NAME

urxvtperl − rxvt−unicode's embedded perl interpreter

SYNOPSIS

```
# create a file grab_test in $HOME:

sub on_sel_grab {
   warn "you selected ", $_[0]->selection;
   ()
}

# start a urxvt using it:

urxvt --perl-lib $HOME -pe grab_test
```

DESCRIPTION

Every time a terminal object gets created, extension scripts specified via the `perl` resource are loaded and associated with it.

Scripts are compiled in a 'use strict ''vars''' and 'use utf8' environment, and thus must be encoded as UTF−8.

Each script will only ever be loaded once, even in urxvtd, where scripts will be shared (but not enabled) for all terminals.

You can disable the embedded perl interpreter by setting both ''perl-ext'' and ''perl-ext-common'' resources to the empty string.

PREPACKAGED EXTENSIONS

A number of extensions are delivered with this release. You can find them in *<libdir>/urxvt/perl/*, and the documentation can be viewed using *man urxvt−<EXTENSIONNAME>*.

You can activate them like this:

```
urxvt -pe <extensionname>
```

Or by adding them to the resource for extensions loaded by default:

```
URxvt.perl-ext-common: default,selection-autotransform
```

Extensions may add additional resources and `actions`, i.e., methods which can be bound to a key and invoked by the user. An extension can define the resources it support using so called META comments, described below. Similarly to builtin resources, extension resources can also be specified on the command line as long options (with `.` replaced by `−`), in which case the corresponding extension is loaded automatically. For this to work the extension **must** define META comments for its resources.

API DOCUMENTATION

General API Considerations

All objects (such as terminals, time watchers etc.) are typical reference-to-hash objects. The hash can be used to store anything you like. All members starting with an underscore (such as `_ptr` or `_hook`) are reserved for internal uses and **MUST NOT** be accessed or modified).

When objects are destroyed on the C++ side, the perl object hashes are emptied, so its best to store related objects such as time watchers and the like inside the terminal object so they get destroyed as soon as the terminal is destroyed.

Argument names also often indicate the type of a parameter. Here are some hints on what they mean:

`$text`

Rxvt-unicode's special way of encoding text, where one ''unicode'' character always represents one screen cell. See ROW_t for a discussion of this format.

`$string`
: A perl text string, with an emphasis on *text*. It can store all unicode characters and is to be distinguished with text encoded in a specific encoding (often locale-specific) and binary data.

`$octets`
: Either binary data or – more common – a text string encoded in a locale-specific way.

`$keysym`
: an integer that is a valid X11 keysym code. You can convert a string into a keysym and viceversa by using `XStringToKeysym` and `XKeysymToString`.

Extension Objects

Every perl extension is a perl class. A separate perl object is created for each terminal, and each terminal has its own set of extension objects, which are passed as the first parameter to hooks. So extensions can use their `$self` object without having to think about clashes with other extensions or other terminals, with the exception of methods and members that begin with an underscore character _: these are reserved for internal use.

Although it isn't a `urxvt::term` object, you can call all methods of the `urxvt::term` class on this object.

Additional methods only supported for extension objects are described in the `urxvt::extension` section below.

META comments

Rxvt-unicode recognizes special meta comments in extensions that define different types of metadata.

Currently, it recognises only one such comment:

#:META:RESOURCE:name:type:desc
: The RESOURCE comment defines a resource used by the extension, where `name` is the resource name, `type` is the resource type, `boolean` or `string`, and `desc` is the resource description.

Hooks

The following subroutines can be declared in extension files, and will be called whenever the relevant event happens.

The first argument passed to them is an extension object as described in the in the `Extension Objects` section.

All of these hooks must return a boolean value. If any of the called hooks returns true, then the event counts as being *consumed*, and the relevant action might not be carried out by the C++ code.

When in doubt, return a false value (preferably `()`).

on_init `$term`
: Called after a new terminal object has been initialized, but before windows are created or the command gets run. Most methods are unsafe to call or deliver senseless data, as terminal size and other characteristics have not yet been determined. You can safely query and change resources and options, though. For many purposes the `on_start` hook is a better place.

on_start `$term`
: Called at the very end of initialisation of a new terminal, just before trying to map (display) the toplevel and returning to the main loop.

on_destroy `$term`
: Called whenever something tries to destroy terminal, when the terminal is still fully functional (not for long, though).

on_reset `$term`
: Called after the screen is ''reset'' for any reason, such as resizing or control sequences. Here is where you can react on changes to size-related variables.

on_child_start `$term`, `$pid`
: Called just after the child process has been `fork`ed.

on_child_exit `$term`, `$status`
: Called just after the child process has exited. `$status` is the status from `waitpid`.

on_sel_make `$term`, `$eventtime`
: Called whenever a selection has been made by the user, but before the selection text is copied, so changes to the beginning, end or type of the selection will be honored.

  Returning a true value aborts selection making by urxvt, in which case you have to make a selection yourself by calling `$term->selection_grab`.

on_sel_grab `$term`, `$eventtime`
: Called whenever a selection has been copied, but before the selection is requested from the server. The selection text can be queried and changed by calling `$term->selection`.

