ESA
A library for writing Emacs-Style Applications.

Robert Strandh

2016
Contents

1 Introduction ............................................. 1
   1.1 Purpose .............................................. 1

2 Packages ................................................. 3

3 Basic use of ESA ......................................... 5
   3.1 Mixin classes .......................................... 5
   3.2 The info pane .......................................... 6
   3.3 The minibuffer pane ..................................... 6
   3.4 Command tables ......................................... 6
      3.4.1 Command tables supplied by ESA ................. 6
      3.4.2 Finding a command table ......................... 7

4 Functions for input and output ............................ 9

Bibliography ................................................. 13
Chapter 1

Introduction

1.1 Purpose

ESA is a library that helps the programmer write applications with a look and feel that resembles that of Emacs ([LLS14, Fin91, Fin80]).

ESA supplies a command processor that is different from the one supplied by standard CLIM, making it practical to use multi-keystroke command invocation, and invocation of commands with no keyboard binding possible through M-x.

ESA assumes that an application that displays a certain number of buffers in a certain number of windows, and that at all times, there is a current buffer that is being worked on.
Chapter 2

Packages

ESA defines four packages: esa-utils, esa-io, esa-buffer, and esa.
Chapter 3

Basic use of ESA

3.1 Mixin classes

For basic use of the ESA library, the application needs to supply it with certain functionality. The basic application document should be a class that inherits from the class \texttt{esa-buffer:esa-buffer-mixin}. This class supplies functionality for associating the buffer with a file, to determine whether the buffer has been modified since last saved, and whether the buffer is read-only.

Application panes should inherit from the class \texttt{esa:esa-pane-mixin}. This class supplies a slot for storing a command table and a slot for storing the previous command.

Application frames should inherit from the class \texttt{esa:esa-frame-mixin}. This class supplies a slot that stores a list of the windows used by the application, and an accessor \texttt{esa:windows} that can be used by application code to return or to modify the list of windows used. Notice that the class definition for the application frame must explicitly inherit not only from \texttt{esa-frame-mixin}, but also from \texttt{standard-application-frame}, since the latter is automatically supplied only if the list of superclasses is empty.

Applications should supply a method on the generic function \texttt{esa:buffers} which takes a single argument, the application frame. It should return a list of all the application documents (buffers) that the application is currently
manipulating.

Applications should also supply a method on the generic function `esa:esa-current-buffer`, which also take a single argument, the application frame. The method should return the current buffer, i.e. the buffer that is currently being manipulated by the user. This might be the buffer that is on display in the window with the current keyboard focus. This method is called by functions that require the current buffer, in particular in order to save the current buffer to file, or to toggle the read-only flag of the current buffer.

3.2 The info pane

ESA supplies a class `esa:info-pane` which is typically used to display something similar to the status line of Emacs. It supplies a slot that contains a main application pane. This slot can be initialized with the :initarg :master-pane and can be read using the reader master-pane. An application typically supplies a CLIM display-function for an info pane that displays some data about its master pane.

3.3 The minibuffer pane

ESA supplies a class `esa:minibuffer-pane` that is used to display messages to the user of the application, and also to acquire arguments to commands. Applications should make sure the application frame contains an instance of this class, or of a subclass of it.

3.4 Command tables

3.4.1 Command tables supplied by ESA

Typically, an application using the ESA library will need a number of CLIM command tables. ESA supplies a number of such command tables that the application can inherit from.
3.4. COMMAND TABLES

⇒ esa:global-esa-table

This command table contains a few basic commands that every application using the ESA library will need.

⇒ esa:com-quit

This command quits the application by invoking the CLIM function frame-exit on the application frame. It is included in the global-esa-table, together with the standard key bindings C-x C-c.

The global-esa-table also contains the keyboard binding M-x which invokes the command esa:com-extended-command. This command prompts the user for the name of a command in the minibuffer, and executes that command.

⇒ esa:keyboard-macro-table

This command table contains three commands, com-start-kbd-macro (C-x (), com-end-kbd-macro (C-x )) and com-call-last-kbd-macro (C-x e). Applications that want to use Emacs-style keyboard macros should include this table in the global application command table.

3.4.2 Finding a command table

A typical ESA application may require a different command table depending on what part of the application is being manipulated. ESA makes this possible in two different ways.

⇒ find-applicable-command-table frame

This function provides the most general way in which ESA finds the command table to use for each command. It has a single argument, which is the ESA application frame. Client code may supply a primary method on this generic function, specialized to its particular application-frame class.

⇒ find-applicable-command-table (frame esa-frame-mixin)

This default method on find-applicable-command-table accesses a slot in the current pane (i.e., the first pane in the list returned by a call to the generic function windows) that stores a command table associated with that pane.

