Cameleon2/Chamo

Maxence Guesdon - SED, INRIA Paris-Rocquencourt

OCaml meeting 2009 - Grenoble, France

Plan

Plan

Plan

What is Cameleon2 ?

- A collection of tools and libraries, as components to create an IDE for OCaml,
- Started in 2001,
- Based on LablGtk, then LablGtk2,
- Distributed under LGPL,
- http://home.gna.org/cameleon/

What is Cameleon2 ?

- A collection of tools and libraries, as components to create an IDE for OCaml,
- Started in 2001,
- Based on LablGtk, then LablGtk2,
- Distributed under LGPL,
- http://home.gna.org/cameleon/

What is Cameleon2 ?

- A collection of tools and libraries, as components to create an IDE for OCaml,
- Started in 2001,
- Based on LablGtk, then LablGtk2,
- Distributed under LGPL,
- http://home.gna.org/cameleon/

What is Cameleon2 ?

- A collection of tools and libraries, as components to create an IDE for OCaml,
- Started in 2001,
- Based on LablGtk, then LablGtk2,
- Distributed under LGPL,
- http://home.gna.org/cameleon/

What is Cameleon2 ?

- A collection of tools and libraries, as components to create an IDE for OCaml,
- Started in 2001,
- Based on LablGtk, then LablGtk2,
- Distributed under LGPL,
- http://home.gna.org/cameleon/

Libraries

- **Config_file** : defining, loading and saving options files,
- **Configwin** : creating input dialog boxes for LablGtk2 applications ; used in CoqIDE,
- **Odiff** : parsing, printing, displaying and merging differences in diff format,
- **Odot** : parsing and printing Graphviz dot files,
- **Okey** : a module to add handlers for key press events in LablGtk2,
- **Rss** : editing, reading and writing RSS 2.0 files,
- **Tdl** : editing, reading and storing to-do lists in XML files,
- **Gtksv_utils** : sharing configuration of sourceviews between applications based on LablGtkSourceView,
- **Custop** : building graphical interfaces for toplevels,
- + various utilities (Gmylist, Gmytree, Gdir, Tmpl-engine, Sqml).

Development tools

- **OCamlcvs** : graphical front-end to CVS,
- **Report** : graphical designer of XML templates,
- **Docbrowser** : browser of ocamldoc dumps,
- **Topcameleon** : graphical front-end to OCaml toplevel,
- **DBForge** : describing database tables and queries and generating OCaml and SQL code to access such databases. Check queries at compile time against the table description.