  Returning a true value aborts selection grabbing. It will still be highlighted.

on_sel_extend `$term`
: Called whenever the user tries to extend the selection (e.g. with a double click) and is either supposed to return false (normal operation), or should extend the selection itself and return true to suppress the built-in processing. This can happen multiple times, as long as the callback returns true, it will be called on every further click by the user and is supposed to enlarge the selection more and more, if possible.

  See the *selection* example extension.

on_view_change `$term`, `$offset`
: Called whenever the view offset changes, i.e. the user or program scrolls. Offset `0` means display the normal terminal, positive values show this many lines of scrollback.

on_scroll_back `$term`, `$lines`, `$saved`
: Called whenever lines scroll out of the terminal area into the scrollback buffer. `$lines` is the number of lines scrolled out and may be larger than the scroll back buffer or the terminal.

  It is called before lines are scrolled out (so rows 0 .. min ($lines − 1, `$nrow` − 1) represent the lines to be scrolled out). `$saved` is the total number of lines that will be in the scrollback buffer.

on_osc_seq `$term`, `$op`, `$args`, `$resp`
: Called on every OSC sequence and can be used to suppress it or modify its behaviour. The default should be to return an empty list. A true value suppresses execution of the request completely. Make sure you don't get confused by recursive invocations when you output an OSC sequence within this callback.

  `on_osc_seq_perl` should be used for new behaviour.

on_osc_seq_perl `$term`, `$args`, `$resp`
: Called whenever the **ESC] 777 ; string ST** command sequence (OSC = operating system command) is processed. Cursor position and other state information is up-to-date when this happens. For interoperability, the string should start with the extension name (sans −osc) and a semicolon, to distinguish it from commands for other extensions, and this might be enforced in the future.

  For example, `overlay-osc` uses this:

```
sub on_osc_seq_perl {
   my ($self, $osc, $resp) = @_;

   return unless $osc =~ s/^overlay;//;

   ... process remaining $osc string
}
```

Be careful not ever to trust (in a security sense) the data you receive, as its source can not easily be controlled (e–mail content, messages from other users on the same system etc.).

For responses, `$resp` contains the end-of-args separator used by the sender.

on_add_lines `$term`, `$string`
: Called whenever text is about to be output, with the text as argument. You can filter/change and output the text yourself by returning a true value and calling `$term->scr_add_lines` yourself. Please note that this might be very slow, however, as your hook is called for **all** text being output.

on_tt_write `$term`, `$octets`
: Called whenever some data is written to the tty/pty and can be used to suppress or filter tty input.

on_tt_paste `$term`, `$octets`
: Called whenever text is about to be pasted, with the text as argument. You can filter/change and paste the text yourself by returning a true value and calling `$term->tt_paste` yourself. `$octets` is locale-encoded.

on_line_update `$term`, `$row`
: Called whenever a line was updated or changed. Can be used to filter screen output (e.g. underline urls or other useless stuff). Only lines that are being shown will be filtered, and, due to performance reasons, not always immediately.

The row number is always the topmost row of the line if the line spans multiple rows.

Please note that, if you change the line, then the hook might get called later with the already-modified line (e.g. if unrelated parts change), so you cannot just toggle rendition bits, but only set them.

on_refresh_begin `$term`
: Called just before the screen gets redrawn. Can be used for overlay or similar effects by modifying the terminal contents in refresh_begin, and restoring them in refresh_end. The built-in overlay and selection display code is run after this hook, and takes precedence.

on_refresh_end `$term`
: Called just after the screen gets redrawn. See `on_refresh_begin`.

on_action `$term`, `$string`
: Called whenever an action is invoked for the corresponding extension (e.g. via a `extension:string` builtin action bound to a key, see description of the **keysym** resource in the **urxvt** (1) manpage). The event is simply the action string. Note that an action event is always associated to a single extension.

on_user_command `$term`, `$string` *DEPRECATED*
: Called whenever a user-configured event is being activated (e.g. via a `perl:string` action bound to a key, see description of the **keysym** resource in the **urxvt** (1) manpage).

The event is simply the action string. This interface is going away in preference to the `on_action` hook.

on_resize_all_windows `$term`, `$new_width`, `$new_height`
: Called just after the new window size has been calculated, but before windows are actually being resized or hints are being set. If this hook returns a true value, setting of the window hints is being skipped.

on_x_event `$term`, `$event`
: Called on every X event received on the vt window (and possibly other windows). Should only be used as a last resort. Most event structure members are not passed.

on_root_event `$term`, `$event`
: Like `on_x_event`, but is called for events on the root window.

on_focus_in `$term`
: Called whenever the window gets the keyboard focus, before rxvt-unicode does focus in processing.

on_focus_out `$term`
: Called whenever the window loses keyboard focus, before rxvt-unicode does focus out processing.

on_configure_notify `$term`, `$event`
on_property_notify `$term`, `$event`
on_key_press `$term`, `$event`, `$keysym`, `$octets`
on_key_release `$term`, `$event`, `$keysym`
on_button_press `$term`, `$event`
on_button_release `$term`, `$event`
on_motion_notify `$term`, `$event`
on_map_notify `$term`, `$event`
on_unmap_notify `$term`, `$event`
: Called whenever the corresponding X event is received for the terminal. If the hook returns true, then the event will be ignored by rxvt-unicode.

