argument.

The event is returned by AquaTerm as a string but is decoded internally and passed back as follows:

Return value	**Meaning**
`#f`	No event was received.
`('mouse (x y) button)`	The mouse-button with the number `button` was clicked at the position `x, y` in plot coordinates.
`('key (x y) key)`	The key corresponding to the character `key` was clicked while the mouse was at the position `x, y` in plot coordinates.

Should the event returned from the library encode an error, a continuable exception of the composite kind `(exn aquaterm)` is thrown, which has an additional property `type` holding either the symbol `server` or `general` depending on the source of the error.

`aqt:wait-next-event` [procedure]

```
(aqt:wait-next-event) => <list|boolean>
```

Works analogous to `aqt:get-last-event` but blocks and waits for an event instead of returning immediately.

3 Examples

The following code is a free translation of the C API demo program shipped with AquaTerm. Most of the functionality of the library is demonstrated here.

```
;;;; amb-demo.scm
;;;; A solution for the Kalotan puzzle using amb

(require-extension amb)

(define (xor a? b?)
  (if (and a? b?) #f (or a? b?)))

(define (solve-kalotan-puzzle)
  (let ((parent1 (amb 'm 'f))
        (parent2 (amb 'm 'f))
        (kibi (amb 'm 'f))
        (kibi-self-desc (amb 'm 'f))
        (kibi-lied? (amb #t #f)))
    (amb-assert
     (not (eq? parent1 parent2)))
    (if kibi-lied?
        (amb-assert
         (xor
          (and (eqv? kibi-self-desc 'm)
               (eqv? kibi 'f))
          (and (eqv? kibi-self-desc 'f)
               (eqv? kibi 'm)))))
    (if (not kibi-lied?)
        (amb-assert
         (xor
          (and (eqv? kibi-self-desc 'm)
               (eqv? kibi 'm))
          (and (eqv? kibi-self-desc 'f)
               (eqv? kibi 'f)))))
    (if (eqv? parent1 'm)
        (amb-assert
         (and
          (eqv? kibi-self-desc 'm)
          (xor
           (and (eqv? kibi 'f)
                (eqv? kibi-lied? #f))
           (and (eqv? kibi 'm)
                (eqv? kibi-lied? #t))))))
    (if (eqv? parent1 'f)
        (amb-assert
         (and
          (eqv? kibi 'f)
```

```
          (eqv? kibi-lied? #t))))
    (list parent1 parent2 kibi)))

(write (amb-collect (solve-kalotan-puzzle)))
(newline)
```

4 License

Index