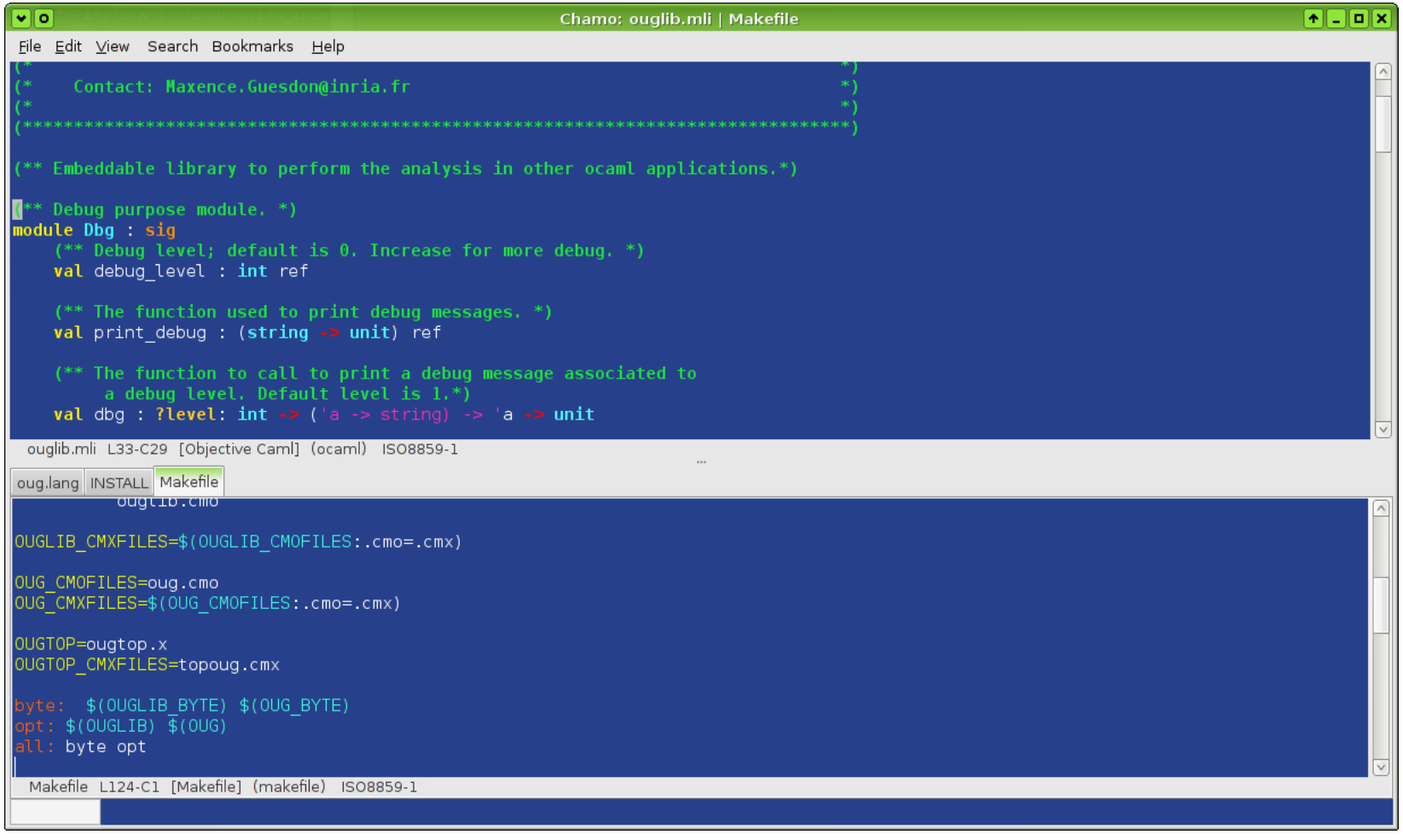
Most of the tools come with a library to be able to include the tools' features into another OCaml/Lablgtk application.

Cameleon itself

- Includes a documentation browser (based on ocamldoc and a library version of docbrowser),
- Offers views (CVS front-end, directory, ...) on ressources (files, directories, ...),
- Include Chamo to provide source code edition ability,
- Should provide "project" features (yet to be defined).