  The event is a hash with most values as named by Xlib (see the XEvent manpage), with the additional members `row` and `col`, which are the (real, not screen-based) row and column under the mouse cursor.

  `on_key_press` additionally receives the string rxvt-unicode would output, if any, in locale-specific encoding.

on_client_message `$term`, `$event`
on_wm_protocols `$term`, `$event`
on_wm_delete_window `$term`, `$event`
: Called when various types of ClientMessage events are received (all with format=32, WM_PROTOCOLS or WM_PROTOCOLS:WM_DELETE_WINDOW).

on_bell `$term`
: Called on receipt of a bell character.

Variables in the `urxvt` Package

`$urxvt::LIBDIR`
: The rxvt-unicode library directory, where, among other things, the perl modules and scripts are stored.

`$urxvt::RESCLASS`, `$urxvt::RESCLASS`
: The resource class and name rxvt-unicode uses to look up X resources.

`$urxvt::RXVTNAME`
: The basename of the installed binaries, usually `urxvt`.

`$urxvt::TERM`
: The current terminal. This variable stores the current `urxvt::term` object, whenever a callback/hook is executing.

`@urxvt::TERM_INIT`
: All code references in this array will be called as methods of the next newly created `urxvt::term` object (during the `on_init` phase). The array gets cleared before the code references that were in it are being executed, so references can push themselves onto it again if they so desire.

  This complements to the perl-eval command line option, but gets executed first.

`@urxvt::TERM_EXT`
: Works similar to `@TERM_INIT`, but contains perl package/class names, which get registered as normal extensions after calling the hooks in `@TERM_INIT` but before other extensions. Gets cleared just like `@TERM_INIT`.

Functions in the `urxvt` Package

urxvt::fatal `$errormessage`
: Fatally aborts execution with the given error message (which should include a trailing newline). Avoid at all costs! The only time this is acceptable (and useful) is in the init hook, where it prevents the terminal from starting up.

urxvt::warn `$string`
: Calls `rxvt_warn` with the given string which should include a trailing newline. The module also overwrites the `warn` builtin with a function that calls this function.

  Using this function has the advantage that its output ends up in the correct place, e.g. on stderr of the connecting urxvtc client.

  Messages have a size limit of 1023 bytes currently.

`@terms` = urxvt::termlist
: Returns all urxvt::term objects that exist in this process, regardless of whether they are started, being destroyed etc., so be careful. Only term objects that have perl extensions attached will be returned (because there is no urxvt::term object associated with others).

`$time` = urxvt::NOW
: Returns the ''current time'' (as per the event loop).

urxvt::CurrentTime
urxvt::ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask, Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, AnyModifier
urxvt::NoEventMask, KeyPressMask, KeyReleaseMask, ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeymapStateMask, ExposureMask, VisibilityChangeMask, StructureNotifyMask, ResizeRedirectMask, SubstructureNotifyMask, SubstructureRedirectMask, FocusChangeMask, PropertyChangeMask, ColormapChangeMask, OwnerGrabButtonMask
urxvt::KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, FocusIn, FocusOut, KeymapNotify, Expose, GraphicsExpose, NoExpose, VisibilityNotify, CreateNotify, DestroyNotify, UnmapNotify, MapNotify, MapRequest, ReparentNotify, ConfigureNotify, ConfigureRequest, GravityNotify, ResizeRequest, CirculateNotify, CirculateRequest, PropertyNotify, SelectionClear, SelectionRequest, SelectionNotify, ColormapNotify, ClientMessage, MappingNotify
: Various constants for use in X calls and event processing.

RENDITION

Rendition bitsets contain information about colour, font, font styles and similar information for each screen cell.

The following ''macros'' deal with changes in rendition sets. You should never just create a bitset, you should always modify an existing one, as they contain important information required for correct operation of rxvt-unicode.

`$rend` = urxvt::DEFAULT_RSTYLE
: Returns the default rendition, as used when the terminal is starting up or being reset. Useful as a base to start when creating renditions.

`$rend` = urxvt::OVERLAY_RSTYLE
: Return the rendition mask used for overlays by default.

`$rendbit` = urxvt::RS_Bold, urxvt::RS_Italic, urxvt::RS_Blink, urxvt::RS_RVid, urxvt::RS_Uline
: Return the bit that enabled bold, italic, blink, reverse-video and underline, respectively. To enable such a style, just logically OR it into the bitset.

`$foreground` = urxvt::GET_BASEFG `$rend`
`$background` = urxvt::GET_BASEBG `$rend`
: Return the foreground/background colour index, respectively.

`$rend` = urxvt::SET_FGCOLOR `$rend`, `$new_colour`
`$rend` = urxvt::SET_BGCOLOR `$rend`, `$new_colour`
`$rend` = urxvt::SET_COLOR `$rend`, `$new_fg`, `$new_bg`
: Replace the foreground/background colour in the rendition mask with the specified one.

`$value` = urxvt::GET_CUSTOM `$rend`
: Return the "custom" value: Every rendition has 5 bits for use by extensions. They can be set and changed as you like and are initially zero.

`$rend` = urxvt::SET_CUSTOM `$rend`, `$new_value`
: Change the custom value.

The `urxvt::term::extension` class

Each extension attached to a terminal object is represented by a `urxvt::term::extension` object.

