Epoch

GNU Emacs for the X Windowing System

Release 4.0 Patchlevel 0

Based on GNU Emacs 18.58

March 1992

Copyright © 1989, 1990, 1991, 1992 by
The Epoch Development Team:

Simon Kaplan (kaplan@cs.uiuc.edu);
  Leadership, design, manual

Christopher Love (love@cs.uiuc.edu);
  Implementation, design, manual, support

Alan M. Carroll (carroll@cs.uiuc.edu);
  Implementation, design, manual

Daniel M. LaLiberte (liberte@cs.uiuc.edu);
  Manual support

This documents release 4.0 of Epoch which is based on GNU Emacs 18.58. Epoch was built as part of the ConversationBuilder project under Simon Kaplan. Epoch was originally designed and implemented by Kaplan and Carroll. The manual was written by Kaplan, Carroll, Love, and LaLiberte. Epoch 3.2 was designed and implmented by Kaplan, Carroll, and Love. Epoch 4.0 was designed by Kaplan, Love, and Carroll and was implemented by Love. Change hooks were provided by LaLiberte.

Published by ...

1304 W Springfield
Department of Computer Science
University of Illinois, Urbana-Champaign

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the ...University of Illinois, Urbana-Champaign.

1. Introduction

Epoch is a modified version of GNU Emacs with several major enhancements:

- Multiple X-Window support
- Marked and attributed regions of text
- Support for proportional and variably-sized fonts
- Support for 8-bit clean fonts (ISO, Latin-1, etc.)
- Asynchronous communication with other X clients (usually the window manager).
- Mouse-dragging with highlighting to support mouse-based cut-and-paste.
- Access to raw X-Window objects.

Some minor enhancements include various bug fixes and improved option handling.

As much compatability as possible with previous versions of GNU Emacs has been maintained. Existing elisp code should run under Epoch, although it will not automatically use the new features of Epoch, and the behavior may not be as expected when running with proportional fonts. Almost all the X support code in GNU Emacs was removed because of incompatabilities. Almost all features except the X menu code were reproduced; we believe that the window manager should provide menus (see Section 2.5.2 [Menus], page 13).

Epoch requires X-Windows to function. Support for editing on non-graphic terminals is provided, but is equivalent to running an ASCII version of GNU Emacs with no Epoch functionality. The mode of operation is determined as with GNU Emacs: if the "-nw" command-line flag is specified or if the environment variable $DISPLAY is unset, then Epoch will run in ASCII mode; otherwise it will run under X with full functionality provided. **NOTE:** Please refer to the file '`INSTALL`' for insuring appropriate ASCII support.

Most of this manual is intended for the "Emacs guru" who wants to play with and extend Epoch's functionality. Information intended for "ordinary Emacs users" is found in see Chapter 2 [General Information], page 5. There is now a list of frequently asked questions; refer to the file '`etc/EPOCH-FAQ`'.

1.1 Support and Mailing List

For sharing ideas about applications of Epoch, code, and bug reports, there is now a news-group:

`gnu.epoch.misc`. Articles posted to this newsgroup are gatewayed to and from the Epoch mailing list for those without news access.

To post to the mailing list send mail to one of these addresses:

```
epoch@cs.uiuc.edu
uunet!uiucdcs!epoch
epoch%cs.uiuc.edu@uiucvmd.bitnet
```

For administrative messages (joining the mailing list, etc), substitute '`epoch`' with '`epoch-request`' in the above.

1.2 Getting Epoch

Epoch is available by anonymous ftp on cs.uiuc.edu. We are negotiating to have it available through other sites, and through uucp mail request. Send mail to epoch-request to find out the current status with this if you do not have anonymous ftp.

Epoch is available on tape: The cost at the time of creating version 4.0 was $175.00 for cartridge tapes. We can currently make 1600 BPI reel tapes and cartridge tapes for SUN and HP 9000 series machines. The price covers the cost of purchasing, making and shipping the tapes. You should contact the epoch-request mailing list to confirm these rates. Payment should be by cheque made payable to the University of Illinois, and sent to:

```
Epoch Distribution, attn: R. Canaday
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana
Illinois 61801, USA.
```

Announcements regarding official (i.e. supported) patches and subsequent releases will be made to `gnu.epoch.misc`; patches will be available via ftp and email upon request.

1.3 Acknowledgements

Simon would like to thank the Computer Science Department at the University of Illinois, AT&T and the National Science Foundation for their support of parts of this work, and also Philippa and

Ronan for putting up with him.

Dan would like to thank Simon and Alan for their fine work, and Clare and Carey for putting him up.

Chris would like to thank Simon and Alan for valuable contributions to the design of Epoch 4.0, and members of the ConversationBuilder research group for serving as "pre-Alpha" guinea pigs.

Susan Hinrichs, Alan's lovely wife, provided emotional, intellectual, and code support.

We would all like to thank Richard Stallman and the Free Software Foundation for GNU Emacs, and Colas Nahaboo and Bull for the GWM window manager.

We would also like to thank all those who have provided invaluable advice, assistance, and contributions in getting Epoch to this stage of development through Alpha and Beta testing, including members of the `epoch-design` mailing list.

2. General Information

2.1 Getting Started

We assume you are reasonably familiar with GNU Emacs. You can essentially use Epoch just like GNU Emacs.

Epoch is started just like GNU Emacs. It opens it own X window(s), so it should be run in background. Depending on its mode of operation, Epoch will create one or two windows by default. The first will contain the initial edit buffer; if created, the second will contain the minibuffer. More on this below (See Section 2.2 [New Features], page 9).

If you wish to use Epoch without having to do any Elisp programming, while still specifying window properties, the following sections are for you.

2.1.1 X Window Program Defaults

Epoch reads the default values associated with X Window programs, or *X defaults*, set up by the user. The possible sources of these defaults are listed below in the order in which epoch will discover them. These values are loaded into an internal database, called the global *default-set*, which is initially filled in with hard-wired defaults (which are listed below).

The resulting global *default-set* is stored, and remains static for the rest of the program execution. It serves as a basis for all screen creation, in order to guarantee a value for every property. (The minibuffer has its own default-set, which is used once to create the minibuffer screen.) See Section 3.2 [Screen Properties], page 17 for more background on how X defaults are used.

The X default specifications are determined in the following order, with later specifications superceding earlier values for the same options:

1. Hard-wired X defaults (see below).
2. <resource-name> or <class-name> in either $XAPPLRESDIR or the appropriate app-defaults directory.
3. $HOME/.Xdefaults if "-ud" command line flag is set.
4. File named by the environment variable $XENVIRONMENT

5. If $XENVIRONMENT is not set, $HOME/.Xdefaults-<hostname>
6. Resources database (see '`xrdb`')
7. Command line options

The X default options recognized by Epoch are listed in the table below. The window resource name/class defaults are special; they default to the program resource name/class respectively (see Section 2.1.2 [Command Line Options], page 8).

Furthermore, the *program* resource name/class command line options are special. They are not put into the internal database - they affect how the database is queried for X default values. When searching for names and classes, the leftmost qualifier is the resource name/class name. The default for the name is the name of the program (normally "epoch") and the default for the class is "Emacs". If these are set by command line options, the new values become the X defaults.

Epoch also accepts class defaults (as distinct from resource defaults), which are exactly as below except that the first letter of each field is capitalized.

`Option` Default Value, Effect

`.screen.background`
: White, Edit screen background color

`.minibuf.background`
: White, Minibuffer screen background color

`.screen.border.color`
: Black, Edit screen border color

`.minibuf.border.color`
: Black, Minibuffer screen border color

`.screen.border.width`
: 2, Edit screen border width

`.minibuf.border.width`
: 2, Minibuffer screen border width

`.screen.cursor.background`
: White, Edit screen X cursor background color

`.minibuf.cursor.background`
: White, Minibuffer screen X cursor background color

`.screen.cursor.color`
: Black, Edit screen text cursor color

`.minibuf.cursor.color`
: Black, Minibuffer screen text cursor color

`.screen.cursor.foreground`
: Black, Edit screen X cursor foreground color

`.minibuf.cursor.foreground`
: Black, Minibuffer screen X cursor foreground color

`.screen.cursor.glyph`
: 86, Edit screen X cursor glyph

`.minibuf.cursor.glyph`
: 86, Minibuffer screen X cursor glyph

`.screen.internal.border.width`
: 2, Edit screen internal border width

`.minibuf.internal.border.width`
: 2, Minibuffer screen internal border width

`.screen.class`
: <special>, Resource class of the edit screen

`.minibuf.class`
: <special>, Resource class of the minibuffer screen

`.nonlocal.minibuf`
: false, Minibuffer location (local or non-local)

`.display` ""
: , X Window display

`.motion` Off
: , Whether motion-hints should come through to elisp

`.screen.font`
: "fixed", Edit screen font

`.minibuf.font`
: "fixed", Minibuffer screen font

`.screen.foreground`
: Black, Edit screen foreground color

`.minibuf.foreground`
: Black, Minibuffer screen foreground color

`.screen.geometry`
: "80x24", Geometry of the initial edit screen

`.minibuf.geometry`
: "80x1", Geometry of the minibuffer screen

`.screen.name`
: "Edit", Name of the edit screen

`.minibuf.name`
: "Minibuffer", Name of the minibuffer screen

`.screen.resource`
: <special>, Resource name of the edit screen

`.minibuf.resource`
: <special>, Resource name of the minibuffer screen

`.screen.reverse`
: False, Reverse colors for edit screen

`.minibuf.reverse`
: False, Reverse colors for minibuffer screen

`.screen.icon.name`
: "", Icon Name

2.1.2 Command Line Options

Epoch also accepts command line options. These override values in the X defaults. A complete list is in the table below. Note that the program resource name/class can be overridden by specifying different values on the command line (see Section 2.1.1 [X Window Program Defaults], page 5 for how the name and class are used).

Option	Resource, Effect
-bg	*background, Sets background colors
-background	*background, Sets background colors
-fg	*foreground, Sets foreground color
-foreground	*foreground, Sets foreground color
-geometry	*screen.geometry, Sets X-window geometry
-fn	*font, Sets the font
-font	*font, Sets the font
-nm	*nonlocal.minibuf, Selects local minibuffer to each edit screen
-wn	*name, Sets the X-window title
-window	*name, Sets the X-window title
-d	*display, Sets the display to use
-display	*display, Sets the display to use
-r	*reverse, Reverses foreground / background colors (no argument)
-rn	NA, Sets the resource name
-resource	NA, Sets the resource name
-name	NA, Sets the resource name
-cn	NA, Sets the resource class
-class	NA, Sets the resource class
-ud	NA, Looks at $HOME/.Xdefaults for program defaults
-motion	*motion, Turns on passing motion events to elisp for all screens
-xrm	NA, Overrides a resource manager value

2.2 New Features

This section provides a brief overview of the new features in Epoch. More detail is provided in subsequent chapters.

In this document, the term *screen* will mean an X-window, and *window* will refer to an editor window to avoid confusion. In most cases, working within a screen works the same as in GNU Emacs.

At any time there is a single *selected screen*, similar to having a selected window. This can also be called the *current screen*. Standard window functions work, but only apply to the current screen. E.g., `one-window-p` returns non-`nil` if there is only one window on the current screen. Buffers are shared among all screens, so any buffer in any window can be accessed, and will show up in buffer lists. Killing buffers removes them from all windows.

Epoch supports two different kinds of minibuffers. The default is a *non-local* minibuffer which is displayed in its own distinct screen; this is what Epoch has traditionally done. The alternative is to have minibuffer windows *local* to each edit screen; this is similar to traditional GNU Emacs. In the case of *non-local* minibuffers, there will always be exactly one minibuffer screen, and one or more edit screens; a single edit screen and minibuffer screen together act similarly to normal GNU Emacs. For *local* minibuffers, the real minibuffer will be located at the bottom of the current edit screen.

Buttons are regions of text in a buffer than can have data attached to them and be displayed in different styles (background/foreground color, stipple pattern, font) than surrounding text. Buttons provide both the ability to do highlighting and also to attach data to text regions that can be retrieved based on buffer location.

Epoch supports the use of either fixed-size or variable-sized (*proportional*) fonts for display. This includes support for mixtures of fonts (using buttons) which feature potentially different attributes (default or quad width, ascent, descent, etc).

Epoch supports event sending and reception, including client messages and X-window properties, and asynchronous reception of client and property change events. This allows Epoch to communicate with other clients or the window manager.

2.3 Key Bindings

We provide a large range of useful functions to create, delete and change the attributes of screens (including changing colors, size, etc). We provide a default set of key bindings to these functions, shown below. The basic idea was to take the `C-X` binding for Emacs window and buffer operations, and provide analogs for screen operations using `C-Z` as the prefix key. (We use `C-Z` because a suspend operator is not all that useful in a multi-window environment, and anyway `C-X C-Z` still provides the same effect as the standard.) We do not believe we have a full set of bindings yet, and look forward to user contributions.

We supply a standard set of elisp files See Section 7.3 [Packages], page 66. They provide a good

introduction to using the primitive Epoch facilities described in this document. The file '`epoch.el`' installs the basic Epoch key bindings. See below for more information.

KEY	Functionality
`C-Z 4 F`	`find-file-other-screen` Find a file in another screen. Prompts for a filename. If it is already in a window on a screen, warp the cursor there, otherwise create a new screen and edit the file in it.
`C-Z 4 B`	`find-buffer-other-screen` Selects a buffer in another screen. Prompts for a buffer name. If it is already in a window on a screen, warp the cursor there, otherwise create a new screen with one window for the buffer.
`C-Z 0`	`remove-screen` Delete current screen.
`C-Z 0`	`switch-screen` Circulate through mapped screens in forwards direction.
`C-Z P`	`prev-switch-screen` Circulate through mapped screens in reverse direction.
`C-Z 2`	`duplicate-screen` Duplicate current screen.
`C-Z R`	`raise-current-screen` Raise current screen
`C-Z L`	`lower-current-screen` Lower current screen
`C-Z M`	`raise-minibuf` Raise minibuffer screen
`C-Z I`	`iconify-screen` Iconify current screen
`C-Z E`	`display-event-status` Display event handler status

2.4 Conventions

Several conventions are used in the following chapters.

- Functions and variables are described with the format used in the Emacs Lisp Reference Manual.
- Any *screen* argument can be a screen ID or a screen object. Optional screen arguments default to the current screen.
- Any *color* argument can be an X Resource of type X-Cardinal See Section 6.1 [X Resources], page 57, a vector or a string. Strings and vectors are to X-Cardinals converted internally.

Each epoch primitive has a leading '`epoch::`' in a fashion reminiscent of Common Lisp packages, e.g., `epoch::create-screen`. The point is that we want to build around these functions at the elisp level, so we deliberately make the primitives uncomfortable to use. The files '`wrapper.el`' and '`epoch.el`' provide a full set of wrappers that remove the '`epoch::`' leading characters and add additional functionality (e.g. there is a function `create-screen` that provides a higher-level

interface to the primitives). If you are going to write your own elisp code you should probably use these wrapper functions, not the raw primitives.

2.5 Menus

Unlike GNU Emacs and some other X applications, Epoch provides no internal support for pop-up menus or menu bar facilities. We feel that it is more appropriate for X11 clients to depend on the window manager (or some sort of *widget server*) for menu support.

Two primary options exist for creating pop-up menus under Epoch: requesting the Window Manager (i.e. GWM) to supply them, or using an external X client (XMENU) to provide them. Examples of each method are available in the `epoch/contrib` directories.

2.5.1 GWM

GWM (generic window manager) is an Emacs-like window manager in that it provides a basic kernel of window-managing facilities written in C and a lisp level for extensive user customization. GWM is part of the X11 release (the most up-to-date version is always available from '`expo.lcs.mit.edu`').

One can attach state machines to objects (such as X-windows) in GWM, and these then receive and handle events. GWM can receive and initiate property change events, as can Epoch, so this provides the basis for communication between the two.

GWM also provides the ability to decorate windows with up to four bars (one on each side, top and bottom). Each bar can have "plugs" (icons) which can have specialized state machines attached. This allows one to have title bars, bars with pulldown menus, etc, all maintained by GWM not by the client, but with communication allowing selections to be communicated.

GWM is easily tailored to emulate other window managers and comes with reasonable emulations of MWM and TWM as well as a "home-grown" look-and-feel, so most users can use it as a replacement for previous managers quite easily.

The interested reader should see the GWM documentation for more details,

2.5.2 XMENU

XMENU is a toolkit program which producesa popup menu taking command-line arguments as menu entries. It is possible to execute xmenu as a process from Epoch (or GNU Emacs) and wait for the process output. XMENU is included in the directory `epoch/contrib/xmenu`.

3. Screens

This chapter describes screen features supported by Epoch, except for the elisp extensions described in Chapter 7 [Miscellaneous], page 65.

3.1 Screen Basics

In this document, the term *screen* will mean an X window, and *window* will refer to an editor window to avoid confusion. In most cases working within a screen will be the same as in GNU Emacs.