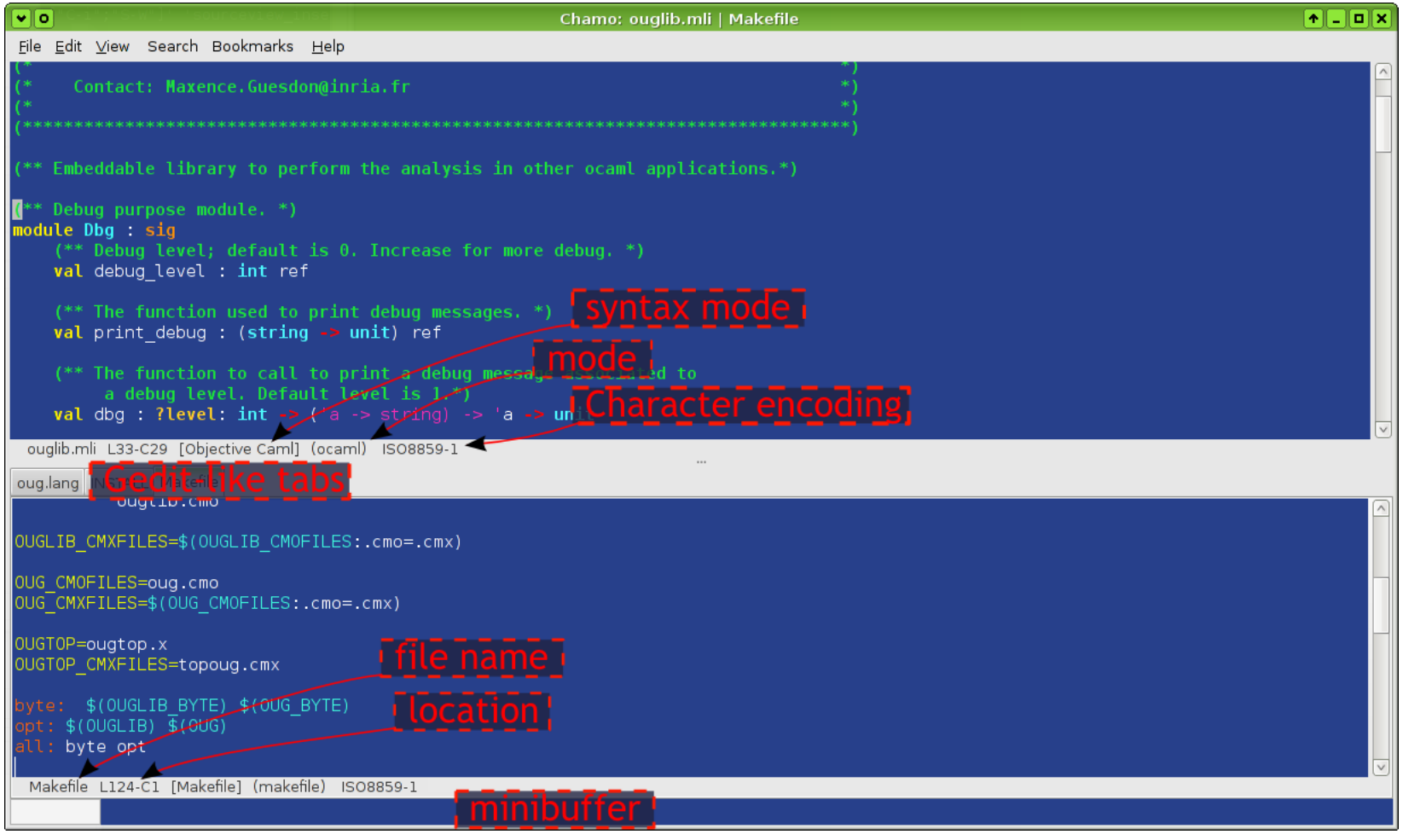
Chamo

An Emacs-like editor with OCaml replacing Emacs Lisp.



Chamo

An Emacs-like editor with OCaml replacing Emacs Lisp.



Commands

Shell-like internal commands : `command-name arg1 arg2 ...`
Ex : `sourceview_switch_line_numbers`, `open_file foo.txt`.

Based on the `Cam_commands` module of Cameleon. Developer and user can define new commands :

```
type command = string array -> unit
type command_desc = {
 com_name : string ;
 com_args : string array ;
 com_more_args : string option ;
 com_f : command ;
}
val register : ?table :(string, command_desc) Hashtbl.t ->
        ?replace :bool -> command_desc -> unit
val register_before : ...
val register_after : ...
```