You can use these objects, which are passed to all callbacks to store any state related to the terminal and extension instance.

The methods (And data members) documented below can be called on extension objects, in addition to call methods documented for the <urxvt::term> class.

`$urxvt_term` = `$self->{term}`
: Returns the `urxvt::term` object associated with this instance of the extension. This member *must not* be changed in any way.

`$self->`enable ($hook_name => `$cb`[, `$hook_name` => `$cb`..])
: Dynamically enable the given hooks (named without the `on_` prefix) for this extension, replacing any hook previously installed via `enable` in this extension.

  This is useful when you want to overwrite time-critical hooks only temporarily.

  To install additional callbacks for the same hook, you can use the `on` method of the `urxvt::term` class.

`$self->`disable ($hook_name[, `$hook_name`..])
: Dynamically disable the given hooks.

`$guard` = `$self->`on ($hook_name => `$cb`[, `$hook_name` => `$cb`..])
: Similar to the `enable` enable, but installs additional callbacks for the given hook(s) (that is, it doesn't replace existing callbacks), and returns a guard object. When the guard object is destroyed the callbacks are disabled again.

`$self->`bind_action ($hotkey, `$action`)
`$self->`x_resource ($pattern)
`$self->`x_resource_boolean ($pattern)
: These methods support an additional `%` prefix for `$action` or `$pattern` when called on an extension object, compared to the `urxvt::term` methods of the same name – see the description of these methods in the `urxvt::term` class for details.

The `urxvt::anyevent` Class

The sole purpose of this class is to deliver an interface to the `AnyEvent` module – any module using it will work inside urxvt without further programming. The only exception is that you cannot wait on condition variables, but non-blocking condvar use is ok.

In practical terms this means is that you cannot use blocking APIs, but the non-blocking variant should work.

The `urxvt::term` Class

`$term` = new urxvt::term `$envhashref`, `$rxvtname`, [arg...]
: Creates a new terminal, very similar as if you had started it with system `$rxvtname, arg...`. `$envhashref` must be a reference to a `%ENV`-like hash which defines the environment of the new terminal.

  Croaks (and probably outputs an error message) if the new instance couldn't be created. Returns `undef` if the new instance didn't initialise perl, and the terminal object otherwise. The `init` and `start` hooks will be called before this call returns, and are free to refer to global data (which is race free).

`$term->destroy`
Destroy the terminal object (close the window, free resources etc.). Please note that urxvt will not exit as long as any event watchers (timers, io watchers) are still active.

`$term->exec_async` ($cmd[, `@args`])
Works like the combination of the `fork`/`exec` builtins, which executes ("starts") programs in the background. This function takes care of setting the user environment before exec'ing the command (e.g. `PATH`) and should be preferred over explicit calls to `exec` or `system`.

Returns the pid of the subprocess or `undef` on error.

`$isset` = `$term->option` ($optval[, `$set`])
Returns true if the option specified by `$optval` is enabled, and optionally change it. All option values are stored by name in the hash `%urxvt::OPTION`. Options not enabled in this binary are not in the hash.

Here is a likely non-exhaustive list of option names, please see the source file */src/optinc.h* to see the actual list:

```
borderLess buffered console cursorBlink cursorUnderline hold iconic
insecure intensityStyles iso14755 iso14755_52 jumpScroll loginShell
mapAlert meta8 mouseWheelScrollPage override_redirect pastableTabs
pointerBlank reverseVideo scrollBar scrollBar_floating scrollBar_right
scrollTtyKeypress scrollTtyOutput scrollWithBuffer secondaryScreen
secondaryScroll skipBuiltinGlyphs skipScroll transparent tripleclickwords
urgentOnBell utmpInhibit visualBell
```

`$value` = `$term->resource` ($name[, `$newval`])
Returns the current resource value associated with a given name and optionally sets a new value. Setting values is most useful in the `init` hook. Unset resources are returned and accepted as `undef`.

The new value must be properly encoded to a suitable character encoding before passing it to this method. Similarly, the returned value may need to be converted from the used encoding to text.

Resource names are as defined in *src/rsinc.h*. Colours can be specified as resource names of the form `color+<index>`, e.g. `color+5`. (will likely change).

Please note that resource strings will currently only be freed when the terminal is destroyed, so changing options frequently will eat memory.

Here is a likely non-exhaustive list of resource names, not all of which are supported in every build, please see the source file */src/rsinc.h* to see the actual list:

```
answerbackstring backgroundPixmap backspace_key blurradius
boldFont boldItalicFont borderLess buffered chdir color cursorBlink
cursorUnderline cutchars delete_key depth display_name embed ext_bwidth
fade font geometry hold iconName iconfile imFont imLocale inputMethod
insecure int_bwidth intensityStyles iso14755 iso14755_52 italicFont
jumpScroll letterSpace lineSpace loginShell mapAlert meta8 modifier
mouseWheelScrollPage name override_redirect pastableTabs path perl_eval
perl_ext_1 perl_ext_2 perl_lib pointerBlank pointerBlankDelay
preeditType print_pipe pty_fd reverseVideo saveLines scrollBar
scrollBar_align scrollBar_floating scrollBar_right scrollBar_thickness
scrollTtyKeypress scrollTtyOutput scrollWithBuffer scrollstyle
secondaryScreen secondaryScroll shade skipBuiltinGlyphs skipScroll
term_name title transient_for transparent tripleclickwords urgentOnBell
utmpInhibit visualBell
```

`$value` = `$term->x_resource` ($pattern)
Returns the X-Resource for the given pattern, excluding the program or class name, i.e. `$term->x_resource ("boldFont")` should return the same value as used by this instance of

rxvt-unicode. Returns `undef` if no resource with that pattern exists.