Each screen contains its own Emacs window hierarchy. (This terminology is a little confusing: when GNU Emacs uses the term *window* this refers to the logical structure in which a buffer is displayed. The name "window" was already appropriated, so we had to call our "X-windows" something different; they are therefore called *screens*.) When a screen is first created, it contains a single window with its mode line covering the entire screen. The buffer for this window is set to `*scratch*` unless a different buffer is specified.

Epoch supports two types of screens: *edit screens* and *minibuffer screens*. There will always be at least one edit screen. If Epoch is operating with non-local minibuffers, then there will always be exactly one minibuffer screen; otherwise, a local minibuffer window will appear at the bottom of each edit screen. Emacs assumes there is at least one edit window at all times. A single edit screen and the minibuffer window together act much like normal GNU Emacs.

epoch::nonlocal-minibuffer Variable

This variable is set to `t` if there is a distinct minibuffer screen, `nil` otherwise.

epoch::synchronize-minibuffers Variable

If Epoch is operating with local minibuffers, and this variable is set to `t`, then the contents of all local minibuffer windows will be synchronized with the contents of the real minibuffer window. This variable defaults to `nil`.

At any time there is a single *selected screen*, analogous to the selected window, also called the *current screen*. Standard window functions work, but they only apply to the current screen. E.g., `one-window-p` returns non-`nil` if there is only one window on the current screen. Buffers are shared among all screens, so any buffer in any window can be accessed, and will show up in buffer lists.

Killing buffers removes them from all windows. The value of `epoch::global-update` determines whether Epoch updates non-current screens or not (see See Section 3.7 [Screen Updating], page 27).

Epoch adds *screens* as a new Emacs Lisp primitive type. Screens are printed as `#<screen` *n*`>`, where *n* is the *screen ID*. This ID is a sequence number attached to a screen when it is created. It serves primarily as a human convenience. Any argument to a primitive listed as *screen* can be either a screen object or a screen ID. However, when the result of a function is a screen, this means a screen object is returned unless specifically mentioned to do otherwise. Optional screen arguments default to the current screen unless otherwise specified.

A screen that is displayed is said to be *mapped*; if it is only displayed as an icon, it is said to be *unmapped*. By X Window System conventions, a screen is mapped iff its X window is mapped.

inhibit-initial-screen-mapping Variable

This variable, when bound and set to `t`, will inhibit the mapping of the first edit screen when Epoch is starting up. In this case, the only screen which will be mapped initially will be the minibuffer screen. If Epoch is run with local minibuffers, then the first edit screen will be mapped regardless. This variable is not bound by default, but can be set in the user's '`.emacs`' file.

There is one significant difference between screens and windows: screens are known to external programs (e.g., the X-server). This means that the GNU Emacs scheme for window activities is not sufficient for screen activities. In particular, the creation of screens is much more complex than window creation.

epoch::create-screen &optional *buffer alist* Function

Creates a new screen (unmapped), and returns it. If the *buffer* argument is present, that buffer is attached to the single window originally present in the screen, otherwise the `*scratch*` buffer is used. The *alist* argument is used to override screen creation defaults. See `*create-screen-alist-hook*` below.

create-screen &optional *buffer alist* Function

Calls `epoch::create-screen` but does some additional things. The alist passed to `epoch::create-screen` is a copy of *alist* appended with a copy of `epoch-mode-alist` and two additional properties. These additional properties are `icon-name` and `title` (which here default to *buffer name*`@`*system name*). Also see `*create-screen-alist-hook*` below.

create-screen-alist-hook Variable

This is a hook variable used by `create-screen`. Just before the call to `epoch::create-screen`, each function in the hook is called with the screen property alist `create-screen` has generated. Each function should return a new version of the alist to be used instead. Normally, the hook function will add or modify some of the entries in the alist. See the code in See Section 7.4 [Screen Pools], page 70 for an example.

3.2 Screen Properties

Screens have a large number of properties (or attributes) that can differ between screens, and are externally visible (e.g., colors, cursor shape and color, font, size, title, class). Most of the complexity of screen creation is involved in dealing with these properties. Epoch deals with *default-sets* which are collections of X window property types and values. When a screen is created, a *default-set* is used to set the various properties of the screen. For information on default sets See Section 2.1.1 [X Window Program Defaults], page 5.

In order to allow more variety in screens, Epoch provides more ways of overriding the global X default set. A global variable `epoch::screen-properties` contains an alist of screen properties. At screen creation time, a copy of the global default-set is made, and entries are overwritten with values from `epoch::screen-properties`. After that, the alist given as an argument to `create-screen` (if any) is used to override any previous values.

The order of determination of the default set for each screen is as follows (in increasing order of precedence):

1. X Window Program Defaults (see Section 2.1.1 [X Window Program Defaults], page 5)
2. Values in `epoch::screen-properties`.
3. Alist argument to `epoch::create-screen`.
4. Mode-specific values using `epoch-mode-alist`, if `create-screen` is used for creation.
5. Alist argument to `create-screen`.

epoch::screen-properties Variable

This variable should contain an alist, the key values being screen property keys. The primitive attempts to bypass bad entries, but if the alist is improperly formed, Epoch may experience bizarre failures.

epoch-mode-alist Variable

This variable allows you to associate a set of X defaults with an emacs mode. The form is a list, each element of which is a form (`mode-name` . *alist-of-defaults*). Whenever `create-screen` is called, the mode of the *buffer* argument (or the `scratch` buffer's mode if *buffer* is `nil`) is used to look in the alist.

This feature allows you to set geometry, colors, font, etc by mode. It's useful to allow (for example) 80x60 windows for '`.c`' and '`.h`' files, but a regular size window for other file types. See the definition of '`epoch-mode-alist`' in '`epoch.el`' for some examples.

Thus you can set up a default mode to epoch upon entry and then override portions of it at the time you call `create-screen`.

The table below is a list of all screen property keys recognized by the screen creation routines. The defaults listed are the hard-wired defaults, but they may be superceded as described in Section 2.1.1 [X Window Program Defaults], page 5. Values listed as `flag` are false if the value is `nil`, true for any other value.

As an example, to set screens to have the font '`9x15`' and the '`Gumby`' cursor, the following Elisp code would be used:

```
(setq epoch::screen-properties
  (cons '(font . "9x15")
        (cons '(cursor-glyph . 56) epoch::screen-properties)))
```

`Name` Type, Default, Description

`foreground`
String, Black, Screen foreground

`background`
String, White, Screen background

`cursor-color`
String, Black, Text cursor color

`cursor-foreground`
String, White, cursor foreground color

`cursor-background`
String, Black, cursor background color

`border-color`
String, White, X border color

`in-border-width`
: Number, 2, Internal border width

`ex-border-width`
: Number, 1, External border width

`title` : String, "Edit", Screen title

`name` : String, "epoch", Screen resource name

`class` : String, "Emacs", Screen class name

`update` : String, false, Flag to indicate non-local screen updating

`geometry` : String, 80x24, Geometry specifier. Size units are characters. Screen must be at least 2 lines high.

`cursor-glyph`
: Number, 86, X cursor glyph. Index into the cursor font.

`reverse` : Flag, False, Reverse foreground / background

`font` : String, "fixed", Font name

`icon-name`
: String, "", Name of Icon for Screen

`motion` : Flag, nil, `t` will cause motion-events to be sent to screen, `nil` will inhibit this.

`parent` : X resource, nil, If non-`nil`, must be an X "resource id". The new screen is created as a child of this resource. Useful to provide clipping-parents, for example when trying to put screen in a form of some kind. *This option is extremely dangerous and you should not use it if you don't know what you are doing.*

`initial-state`
: Flag, True, Starting state of screen. True means NormalState, false is IconicState.

3.3 Controlling Screens

The primitives in this section concern internal manipulations of screens.

epoch::get-screen &optional *screen* Function

Coerces a *screen* or screen ID argument to the corresponding screen object.

epoch::get-screen-id &optional *screen* Function

Coerces a *screen* or screen ID to the corresponding screen ID. Returns `nil` if the screen is deleted.

epoch::screen-p *screen* Function

Returns `t` if the argument is of type *screen*, `nil` otherwise.

epoch::screen-list &optional *unmapped* Function

Returns a list of all mapped screens, except the minibuffer which is never in the list. If the argument *unmapped* non-`nil` then unmapped screens are also included.

epoch::next-screen &optional *screen unmapped* Function

Returns the next screen in the internal ordering. With no arguments, it is the next screen after the current screen. A *screen* argument returns the next screen past the argument. Unmapped screens are not returned, unless the second argument *unmapped* is non-`nil`. The minibuffer screen is never returned.

epoch::prev-screen &optional *screen unmapped* Function

Same as `epoch::next-screen` except the internal screen list is traversed in the opposite order.

epoch::select-screen &optional *screen* Function

Makes the argument *screen* the current edit screen. The default is the next screen, as defined by `epoch::next-screen`. The minibuffer screen cannot be selected. Unmapped screens may be selected, but only explicitly or if only unmapped screens are left. Epoch attempts to remember what window was last selected in each screen, and selects that one. The hook **select-screen-hook** is run at this time.

Returns the selected screen, or `nil` if given a bad argument or if *screen* has been deleted.

Note: If a window is selected by `select-window`, an implicit `epoch::select-screen` is done to select the screen the window is on. This insures that the current window is on the current screen whether `epoch::select-screen` or `select-window` was used. In fact, this primitive actually works by selecting a window in the new screen.

select-screen-hook Variable

This hook is run by `epoch::select-screen`.

epoch::current-screen Function

Returns the current edit screen. The minibuffer screen is never returned.

epoch::minibuf-screen Function

Returns the minibuffer screen. If no distinct minibuffer screen exists, then `nil` is returned.

epoch::delete-screen &optional *screen* Function

Deletes a screen. Returns `t` if successful, `nil` otherwise. It is an error to delete the sole remaining screen, and Epoch will not allow this.

If you delete a screen and it is the current screen, it first uses `epoch::select-screen` to select a new screen.

BUG: Window managers can delete the last edit screen. If this happens, Epoch will crash and burn in short order. Window managers *should* obey the ICCCM standard and send a delete signal to the client; then Epoch can grab this and terminate gracefully.

3.4 Screens and Windows

Epoch supports all of the windowing ability of GNU Emacs on each screen. Every window is on a screen, and can never be moved from that screen. Epoch provides some primitives to help in the interaction between screens and windows.

NOTE: The concept of *selected-window* under Epoch changes slightly from that of GNU Emacs in that asynchronous events (i.e. Focus events) may cause the selected window to change without any user-originated command to do so. This will cause problems for elisp packages which make the assumption that the selected window can not change in the midst of their processing.

epoch::get-buffer-window *buffer* Function

Searchs for a window displaying *buffer* on all screens. Returns such a window if successful, `nil` otherwise. This is effectively an updated version of the normal primitive `get-buffer-window`, which searches only the current screen and minibuffer.

Note: selecting a window also selects the screen it is on. Remember that the window may be on an unmapped screen. This function has no wrapper since that would collide with the standard function `get-buffer-window`.

epoch::screens-of-buffer *buffer* Function

Returns a list of all screens on which *buffer* is displayed in a window.

epoch::screen-of-window &optional *window* Function

Returns the screen that *window* is on. Returns the current screen if *window* is `nil`.

epoch::first-window &optional *screen* Function

Returns the first window in canonical order for *screen*. Use `next-window` to find the other windows in the *screen*.

epoch::selected-window &optional *screen* Function

Returns the selected window for *screen*. This is the window that would be selected if the screen it is on were selected, or equivalently the selected window the last time *screen* was the current screen.

Note: There is no wrapper for this function, since that would collide with the standard function `selected-window`.

3.5 Variable sized Fonts

Epoch now specifies the dimensions of Emacs windows in pixels. The primitives `window-height` and `window-width` now calculate character dimensions based on pixel dimensions and the base font for that window. It is important to realize that only the pixel values are "constant"; character/line values will fluxuate based on the presence of any proportional fonts or variable-height lines.

NOTE: The behavior of Epoch with standard GNU Emacs elisp packages should be as expected if fixed fonts are being used; different behavior may be seen otherwise.

window-pixheight &optional *window* Function

Returns the height of window in pixels. Defaults to selected window.

window-pixwidth &optional *window* Function

Returns the width of window in pixels. Defaults to selected window.

window-pixedges &optional *window* Function

Returns a list of edge coordinates of window in pixels. Defaults to selected window.

window-height &optional *window* Function

Returns the height of window in characters. Defaults to selected window.

window-width &optional *window* Function

Returns the width of window in characters. Defaults to selected window.

window-edges &optional *window* Function

Returns a list of the edge coordinates of the window in characters. (LEFT TOP RIGHT BOTTOM). **NOTE:** LEFT and TOP will be approximate, according to the base font for the screen; RIGHT will be one more than the rightmost column in the window; and BOTTOM is one more than the bottommost row used by the window and its mode-line.

Epoch now provides pixel-based functions and variables analogous to several character-based functions:

current-pixel Function

Returns a pixel position corresponding to current location of point. Does not take window dimensions into consideration; may return pixel values past the right edge of the window.

move-to-pixel *pixel* Function

Moves point to the character position corresponding to pixel position *pixel*. Ignores values of *pixel* which are past the edge of the window.

fill-pixel Variable

Pixel column beyond which automatic line-wrapping should happen. Automatically becomes local when set in any fashion. If *fill-column* is set to nil, the value of *auto-fill-hook* is called if the current pixel position is past this value. The value of *line-fill-hook* is called if text insertion will cause the current line to extend past *fill-pixel*.

text-width *string* &optional *font* Function

Return the length of text in *string* as displayed using *font*, or the base font of the current screen and window.

3.6 Manipulating Screens

These primitives allow control of the windowing system aspects of screens. Most of them correspond directly to X-window calls.

Each of these functions return the screen, if successful, or `nil` otherwise.

The *screen-or-xwin* arguments may be screen objects or X resource object with type window (see Section 6.1 [X Resources], page 57).

epoch::raise-screen &optional *screen-or-xwin* Function

Raises the *screen-or-xwin* to the top of the display. This is called by `epoch::select-screen`.

epoch::lower-screen &optional *screen-or-xwin* Function

Lowers the *screen-or-xwin* to the bottom of the display.

epoch::map-screen &optional *screen-or-xwin* Function

Maps the *screen-or-xwin* onto the display. No effect if the screen is already mapped.

epoch::unmap-screen &optional *screen-or-xwin* Function

Unmaps the *screen-or-xwin*. This is for sophisticated users only, as it does not check to see if the screen is the edit screen or the minibuffer. If the unmapped screen is the edit screen, it remains so - you just won't be able to see it.

epoch::mapraised-screen &optional *screen-or-xwin* Function

This does a MapRaised call on the *screen-or-xwin*. This is equivalent to doing a map followed atomically by a raise. If no argument is present, it uses the current screen.

epoch::iconify-screen &optional *screen* Function

Sends ICCCM client message requesting iconification of screen (or current screen if no argument present).

Note:

- Some window managers are not ICCCM compliant and will not handle this correctly.
- ICCCM standard says that to uniconify, an `epoch::map-screen` should be used. This doesn't seem to work for many window managers at present.

epoch::screen-mapped-p &optional *screen* Function

Returns `t` if the *screen* exists and is mapped, `nil` otherwise.

epoch::screen-height &optional *screen* Function

Returns the height (in characters) of the *screen*. **NOTE:** there is no Epoch wrapper function for this primitive, as it would conflict with the existing GNU Emacs `screen-height` primitive. The height returned is calculated according to the base font for that screen.

epoch::screen-width &optional *screen* Function

Returns the width (in characters) of the *screen*. **NOTE:** there is no Epoch wrapper function for this primitive, as it would conflict with the existing GNU Emacs `screen-width` primitive. The width returned is calculated according to the base font for that screen.

epoch::change-screen-size &optional *width height screen* Function

Attempts to resize the *screen*, leaving the upper left corner fixed. The *width* and *height* are in characters, and default to the current width and height. This uses the `XResizeWindow()` call, which may be intercepted by the window manager. The *width* and *height* values are adjusted be be not more than 150. The *height* is adjusted to at least 1 for the minibuffer and 2 for edit screens.

epoch::font &optional *font screen* Function

If called with `nil` *font* argument, this function returns the current font. If called with `font` argument, it changes *screen*'s font to be that font. It will resize the screen so that it still has the same character geometry. It operates on the current screen if *screen* is `nil`. The result is a list of three elements: Font name, character-width, character-height in pixels.

epoch::title &optional *name screen* Function

If called with `nil` *name* argument, this function returns the X screen name for *screen* or the current screen. If `name` is a string, *screen*'s title is changed accordingly.

epoch::icon-name &optional *name screen* Function

If called with `nil` *name* argument, this function returns the X screen icon name for *screen* or the current screen. If `name` is a string, *screen*'s icon name is changed accordingly.

epoch::plane-size Function

Returns a cons pair. Car is plane width, cdr is plane height in pixels. (A plane is a "minor screen" on a display: on mono screens, there is just one plane, color screens sometimes have several).

epoch::screen-information &optional *screen-or-xwin* Function

Returns information about *screen-or-xwin*, or the current screen if *screen-or-xwin* is `nil`. The information is a list with the following elements:

1. X location of screen in pixels
2. Y location of screen in pixels
3. Width of screen in pixels
4. Height of screen in pixels
5. External Borderwidth in pixels
6. Internal Borderwidth; for an xwin, this is always 0.
7. Map state: `t` if normal, `nil` if iconic.

epoch::move-screen *x y* &optional *screen-or-xwin* Function

Sends request to window manager to move upper-left corner of *screen-or-xwin* (or current screen if no third argument) to given (*x*,*y*) coordinates in pixels. Note that some window managers may choose to ignore this request.

epoch::foreground &optional *color screen* Function

Set/get the foreground color of the screen. If *color* is `nil` then the function returns the current foreground color as an X Resource. Otherwise it attempts to set the foreground color, and returns `t` on success and `nil` otherwise.

epoch::background &optional *color screen* Function

Set/get the background color of the screen. If *color* is `nil` then the function returns the current background color as an X Resource. Otherwise it attempts to set the background color, and returns `t` on success and `nil` otherwise.

epoch::cursor-color &optional *color screen* Function

Set/get the text cursor color of the screen. If *color* is `nil` then the function returns the current text cursor color as an X Resource. Otherwise it attempts to set the text cursor color, and returns `t` on success and `nil` otherwise.

epoch::cursor-glyph &optional *glyph screen* Function

Set/get the X cursor glyph. If *glyph* is `nil`, returns the current glyph number, otherwise attempts to set the glyph.

epoch::flash-screen &optional *screen-or-xwin* Function

Flash the *screen-or-xwin*. If the argument is a screen object, the flashing is done by exchanging the foreground and background colors, pausing .25 seconds, and then exchanging them again. For an X-window resource (even if it is an Epoch screen), the flashing is done by inverting the pixels, pausing, and then inverting again. This is effective on a monochrome display, but is not as effective for color, depending on the foreground and background colors.