Views

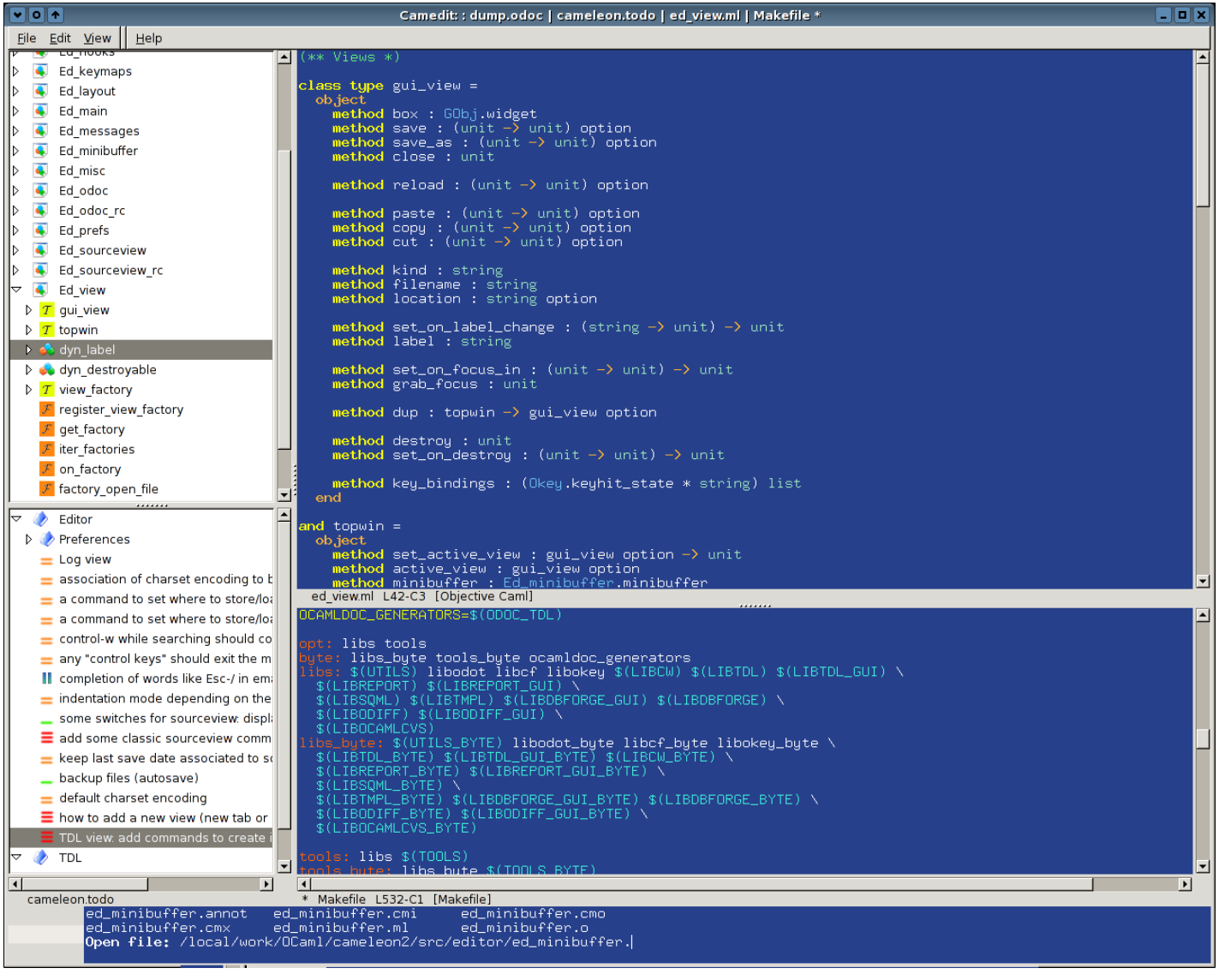
Each window can contain various *views*. A view is any object implementing the `Ed_view.gui_view` class type :

```
class type gui_view =
  object
    method box : GObj.widget
    method close : unit
    method destroy : unit
    method dup : topwin -> gui_view option
    method reload : (unit -> unit) option
    method kind : string
    method filename : string
    method attributes : (string * string) list
    method save : (unit -> unit) option
    method save_as : (unit -> unit) option
    method paste : (unit -> unit) option
    method copy : (unit -> unit) option
    method cut : (unit -> unit) option
    method label : string
    method key_bindings : (Okey.keyhit_state * string) list

    (** The menus to add when this view is activated. The label
        should already be in UTF-8. *)
    method menus : (string * GToolbox.menu_entry list) list

    (* ... *)
  end
```