Extensions that define extra resources also need to call this method to access their values.

If the method is called on an extension object (basically, from an extension), then the special prefix `%.` will be replaced by the name of the extension and a dot, and the lone string `%` will be replaced by the extension name itself. This makes it possible to code extensions so you can rename them and get a new set of resources without having to change the actual code.

This method should only be called during the `on_start` hook, as there is only one resource database per display, and later invocations might return the wrong resources.

`$value` = `$term->`x_resource_boolean ($pattern)
Like `x_resource`, above, but interprets the string value as a boolean and returns `1` for true values, `0` for false values and `undef` if the resource or option isn't specified.

You should always use this method to parse boolean resources.

`$action` = `$term->`lookup_keysym ($keysym, `$state`)
Returns the action bound to key combination `($keysym, $state)`, if a binding for it exists, and `undef` otherwise.

`$success` = `$term->`bind_action ($key, `$action`)
Adds a key binding exactly as specified via a `keysym` resource. See the `keysym` resource in the **urxvt**(1) manpage.

To add default bindings for actions, an extension should call `->bind_action` in its `init` hook for every such binding. Doing it in the `init` hook allows users to override or remove the binding again.

Example: the `searchable-scrollback` by default binds itself on `Meta-s`, using `$self->bind_action`, which calls `$term->bind_action`.

```
sub init {
   my ($self) = @_;

   $self->bind_action ("M-s" => "%:start");
}
```

`$rend` = `$term->`rstyle ([$new_rstyle])
Return and optionally change the current rendition. Text that is output by the terminal application will use this style.

($row, `$col`) = `$term->`screen_cur ([$row, `$col`])
Return the current coordinates of the text cursor position and optionally set it (which is usually bad as applications don't expect that).

($row, `$col`) = `$term->`selection_mark ([$row, `$col`])
($row, `$col`) = `$term->`selection_beg ([$row, `$col`])
($row, `$col`) = `$term->`selection_end ([$row, `$col`])
Return the current values of the selection mark, begin or end positions.

When arguments are given, then the selection coordinates are set to `$row` and `$col`, and the selection screen is set to the current screen.

`$screen` = `$term->`selection_screen ([$screen])
Returns the current selection screen, and then optionally sets it.

`$term->`selection_make ($eventtime[, `$rectangular`])
Tries to make a selection as set by `selection_beg` and `selection_end`. If `$rectangular` is true (default: false), a rectangular selection will be made. This is the preferred function to make a selection.

`$success` = `$term->`selection_grab ($eventtime[, `$clipboard`])
: Try to acquire ownership of the primary (clipboard if `$clipboard` is true) selection from the server. The corresponding text can be set with the next method. No visual feedback will be given. This function is mostly useful from within `on_sel_grab` hooks.

`$oldtext` = `$term->`selection ([$newtext, `$clipboard`])
: Return the current selection (clipboard if `$clipboard` is true) text and optionally replace it by `$newtext`.

`$term->`selection_clear ([$clipboard])
: Revoke ownership of the primary (clipboard if `$clipboard` is true) selection.

`$term->`overlay_simple ($x, `$y`, `$text`)
: Create a simple multi-line overlay box. See the next method for details.

`$term->`overlay ($x, `$y`, `$width`, `$height`[, `$rstyle`[, `$border`]])
: Create a new (empty) overlay at the given position with the given width/height. `$rstyle` defines the initial rendition style (default: `OVERLAY_RSTYLE`).

  If `$border` is `2` (default), then a decorative border will be put around the box.

  If either `$x` or `$y` is negative, then this is counted from the right/bottom side, respectively.

  This method returns an urxvt::overlay object. The overlay will be visible as long as the perl object is referenced.

  The methods currently supported on `urxvt::overlay` objects are:

  `$overlay->`set ($x, `$y`, `$text`[, `$rend`])
  : Similar to `$term->ROW_t` and `$term->ROW_r` in that it puts text in rxvt-unicode's special encoding and an array of rendition values at a specific position inside the overlay.

    If `$rend` is missing, then the rendition will not be changed.

  `$overlay->`hide
  : If visible, hide the overlay, but do not destroy it.

  `$overlay->`show
  : If hidden, display the overlay again.

`$popup` = `$term->`popup ($event)
: Creates a new `urxvt::popup` object that implements a popup menu. The `$event` *must* be the event causing the menu to pop up (a button event, currently).

`$cellwidth` = `$term->`strwidth ($string)
: Returns the number of screen-cells this string would need. Correctly accounts for wide and combining characters.

`$octets` = `$term->`locale_encode ($string)
: Convert the given text string into the corresponding locale encoding.

`$string` = `$term->`locale_decode ($octets)
: Convert the given locale-encoded octets into a perl string.