⇒ :command-table
Relatively simple ESA applications may supply this :\texttt{initarg} when making an instance of the class \texttt{esa:esa-pane-mixin}. This technique works for applications in which command table to use is always determined by the current pane.
Chapter 4

Functions for input and output

The ESA library provides facilities for loading a buffer from a file, and saving a buffer to a file. The `esa-io` package contains symbols related to this functionality.

⇒ `esa-io:esa-io-table` [Command Table]

This command table contains a number of commands and related key bindings related to input/output. Typically, an application that needs such i/o would inherit from this command table when creating its global command table.

⇒ `com-find-file ((file-name pathname))` [Command]

This command is similar to the Emacs command with the same name.

If no argument is given, this command prompts for a file-name to load into a fresh buffer which then becomes the current buffer.

If a buffer is already visiting that file, it instead switches to that buffer.

If the file-name given does not name an existing file, a fresh buffer is created that is associated with the file-name given, but no file is created.

This command is bound to the keyboard shortcut C-x C-f.

⇒ `esa-io:frame-find-file (application-frame pathname)` [Generic Function]

This generic function is called by the command `com-find-file`, passing the
current application frame and the pathname that it received as an argument.

If a buffer with the file-name *pathname* associated with it already exists, then that buffer is returned. Otherwise, if a file with the name *pathname* exists, a fresh buffer created from the file is returned. Otherwise, a new empty buffer having the associated file name is returned.

Client code may override this method or provide auxiliary methods on it.

⇒ **com-find-file-read-only** (\(\text{file-name pathname}\)) \([\text{Command}]\)

This command is similar to *find-file*, except that after reading the contents of the file into a buffer, it sets the read-only flag of the buffer to *true*.

If no argument is given, this command prompts for a file-name to load into a fresh buffer which then becomes the current buffer.

If a buffer is already visiting that file, it instead switches to that buffer.

If the file-name given does not name an existing file, an error is signaled.

This command is bound to the keyboard shortcut C-x C-r.

⇒ **esa-io:frame-find-file-read-only** (application-frame pathname) \([\text{Generic Function}]\)

This generic function is called by the command **com-find-file-read-only**, passing the current application frame and the pathname that it received as an argument.

⇒ **com-read-only** () \([\text{Command}]\)

This command toggles the read-only flag of the current buffer.

This command is bound to the keyboard shortcut C-x C-q.

⇒ **com-set-visited-file-name** (\(\text{file-name pathname}\)) \([\text{Command}]\)

This command sets the file-name associated with the current buffer.

If no argument is given, this command prompts for a file-name to associate with the current buffer.

This command is not bound to any keyboard shortcut. It is only available as
an extended command.

⇒ com-save-buffer ((file-name pathname)) [Command]

If the current buffer has a file-name associated with it, then this command writes the contents of the current buffer to the file with that name. If not, and if no argument is given, this command prompts for a file-name to save the buffer contents to.

This command is bound to the keyboard shortcut C-x C-s.

⇒ com-write-buffer ((file-name pathname)) [Command]

This command writes the contents of the current buffer to the file whose name is given by the argument.

If no argument is given, this command prompts for a file-name to write the contents of the current buffer to.

The file-name associated with the current buffer is changed to the file name given by the argument, or to the prompt.

This command is bound to the keyboard shortcut C-x C-w.

These commands handle prompting for file names, searching for existing buffers with the file name given, Emacs-style file versioning, and more. The only thing they need help from the application with is for saving a buffer to a stream, and for creating a buffer from a stream. For that, the ESA library calls the generic functions described below.

⇒ esa-buffer:frame-save-buffer-to-stream
   (application-frame buffer stream) [Generic Function]

This generic function is called by the commands that write the contents of a buffer to a file.

Client code that uses the ESA input/output facility must supply a method on this generic function, and that method must be specialized to the application-specific class of its application-frame.

⇒ esa-buffer:frame-make-buffer-from-stream
   (application-frame stream) [Generic Function]
This generic function is called by the commands that read the contents of a file into a new buffer.

Client code that uses the ESA input/output facility must supply a method on this generic function, and that method must be specialized to the application-specific class of its application-frame.

Applications should also provide a method on \texttt{esa-buffer:frame-make-new-buffer} so that the ESA library can create a new buffer whenever a non-existing file name is given.

To implement the i/o functions, the ESA i/o facility calls the generic functions described below:

\begin{itemize}
  \item \texttt{esa-io:frame-set-visited-file-name (application-frame pathname buffer)} \hspace{1cm} \textit{[Generic Function]}
  \item \texttt{esa-io:frame-save-buffer (application-frame buffer)} \hspace{1cm} \textit{[Generic Function]}
  \item \texttt{esa-io:frame-write-buffer (application-frame pathname buffer)} \hspace{1cm} \textit{[Generic Function]}
\end{itemize}

Applications can override these methods, or provide auxiliary methods on them in order to customize their behavior.
Bibliography