Some views


Camedit: : dump.odoc | cameleon.todo | ed_view.ml | Makefile *
File Edit View Help
Ed_keymaps
Ed_layout
Ed_main
Ed_messages
Ed_minibuffer
Ed_misc
Ed_odoc
Ed_odoc_rc
Ed_prefs
Ed_sourceview
Ed_sourceview_rc
Ed_view
gui_view
topwin
dyn_label
dyn_destroyable
view_factory
register_view_factory
get_factory
iter_factories
on_factory
factory_open_file
Editor
Preferences
Log view
association of charset encoding to b
a command to set where to store/loa
a command to set where to store/loa
control-w while searching should co
any "control keys" should exit the m
completion of words like Esc-/ in ema
indentation mode depending on the
some switches for sourceview: displa
add some classic sourceview comm
keep last save date associated to s
backup files (autosave)
default charset encoding
how to add a new view (new tab or
TDL view: add commands to create
TDL
(** Views *)
class type gui_view =
object
method box : GObj.widget
method save : (unit -> unit) option
method save_as : (unit -> unit) option
method close : unit
method reload : (unit -> unit) option
method paste : (unit -> unit) option
method copy : (unit -> unit) option
method cut : (unit -> unit) option
method kind : string
method filename : string
method location : string option
method set_on_label_change : (string -> unit) -> unit
method label : string
method set_on_focus_in : (unit -> unit) -> unit
method grab_focus : unit
method dup : topwin -> gui_view option
method destroy : unit
method set_on_destroy : (unit -> unit) -> unit
method key_bindings : (Okey.keyhit_state * string) list
end
and topwin =
object
method set_active_view : gui_view option -> unit
method active_view : gui_view option
method minibuffer : Ed_minibuffer.minibuffer
ed_view.ml L42-C3 [Objective Caml]
OCAMLDOC_GENERATORS=$(ODOC_TDL)
opt: libs tools
byte: libs_byte tools_byte ocamldoc_generators
libs: $(UTILS) libodot libcf libokey $(LIBCW) $(LIBTDL) $(LIBTDL_GUI) \
$(LIBREPORT) $(LIBREPORT_GUI) \
$(LIBSQML) $(LIBTMPL) $(LIBDBFORGE_GUI) $(LIBDBFORGE) \
$(LIBODIFF) $(LIBODIFF_GUI) \
$(LIBOCAMLCVS)
libs_byte: $(UTILS_BYTE) libodot_byte libcf_byte libokey_byte \
$(LIBTDL_BYTE) $(LIBTDL_GUI_BYTE) $(LIBCW_BYTE) \
$(LIBREPORT_BYTE) $(LIBREPORT_GUI_BYTE) \
$(LIBSQML_BYTE) \
$(LIBTMPL_BYTE) $(LIBDBFORGE_GUI_BYTE) $(LIBDBFORGE_BYTE) \
$(LIBODIFF_BYTE) $(LIBODIFF_GUI_BYTE) \
$(LIBOCAMLCVS_BYTE)
tools: libs $(TOOLS)
tools_byte: libs_byte $(TOOLS_BYTE)
cameleon.todo
* Makefile L532-C1 [Makefile]
ed_minibuffer.annot ed_minibuffer.cmi ed_minibuffer.cmo
ed_minibuffer.cmx ed_minibuffer.ml ed_minibuffer.o
Open file: /local/work/OCaml/cameleon2/src/editor/ed_minibuffer.

View factories and file types

- A view is created by a *view factory* registered with a unique name,
- The user configuration defines *file types* and associates each one to a view factory name,
- The `open_file` command uses the matching factory to create the correct view. A default view factory is also specified by the user's configuration (usually, the "sourceview" view).

See module `Ed_view` for details.

Windows layout

Each window can contain various views organized with tabs and vertical or horizontal splits :

```
class gui_window  :
  ... ->
  object
    ...
    method contents  :
      [ `Notebook of gui_notebook
      | `Paned of gui_paned
      | `View of Ed_view.gui_view ] option
    ...
  end
```

See module `Ed_gui` for details.

The `store_layout` command can be used to store the current layout of all Chamo windows, so that their position, size and view organization are restored at launch time.

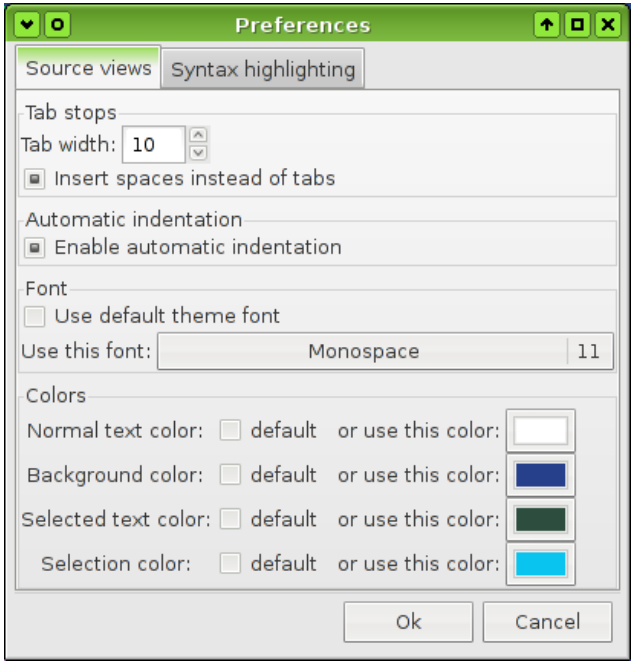
Commands and keyboard shortcuts exist to create, destroy and navigate among views (*C-b*, *C-Tab*, *C-v*, ...).