`$term->`scr_xor_span ($beg_row, `$beg_col`, `$end_row`, `$end_col`[, `$rstyle`])
: XORs the rendition values in the given span with the provided value (default: `RS_RVid`), which *MUST NOT* contain font styles. Useful in refresh hooks to provide effects similar to the selection.

`$term->`scr_xor_rect ($beg_row, `$beg_col`, `$end_row`, `$end_col`[, `$rstyle1`[, `$rstyle2`]])
: Similar to `scr_xor_span`, but xors a rectangle instead. Trailing whitespace will additionally be xored with the `$rstyle2`, which defaults to `RS_RVid | RS_Uline`, which removes reverse video again and underlines it instead. Both styles *MUST NOT* contain font styles.

`$term->scr_bell`
Ring the bell!

`$term->scr_add_lines ($string)`
Write the given text string to the screen, as if output by the application running inside the terminal. It may not contain command sequences (escape codes - see `cmd_parse` for that), but is free to use line feeds, carriage returns and tabs. The string is a normal text string, not in locale-dependent encoding.

Normally its not a good idea to use this function, as programs might be confused by changes in cursor position or scrolling. Its useful inside a `on_add_lines` hook, though.

`$term->scr_change_screen ($screen)`
Switch to given screen - 0 primary, 1 secondary.

`$term->cmd_parse ($octets)`
Similar to `scr_add_lines`, but the argument must be in the locale-specific encoding of the terminal and can contain command sequences (escape codes) that will be interpreted.

`$term->tt_write ($octets)`
Write the octets given in `$octets` to the tty (i.e. as user input to the program, see `cmd_parse` for the opposite direction). To pass characters instead of octets, you should convert your strings first to the locale-specific encoding using `$term->locale_encode`.

`$term->tt_write_user_input ($octets)`
Like `tt_write`, but should be used when writing strings in response to the user pressing a key, to invoke the additional actions requested by the user for that case (`tt_write` doesn't do that).

The typical use case would be inside `on_action` hooks.

`$term->tt_paste ($octets)`
Write the octets given in `$octets` to the tty as a paste, converting NL to CR and bracketing the data with control sequences if bracketed paste mode is set.

`$old_events = $term->pty_ev_events ([$new_events])`
Replaces the event mask of the pty watcher by the given event mask. Can be used to suppress input and output handling to the pty/tty. See the description of `urxvt::timer->events`. Make sure to always restore the previous value.

`$fd = $term->pty_fd`
Returns the master file descriptor for the pty in use, or `-1` if no pty is used.

`$windowid = $term->parent`
Return the window id of the toplevel window.

`$windowid = $term->vt`
Return the window id of the terminal window.

`$term->vt_emask_add ($x_event_mask)`
Adds the specified events to the vt event mask. Useful e.g. when you want to receive pointer events all the times:

```
$term->vt_emask_add (urxvt::PointerMotionMask);
```

`$term->set_urgency ($set)`
Enable/disable the urgency hint on the toplevel window.

`$term->focus_in`
`$term->focus_out`
`$term->key_press ($state, $keycode[, $time])`
`$term->key_release ($state, $keycode[, $time])`
Deliver various fake events to to terminal.

`$window_width` = `$term->`width ([$new_value])
`$window_height` = `$term->`height ([$new_value])
`$font_width` = `$term->`fwidth ([$new_value])
`$font_height` = `$term->`fheight ([$new_value])
`$font_ascent` = `$term->`fbase ([$new_value])
`$terminal_rows` = `$term->`nrow ([$new_value])
`$terminal_columns` = `$term->`ncol ([$new_value])
`$has_focus` = `$term->`focus ([$new_value])
`$is_mapped` = `$term->`mapped ([$new_value])
`$max_scrollback` = `$term->`saveLines ([$new_value])
`$nrow_plus_saveLines` = `$term->`total_rows ([$new_value])
`$topmost_scrollback_row` = `$term->`top_row ([$new_value])
: Return various integers describing terminal characteristics. If an argument is given, changes the value and returns the previous one.

`$x_display` = `$term->`display_id
: Return the DISPLAY used by rxvt-unicode.

`$lc_ctype` = `$term->`locale
: Returns the LC_CTYPE category string used by this rxvt-unicode.

`$env` = `$term->`env
: Returns a copy of the environment in effect for the terminal as a hashref similar to `\%ENV`.

`@envv` = `$term->`envv
: Returns the environment as array of strings of the form `VAR=VALUE`.

`@argv` = `$term->`argv
: Return the argument vector as this terminal, similar to `@ARGV`, but includes the program name as first element.

`$modifiermask` = `$term->`ModLevel3Mask
`$modifiermask` = `$term->`ModMetaMask
`$modifiermask` = `$term->`ModNumLockMask
: Return the modifier masks corresponding to the "ISO Level 3 Shift" (often AltGr), the meta key (often Alt) and the num lock key, if applicable.

`$screen` = `$term->`current_screen
: Returns the currently displayed screen (0 primary, 1 secondary).

`$cursor_is_hidden` = `$term->`hidden_cursor
: Returns whether the cursor is currently hidden or not.

`$view_start` = `$term->`view_start ([$newvalue])
: Returns the row number of the topmost displayed line. Maximum value is `0`, which displays the normal terminal contents. Lower values scroll this many lines into the scrollback buffer.