3.7 Screen Updating

Epoch allows you to control which screens are updated as their contents are changed. The default is to only update the current screen, which is maximally efficient. It is possible, however, to set things up so that all screens or only some subset of the screens are updated. This is useful if you are editing a buffer which is displayed on multiple screens, or if you have a shell or other process running in one buffer and you are working in another. Then you can set things up so that the screen with the shell continues to update as it runs, while you work in another screen. There are several ways to do this, all involving the value of the global variable `epoch::global-update`.

The functions and variables are:

epoch::global-update Variable

There are three classes of values for this variable.

- set to `nil`. Disables updating all but the local screen.
- set to `t`. Enables potential updates: particular screens to be updated conditionally based on the value of the screen-local update *flag*. This flag may be set using `epoch::set-update`, and is stored in the `epoch-mode-alist` variable.
- set to any other value: causes all screens to be updated. This is a shorthand for setting to `t` and then setting the update property on each screen individually.

epoch::set-update *arg* &optional *screen* Function

Sets update flag to *arg* for *screen*, or current screen if nil. *arg* must be `t` for update, `nil` for no update. Returns previous value of flag.

epoch::update-p &optional *screen* Function

Returns `t` if *screen*'s update flag is set, `nil` otherwise. Uses current screen if *screen* argument is not present. This function should be `epoch::updatep`.

In addition you can use the following primitives to help control when screens get updated:

epoch::redisplay-screen &optional *screen* Function

Forces a non-premptive redraw of argument *screen* (current screen if `nil`).

redraw-display Function

Clears current screen and redraws what should be there.

recenter &optional *line* Function

Center point in window and redisplay screen. If *line* is non-`nil`, put point on that line. Operates on current window.

epoch::set-screen-modified &optional *screen* Function

Mark the argument *screen* (current if `nil`) as modified for update next time Epoch gets around to an update.

4. Zones

4.1 Zone Basics

Epoch supports the notion of *zones* (A.K.A. *buttons*), which are regions that are displayed in a different *styles*. Two new data types are used for this: `zone` and `style`.

`zone` Each zone has the following characteristics:

- *start* The start of the zone. A marker.
- *end* The end of the zone. A marker. The zone covers all characters between *start* and *end*.
- *style* Style to be used for displaying the text of the zone.
- *read-only* If set to non-nil, the text of the zone cannot be directly edited. Attempts to insert or delete any character in the zone will generate an error. (See Section 4.7 [Read-only Zones], page 38)
- *data* Not used internally. This field can be any Elisp object, and is provided for the programmer to use to attach data to a zone for later retrieval.
- *transient* Flag to indicate zone should be ignored by undo operations.

`style` A style is an object that describes how to display characters. It has the following fields:

- *foreground* The text foreground color.
- *background* The text background color.
- *cursor-foreground* The character foreground color when the text cursor is on the character.
- *cursor-background* The character background color when the text cursor is on the character.
- *font* The font to be used for displaying the text in this style.
- *stipple* The stipple pattern to use for the text.
- *cursor-stipple* The stipple to use when the cursor is on the text.
- *background-stipple* The stipple to use for the background. Bits that are set in the stipple are displayed in the screen background color. Cleared bits are displayed in the style background color.

underline	The color to use for underlining. If not set, no underlining is done.
tag	Not used internally. This can be any Elisp object.

Zones were intended as a way of providing "hotbutton" capabilities in hypertext-like applications using Epoch as a front-end. In these cases the data field will hold information for the hotbutton. Zones can also be used for many other applications, of course.

Each zone is attached to a buffer. All buffer arguments in the Epoch primitives default to the current buffer. Arguments of type zone must zone objects. A package has been written to store zone information with buffers when they are written to files and then restore zones when the buffer is visisted (See Section 7.5 [Saving Zones], page 71).

NOTE: To maintain compatibility with older Elisp packages, you may do a `(require 'button)` to get "button" wrapper functions for the zone primitives. Old Elisp code may be converted to require zones by using the file '`convert-buttons.el`'.

Functions exist to access and change a zones start/end position or even the zone's buffer, while maintaining the correct position in the zone list. **NOTE:** Elisp code that alters zones by changing the vectors directly will no longer work correctly. See the file '`fix-3.2.el`' for a function which will convert such code.

4.2 Zone Primitives

epoch::make-zone Function

Used to create a new zone. Returns a zone pointing nowhere, in no buffer.

epoch::zonep *object* Function

Returns `t` if object is a zone, otherwise `nil`

epoch::zone-buffer *zone* Function

Returns the buffer a zone is currently in, or `nil`.

epoch::zone-start *zone* Function

Returns the zone's *starting* position as an integer.

epoch::zone-end *zone* Function

Returns the zone's *end* position as an integer

epoch::move-zone *zone* &optional *start end buffer* Function

Moves the specified zone to a new position indicated by *start* and *end* in *buffer*. If either parameter is `nil`, it will remain unchanged. The zone start and end will be swapped, if necessary, so that the start comes before the end. Moving zones will cause the respective buffer(s) to redisplay, but will **not** mark the buffer as modified unless `epoch::zones-modify-buffer` is `t`. Returns the updated zone.

start and *end* may be markers, but only the marker position is used.

epoch::add-zone *start end attribute* &optional *data buffer* Function

Add a zone to *buffer*. *start* and *end* must be markers or integers. *attribute* is a number. *data* is programmer defined, and not used internally, and so can be anything. Returns the new zone, or `nil` on failure.

epoch::zone-at &optional *position buffer* Function

Returns the zone in *buffer* that contains *position*, or `nil`. If *position* is `nil`, point is used.

epoch::zones-at &optional *position buffer* Function

Returns a list of zones in *buffer* that contain *position*, or `nil`. If *position* is `nil`, point is used. The zones are ordered by starting position.

epoch::zone-list &optional *buffer* Function

Returns the list of zones for *buffer*. The list contains the actual zones, not copies. The zones are ordered by starting position.

epoch::zone-style *zone* Function

Returns the zone's current *style* value.

epoch::set-zone-style *zone value* Function

Sets the zone's *style* to *value*.

epoch::zone-data *zone* Function

Returns the zone's current *data* value.

epoch::set-zone-data *zone value* Function

Sets the zone's *data* to *value*.

epoch::zone-transient-p *zone* Function

Returns `t` if *zone* has its transient flag set; else `nil`.

epoch::set-zone-transient *zone zone flag* Function

Sets the zone's *transient* to *flag*. This is useful if the zone should be ignored by undo operations.

zone-text &optional *zone* Function

Returns a string that is the text contained in *zone*. If *zone* is omitted, the `zone-at` point is used, if any. Returns the empty string for an empty zone or no zone.

epoch::zones-modify-buffer Variable

If non-`nil`, changing any zones with Epoch primitives will cause the associated buffer to be marked as changed. Useful if you want `write-file-hooks` that save zone information.

4.3 Deleting Zones

epoch::delete-zone *zone* Function

Removes the *zone* from the buffer that it is in. The argument must be the actual zone itself, not a copy. This can be obtained by saving the result from `epoch::make-zone` or other zone returning primitives. Returns the zone if successful, `nil` otherwise.

epoch::delete-zone-at *position* &optional *buffer* Function

Removes the zone (if any) that contains *position* in *buffer*. Returns the zone if successful, `nil` otherwise. Equivalent to `(epoch::delete-zone (epoch::zone-at position buffer) buffer)`.

epoch::clear-zones &optional *buffer* Function

Removes all zones from *buffer*. Returns `t` if succesful, `nil` otherwise.

4.4 Style Primitives

Epoch supports a completely style-based display, where styles describe the context for displaying normal text, text within zones, text cursors, and modelines. Each style has a number of different fields. Foreground and background colors control how normal characters are displayed in the style. Cursor foreground and background indicate how a character in the text cursor is to be displayed. If cursor colors are not set but text colors are, the cursor will use the inverse of the text colors. The underline color controls if the characters are underlined. If set to a color, a line of that color is drawn under characters using that style. If set to `nil`, no line is drawn. A stipple pattern must be a X-bitmap resource (See Section 6.2 [X Objects], page 58). For text, the on bits in the stipple are displayed in the text foreground; the off bits in the text background. The character background is either solid background color, or the on bits are displayed in the screen background color and the off bits in the text background. For the cursor, the cursor colors are used, and the cursor stipple (if any) is used in place of the background stipple for the cursor background. If the cursor stipple is not set and the background stipple is, the cursor background is not stippled.

When displaying text within a given buffer, a hierarchy of styles exists for determining display attributes:

1. Zone styles
2. Buffer-local styles
3. Screen based styles

NOTE: Screen based styles are changed via primitives such as `epoch::foreground` `epoch::background`, `epoch::cursor-color`, `epoch::font`. All style fields are guaranteed to be valid at the screen level.

epoch::make-style Function

Create a style object. Initially all fields are set to `nil`.

epoch::stylep *object* Function

Return `t` if *object* is a style, else `nil`.

epoch::style-foreground *style* Function

Returns the foreground color of *style*.

epoch::set-style-foreground *style color* Function

Sets the *style* to foreground *color*.

epoch::style-background *style* Function
Returns the background color of *style*.

epoch::set-style-background *style color* Function
Set the *style* to background *color*.

epoch::style-cursor-foreground *style* Function
Returns the cursor foreground color of *style*.

epoch::set-style-cursor-foreground *style color* Function
Sets the *style* to cursor foreground *color*.

epoch::style-cursor-background *style* Function
Returns the background cursor color of *style*.

epoch::set-style-cursor-background *style color* Function
Set the *style* to cursor background *color*.

epoch::style-underline *style* Function
Return the underline color.

epoch::set-style-underline *style color* Function
Set the *style* underline to *color*.

epoch::style-stipple *style* Function
Return the text stipple for the *style*.

epoch::set-style-stipple *style stipple* Function
Set the text stipple pattern for *style* to *stipple*.

epoch::style-background-stipple *style* Function
Return the background stipple for *style*.

epoch::set-style-background-stipple *style stipple* Function
Set the background stipple pattern for *style* to *stipple*.

epoch::style-cursor-stipple *style* Function

Return the stipple pattern for the cursor for *style*.

epoch::set-style-cursor-stipple *style stipple* Function

Set the cursor stipple pattern for *style* to *stipple*.

epoch::style-font *style* Function

Return the font associated with *style*, or `nil` if none is specified.

epoch::set-style-font *style font* Function

Set the font for *style* to be *font*.

epoch::style-tag *style* Function

Return the tag value associated with *style*, or `nil` if none is specified.

epoch::set-style-tag *style tag* Function

Sets the tag value associated with *style* to *tag*.

motion::style Variable

This style, which defaults to underlining, is the style used by Epoch mouse dragging code found in '`motion.el`'. It's value is set at runtime, and may be superceded by values in the file '`.emacs`'.

The following code is used to set up the default style for mouse dragging, and demonstrates the use of Epoch and style primitives.

```
(setq motion::style (make-style))
(set-style-foreground motion::style (foreground))
(set-style-background motion::style (background))
(set-style-underline motion::style (foreground))
```

buffer-style Variable

This variable is set to the style used by Epoch to display text in this buffer. It defaults to `nil`, and becomes local when set in any fashion.

The variable *buffer-style* may also be set using the local variables list in files. The following

code when included in a TEX file would specify a major mode of `tex-mode` and a buffer style of `tex-mode-style` (which could select a proportional font):

In '`.emacs`'

```
(setq tex-mode-style
  (let ((s (make-style)))
    (set-style-font s "variable")
  )
)
```

In file:

```
#;;; Local Variables: ***
#;;; mode: tex ***
#;;; buffer-style: tex-mode-style ***
#;;; End: ***
```

4.5 Modelines

It is also possible to embed styles into the variables which define modeline format. For details on mode line constructs, see section 20.3 of the Emacs Lisp Reference Manual. Note that mode lines being displayed with styles will override the value of `mode-line-inverse-video`.

mode-line-inverse-video Variable

If this variable is non-nil, the modeline will be displayed in inverse video, or other suitable display mode.

default-mode-line-format Variable

This variable holds the default *mode-line-format* for buffers that do not override it. This is the same as `(default-value 'mode-line-format)`.

mode-line-format Variable

The value of this variable is a mode line construct responsible for the mode line format. This variable is buffer-local.

The following is an example of elisp code to embed styles in mode lines.

```
;; hilight "Epoch" and buffer name in current buffer's modeline.
;; s1 - hilight style
(setq s1 (make-style))
(set-style-foreground s1 "red")
(set-style-background s1 "white")
(set-style-font s1 "fixed")

;; s2 - normal style
(setq s2 (make-style))
(set-style-foreground s2 "black")
(set-style-background s2 "white")
(set-style-font s2 "variable")

;; define modeline format
(setq mode-line-format
  (list
    s2 "" mode-line-modified
    s1 mode-line-buffer-identification s2
    "   " global-mode-string "   %[(" mode-name minor-mode-alist
    "%n" mode-line-process ")%]----" (cons -3 "%p") "-%-"
))
```

4.6 Zone Plotting

When a buffer is plotted, zones are extracted from the buffer zone list and used to change the colors of the text and cursor. If the zones are disjoint, then which zone to use for a character is unambiguous. Overlapping zones are more complex; the algorithm for deciding which style are used when zones overlap is described next.

The zone plotter assumes that the zone list is sorted by starting position. If zones overlap, the zone with the largest starting position takes precedence. This allows zones to nest, so that if one zone is entirely inside another, both zones are visible. If two zones overlap without containment, the second zone will be entirely visible, and the first truncated. Whenever there is a reference to the zone "at" a location, that means the zone that was plotted at that location.

A zone with a non-null style always has precedence over a zone with null style. This means that if a zone with an null style follows and overlaps a zone of non-null style, the preceeding zone will be plotted in the overlap region. Null styles are intended for hidden or non-user zones (such as for tagging a region of text with data), and this rule prevents them from interfering with visible zones.

4.7 Read-only Zones

In the case of a *read-only zone*, (the *read-only* field is `t`), most attempts to change text within the zone will result in an error analogous to changing a read-only buffer. Two exceptions to this rule exist at zone boundaries.

1. Attempted changes on the first character of the zone will be allowed, but will be diverted to the left of the zone itself, so the zone is not extended.
2. Attempted changes to the immediate right of the zone will be permitted, but the zone will not be extended. This is really a change outside of the zone.

Essentially, this implies that the size of a read-only zone will only change because of explicit commands to change the zone boundaries; **not** by inserting or deleting text within the zone. Changing the location of a read-only zone by altering text previous to the zone is permitted; this is even possible with adjoining read-only zones (unless they explicitly overlap).

add-read-only-zone *start end attribute* &optional *data buffer* Function

The same as `epoch::add-zone` except that the zone is set to be read-only immediately after being created.

epoch::read-only-region-p *start end* &optional *buffer* Function

Given a region delimited by *start* and *end* in *buffer*, check for read-only zones. Returns `nil` if region contains no read-only zones; otherwise it returns the zone. In cases of overlapping zones, the display rule discussed earlier is obeyed (See Section 4.6 [Zone Plotting], page 37).