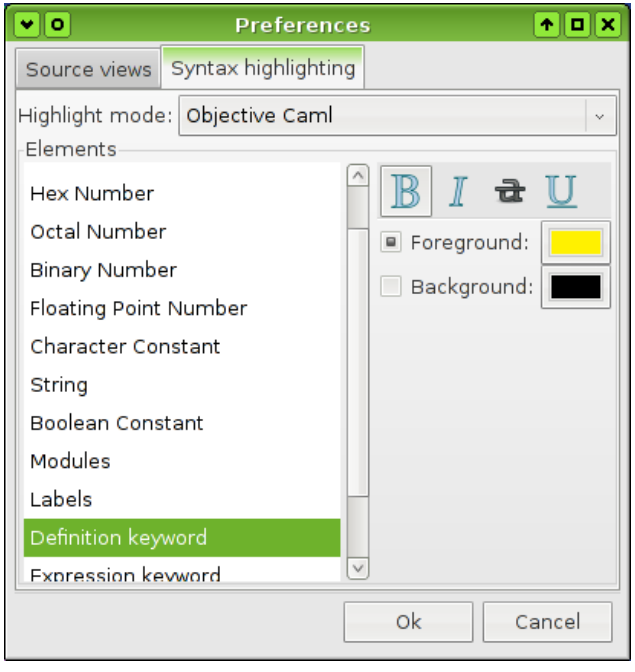
The sourceview view

- Based on the GtkSourceView widget,
- Basic features : query-replace, query-replace regexp, forward and backward incremental search, kill-ring, . . .,
- Location forward-stack (*C-l p* : push, *C-l o* : pop, *C-l f* : forward),
- A buffer can be edited in various sourceviews (à la emacs),
- Each buffer can have its own syntax highlighting mode and its own mode.

Syntax hilighting

- Based on LablgtkSourceView, bindings for GtkSourceView,
- Use Gtksv_utils : configuration of syntax modes and sourceviews are shared with Topcameleon, OugTop, ...
- Graphical interface to choose colors, fonts, ... :





Sourceview modes

A default mode can be associated to each file type. Each mode must implement the `Ed_sourceview.mode` class type :

```
class type mode =
  object
    method key_bindings : (Okey.keyhit_state * string) list
    method menus : (string * GToolbox.menu_entry list) list
    method name : string
    (* ...*)
  end
```

Mapping between file contents and display

Each sourceview mode must have the following methods :

```
method to_display : string -> string
method from_display : string -> string
method set_to_display : (string -> string) -> unit
method set_from_display : (string -> string) -> unit
(* strings are UTF8 strings *)
```

These methods can be used to specify a mapping between contents of a file and its display.

Example : the greek ocaml extension[1] :

```
let φ x = √(x +. 1.);;
print_endline (Printf.sprintf "%f" (φ 2.));;

let α θ κ = let K = κ² in θ /. √K;;
print_endline (Printf.sprintf "%f" (α 2. 10.));;
```

[1] http://home.gna.org/cameleon/snippets/Greek_ocaml.html

Keyboard shortcuts

- Based on the `Okey` library,
- Each window has a common set of keyboard shortcuts (KS),
- Each view defines its own list of additional KS,
- Each sourceview mode defines also additional KS,
- All these lists of KS are aggregated by Okey to create a tree. Each key press event is analyzed to walk (or not) through the tree until a leaf associated to a Chamo command, which is launched.
- All KS are configurable through OCaml code or/and in configuration files.

OCaml as customization language

Bytecode version of Chamo evaluates the contents of :

- `~/.cameleon2/chamo_init.ml` (general configuration),
- `./.chamo_init.ml` (current directory configuration).

These files are regular OCaml code accessing the Chamo modules (Ref. doc. : http://home.gna.org/cameleon/refdoc/index.html).