`$term->`want_refresh
: Requests a screen refresh. At the next opportunity, rxvt-unicode will compare the on-screen display with its stored representation. If they differ, it redraws the differences.

  Used after changing terminal contents to display them.

`$term->`refresh_check
: Checks if a refresh has been requested and, if so, schedules one.

`$text` = `$term->`ROW_t ($row_number[, `$new_text`[, `$start_col`]])
: Returns the text of the entire row with number `$row_number`. Row `$term->top_row` is the topmost terminal line, row `$term->nrow-1` is the bottommost terminal line. Nothing will be returned if a nonexistent line is requested.

  If `$new_text` is specified, it will replace characters in the current line, starting at column `$start_col` (default `0`), which is useful to replace only parts of a line. The font index in the

rendition will automatically be updated.

`$text` is in a special encoding: tabs and wide characters that use more than one cell when displayed are padded with `$urxvt::NOCHAR` (chr 65535) characters. Characters with combining characters and other characters that do not fit into the normal text encoding will be replaced with characters in the private use area.

You have to obey this encoding when changing text. The advantage is that `substr` and similar functions work on screen cells and not on characters.

The methods `$term->special_encode` and `$term->special_decode` can be used to convert normal strings into this encoding and vice versa.

`$rend` = `$term`->ROW_r ($row_number[, `$new_rend`[, `$start_col`]])
Like `$term->ROW_t`, but returns an arrayref with rendition bitsets. Rendition bitsets contain information about colour, font, font styles and similar information. See also `$term->ROW_t`.

When setting rendition, the font mask will be ignored.

See the section on RENDITION, above.

`$length` = `$term`->ROW_l ($row_number[, `$new_length`])
Returns the number of screen cells that are in use (''the line length''). Unlike the urxvt core, this returns `$term->ncol` if the line is joined with the following one.

`$bool` = `$term`->is_longer ($row_number)
Returns true if the row is part of a multiple-row logical ''line'' (i.e. joined with the following row), which means all characters are in use and it is continued on the next row (and possibly a continuation of the previous row(s)).

`$line` = `$term`->line ($row_number)
Create and return a new `urxvt::line` object that stores information about the logical line that row `$row_number` is part of. It supports the following methods:

`$text` = `$line`->t ([$new_text])
Returns or replaces the full text of the line, similar to `ROW_t`

`$rend` = `$line`->r ([$new_rend])
Returns or replaces the full rendition array of the line, similar to `ROW_r`

`$length` = `$line`->l
Returns the length of the line in cells, similar to `ROW_l`.

`$rownum` = `$line`->beg
`$rownum` = `$line`->end
Return the row number of the first/last row of the line, respectively.

`$offset` = `$line`->offset_of ($row, `$col`)
Returns the character offset of the given row|col pair within the logical line. Works for rows outside the line, too, and returns corresponding offsets outside the string.

($row, `$col`) = `$line`->coord_of ($offset)
Translates a string offset into terminal coordinates again.

`$text` = `$term`->special_encode `$string`
Converts a perl string into the special encoding used by rxvt-unicode, where one character corresponds to one screen cell. See `$term->ROW_t` for details.

`$string` = `$term`->special_decode `$text`
Converts rxvt-unicodes text representation into a perl string. See `$term->ROW_t` for details.

`$success` = `$term`->grab_button ($button, `$modifiermask`[, `$window` = `$term`->vt])
`$term`->ungrab_button ($button, `$modifiermask`[, `$window` = `$term`->vt])
Register/unregister a synchronous button grab. See the XGrabButton manpage.

`$success` = `$term`->grab ($eventtime[, `$sync`])
: Calls XGrabPointer and XGrabKeyboard in asynchronous (default) or synchronous (`$sync` is true). Also remembers the grab timestamp.

`$term`->allow_events_async
: Calls XAllowEvents with AsyncBoth for the most recent grab.

`$term`->allow_events_sync
: Calls XAllowEvents with SyncBoth for the most recent grab.

`$term`->allow_events_replay
: Calls XAllowEvents with both ReplayPointer and ReplayKeyboard for the most recent grab.

`$term`->ungrab
: Calls XUngrabPointer and XUngrabKeyboard for the most recent grab. Is called automatically on evaluation errors, as it is better to lose the grab in the error case as the session.

`$atom` = `$term`->XInternAtom ($atom_name[, `$only_if_exists`])
`$atom_name` = `$term`->XGetAtomName ($atom)
`@atoms` = `$term`->XListProperties ($window)
($type,$format,$octets) = `$term`->XGetWindowProperty ($window, `$property`)
`$term`->XChangeProperty ($window, `$property`, `$type`, `$format`, `$octets`)
`$term`->XDeleteProperty ($window, `$property`)
`$window` = `$term`->DefaultRootWindow
`$term`->XReparentWindow ($window, `$parent`, [$x, `$y`])
`$term`->XMapWindow ($window)
`$term`->XUnmapWindow ($window)
`$term`->XMoveResizeWindow ($window, `$x`, `$y`, `$width`, `$height`)
($x, `$y`, `$child_window`) = `$term`->XTranslateCoordinates ($src, `$dst`, `$x`, `$y`)
`$term`->XChangeInput ($window, `$add_events`[, `$del_events`])
`$keysym` = `$term`->XStringToKeysym ($string)
`$string` = `$term`->XKeysymToString ($keysym)
: Various X or X-related functions. The `$term` object only serves as the source of the display, otherwise those functions map more-or-less directly onto the X functions of the same name.