NOTE: When checking for insert operation, use the insertion point for both start and end arguments. If this would normally be inside the zone, but because of insertion rules would be deflected to the side of the zone, this function will not return the zone.

epoch::zone-read-only *zone* Function

Return's the zone's current *read-only* value.

epoch::set-zone-read-only *zone flag* Function

Sets the zone's *read-only* flag to *flag*, which must be either `t` or `nil`.

4.8 Graphical Zones

Epoch now supports displaying graphical images (X pixmaps) in zones. This requires the XPM version 3 library from Groupe Bull, which is on the X11R5 contrib tape, and is available via anonymous ftp from `export.lcs.mit.edu` (18.24.0.12) and `avahi.inria.fr` (192.5.60.47). To enable graphical zones, you must compile Epoch with HAVE_GRAPHIC_ZONES enabled in '`config.h`'.

NOTE: You may need to edit '`button.c`' to include '`xpm.h`' appropriately.

Once Epoch has been built in this manner, several new elisp primitives are available:

add-graphic-zone *pixmap-name start end* &optional *offset data buffer read-only* Function

Adds a graphical zone corresponding to a pixmap named *pixmap-name* from *start* to *end* in *buffer* (or current buffer). Zone will have data value *data*, and will be set read-only if *read-only* is non-nil. *offset* describes the percentage of the graphical image which will appear below the baseline; 0 (default) corresponds to an image totally above the baseline, 100 corresponds to totally below. *pixmap-name* can also be an X Resource of type pixmap. Returns the zone if successful.

epoch::define-opaque-font *name width height* &optional *offset* Function

Defines a psuedo font for displaying graphical images. Font *name* will have width *width*, and height *height*. As above, *offset* describes the percentage of the font which will appear below the baseline. Returns the *name*, or `nil`.

epoch::read-pixmap-file *filename* Function

Using the Xpm library, read in *filename* and create an X pixmap corresponding to it. Return an X Resource of type Pixmap if successful, otherwise `nil`. *filename* must correspond to the complete filename (including extensions) of a text file in the X Pixmap Format, or else an X Resource of type Pixmap.

epoch::query-pixmap *pixmap* Function

Returns a list of the form (width height depth) corresponding to an X pixmap. *pixmap* should be an X Resource of type Pixmap.

epoch::set-style-pixmap *style pixmap* Function

Set style *style* to reference *pixmap*. *pixmap* should be an X Resource of type Pixmap, or `nil`.

epoch::style-pixmap *style* Function

Return the pixmap associated with *style*.

Note: If a buffer containing a graphical zone is being displayed in a window which is too small to display the entire graphical image, in most cases no text will be displayed in the window.

4.9 Zones and Undo

It is possible for undo operations to restore affected zones to their previous positions. This is primarily noticeable for undoing delete operations. This feature is enabled by default, but may be disabled by setting the buffer-local variable `undo-restore-zones`. Any zones with the *transient* field set to `t` will be ignored.

undo-restore-zones t Variable

If set to non-`nil`, then zone information will be stored with all delete operations so that undo may restore the zones to their previous positions. This variable becomes buffer-local when set in any fashion.

5. Events

Epoch provides the ability to handle X window system events in Elisp.

5.1 Event Basics

Events are the mechanism the X window system uses to communicate with client programs (such as Epoch). Certain types of X events, listed below, are stored into a queue, the *event-queue*. Whenever such an event is received by Epoch, it is stored in the event-queue, and an *event-key* is inserted into the keyboard buffer. Whenever the keyboard is checked for keys, these event keys are read and processed until there are none left or an actual key is encountered. For each of these event, the variable `epoch::event` is bound to the value of the event, and the event handler `epoch::event-handler` is called. (See Section 5.2 [Basic Event Handling], page 43 to learn more about that subject.) Each event is returned as a vector of 4 elements. The first is the event type, the second is data associated with the event, the third is the screen on which the event occured, and the fourth is the time-stamp indicating the time the event occured (will be `nil` for `focus`, `map`, `resize`, and `client-message` events)

Epoch currently captures the events described in the following table:

Event	Data
`focus`	`t` for focus-in, `nil` for focus-out
`property-change`	If `epoch::lazy-events` is `nil`, then a cons pair. Car is name of property (a string), cdr is new value of property. Otherwise, the value is just the X-atom resource of the property. (See Section 5.4 [Property Change Events], page 46)
`selection-clear`	*atom* corresponding to selection name (See Section 5.5 [Selection Events], page 47)
`map`	`t` for map, `nil` for unmap
`move`	Cons pair: x-position, y-position in pixels
`resize`	List of 3 elements: new width, new height, outer borderwidth
`client-message`	A triple of the form (*TYPE FROM . DATA*). See description of `epoch::send-client-message` below for an explanation. (See Section 5.6 [Client Message Events], page 49).
`button`	A 5-tuple of form (*Press/Release x-coord y-coord button mod-state*). (See Section 5.7 [Mouse Events], page 50)
`motion`	A triplet of the form (*x-coord y-coord mod-state*). (See Section 5.8 [Motion Events], page 54).

To support asynchronous communication through events, Epoch provides primitives to set and read X Properties (See Section 6.2.2 [X Properties], page 60), and a property event handler. Primitives are also provided to send client messages.

Client messages are sent through the `epoch::send-client-message` function and received through events of type `client-message`.

epoch::lazy-events Variable

If this variable is non-`nil`, then the internal event code will do as little work as possible. Primarily, property events will only report the atom of the property changed, and leave it up to other code to acquire the value if desired. (See Section 5.4 [Property Change Events], page 46)

epoch::mouse-events Variable

If this variable is non-`nil`, then mouse events are placed into the Epoch event queue instead of generating fake key strokes.

5.2 Basic Event Handling

Events are handled by binding the variable `epoch::event-handler` to a function of no arguments. Whenever an event is dispatched as described above, if `epoch::event-handler` is non-`nil`, the value in it is called as a no argument function. This function should then examine the value of `epoch::event` and take appropriate action.

If any error occurs while the event handler is being executed, then the behaviour depends on the value of the variable `epoch::event-handler-abort`. If this is `t`, then the `event-handler` is aborted by resetting the variable to `nil`. This action prevents runaway errors on every following event. If the variable `epoch::event-handler-abort` is `nil` then this action is not taken.

In order to make event handling easier, a standard set of event handling functions are provided. These provide a (somewhat) widget-like interface to events. Selected parts of the interface can be replaced without change to the underlying internals. Event handling is organized hierarchially – the top level handler invokes other handlers to dispatch the final handling routines. The following description is from the top down.

The basic event handler maintains an internal alist to associate event types with event handlers. An *event type* is the symbolic name corresponding to a class of X events. An *event handler* is an elisp function that may be called when an event occurs.

For each event type, a stack of event handlers is kept, and when an event of that type occurs, the function at the top of the stack is called with arguments showed below. This allows you to change the event handler behaviour in a restorable way. Each basic event type must be installed before any events will be dispatched for it; this allows events to be globally disabled and reenabled. Currently the Epoch elisp files (see Section 7.3 [Packages], page 66) install all supported events.

When the basic handler runs, `epoch::event-handler-abort` is initially set to `t`. It then watches for errors during the dispatch of an event, which are detected by checking `epoch::event-handler` after returning from the dispatch. If `epoch::event-handler` has been set to `nil`, an error is presumed to have occured, and `ignore-event` is called on that event. This provides improvements over the built-in system, in that only a particular type of event is ignored (others continue to dispatch), and service can be restored with a simple (`resume-event` *event*), independent of the actual names of the handlers. This also means that handlers can now be anonymous, i.e. `push-event` can be passed a `lambda` expression instead of the symbol name of a function.

Event handler functions should be defined as follows:

```
(defun my-event-handler (event-type value screen)
  "event-type is the type of the event.  value is the value of
the event, and screen is the screen which received the event.
Handlers can call the event handling interface routines, so that a
one-shot handler (which pops itself) can be set up."
  )
```

The following functions and variables form the interface to the event handling system.

epoch::event-handler Variable

Variable that contains `nil` or a function to be called when there are pending property change events. The handler should use the variable `epoch::event` to retrieve the current event. If `epoch::event-handler` has been set to `nil` after an event has been dispatched, an error is presumed to have occured (see above).

epoch::event Variable

Just before the event handler in `epoch::event-handler` is called, this variable is bound to the value of the event that caused the handler call. It is a vector of size four, consisting of the event type, the event value, the screen where the event occurred, and the event's timestamp (if available).

epoch::event-handler-abort Variable

If this variable is non-`nil`, then if an error occurs while the event handler is running, `epoch::event-handler` is set to `nil`. This prevents an error in some part of the event handler code from causing Epoch to lock up with excessive errors.

epoch::get-event Function

All events are stored in an internal event queue. This primitive returns the oldest event in the event queue as an Elisp vector of size three. The zeroth element is the type of the event, the first element is the value of the event, the second element is the screen on which the event occurred, and the third element is the event's timestamp (if available).

install-event *event* Function

This puts *event* into the handler alist. *event* can be anything, but unless it is `eq` to an event type, it will never be dispatched; instead it will be ignored.

remove-event *event* Function

Removes *event* from the handler alist. All handlers associated with it are lost.

push-event *event handler* Function

Installs *handler* for *event*. Handlers are installed in a stack, and the handler at the top of the stack is dispatched for the event.

pop-event *event* Function

Removes and returns the top level handler for *event*.

ignore-event *event* Function

Pushes a marker onto the handler stack for *event* so that the event is not dispatched.

resume-event *event* Function

Pops the ignore marker from the top of the handler stack for *event*. Note that these two functions are different from `remove-event` and `install-event` in that these allow temporary, transparent inhibition of events in a local area, while the latter have global effects.

display-event-status Function

Print the current event handler status into a buffer and display it in a window on the current screen. Normally bound to `C-z e`. The status keys are `H` for being handled, `I` for ignored, and `U` for uninstalled. If a number follows the letter, that is the depth of the handlers or ignore markers.

5.3 Advanced Event Handling

In addition to the standard event handling facilities just described, it is occassionally necessary to handle certain event types specially.

epoch::wait-for-event *event-type handler* Function

This primitive will wait for an *event-type* type to occur, and then call `handler` if non *nil*. `handler` should be a function defined similarly to other event handler functions See Section 5.2 [Basic Event Handling], page 43. Any other events which are received during the wait are stored and dispatched in the order of arrival after the desired event has occured. The function will return after this has occured. **NOTE:** This function could cause deadlock if the selected event type does not occur.

5.4 Property Change Events

The Epoch elisp files set up a handler for the `property-change` event which then dispatches based on the property, in a manner analagous to the basic event handling. This property event handler uses *lazy events* (see `epoch::lazy-events` in See Section 5.2 [Basic Event Handling], page 43.) This means that a property change event has a value of just the X-atom resource corresponding to the property that changed.

> Implementation note: Property events are passed as lazy events to save the expense of unneeded round-trip server requests. With non-lazy events, all property events would involve 2 round-trip requests, one for the atom and another for the value. However, most property events are not handled, and so such information is simply thrown away. With this system, only those property events that are actually dispatched invoke this expense, and in many cases the name of the property is unnecessary. (If the handler function is attached to only one property, then it implicitly knows the property name value.) The primitive `equal` has been extened to provide comparisons for X-Resources See Section 6.1 [X Resources], page 57.

Each installed property is stored as the X-atom resource and a stack of handlers. If the top entry is a function, then it is called with the same parameters as for basic event handling. (The event type will always be `property-change`, but this allows a consistent handler interface for all handler functions.) If the property is not in the alist, or the top entry is not a function, the property change is ignored.

In contrast to basic event handling, if the property is not installed, pushing the property handler will install it. The inhibition of handling for certain properties is not as useful as for events. Since the number of possible property events is far larger than the number of possible event types, the extra hassle of having to seperately install properties was judged to be too much. The `install-property` and `remove-property` should be used seldom, if at all, and are provided primarily for conformity with basic event handling.

Property change handler functions should be of the same form as the general event handlers. Note that the value of the property change is passed as an X-atom. If the value of the property is desired, the function `get-property` can be used. If the name of the property is needed, use `unintern-atom`.

One way that property change events can be used is to send messages to and receive messages from the window manager. We use this facility extensively to display menus to the user, change window names and decorations, etc. X properties are manipulated and accessed through the `epoch::set-property` and `epoch::get-property` functions (see Section 6.2.2 [X Properties], page 60).

Epoch maintains two properties on the screens automatically. These are `EPOCH_SCREEN_ID`, which is an integer that is the screen ID, and `EPOCH_CURRENT`, which is a string, either "yes" if that screen is the current edit screen, "no" if it is not, or "minibuf" if it is the minibuffer.

The following functions are the interface to the property event handling system.

install-property *property* Function

This puts *property* into the handler alist. *property* should be the name of an X Property (i.e. a string or X Atom).

remove-property *property* Function

Removes *property* from the handler alist. All handlers associated with it are lost.

push-property *property handler* Function

Installs *handler* for *property*. Handlers are installed in a stack, and the handler at the top of the stack is dispatched for the property. If the property is not already in the handler list, it is added.

pop-property *property* Function

Removes and returns the top level handler for *property*.

ignore-property *property* Function

Pushes a marker onto the handler stack for *property* so that the property is not dispatched.

resume-property *property* Function

Pops the ignore marker from the top of the handler stack for *property*.

5.5 Selection Events

There are three types of events associated with X11 selections, two of which may be handled in elisp (See Section 6.2.3 [X Selections], page 61). `selection-clear` events are handled based on selection name with a handler similar to other events. `selection-request` events are handled internally unless the requested target atom is non-standard.

epoch::convert-selection-hook convert-to-target Variable

The function named by this hook will be called for any selection-requests with non-standard target atoms.

add-selection-target *target function* Function

Adds *target* to list of user-defined target atoms. *function* will be called for conversion requests of this type.

convert-to-target *target* Function

Called by `epoch::convert-selection-hook` to call appropriate conversion function based on *target*. If no function exists, return `nil`.

install-selection *selection* Function

Puts *selection* on handler alist. *selection* should be the name of an X Selection (i.e. a string or X Atom).

remove-selection *selection* Function

Removes *selection* from handler alist. All handlers associated with it are lost.

push-selection *selection handler* Function

Installs *handler* for *selection*. Handlers are installed in a stack, with the handler at the top of the stack is dispatched for the selection. If the selection is not already on the handler list, it is added.

pop-selection *selection* Function

Removes and returns the top level handler for *selection*.

ignore-selection *selection* Function

Pushes a marker onto the handler stack for *selection* so that the selection is not dispatched.

resume-selection *selection* Function

Pops the ignore marker from the top of the handler stack for *selection*.

NOTE: If no handler is installed for a given selection, then `selection-clear` events received will be handled *generically*, with any data for that selection being deleted from `epoch::selection-`

`alist`. If a handler has been installed for a selection, it is the responsibility of that handler to remove data from `epoch::selection-alist` if this behavior is desired.

5.6 Client Message Events

The Epoch elisp files set up a handler for the `client-message` event which then dispatches based on the message type. Each installed property is stored as the X-atom resource and a stack of handlers. If the top entry is a function, then it is called, with the same parameters as for basic event handling. If the message type is not in the alist, or the top entry is not a function, the message is ignored.

The value of a client message is a list of (*type source* . *data*) where *type* is the message type X-atom resource, *source* is an X-window resource specifying the source window, and *data* is the data field of the message converted to Elisp data in the same manner as X property values (see Section 6.2.2 [X Properties], page 60). To send client mesages, see Section 6.2.1 [Client Messages], page 59.

If the message type is not installed, pushing a handler will install it. `install-message` and `remove-message` should be used seldom, if at all, and are provided primarily for conformity with basic event handling.

The following functions are the interface to the client message event handling system.

install-message *message* Function

This puts *message* into the handler alist. *message* should be either a string or X Atom.

remove-message *message* Function

Removes *message* from the handler alist. All handlers associated with it are lost.

push-message *message handler* Function

Installs *handler* for *message*. Handlers are installed in a stack, and the handler at the top of the stack is dispatched for the message. If the message is not already in the handler list, it is added.

pop-message *message* Function

Removes and returns the top level handler for *message*.

ignore-message *message* Function

Pushes an ignore marker onto the handler stack for *message* so that the message is not dispatched.

resume-message *message* Function

Pops the ignore marker from the top of the handler stack for *message*.

5.7 Mouse Events

Mouse events go into the event queue with event type `button` (which have nothing to do with epoch buttons) and event value the 5-tuple (*press/release x-coord y-coord button mod-state*). *press/release* indicates if the event was a button press or release; the coordinates are character coordinates within the screen; *button* indicates which button; and the *mod-state* is a bitmask indicating the state of the button modifiers.

Note: Mouse events are only received on the keyboard queue when `epoch::mouse-events` is `nil`. This is the default value, but the Epoch Elisp files set it to `t`.

The Epoch Elisp files install a handler for the `button` events, which in turn dispatches functions for each different mouse event.

Mouse event dispatching uses two tables, one global and the other local to each buffer. When a mouse event is dispatched, the local table is checked first. If the table is missing or the entry is `nil`, then the global table is used. If a function is found, then it is dispatched. The mouse event handler should be defined as follows:

```
(defun my-mouse-handler (mouse-data)
  "mouse-data is a list of ( point buffer window screen ).
These all refer to the character location
at which the button on the mouse was pressed or released.  Note that at
the time the handler is called,  no change to point, current buffer,
current window or current screen has been made.  It is entirely up to the
handler how much to change such information.")
```

When specifying a particular mouse event, both the mouse button and keyboard *modifier* states must be specified, e.g. the middle button with Control. Note that separate mouse button codes exist for button presses in the window, mode line, and minibuffer. These values are specified as numbers, but for ease of programming, constants are defined for them, listed below.

These are the predefined constants for specifying the mouse button: Note that these now match codes in `mouse::event-data`.

Name	Value
`mouse-left`	1
`mouse-middle`	2
`mouse-right`	3
`mouse-mode-left`	4
`mouse-mode-middle`	5
`mouse-mode-right`	6
`mouse-minibuf-left`	7
`mouse-minibuf-middle`	8
`mouse-minibuf-right`	9

These are the predefined constants for specifying the keyboard modifier state. Names without a trailing `-up` signify the down button event.

Name	Value
`mouse-down`	0
`mouse-up`	1
`mouse-shift`	2
`mouse-shift-up`	3
`mouse-control`	4
`mouse-control-up`	5

`mouse-control-shift`
6

`mouse-control-shift-up`
7

`mouse-meta`
8

`mouse-meta-up`
9

`mouse-meta-shift`
10

`mouse-meta-shift-up`
11

`mouse-meta-control`
12

`mouse-meta-control-up`
13

`mouse-meta-control-shift`
14

`mouse-meta-control-shift-up`
15

Note: This version of the mouse handling code is very different from the previous version, whose main advantage was that it was the same as regular Emacs. In that version, the passed argument was a list of the form (*X Y screen*) of the mouse event. This data was not particularly useful, and had to be converted into more useful data by actually setting point with the `x-mouse-set-point` call. A look through the mouse handling code indicated that, in fact, every function went ahead and did this in order to calculate the point.

The following functions are modeled after keymaps.

create-mouse-map &optional *source-map* Function
Creates a mouse map for use in dispatching. If *source-map* is non-`nil`, then the contents of that map are copied into the new map.

copy-mouse-map *source-map dest-map* Function
Copies a mouse map from *source-map* to *dest-map*.

use-local-mouse-map *mouse-map* &optional *buffer* Function
Sets *mouse-map* to be the local mouse map in *buffer*.

kill-local-mouse-map &optional *buffer* Function
Removes the local mouse map from *buffer*.

define-mouse *mouse-map button modifier function* Function
Sets the entry in *mouse-map* for the mouse *button* with *modifier* to be *function*. *button* should be one of the mouse button constants, and *modifier* should be one of the button modifiers.

local-set-mouse *button modifier function* Function
Sets the entry in the local mouse map for *button* and *modifier* to *function*.

global-set-mouse *button modifier function* Function
Sets the entry in the global mouse map for *button* and *modifier* to *function*.

mouse::set-point *mouse-data* Function
Sets the point to the value specified in *mouse-data*, which should be the same form as the list passed to mouse handler functions, (`point buffer window screen`)

coordinates-in-window-p *position window* Function
Returns `t` if position described by *position* is in *window*. *position* is a list of the form (screen-x screen-y).

epoch::coords-to-point *x y screen* Function
Converts a pixel (x, y) location on a given *screen* into a list of the form (`point buffer window screen`) or `nil` if the location is not on the *screen*. *point* is returned as `nil` if location is in window's modeline.

Data pertaining to the most recent button press event is available in the following variables. Note that this enables the detection of multiple mouse clicks.

mouse::interval 200 Variable
The number of milliseconds allowed between multiple mouse clicks. Interval is clocked between down-click and the previous up-click.

mouse::x Variable

X screen position of last mouse button press, in pixels.

mouse::y Variable

Y screen position of last mouse button press, in pixels.

mouse::last-spot Variable

Mouse data of the last event (point buffer window screen).

mouse::time-stamp Variable

Millisecond time of the last up-click.

mouse::clicks Variable

The number of times mouse button was pressed and released.

5.8 Motion Events

Epoch allows the user to receive motion events, which occur when the mouse moves. By default this feature is turned off. It can be enabled in a number of ways:

- using command line option `-motion`: makes motion events appear for all screens.
- using `*motion` in resources: makes motion events appear for all screens.
- using `motion` in the property alist; `t` turns on motion events at screen creation time, `nil` disables them.
- using primitive `epoch::set-motion-hints`: turns motion events on or off by screen.

Note that motion events go into the event-queue but must be specifically solicited by functions rather than being automatically provided. Each time you execute a function affecting the mouse, (for example `epoch::query-pointer`), the next motion event (if any) is sent to the primitive. This prevents motion events from coming so fast that Epoch would thrash to death.

The relevent primitives are:

epoch::motion-hints-p &optional *screen* Function

Returns `t` or `nil` for *screen* (default: current screen) depending if motion events are enabled or disabled for that screen.

epoch::set-motion-hints *flag* &optional *screen* Function

Flag must be `t` or `nil`; enables or disables motion hints for *screen* (default: current screen).

5.9 On Event Handling

As part of the standard event handling, a facility to install one-shot handlers for events on particular screens is provided. After the action function has been called for the particular event, it is removed.

on-map-do *screen action* Function

On the next map event for *screen*, call *action* with a single argument that is the screen of the event.

on-unmap-do *screen action* Function

On the next unmap event for *screen*, call *action* with a single argument that is the screen of the event.

on-move-do *screen action* Function

On the next move event for *screen*, call *action* with 2 arguments, the screen and the move event value.

on-resize-do *screen action* Function

On the next resize event for *screen*, call *action* with 2 arguments, the screen and the resize event value.

6. X11 Primitives

Various primitive X11 facilities are provided which we describe here.

6.1 X Resources

Epoch X Resources give the ability to work with raw "X resources", such as the X server's internal ID numbers for screens (X "windows"). These numbers are too big to fit into the standard GNU Emacs number size, so they are made a special opaque type called an *X-resource*. X-resources contain X-window id information (32 bit quantity), and a type, which is always an X window system atom. When the type of an Epoch X Resources is a predefined value (such as "X-Atom"), then it is referred to by that type, (e.g. X-atom resource). Currently recognized types are `Arc`, `Atom`, `Bitmap`, `Cardinal`, `Cursor`, `Drawable`, `Font`, `Integer`, `Pixmap`, `Point`, `Rectangle`, `String`, `Window`, `WMHints`, `WM Size Hints` or `Resource` (untyped).

epoch::intern-atom *name* Function

Interns the string *name* as an X-atom resource (as opposed to Elisp atom). *name* should be a string. X-atom resources are 32-bit numbers that represent strings in the X window system. They are used for typing information on X window data objects.

epoch::unintern-atom *name* Function

Converts the X-atom resource *name* to an Elisp string. Note that the atom is not removed from the server.

epoch::resourcep *arg* Function

Returns `t` if *arg* is an X resource, `nil` otherwise.

In addition, the primitive `equal` has been extended to accept X resources. Two resources are `equal` if their values are the same, regardless of type.

epoch::string-to-resource *string type* Function

Converts an elisp *string* to an X resource (raw window id, for example) of given *type*. *string* is assumed to represent a 32-bit numeric value in the C language numeric literal format. *type* must be an X-atom resource.

epoch::resource-to-type *resource* Function

Returns an X resource whose type is `atom` and whose value is the type of the *resource*.

epoch::resource-to-string *resource* &optional *base* Function

Convert the id of the *resource* to a numeric string. Optional *base* specifies the base for the conversion (may be 2..36 inclusive). Note that no special notation is used to signify the base.

epoch::xid-of-screen &optional *screen* Function

Returns the X-window resource value for the *screen* (default: current-screen).

epoch::set-resource-type *resource type* Function

Sets the type field of *resource* to the value of *type*. *type* must be an X-atom resource. Returns *resource* if it was successfully modified, nil otherwise.

6.2 X Objects

Several types of X objects can be encapsulated into Elisp. Each is stored in Epoch as its X-window resource ID, in an X-resource data object. When a function claims that it returns the X-window object, this should be taken to mean that it returns an X-resource with the resource ID and type stored in it. These resources can be manipulated with various functions, see Section 6.1 [X Resources], page 57.

epoch::get-font *name* Function

Loads the font into the server, and returns the X Resource associated with it, or `nil`.

epoch::get-color *color* Function

Allocates a slot in the X server default colormap for the *color*. Returns an X Resource of type Cardinal, which is the pixel value for the *color*. *color* can be a color name, a vector of length 3 specifying the red, green, blue components, or an X Resource of type Cardinal. If the last, it is simply returned (this is so that the function can be called on any representation without generating an error).

epoch::free-color *color* Function

Releases the X server colormap slot corresponding to the *color*. *color* must be an X Resource of type Cardinal. No checking is done to verify that the color was allocated

by Epoch, or is available for release. Once released, the pixel value can be reused for a different color, so that anything still displayed in the color may change at any time (through the action of another X client, or Epoch).

epoch::color-components *color* Function

Return a 3-vector containing the red, green and blue components of *color*, which should be an X Resource of type Cardinal.

epoch::make-bitmap *width height byte-string* Function

Creates a bitmap, and returns it. The *width* and *height* should be positive integers, specifying the size of the bitmap. There is no way to change the size once the bitmap has been created. The *byte-string* should be a string, which will be interpreted as an array of bytes, not characters per se. The format corresponds directly to the standard X11 bitmap format. Each row of the bitmap is in integral number of bytes, floor(*width*+7 / 8) long, and should consist of *height* rows.

epoch::free-bitmap *bitmap* Function

Releases a bitmap from the X server. The bitmap should *not* be used again, even indirectly (e.g., in a style).

6.2.1 Client Messages

epoch::send-client-message *to* &optional *from data type format* Function

Sends a client message of *type* with *data* to the destination *to* using *format* and *data*. The parameters have the following meaning:

to A screen or an X Resource of type Window. See Chapter 6 [X11 Primitives], page 57 for more information.

from A screen or an X Resource of type Window that serves as the source of the message. This defaults to the current screen.

data The message data may be `nil` if there is no data, integer, string, or a list or vector of integers or X-resources. If it is simply a string, the first 20 characters are transmitted, and interpreted as 1, 2 or 4-byte data on the other end according to the *format* parameter. If *data* is `nil`, 0s are sent.

type This is an X-atom (as opposed to an elisp atom) and can be created by `intern-atom`. Usually something which conveys the meaning of the message, e.g., `WM_CHANGE_STATE`.

format The data for a client message is always 20 bytes long. The format statement tells how to interpret this. The value must be one of the following:

`nil`	The format defaults to a value appropriate for the data, as determined by Epoch.
`8`	for 20 1-byte data objects.
`16`	for 10 2-byte objects.
`32`	for 5 4-byte objects.

```
;; Fake an iconify message
(setq xroot (car (query-tree))) ; get the root window
;; IconicState is 3
;; Send to the root window, from the current screen,
;; with data 3 of type WM_CHANGE_STATE.
(send-client-message xroot nil 3
  (intern-atom "WM_CHANGE_STATE"))
```

6.2.2 X Properties

epoch::get-property *name* &optional *screen* Function

Returns the X-Property *name* on the *screen*. *screen* can be an X-window resource. *name* can be an X-atom resource or a string. The property value is returned in the most (in Epoch's opinion) convenient form.

If the property is an array of items (e.g., the *count* return value is more than 1) then it is returned as a list. Strings are converted to Elisp strings, integers into Elisp integers (with a possible loss of precision, since Elisp integers are less than 32 bits), and other 32-bit types into X-resources. Strings are handled specially - if nulls are found in the returned byte array, they are assumed to represent string seperators, and that the property is an array of these null-seperated substrings.

If there is a failure (bad screen, no property, bad type, etc.), then `nil` is returned.

epoch::set-property *name value* &optional *screen* Function

Sets the X-property *name* to *value* on the X-window *screen*. *screen* can be an X-window resource. *name* can be an X-Atom resource or a string. *value* must be a string, an integer, an X-resource, or an vector or list of these types. If it is a vector or list, the types of all the sequence elements must be the same, and for X-resources the X-resource types must also be the same.

6.2.3 X Selections

epoch::get-selection-owner *selection* Function

Returns an X Resource of type Window corresponding to the X11 client owning *selection*. *selection* may be an X-Atom Resource or a string. If *selection* is not owned by any client, *nil* is returned.

epoch::acquire-selection *selection* &optional *screen* Function

Asserts ownership of *selection* for the X-Window *screen*. *screen* can be an X-Window resource. Uses current screen if no *screen* is given. If *screen* is `t`, set *selection* owner to `None`, and send Epoch a `selection-clear` event.

Note that *selection*'s data should be stored in `epoch::selection-alist` so that Epoch can internally handle conversion requests and possible loss of ownership to other X11 clients.

epoch::convert-selection *selection target property* &optional *screen* Function

Requests owner of *selection* to convert selection's data to type *target* and store result in *property*. Property will be hung on *screen*, or current screen if no *screen* argument is given. This function will wait internally for the selection value to be available, and will return this value if the selection is owned by a client and no timeout occurs waiting for the client response.

epoch::selection-timeout 2 Variable

This variable describes the amount of time (in seconds) for the epoch::convert-selection function to wait for a response before returning nil. This pause will not occur in the event the selection is unowned.

epoch::selection-alist Variable

This variable will hold an alist of (`atom . value`) for all selections owned by Epoch X-Windows. It is the responsibility of the elisp code under Epoch to store data in this alist upon asserting ownership of any selections. When `selection-clear` events are processed, the corresponding entry is deleted from the alist.

When `selection-request` events are processed internally, Epoch looks in this alist for selection data to be converted. Currently, selection data must be of type `String`; the selection-notify event structure is set to `None` for invalid requests or requests of selections with `null` data.

epoch::convert-selection-alist Variable

This variable will hold an alist of (`atom function`) for any user-supplied target atoms. *function* will be called with no arguments, and should return either a string or `nil`.

6.2.4 X Cursor

epoch::query-pointer &optional *screen-or-xwin* Function

Returns a list containing x-coordinate, y-coordinate (in pixels) and state of mouse. Uses current screen if no *screen* argument is given.

epoch::warp-pointer *x y* &optional *screen-or-xwin* Function

Warps cursor to (*x*,*y*) location (in pixels) on *screen* relative to the upper-left corner. Uses current screen if no *screen* argument is given.

epoch::ungrab-pointer Function

Ungrabs the pointer. This is useful after a mouse down event, so that another X client can act before the mouse up event.

epoch::query-mouse &optional *screen* Function

Returns a list containing x-coordinate, y-coordinate (in character position) and state of mouse. Uses current screen if no *screen* argument is given.

epoch::warp-mouse *x y* &optional *screen* Function

Warps cursor to (*x*,*y*) location (in character position) on *screen*. Uses current screen if no *screen* argument is given.

epoch::query-cursor &optional *screen* Function

Returns a list (*x* . *y*) corresponding to the cursor position in characters. Uses current screen if no *screen* argument is given.

epoch::query-cursor-pixels &optional *screen* Function

Returns a list (*x* . *y*) corresponding to the cursor position in pixels (Upperleft corner of cursor). Uses current screen if no *screen* argument is given.

6.3 Other X Stuff

epoch::get-default *name* &optional *class* Function

Does a lookup into X Resources database for *name* and *class* (if specified). Returns a string for the definition, or `nil` if lookup request failed.

epoch::rebind-key *keysym shiftmask string* Function

Rebinds a raw X key on the fly. Takes as arguments the *keysym* to rebind, the *shiftmask* for the rebinding, and a *string* to send when that key is depressed. *keysym* should be a string, naming the keysym. *shiftmask* should be one of the following:

`integer`	This is a bit-mask, with a bit for every modifier, just like an X shift state.
`symbol`	The symbol should be one of `'shift`, `'lock`, `'control`, `'meta`, `'mod1` ... `'mod5`. `'meta` and `'mod1` are equivalent.
`list`	A list of symbols, indicating multiple modifiers.

This function does not affect keybindings for other X clients, but does affect all Epoch screens.

epoch::mod-to-shiftmask *index* Function

Input is an X modifier *index*, output a shiftmask that can be passed to `epoch::rebind-key`. OBSOLETE.

epoch::query-tree &optional *screen-or-xwin* Function

Returns a list of the form `(root parent . children)` where all the elements are X-window resources. `root` is the X root window, `parent` is the X parent window of *screen-or-xwin* (`nil` if *screen-or-xwin* is the X root window), and `children` is a list of the X child windows of *screen-or-xwin*.

epoch::set-bell *arg* Function

If *arg* is `nil`, Epoch will use an audible bell. If non-`nil`, then a visual bell will be used.

epoch::bell-volume Variable

Controls the X bell volume. If a number from -100 to 100, then the value is used as the bell volume, otherwise it is ignored and 50 is used. Note that on many systems, the hardware does not support different bell volumes, and so this may not be effective.