The `urxvt::popup` Class

`$popup`->add_title ($title)
: Adds a non-clickable title to the popup.

`$popup`->add_separator ([$sepchr])
: Creates a separator, optionally using the character given as `$sepchr`.

`$popup`->add_button ($text, `$cb`)
: Adds a clickable button to the popup. `$cb` is called whenever it is selected.

`$popup`->add_toggle ($text, `$initial_value`, `$cb`)
: Adds a toggle/checkbox item to the popup. The callback gets called whenever it gets toggled, with a boolean indicating its new value as its first argument.

`$popup`->show
: Displays the popup (which is initially hidden).

The `urxvt::timer` Class

This class implements timer watchers/events. Time is represented as a fractional number of seconds since the epoch. Example:

```
   $term->{overlay} = $term->overlay (-1, 0, 8, 1, urxvt::OVERLAY_RSTYLE, 0);
   $term->{timer} = urxvt::timer
                    ->new
                    ->interval (1)
                    ->cb (sub {
                       $term->{overlay}->set (0, 0,
                          sprintf "%2d:%02d:%02d", (localtime urxvt::NOW)[2,1,0]);
                    });
```

`$timer` = new urxvt::timer
: Create a new timer object in started state. It is scheduled to fire immediately.

`$timer` = `$timer`->cb (sub { my ($timer) = `@_`; ... })
: Set the callback to be called when the timer triggers.

`$timer` = `$timer`->set ($tstamp[, `$interval`])
: Set the time the event is generated to `$tstamp` (and optionally specifies a new `$interval`).

`$timer` = `$timer`->interval ($interval)
: By default (and when `$interval` is `0`), the timer will automatically stop after it has fired once. If `$interval` is non-zero, then the timer is automatically rescheduled at the given intervals.

`$timer` = `$timer`->start
: Start the timer.

`$timer` = `$timer`->start ($tstamp[, `$interval`])
: Set the event trigger time to `$tstamp` and start the timer. Optionally also replaces the interval.

`$timer` = `$timer`->after ($delay[, `$interval`])
: Like `start`, but sets the expiry timer to c<urxvt::NOW + `$delay`>.

`$timer` = `$timer`->stop
: Stop the timer.

The `urxvt::iow` **Class**

This class implements io watchers/events. Example:

```
  $term->{socket} = ...
  $term->{iow} = urxvt::iow
                 ->new
                 ->fd (fileno $term->{socket})
                 ->events (urxvt::EV_READ)
                 ->start
                 ->cb (sub {
                   my ($iow, $revents) = @_;
                   # $revents must be 1 here, no need to check
                   sysread $term->{socket}, my $buf, 8192
                      or end-of-file;
                 });
```

`$iow` = new urxvt::iow
: Create a new io watcher object in stopped state.

`$iow` = `$iow`->cb (sub { my ($iow, `$reventmask`) = `@_`; ... })
: Set the callback to be called when io events are triggered. `$reventmask` is a bitset as described in the `events` method.

`$iow` = `$iow`->fd ($fd)
: Set the file descriptor (not handle) to watch.

`$iow` = `$iow`->events ($eventmask)
: Set the event mask to watch. The only allowed values are `urxvt::EV_READ` and `urxvt::EV_WRITE`, which might be ORed together, or `urxvt::EV_NONE`.

`$iow` = `$iow->`start
Start watching for requested events on the given handle.

`$iow` = `$iow->`stop
Stop watching for events on the given file handle.

The `urxvt::iw` Class
This class implements idle watchers, that get called automatically when the process is idle. They should return as fast as possible, after doing some useful work.

`$iw` = new urxvt::iw
Create a new idle watcher object in stopped state.

`$iw` = `$iw->`cb (sub { my ($iw) = `@_`; ... })
Set the callback to be called when the watcher triggers.

`$timer` = `$timer->`start
Start the watcher.

`$timer` = `$timer->`stop
Stop the watcher.

The `urxvt::pw` Class
This class implements process watchers. They create an event whenever a process exits, after which they stop automatically.

```
my $pid = fork;
...
$term->{pw} = urxvt::pw
                  ->new
                  ->start ($pid)
                  ->cb (sub {
                     my ($pw, $exit_status) = @_;
                     ...
                  });
```

`$pw` = new urxvt::pw
Create a new process watcher in stopped state.

`$pw` = `$pw->`cb (sub { my ($pw, `$exit_status`) = `@_`; ... })
Set the callback to be called when the timer triggers.

`$pw` = `$timer->`start ($pid)
Tells the watcher to start watching for process `$pid`.

`$pw` = `$pw->`stop
Stop the watcher.

ENVIRONMENT

URXVT_PERL_VERBOSITY
This variable controls the verbosity level of the perl extension. Higher numbers indicate more verbose output.

== 0 – fatal messages
>= 3 – script loading and management
>=10 – all called hooks
>=11 – hook return values

AUTHOR

Marc Lehmann <schmorp@schmorp.de>
http://software.schmorp.de/pkg/rxvt-unicode