7. Miscellaneous

Several unrelated features of Epoch are described here.

7.1 Standard Extensions

Epoch contains some extensions of GNU Emacs that are not specifically related to X Windows.

symbol-buffer-value *symbol buffer* Function

Returns the value of *symbol* in *buffer*, without the expense of using `set-buffer`.

epoch::function-key-mapping t Variable

If this variable is non-`nil`, then functions keys will be mapped into an extended form starting with ESC [, as is done in normal GNU-Emacs.

If this variable is set to `nil`, then no mapping will be done, and the key will be ignored. Keys rebound (see Section 6.3 [Other X Stuff], page 63, `epoch::rebind-key`) to a non-empty string will not be affected by this value. If set to `t` then keys which have not been rebound will be mapped to various escape strings.

equal *object1 object2* Function

This function returns `t` if *object1* and *object2* have equal components; `nil` otherwise. If *object1* and *object2* are both X Resources, see Section 6.1 [X Resources], page 57, then `t` is returned if their xid's are equal regardless of type.

7.2 Epoch Version

epoch::version Variable

This built-in variable will contain the version number of Epoch. Thus, `(boundp 'epoch::version)` will be `t` on epoch and `nil` on any other version of GNU Emacs. You can use this to customize your elisp code conditionally. The value of this variable will of the form

```
"Epoch 4.0 Patchlevel 0"
```

and will indicate the highest official patch applied.

7.3 Packages

Epoch comes with a set of standard elisp files, which provide functionality to the user. Beginning with version 3.2, a majority of the standard files are loaded when Epoch is built. See '`ymakefile`' and '`loadup.el`' for which files are loaded. Any code which can not be executed except at runtime may be loaded from '`.emacs`', or run via a hook:

epoch-setup-hook nil Variable

Functions to be executed at runtime, prior to '`.emacs`' being loaded and `term-setup-hook` and `window-setup-hook` have been run.

The following files are included in this version of the Epoch distribution, in the `epoch-lisp` directory:

'`dot.emacs`'
: Things that should go in your `.emacs` file, or `(load "dot.emacs")`. Pretty straightforward; read the code.

'`epoch.el`'
: Installs standard keybindings, variables that you can set to change things, etc. This file also defines the default event handler code.

'`epoch-util.el`'
: Some utility functions used throughout Epoch lisp code.

'`mini-cl.el`'
: Provides some common lisp primitives to Emacs lisp. This is a subset of the common lisp package provided by '`cl.el`'.

'`wrapper.el`'
: Every primitive function introduced into GNU Emacs to support Epoch is of the form `epoch::`*name*. This is like common lisp internal package symbols, and for each we provide a form without the leading `epoch::`. These are the forms you should use; this way, you can add additional functionality to the primitives very easily. For example, the definition of `create-screen` determines what to take from the `epoch-mode-alist` before creating the screen with `epoch::x-create-screen`.

'`zone.el`' Extensions to zone primitives; primitives for style manipulation.

'`event.el`'
Standard event handler code.

'`mouse.el`'
Mouse event handling for epoch. (See Section 7.3.1 [Mouse Dragging], page 67)

'`motion.el`'
Handles mouse motion events. Provides dragging and pasting operations. (See Section 7.3.1 [Mouse Dragging], page 67)

'`message.el`'
Provides a client message handler. Provides a handler for the WM_DELETE_WINDOW client message from the window manager.

'`property.el`'
Provides the basic property handlers.

'`selection.el`'
Provides support code for selections.

'`button.el`'
Provides wrappers for zone functions to maintain compatibility with older Elisp packages.

'`convert-buttons.el`'
Provides a function to convert old Elisp code using buttons to use zones instead.

Additional lisp code is provided in the '`contrib`' directory, under the name of the author or package. This code has been verified to be minimally compatible with Epoch version 4.0, but is not considered to be part of the standard distribution.

7.3.1 Mouse Dragging

We have implemented drag, scrolling drag, cut and paste with the mouse. The files '`mouse.el`' and '`motion.el`' must be loaded to use this feature.

The functionality of buttons with this feature loaded is shown in following table.

Button Action

left-down Clear drag region, set point.

shift left-down
: Display line number and buffer name at mouse pointer.

left-down and drag
: Drag out a region. Highlight (buffer-local), place in PRIMARY selection and kill-ring. Set point and mark around highlighted region.

middle-down
: Paste at the mouse location. (Use `yank` to paste at the current point.)

right-down
: Extend highlighted region from a left-down and drag.

right-down and drag
: Extend highlighted region initially, then adjust by drag.

Scrolling Drag means that when the mouse moves out of the window with a button down, the scroll continues until the mouse button is raised. The speed of the drag can be controlled by setting the following variables:

horizontal-drag-inc 5 Variable

The number of characters the buffer is scrolled horizontally in a single scroll.

vertical-drag-inc 2 Variable

The number of lines the buffer is scrolled vertically in a single scroll.

Dragged regions are global, so you can have only one drag between all Epoch buffers. Dragged region will no longer be highlighted if some other X11 client asserts ownership of PRIMARY selection. (This is the standard behavior for xterms, etc.) Because point and mark are set around drags, you can use a combination of mouse drag and Epoch keystrokes to do editing.

NOTE: When cutting and pasting, Epoch will first request data from the selection indicated by `mouse::selection`, and subsequently look to the X cutbuffer. This is to maintain compatibility with older Xterms which do not support selections, and/or do not use the standard selection for cutting and pasting:

mouse::selection (`intern-atom "PRIMARY"`) Variable

This variable indicates the name of the selection to use. This defaults to the atom XA_PRIMARY, but may need to be set differently for xterms which don't use the standard primary selection.

7.3.2 Screen Naming

include-system-name Variable

Set the variable `include-system-name` to `t` if you would like screen names to include the name of the system they are running on, for example the name of the screen with buffer `foo` selected on machine `bar` would appear `foo @ bar`.

7.3.3 Autoraise

auto-raise-screen Variable

The variable `auto-raise-screen` is a flag to indicate which (if any) screens will be raised when a new edit screen is selected with `select-screen`. When set to `t`, both the edit screen and minibuffer screen (if distinct) will be raised. If set to `'screen`, only the edit screen will be raised; if set to `'minibuf`, only the minibuffer screen (if distinct) is raised. Finally, if set to `nil`, no screens are raised.

7.3.4 Multiple Screen Updates

The '`epoch.el`' file sets things up so all screens are updated automatically. You will have to change this if you would like a different convention. For example, you could use the `epoch-mode-alist` to set things up so that only screens created for buffers with mode `shell-mode` would be updated. See Section 3.7 [Screen Updating], page 27 for more information.

7.3.5 Display of Control Characters

Epoch 4.0 includes the well-known patches to support full 8-bit character sets. (i.e. ISO, LATIN-1, etc.).

ctl-arrow Variable

This buffer-local variable, will display control characters with uparrow if set to `t`, as backslash and octal digits if set to `nil`, and as regular characters if set to any other values.

This behavior may be selected for all buffers by using the code

```
(setq-default ctl-arrow 'foo)
```

7.4 Screen Pools

The package in '`scr-pool.el`', which is not dumped or loaded by default, provides a set of functions for creating pools of screens. This is a set of screens of a fixed size that are used in a package. Screens pools are useful when a package wants to use multiple screens, but wants a limit on the number of screens in use to prevent excessive clutter or resource consumption.

7.4.1 Screen Pool Basics

A screen pool consists of

Size The maximum number of screens allowed in the pool. When this many screens are in the pool, and another is requested, one of the screens already in the pool is recycled and returned instead of a new screen.

Create-Function
This is a function that is used to create a new screen when needed, either because there are less than *size* screens in the pool, or one of the screens in the pool is dead. This function is optional, and if missing `create-screen` is used. No arguments are passed to this function.

Cleanup-Function
This function is called on a screen just before it is recycled, with the screen as its single argument. If missing, no function is called. This is useful, for example, when buffers displayed in a screen should be killed when the screen is recycled.

The screen pool attempts to keep track of the least recently used screen and will recycle that screen first. A screen is marked as used whenever it is returned as a requested screen. In addition, there are functions to explicitly mark screens as either most or least recently used. These function should not be called on screens not in the pool, since this will cause the screen to be added to the pool.

7.4.2 Screen Pool Functions

These are the functions provided by the Screen Pool package.

pool:create *size* &optional *create-function cleanup-function* Function

Creates and returns a screen pool with no screens in it.

pool:delete *pool* Function

Deletes all the screens in the *pool* using `delete-screen`. The cleanup-function, if any, is called on each screen first.

pool:get-screen *pool* Function

Returns a screen from the *pool*. If there are fewer screens than the maximum size, a new screen is created, otherwise an existing screen is recycled. If a screen is recycled, the cleanup-function (if any) is called on the screen.

pool:get-screen-with-buffer *pool buffer* Function

Similar to `pool:get-screen`, except that if one of the screens in the pool is already displaying *buffer*, it is returned instead of creating a new screen or recycling another screen, and the cleanup-function is not called.

pool:get-shrink-wrapped-screen *pool buffer limits* Function

Similar to `pool:get-screen`. If *buffer* is displayed on any screen in the *pool*, then that screen is used, otherwise a screen selected as in `pool:get-screen` is used. This screen is selected, all but one window is deleted, and that window is set to display *buffer*. The screen is then shrunk to fit the buffer, in height and width, up to *limits*. *limits* should be a list of 4 numbers, of the form `(min-width, max-width, min-height, max-height)`.

pool:mark-screen *pool screen* Function

Mark *screen* as being the most recently used *screen* in the *pool*. This means it will be the last to be recycled.

pool:unmark-screen *pool screen* Function

Mark *screen* as being the least recently used *screen* in the *pool*. This means it will be the first screen to be recycled.

7.5 Saving Zones

The package in '`save-zones.el`', which is not dumped or loaded by default, provides the basis for saving a buffer's zone information when it is saved for restoration later. Zone information is

stored at the end of the buffer, commented out appropriately according to the buffer's mode. The information is interpreted and deleted when the buffer is loaded, and is recreated and inserted when the file is stored.

The following zone information is archived:

- zone start
- zone end
- zone data
- zone style tag (or `nil`)

When recreating zones for a buffer, the style tag information is used to determine the appropriate style to assign to the zone. If the style tag is non-`nil`, then the value of *find-style-hook* is called with the style tag as an argument. It is expected that this function will return either a style object, or `nil`. The code provides one possible scheme for using style tags, in which the tag is assumed to be an Elisp symbol whose value is the appropriate style. The hook can be made buffer local if necessary. Any buttons with the *transient* field set to `t` will be ignored.

find-style-hook nil Variable

The function called to find a style corresponding to a *style-tag*.

7.6 Menus

The package in `contrib/wm-menu` '`menu.el`', which is not dumped or loaded by default, provides the basis for doing popup menus between Epoch and GWM 1.6. A second file, '`emenu.el`', gives an example of using the menu package to implement a menu for various options which call Elisp functions, and binds this menu to an event in the mouse map (`control-right-up`).

Menu lists passed to this function should be of the following form:

```
("menu title"
  option 1
  option 2
  ...
  option N
)
```

Each `option` should be a list `(return-symbol "option name")`, and in the case of submenu options, should contain a menu list structure of the same format corresponding to that submenu.

menu::popup *menu* Function

This function will produce a popup menu containing options described in the *menu* list. It will then wait until the user has selected an option and then return the `return-symbol` defined in the menu list.

7.7 Colors

Colors can be specified as either a string, which is used for lookup in the X color database, or a vector with three components, the red, green, and blue components. Vector values are such that `65535 (==2^16-1)` is the maximum for the display device, and 0 is off. Epoch stores colors as X Resources of type X-Cardinal See Section 6.1 [X Resources], page 57.

epoch::number-of-colors Function

Returns the number of color cells on the display. This function is used to distinguish between monochrome and color systems. A result of `2` indicates monochrome, a larger number indicates a grey-scale or color display.

epoch::get-color *name* Function

Given a color name, this function converts the name into an X-Cardinal resource (a pixel value).

7.8 Change Hooks

Two variables were added to Epoch that hold functions to call before and after, respectively, each change is made to a buffer. These variables are not buffer-local, so if you wish to have your change function called only for specific buffers, first make the variables buffer-local, then assign the name of the function to be called.

These variables are only defined if you compile Epoch with DEFINE_CHANGE_FUNCTIONS defined in the '`config.h`' file.

before-change-function Variable

Function to call before each text change. Two arguments are passed to the function: the position of the change and the position of end of the region deleted. If the two positions are equal, then the change is an insertion.

While executing the `before-change-function`, changes to any buffers do not cause calls to any `before-change-function`, `after-change-function`, or `after-movement-function`.

after-movement-function Variable

Function to call after cursor (point) movement which was not due to buffer changes. No arguments are passed to the function.

after-change-function Variable

Function to call after each text change. Three arguments are passed to the function: the position of the change, the position of the end of the inserted text, and the length of the deletion, or `0` if none.

While executing the `after-change-function`, changes to any buffers do not cause calls to any `before-change-function`, `after-change-function`, or `after-movement-function`.

While executing the `after-movement-function`, changes to any buffers will not cause calls to any `before-change-function`, `after-change-function`, or `after-movement-function`.

```
(defun after-change (pos inspos dellen)
  "Called after each change"
  (condition-case err
      (progn
       (cond ((= dellen 0)
              (message "insertion: (%s %s)" pos inspos))
             ((= dellen (- inspos pos))
              (message "replacement: (%s %s)" pos inspos))
             ((= pos inspos)
              (message "deletion: %s of length %s" pos dellen))
             (t
              (message "other change at %s: ins %s, del %s"
                       pos inspos dellen))
             )
       )
    (error
```

```
      (setq after-change-function nil)
      (message "*Change error: %s" (prin1-to-string err)))
    )
  )

;; first evaluate the above function
;; then eval the next two lines to activate the function

(make-local-variable 'after-change-function)
(setq after-change-function 'after-change)
```

7.9 Icons

You can set the names of icons for screens (accessible through the property `WM_ICON_NAME` using the global `.Xdefaults`, the screen property list, or the function `epoch::icon-name`.)

epoch::icon-name &optional *value screen* Function

If value is `nil`, return current icon name. Otherwise set icon name to value. If screen is `nil`, use current screen.

7.10 dbx

dbx Function

Calls a C function named DEBUG which does nothing. This serves as an entry into a debugger if a breakpoint was set upon entry to this function.

Index

$

$HOME/.Xdefaults-<hostname> 5

*

create-screen-alist-hook 16
select-screen-hook 20

<

<hostname>, $HOME/.Xdefaults- 5

A

abort, epoch::event-handler- 44
acquire-selection, epoch:: 61
add-graphic-zone 39
add-read-only-zone 38
add-selection-target 48
add-zone, epoch:: 31, 38
after-change-function 74
after-movement-function 74
alist properties alist, epoch-mode- 17
alist, create-screen properties 17
alist, epoch-mode- 17
alist, epoch-mode-alist properties 17
alist, epoch::convert-selection- 62
alist, epoch::create-screen properties 17
alist, epoch::screen-properties 17
alist, epoch::selection- 61
alist-hook*, *create-screen- 16
app-defaults 5
arrow, ctl- 69
ASCII Support 1
Asked Questions, Frequently 1
asynchronous communication 42
at, epoch::delete-zone- 32
at, epoch::zone- 31
at, epoch::zones- 31
atom, epoch::intern- 57
atom, epoch::unintern- 57
atom, property X- 46
attributes, screen properties or 17
auto-raise-screen 69
autoraise, screen 69

B

background color 9
background color, cursor 18
background, epoch:: 26
background, epoch::set-style- 34
background, epoch::set-style-cursor- 34
background, epoch::style- 34
background, epoch::style-cursor- 34
background, minibuffer 6
background, screen 6, 18
background, screen cursor 6
background-stipple, epoch::set-style- 34
background-stipple, epoch::style- 34
before-change-function 73
bell, epoch::set- 63
bell-volume, epoch:: 63
bindings, key 10
bitmap, epoch::free- 59
bitmap, epoch::make- 59
blue components, red green 73
border color 18
border color, minibuffer 6
border color, screen 6
border width, external 19
border width, internal 19
border width, minibuffer 6
border width, minibuffer internal 7
border width, screen 6
border width, screen internal 7
borderwidth of screen, external and internal 26
buffer, epoch::screens-of- 21
buffer, epoch::zone- 30
buffer, epoch::zones-modify- 32
buffer, pool:get-screen-with- 71
buffer-style 35
buffer-value, symbol- 65
buffer-window, epoch::get- 21
bug reports 1
button event 42

button events 50
button.el 67
buttons.el, convert- 67

C

C-Z 11
C-Z e 45
change event, property- 42
change events, property- 46
change hooks 73
change-function, after- 74
change-function, before- 73
change-screen-size, epoch:: 25
cl.el, mini- 66
class defaults, X 6
class name, screen 19
class name, X program or window 6
class, minibuffer 7
class, resource 9
class, screen 7
clear event, selection- 42
clear-zones, epoch:: 32
clicks, mouse:: 54
clicks, multiple mouse 54
client message event 42
client messages 59
client-message, epoch::send- 59
color vector or string 11
color, background 9
color, border 18
color, cursor background 18
color, cursor foreground 18
color, epoch::cursor- 26
color, epoch::free- 58
color, epoch::get- 58, 73
color, foreground 9
color, minibuffer border 6
color, minibuffer cursor 6
color, screen border 6
color, screen cursor 6
color, text cursor 18
color-components, epoch:: 59
colors 73
colors, epoch::number-of- 73
colors, reverse 9
command line options 8
command line options, defaults in 5
communication, asynchronous 42
compatability for deletion, ICCCM 21
compatibility, GNU Emacs 1
compatibilty for iconify, ICCCM 24
components, epoch::color- 59
components, red green blue 73
configuration, window 15
controls, windowing system 23
convert-buttons.el 67
convert-selection, epoch:: 61
convert-selection-alist, epoch:: 62
convert-selection-hook, epoch:: 48
convert-to-target 48
coordinates-in-window-p 53
coords-to-point, epoch:: 53
copy-mouse-map 52
create screen 16, 17
create, pool: 71
create-mouse-map 52
create-screen 16
create-screen properties alist 17
create-screen properties alist, epoch:: 17
create-screen, epoch:: 16
creation, zone 30
ctl-arrow 69
current screen 15
current-pixel 23
current-screen, epoch:: 20
cursor background color 18
cursor background, screen 6
cursor color, minibuffer 6
cursor color, screen 6
cursor color, text 18
cursor foreground color 18
cursor foreground, minibuffer 7
cursor foreground, screen 7
cursor glyph 19
cursor glyph, minibuffer 7
cursor glyph, screen 7

cursor, epoch::query- 62
cursor, X 62
cursor-background, epoch::set-style- 34
cursor-background, epoch::style- 34
cursor-color, epoch:: 26
cursor-foreground, epoch::set-style- 34
cursor-foreground, epoch::style- 34
cursor-glyph, epoch:: 27
cursor-pixels, epoch::query- 62
cursor-stipple, epoch::set-style- 35
cursor-stipple, epoch::style- 35
cut and paste 67

D

data, epoch::set-zone- 32
data, epoch::zone- 31
database, X resources 5
dbx 75
default sets 5
default, epoch::get- 63
default-mode-line-format 36
default-sets, screen 17
defaults in command line options 5
defaults, app- 5
defaults, X class 6
defaults, X Window program 5
define-mouse 53
define-opaque-font, epoch:: 39
DEFINE_CHANGE_FUNCTIONS 73
delete, pool: 71
delete-screen, epoch:: 21
delete-zone, epoch:: 32
delete-zone-at, epoch:: 32
deleting zones 32
deletion, ICCCM compatability for 21
display, redraw- 28
display, X window 9
display, X Window 7
display, zone 33
display-event-status 45
do, on-map- 55
do, on-move- 55
do, on-resize- 55
do, on-unmap- 55
dot.emacs 66
drag, scrolling 68
drag-inc, horizontal- 68
drag-inc, vertical- 68
dragging, mouse 67

E

e, C-Z 45
edges, window- 23
edit screens 15
Emacs compatibility, GNU 1
Emacs windows, X windows and GNU 15
end, epoch::zone- 31
epoch version 65
epoch-mode-alist 17
epoch-mode-alist properties alist 17
epoch-setup-hook 66
epoch-util.el 66
epoch.el 66
epoch:: 11, 66
epoch::acquire-selection 61
epoch::add-zone 31, 38
epoch::background 26
epoch::bell-volume 63
epoch::change-screen-size 25
epoch::clear-zones 32
epoch::color-components 59
epoch::convert-selection 61
epoch::convert-selection-alist 62
epoch::convert-selection-hook 48
epoch::coords-to-point 53
epoch::create-screen 16
epoch::create-screen properties alist 17
epoch::current-screen 20
epoch::cursor-color 26
epoch::cursor-glyph 27
epoch::define-opaque-font 39
epoch::delete-screen 21
epoch::delete-zone 32
epoch::delete-zone-at 32
epoch::event 44
epoch::event-handler 44

epoch::event-handler-abort 44
epoch::first-window 22
epoch::flash-screen 27
epoch::font 25
epoch::foreground 26
epoch::free-bitmap 59
epoch::free-color 58
epoch::function-key-mapping 65
epoch::get-buffer-window 21
epoch::get-color 58, 73
epoch::get-default 63
epoch::get-event 44
epoch::get-font 58
epoch::get-property 60
epoch::get-screen 19
epoch::get-screen-id 19
epoch::get-selection-owner 61
epoch::global-update 27
epoch::icon-name 25, 75
epoch::iconify-screen 24
epoch::intern-atom 57
epoch::lazy-events 42
epoch::lower-screen 24
epoch::make-bitmap 59
epoch::make-style 33
epoch::make-zone 30
epoch::map-screen 24
epoch::mapraised-screen 24
epoch::minibuf-screen 21
epoch::mod-to-shiftmask 63
epoch::motion-hints-p 54
epoch::mouse-events 42
epoch::move-screen 26
epoch::move-zone 31
epoch::next-screen 20
epoch::nonlocal-minibuffer 15
epoch::number-of-colors 73
epoch::plane-size 26
epoch::prev-screen 20
epoch::query-cursor 62
epoch::query-cursor-pixels 62
epoch::query-mouse 62
epoch::query-pixmap 39
epoch::query-pointer 62
epoch::query-tree 63
epoch::raise-screen 24
epoch::read-only-region-p 38
epoch::read-pixmap-file 39
epoch::rebind-key 63
epoch::redisplay-screen 28
epoch::resource-to-string 58
epoch::resource-to-type 58
epoch::resourcep 57
epoch::screen-height 25
epoch::screen-information 26
epoch::screen-list 20
epoch::screen-mapped-p 24
epoch::screen-of-window 22
epoch::screen-p 20
epoch::screen-properties 17
epoch::screen-properties alist 17
epoch::screen-width 25
epoch::screenp 20
epoch::screens-of-buffer 21
epoch::select-screen 20
epoch::selected-window 22
epoch::selection-alist 61
epoch::selection-timeout 61
epoch::send-client-message 59
epoch::set-bell 63
epoch::set-motion-hints 55
epoch::set-property 60
epoch::set-resource-type 58
epoch::set-screen-modified 28
epoch::set-style-background 34
epoch::set-style-background-stipple 34
epoch::set-style-cursor-background 34
epoch::set-style-cursor-foreground 34
epoch::set-style-cursor-stipple 35
epoch::set-style-font 35
epoch::set-style-foreground 33
epoch::set-style-pixmap 39
epoch::set-style-stipple 34
epoch::set-style-tag 35
epoch::set-style-underline 34
epoch::set-update 27

epoch::set-zone-data ... 32
epoch::set-zone-read-only ... 38
epoch::set-zone-style ... 31
epoch::set-zone-transient ... 32
epoch::string-to-resource ... 57
epoch::style-background ... 34
epoch::style-background-stipple ... 34
epoch::style-cursor-background ... 34
epoch::style-cursor-foreground ... 34
epoch::style-cursor-stipple ... 35
epoch::style-font ... 35
epoch::style-foreground ... 33
epoch::style-pixmap ... 40
epoch::style-stipple ... 34
epoch::style-tag ... 35
epoch::style-underline ... 34
epoch::stylep ... 33
epoch::synchronize-minibuffers ... 15
epoch::title ... 25
epoch::ungrab-pointer ... 62
epoch::unintern-atom ... 57
epoch::unmap-screen ... 24
epoch::update-p ... 28
epoch::version ... 65
epoch::wait-for-event ... 45
epoch::warp-mouse ... 62
epoch::warp-pointer ... 62
epoch::xid-of-screen ... 58
epoch::zone-at ... 31
epoch::zone-buffer ... 30
epoch::zone-data ... 31
epoch::zone-end ... 31
epoch::zone-list ... 31
epoch::zone-read-only ... 38
epoch::zone-start ... 30
epoch::zone-style ... 31
epoch::zone-transient-p ... 32
epoch::zonep ... 30
epoch::zones-at ... 31
epoch::zones-modify-buffer ... 32
equal ... 65
errors, event handler ... 43
event handler errors ... 43
event handlers, one-shot ... 55
event handling, on ... 55
event inhibition ... 45
event key ... 41
event queue ... 41
event queue, internal ... 44
event screen ... 41
event types ... 41
event value ... 41
event, button ... 42
event, client message ... 42
event, epoch:: ... 44
event, epoch::get- ... 44
event, epoch::wait-for- ... 45
event, ignore- ... 45
event, install- ... 44
event, motion ... 42
event, move ... 42
event, pop- ... 45
event, property-change ... 42
event, push- ... 45
event, remove- ... 44
event, resize ... 42
event, resume- ... 45
event, selection-clear ... 42
event-handler, epoch:: ... 44
event-handler-abort, epoch:: ... 44
event-status, display- ... 45
event.el ... 67
events, button ... 50
events, epoch::lazy- ... 42
events, epoch::mouse- ... 42
events, map and unmap ... 42
events, motion ... 7, 9, 19, 54
events, mouse ... 50
events, property-change ... 46
events, waiting for ... 45
extensions, fill mode ... 23
extensions, standard ... 65
external and internal borderwidth of screen ... 26
external border width ... 19

F

file, epoch::read-pixmap- 39
fill mode extensions 23
fill-hook, line- 23
fill-pixel 23
find-style-hook 72
first-window, epoch:: 22
flash-screen, epoch:: 27
font 9, 19
font, epoch:: 25
font, epoch::define-opaque- 39
font, epoch::get- 58
font, epoch::set-style- 35
font, epoch::style- 35
font, minibuffer 7
font, screen 7
for-event, epoch::wait- 45
foreground color 9
foreground color, cursor 18
foreground, epoch:: 26
foreground, epoch::set-style- 33
foreground, epoch::set-style-cursor- 34
foreground, epoch::style- 33
foreground, epoch::style-cursor- 34
foreground, minibuffer 7
foreground, minibuffer cursor 7
foreground, screen 7, 18
foreground, screen cursor 7
format, default-mode-line- 36
format, mode-line- 36
format, Modeline 36
free-bitmap, epoch:: 59
free-color, epoch:: 58
Frequently Asked Questions 1
function, after-change- 74
function, after-movement- 74
function, before-change- 73
function-key-mapping, epoch:: 65

G

generic window manager 12
geometry, minibuffer 7
geometry, screen 7, 9, 19
get-buffer-window, epoch:: 21
get-color, epoch:: 58, 73
get-default, epoch:: 63
get-event, epoch:: 44
get-font, epoch:: 58
get-property, epoch:: 60
get-screen, epoch:: 19
get-screen, pool: 71
get-screen-id, epoch:: 19
get-screen-with-buffer, pool: 71
get-selection-owner, epoch:: 61
get-shrink-wrapped-screen, pool: 71
global-set-mouse 53
global-update, epoch:: 27
glyph, cursor 19
glyph, epoch::cursor- 27
glyph, minibuffer cursor 7
glyph, screen cursor 7
GNU Emacs compatibility 1
GNU Emacs windows, X windows and 15
graphic-zone, add- 39
Graphical zones 39
green blue components, red 73
GWM 12

H

handler errors, event 43
handler, epoch::event- 44
handler-abort, epoch::event- 44
handlers, one-shot event 55
handling, on event 55
height of screen, width and 26
height, epoch::screen- 25
height, window- 22
hierarchy, Style 33
hints, epoch::set-motion- 55
hints-p, epoch::motion- 54
hook*, *create-screen-alist- 16
hook*, *select-screen- 20
hook, epoch-setup- 66
hook, epoch::convert-selection- 48
hook, find-style- 72
hook, line-fill- 23

hooks 17
hooks, change 73
horizontal-drag-inc 68
hypertext 30

I

ICCCM compatability for deletion 21
ICCCM compatibilty for iconify 24
icon name 19
icon name, screen 8
icon-name, epoch:: 25, 75
IconicState 19
iconify, ICCCM compatibilty for 24
iconify-screen, epoch:: 24
ID or screen object, screen 11
id, epoch::get-screen- 19
id, make- 19
id, X parent resource 19
ignore-event 45
ignore-message 50
ignore-property 47
ignore-selection 48
inc, horizontal-drag- 68
inc, vertical-drag- 68
include-system-name 69
information, epoch::screen- 26
inhibit-initial-screen-mapping 16
inhibition, event 45
initial state, screen 19
initial-screen-mapping, inhibit- 16
install-event 44
install-message 49
install-property 47
install-selection 48
intern-atom, epoch:: 57
internal border width 19
internal border width, minibuffer 7
internal border width, screen 7
internal borderwidth of screen, external and 26
internal event queue 44
interval, mouse:: 53
inverse-video, mode-line- 36

K

key bindings 10
key, epoch::rebind- 63
key, event 41
key-mapping, epoch::function- 65
keyboard queue 50
kill-local-mouse-map 53

L

last-spot, mouse:: 54
lazy-events, epoch:: 42
line options, command 8
line options, defaults in command 5
line-fill-hook 23
line-format, default-mode- 36
line-format, mode- 36
line-inverse-video, mode- 36
lisp packages 66
list support, mailing 1
list, epoch::screen- 20
list, epoch::zone- 31
list, unmapped screens 20
local-mouse-map, kill- 53
local-mouse-map, use- 52
local-set-mouse 53
location of screen, X and Y 26
lower-screen, epoch:: 24

M

mailing list support 1
make-bitmap, epoch:: 59
make-id 19
make-screen 19
make-style, epoch:: 33
make-zone, epoch:: 30
manager, generic window 12
manager, resource 9
map and unmap events 42
map, copy-mouse- 52
map, create-mouse- 52
map, kill-local-mouse- 53
map, use-local-mouse- 52
map-do, on- 55

map-screen, epoch:: ... 24
mapped screen ... 16
mapped state, screen ... 26
mapped-p, epoch::screen- ... 24
mapping, epoch::function-key- ... 65
mapping, inhibit-initial-screen- ... 16
mapraised-screen, epoch:: ... 24
mark-screen, pool: ... 71
menu::popup ... 73
menus ... 46, 72
message event, client ... 42
message, epoch::send-client- ... 59
message, ignore- ... 50
message, install- ... 49
message, pop- ... 49
message, push- ... 49
message, remove- ... 49
message, resume- ... 50
message.el ... 67
messages, client ... 59
mini-cl.el ... 66
minibuf-screen, epoch:: ... 21
minibuffer background ... 6
minibuffer border color ... 6
minibuffer border width ... 6
minibuffer class ... 7
minibuffer cursor color ... 6
minibuffer cursor foreground ... 7
minibuffer cursor glyph ... 7
minibuffer font ... 7
minibuffer foreground ... 7
minibuffer geometry ... 7
minibuffer internal border width ... 7
minibuffer modes ... 7
minibuffer name ... 7
minibuffer resource name ... 8
minibuffer reverse ... 8
minibuffer screens ... 15
minibuffer title ... 7
minibuffer, epoch::nonlocal- ... 15
minibuffers, epoch::synchronize- ... 15
minor screens ... 26
mod-to-shiftmask, epoch:: ... 63
mode extensions, fill ... 23
mode-alist properties alist, epoch- ... 17
mode-alist, epoch- ... 17
mode-line-format ... 36
mode-line-format, default- ... 36
mode-line-inverse-video ... 36
Modeline format ... 36
modes, minibuffer ... 7
modified, epoch::set-screen- ... 28
modify-buffer, epoch::zones- ... 32
motion event ... 42
motion events ... 7, 9, 19, 54
motion-hints, epoch::set- ... 55
motion-hints-p, epoch:: ... 54
motion.el ... 67
motion::style ... 35
mouse clicks, multiple ... 54
mouse dragging ... 67
mouse events ... 50
mouse, define- ... 53
mouse, epoch::query- ... 62
mouse, epoch::warp- ... 62
mouse, global-set- ... 53
mouse, local-set- ... 53
mouse-events, epoch:: ... 42
mouse-map, copy- ... 52
mouse-map, create- ... 52
mouse-map, kill-local- ... 53
mouse-map, use-local- ... 52
mouse.el ... 67
mouse::clicks ... 54
mouse::interval ... 53
mouse::last-spot ... 54
mouse::selection ... 68
mouse::set-point ... 53
mouse::time-stamp ... 54
mouse::x ... 54
mouse::y ... 54
move event ... 42
move-do, on- ... 55
move-screen, epoch:: ... 26
move-to-pixel ... 23
move-zone, epoch:: ... 31

movement-function, after- 74
multiple mouse clicks 54
multiple screen updates 69

N

name, epoch::icon- 25, 75
name, icon 19
name, include-system- 69
name, minibuffer 7
name, minibuffer resource 8
name, resource 9
name, screen 7
name, screen class 19
name, screen icon 8
name, screen resource 7, 19
name, X program or window class 6
name, X program or window resource 6
naming, screen 69
next-screen, epoch:: 20
nonlocal-minibuffer, epoch:: 15
NormalState 19
null zone styles 37
number-of-colors, epoch:: 73

O

object, screen 16
object, screen ID or screen 11
object, zone 29
objects, X 58
one-shot event handlers 55
only zones, read- 38
only, epoch::set-zone-read- 38
only, epoch::zone-read- 38
only-region-p, epoch::read- 38
only-zone, add-read- 38
opaque-font, epoch::define- 39
options, command line 8
options, defaults in command line 5
or attributes, screen properties 17
or screen object, screen ID 11
or string, color vector 11
or window class name, X program 6
or window resource name, X program 6
overlapping zones 37
owner, epoch::get-selection- 61

P

p, coordinates-in-window- 53
p, epoch::motion-hints- 54
p, epoch::read-only-region- 38
p, epoch::screen- 20
p, epoch::screen-mapped- 24
p, epoch::update- 28
p, epoch::zone-transient- 32
packages, lisp 66
parent resource id, X 19
paste, cut and 67
pixedges, window- 22
pixel, current- 23
pixel, fill- 23
pixel, move-to- 23
pixels, epoch::query-cursor- 62
pixheight, window- 22
pixmap, epoch::query- 39
pixmap, epoch::set-style- 39
pixmap, epoch::style- 40
pixmap-file, epoch::read- 39
pixwidth, window- 22
plane-size, epoch:: 26
plotting, zone 37
point, epoch::coords-to- 53
point, mouse::set- 53
pointer, epoch::query- 62
pointer, epoch::ungrab- 62
pointer, epoch::warp- 62
pool.el, scr- 70
pool:create 71
pool:delete 71
pool:get-screen 71
pool:get-screen-with-buffer 71
pool:get-shrink-wrapped-screen 71
pool:mark-screen 71
pool:unmark-screen 71
pools, screen 70
pop-event 45
pop-message 49

pop-property 47
pop-selection 48
popup, menu:: 73
prev-screen, epoch:: 20
Primitives, Style 33
program defaults, X Window 5
program or window class name, X 6
program or window resource name, X 6
properties alist, create-screen 17
properties alist, epoch-mode-alist 17
properties alist, epoch::create-screen 17
properties alist, epoch::screen- 17
properties or attributes, screen 17
properties, epoch::screen- 17
properties, X 60
property X-atom 46
property, epoch::get- 60
property, epoch::set- 60
property, ignore- 47
property, install- 47
property, pop- 47
property, push- 47
property, remove- 47
property, resume- 47
property-change event 42
property-change events 46
property.el 67
push-event 45
push-message 49
push-property 47
push-selection 48

Q

query-cursor, epoch:: 62
query-cursor-pixels, epoch:: 62
query-mouse, epoch:: 62
query-pixmap, epoch:: 39
query-pointer, epoch:: 62
query-tree, epoch:: 63
Questions, Frequently Asked 1
queue, event 41
queue, internal event 44
queue, keyboard 50

R

raise-screen, auto- 69
raise-screen, epoch:: 24
read-only zones 38
read-only, epoch::set-zone- 38
read-only, epoch::zone- 38
read-only-region-p, epoch:: 38
read-only-zone, add- 38
read-pixmap-file, epoch:: 39
rebind-key, epoch:: 63
recenter 28
red green blue components 73
redisplay-screen, epoch:: 28
redraw-display 28
region-p, epoch::read-only- 38
remove-event 44
remove-message 49
remove-property 47
remove-selection 48
reports, bug 1
required, X Windows 1
resize event 42
resize-do, on- 55
resource class 9
resource id, X parent 19
resource manager 9
resource name 9
resource name, minibuffer 8
resource name, screen 7, 19
resource name, X program or window 6
resource, epoch::string-to- 57
resource-to-string, epoch:: 58
resource-to-type, epoch:: 58
resource-type, epoch::set- 58
resourcep, epoch:: 57
resources database, X 5
resources, X 57
restore-zones, undo- 40
resume-event 45
resume-message 50
resume-property 47
resume-selection 48
reverse 19

reverse colors 9
reverse, minibuffer 8
reverse, screen 8

S

Saving Zones 71
scr-pool.el 70
screen autoraise 69
screen background 6, 18
screen border color 6
screen border width 6
screen class 7
screen class name 19
screen cursor background 6
screen cursor color 6
screen cursor foreground 7
screen cursor glyph 7
screen default-sets 17
screen font 7
screen foreground 7, 18
screen geometry 7, 9, 19
screen icon name 8
screen ID or screen object 11
screen initial state 19
screen internal border width 7
screen mapped state 26
screen name 7
screen naming 69
screen object 16
screen object, screen ID or 11
screen pools 70
screen properties alist, create- 17
screen properties alist, epoch::create- 17
screen properties or attributes 17
screen resource name 7, 19
screen reverse 8
screen title 7, 19
screen updates, multiple 69
screen updating 19, 27
screen, auto-raise- 69
screen, create 16, 17
screen, create- 16
screen, current 15
screen, epoch::create- 16
screen, epoch::current- 20
screen, epoch::delete- 21
screen, epoch::flash- 27
screen, epoch::get- 19
screen, epoch::iconify- 24
screen, epoch::lower- 24
screen, epoch::map- 24
screen, epoch::mapraised- 24
screen, epoch::minibuf- 21
screen, epoch::move- 26
screen, epoch::next- 20
screen, epoch::prev- 20
screen, epoch::raise- 24
screen, epoch::redisplay- 28
screen, epoch::select- 20
screen, epoch::unmap- 24
screen, epoch::xid-of- 58
screen, event 41
screen, external and internal borderwidth of 26
screen, make- 19
screen, mapped 16
screen, pool:get- 71
screen, pool:get-shrink-wrapped- 71
screen, pool:mark- 71
screen, pool:unmark- 71
screen, selected 15
screen, unmapped 16
screen, width and height of 26
screen, X and Y location of 26
screen-alist-hook*, *create- 16
screen-height, epoch:: 25
screen-hook*, *select- 20
screen-id, epoch::get- 19
screen-information, epoch:: 26
screen-list, epoch:: 20
screen-mapped-p, epoch:: 24
screen-mapping, inhibit-initial- 16
screen-modified, epoch::set- 28
screen-of-window, epoch:: 22
screen-p, epoch:: 20
screen-properties alist, epoch:: 17
screen-properties, epoch:: 17

screen-size, epoch::change- 25
screen-width, epoch:: 25
screen-with-buffer, pool:get- 71
screenp, epoch:: 20
screens 15
screens list, unmapped 20
screens, edit 15
screens, minibuffer 15
screens, minor 26
screens-of-buffer, epoch:: 21
scrolling drag 68
select-screen, epoch:: 20
selected screen 15
selected-window 21
selected-window, epoch:: 22
selection, epoch::acquire- 61
selection, epoch::convert- 61
selection, ignore- 48
selection, install- 48
selection, mouse:: 68
selection, pop- 48
selection, push- 48
selection, remove- 48
selection, resume- 48
selection-alist, epoch:: 61
selection-alist, epoch::convert- 62
selection-clear event 42
selection-hook, epoch::convert- 48
selection-owner, epoch::get- 61
selection-target, add- 48
selection-timeout, epoch:: 61
selection.el 67
send-client-message, epoch:: 59
set-bell, epoch:: 63
set-motion-hints, epoch:: 55
set-mouse, global- 53
set-mouse, local- 53
set-point, mouse:: 53
set-property, epoch:: 60
set-resource-type, epoch:: 58
set-screen-modified, epoch:: 28
set-style-background, epoch:: 34
set-style-background-stipple, epoch:: 34
set-style-cursor-background, epoch:: 34
set-style-cursor-foreground, epoch:: 34
set-style-cursor-stipple, epoch:: 35
set-style-font, epoch:: 35
set-style-foreground, epoch:: 33
set-style-pixmap, epoch:: 39
set-style-stipple, epoch:: 34
set-style-tag, epoch:: 35
set-style-underline, epoch:: 34
set-update, epoch:: 27
set-zone-data, epoch:: 32
set-zone-read-only, epoch:: 38
set-zone-style, epoch:: 31
set-zone-transient, epoch:: 32
sets, default 5
sets, screen default- 17
setup-hook, epoch- 66
shiftmask, epoch::mod-to- 63
shot event handlers, one- 55
shrink-wrapped-screen, pool:get- 71
size, epoch::change-screen- 25
size, epoch::plane- 26
spot, mouse::last- 54
stamp, mouse::time- 54
standard extensions 65
start, epoch::zone- 30
state, screen initial 19
state, screen mapped 26
status, display-event- 45
stipple, epoch::set-style- 34
stipple, epoch::set-style-background- 34
stipple, epoch::set-style-cursor- 35
stipple, epoch::style- 34
stipple, epoch::style-background- 34
stipple, epoch::style-cursor- 35
string, color vector or 11
string, epoch::resource-to- 58
string-to-resource, epoch:: 57
Style hierarchy 33
Style Primitives 33
style, buffer- 35
style, epoch::make- 33
style, epoch::set-zone- 31

style, epoch::zone- 31
style, motion:: 35
style-background, epoch:: 34
style-background, epoch::set- 34
style-background-stipple, epoch:: 34
style-background-stipple, epoch::set- 34
style-cursor-background, epoch:: 34
style-cursor-background, epoch::set- 34
style-cursor-foreground, epoch:: 34
style-cursor-foreground, epoch::set- 34
style-cursor-stipple, epoch:: 35
style-cursor-stipple, epoch::set- 35
style-font, epoch:: 35
style-font, epoch::set- 35
style-foreground, epoch:: 33
style-foreground, epoch::set- 33
style-hook, find- 72
style-pixmap, epoch:: 40
style-pixmap, epoch::set- 39
style-stipple, epoch:: 34
style-stipple, epoch::set- 34
style-tag, epoch:: 35
style-tag, epoch::set- 35
style-underline, epoch:: 34
style-underline, epoch::set- 34
stylep, epoch:: 33
styles 33
styles, null zone 37
Support, ASCII 1
support, mailing list 1
symbol-buffer-value 65
synchronize-minibuffers, epoch:: 15
system controls, windowing 23
system-name, include- 69

T

tag, epoch::set-style- 35
tag, epoch::style- 35
target, add-selection- 48
target, convert-to- 48
text cursor color 18
text, zone- 32
text-width 23
time-stamp, mouse:: 54
timeout, epoch::selection- 61
title, epoch:: 25
title, minibuffer 7
title, screen 7, 19
title, window 9
transient, epoch::set-zone- 32
transient-p, epoch::zone- 32
tree, epoch::query- 63
type, epoch::resource-to- 58
type, epoch::set-resource- 58
types, event 41

U

underline, epoch::set-style- 34
underline, epoch::style- 34
Undo, Zones and 40
undo-restore-zones 40
ungrab-pointer, epoch:: 62
unintern-atom, epoch:: 57
unmap events, map and 42
unmap-do, on- 55
unmap-screen, epoch:: 24
unmapped screen 16
unmapped screens list 20
unmark-screen, pool: 71
update, epoch::global- 27
update, epoch::set- 27
update-p, epoch:: 28
updates, multiple screen 69
updating, screen 19, 27
use-local-mouse-map 52
util.el, epoch- 66

V

value, event 41
value, symbol-buffer- 65
vector or string, color 11
version, epoch 65
version, epoch:: 65
vertical-drag-inc 68
video, mode-line-inverse- 36
volume, epoch::bell- 63

W

wait-for-event, epoch:: ... 45
waiting for events ... 45
warp-mouse, epoch:: ... 62
warp-pointer, epoch:: ... 62
width and height of screen ... 26
width, epoch::screen- ... 25
width, external border ... 19
width, internal border ... 19
width, minibuffer border ... 6
width, minibuffer internal border ... 7
width, screen border ... 6
width, screen internal border ... 7
width, text- ... 23
width, window- ... 23
window class name, X program or ... 6
window configuration ... 15
window display, X ... 9
Window display, X ... 7
window manager, generic ... 12
Window program defaults, X ... 5
window resource name, X program or ... 6
window title ... 9
window, epoch::first- ... 22
window, epoch::get-buffer- ... 21
window, epoch::screen-of- ... 22
window, epoch::selected- ... 22
window, selected- ... 21
window-edges ... 23
window-height ... 22
window-p, coordinates-in- ... 53
window-pixedges ... 22
window-pixheight ... 22
window-pixwidth ... 22
window-width ... 23
windowing system controls ... 23
windows and GNU Emacs windows, X ... 15
Windows required, X ... 1
windows, X windows and GNU Emacs ... 15
with-buffer, pool:get-screen- ... 71
WM_DELETE_WINDOW ... 67
WM_ICON_NAME ... 75
wrapped-screen, pool:get-shrink- ... 71
wrapper.el ... 66

X

X and Y location of screen ... 26
X class defaults ... 6
X cursor ... 62
X objects ... 58
X parent resource id ... 19
X program or window class name ... 6
X program or window resource name ... 6
X properties ... 60
X resources ... 57
X resources database ... 5
X window display ... 9
X Window display ... 7
X Window program defaults ... 5
X windows and GNU Emacs windows ... 15
X Windows required ... 1
x, mouse:: ... 54
X-atom, property ... 46
XAPPLRESDIR ... 5
XENVIRONMENT ... 5
xid-of-screen, epoch:: ... 58
xmenu ... 13
Xpm ... 39
xrdb ... 5

Y

Y location of screen, X and ... 26
y, mouse:: ... 54

Z

Z e, C- ... 45
Z, C- ... 11
zone creation ... 30
zone display ... 33
zone object ... 29
zone plotting ... 37
zone styles, null ... 37
zone, add-graphic- ... 39
zone, add-read-only- ... 38
zone, epoch::add- ... 31, 38
zone, epoch::delete- ... 32

zone, epoch::make- 30
zone, epoch::move- 31
zone-at, epoch:: 31
zone-at, epoch::delete- 32
zone-buffer, epoch:: 30
zone-data, epoch:: 31
zone-data, epoch::set- 32
zone-end, epoch:: 31
zone-list, epoch:: 31
zone-read-only, epoch:: 38
zone-read-only, epoch::set- 38
zone-start, epoch:: 30
zone-style, epoch:: 31
zone-style, epoch::set- 31
zone-text 32
zone-transient, epoch::set- 32
zone-transient-p, epoch:: 32
zone.el 67
zonep, epoch:: 30
zones 29
Zones and Undo 40
zones, deleting 32
zones, epoch::clear- 32
zones, Graphical 39
zones, overlapping 37
zones, read-only 38
Zones, Saving 71
zones, undo-restore- 40
zones-at, epoch:: 31
zones-modify-buffer, epoch:: 32

Short Contents

1 Introduction 1
2 General Information 5
3 Screens 15
4 Zones 29
5 Events 41
6 X11 Primitives 57
7 Miscellaneous 65
Index 77

Table of Contents

1 Introduction 1

1.1 Support and Mailing List 1
1.2 Getting Epoch 2
1.3 Acknowledgements 2

2 General Information 5

2.1 Getting Started 5
2.1.1 X Window Program Defaults 5
2.1.2 Command Line Options 8
2.2 New Features 9
2.3 Key Bindings 10
2.4 Conventions 11
2.5 Menus 12
2.5.1 GWM 12
2.5.2 XMENU 13

3 Screens 15

3.1 Screen Basics 15
3.2 Screen Properties 17
3.3 Controlling Screens 19
3.4 Screens and Windows 21
3.5 Variable sized Fonts 22
3.6 Manipulating Screens 23
3.7 Screen Updating 27

4 Zones 29

4.1 Zone Basics 29
4.2 Zone Primitives 30
4.3 Deleting Zones 32
4.4 Style Primitives 33
4.5 Modelines 36
4.6 Zone Plotting 37
4.7 Read-only Zones 38
4.8 Graphical Zones 39
4.9 Zones and Undo 40

5 Events 41

5.1 Event Basics 41
5.2 Basic Event Handling 43
5.3 Advanced Event Handling 45
5.4 Property Change Events 46
5.5 Selection Events 47
5.6 Client Message Events 49
5.7 Mouse Events 50
5.8 Motion Events 54
5.9 On Event Handling 55

6 X11 Primitives 57

6.1 X Resources 57
6.2 X Objects 58
6.2.1 Client Messages 59
6.2.2 X Properties 60
6.2.3 X Selections 61
6.2.4 X Cursor 62
6.3 Other X Stuff 63

7 Miscellaneous 65

7.1 Standard Extensions 65
7.2 Epoch Version 65
7.3 Packages 66
7.3.1 Mouse Dragging 67
7.3.2 Screen Naming 69
7.3.3 Autoraise 69
7.3.4 Multiple Screen Updates 69
7.3.5 Display of Control Characters 69
7.4 Screen Pools 70
7.4.1 Screen Pool Basics 70
7.4.2 Screen Pool Functions 70
7.5 Saving Zones 71
7.6 Menus 72
7.7 Colors 73
7.8 Change Hooks 73
7.9 Icons 75
7.10 dbx 75

Index 77