Moreover, the `eval` chamo command can evaluate OCaml code. For example :

```
eval "GToolbox.message_box \"Message\" \"Hello world !\""
```

Native-code Chamo

Building your own customized native code version of Chamo :

```
make_my_chamo -o mychamo ~/.cameleon2/chamo_init.ml foo.ml ...
```

OCaml-specific features

- indentation of line (`ocaml_indent_line`, *Tab*) or whole buffer (`ocaml_indent_buffer`) ; based on a modified lexer,
- display of type annotations (`ocaml_display_type`, *M-t*),
- switching between `.ml` and `.mli` files (`ocaml_switch_file`, *C-x C-a*),
- launching `ocamlbuild` (`ocaml_build`, *C-o C-c*) and storing the compilation command specified for each file,
- jumping to and highlighting compilation error locations (using a library function analyzing ocaml compilation output).

OCaml-specific features

- indentation of line (`ocaml_indent_line`, *Tab*) or whole buffer (`ocaml_indent_buffer`) ; based on a modified lexer,
- display of type annotations (`ocaml_display_type`, *M-t*),
- switching between `.ml` and `.mli` files (`ocaml_switch_file`, *C-x C-a*),
- launching `ocamlbuild` (`ocaml_build`, *C-o C-c*) and storing the compilation command specified for each file,
- jumping to and highlighting compilation error locations (using a library function analyzing ocaml compilation output).

OCaml-specific features

- indentation of line (`ocaml_indent_line`, *Tab*) or whole buffer (`ocaml_indent_buffer`) ; based on a modified lexer,
- display of type annotations (`ocaml_display_type`, *M-t*),
- switching between `.ml` and `.mli` files (`ocaml_switch_file`, *C-x C-a*),
- launching `ocamlbuild` (`ocaml_build`, *C-o C-c*) and storing the compilation command specified for each file,
- jumping to and highlighting compilation error locations (using a library function analyzing ocaml compilation output).

OCaml-specific features

- indentation of line (`ocaml_indent_line`, *Tab*) or whole buffer (`ocaml_indent_buffer`) ; based on a modified lexer,
- display of type annotations (`ocaml_display_type`, *M-t*),
- switching between `.ml` and `.mli` files (`ocaml_switch_file`, *C-x C-a*),
- launching `ocamlbuild` (`ocaml_build`, *C-o C-c*) and storing the compilation command specified for each file,
- jumping to and highlighting compilation error locations (using a library function analyzing ocaml compilation output).

OCaml-specific features

- indentation of line (`ocaml_indent_line`, *Tab*) or whole buffer (`ocaml_indent_buffer`) ; based on a modified lexer,
- display of type annotations (`ocaml_display_type`, *M-t*),
- switching between `.ml` and `.mli` files (`ocaml_switch_file`, *C-x C-a*),
- launching `ocamlbuild` (`ocaml_build`, *C-o C-c*) and storing the compilation command specified for each file,
- jumping to and highlighting compilation error locations (using a library function analyzing ocaml compilation output).

Writing extensions - A simple LaTeXmode

We define a LaTeXmode by adding this code to our `~/.cameleon2/chamo_init.ml` file :

```
module Latex =
  struct
    (* we define a "latex" output window. These output windows
       are displayed in a separate window dedicated to
       displaying command outputs. *)
    let output_name = "latex"
    let latex_output = ref None
    let latex_output () =
      match !latex_output with
        None ->
          let o = new Ed_outputs.text_output
            ~on_destroy : (fun () -> latex_output  := None)
              output_name
          in
          latex_output  := Some o ;
          o
      | Some o -> o
```

```
(* We define a new function, pdflatex, which launches a xpdf command on
   the pdf file corresponding to the focused .tex file, if any. *)
let pdflatex args =
  (* get the active sourceview, if any *)
  match !Ed_sourceview.active_sourceview with
    None -> ()
  | Some v ->
    (* get the name of the file edited in this view *)
    let file = v#file#filename in
    let dir = Filename.dirname file in
    if Filename.check_suffix file ".tex" then
      begin
        let command = Printf.sprintf
          "(cd %s && pdflatex %s)"
            (Filename.quote dir)
            (Filename.quote (Filename.basename file))
        in
        (* launch the command, displaying the output in the
            our "latex" window.*)
        Ed_ocamlbuild.run ~output : (latex_output()) command
      end
        else
      ()
(* we register this new function, as a command named "pdflatex" *)
let _ = Cam_commands.register
  (Cam_commands.unit_com "pdflatex" pdflatex)
```

```
(* We define our mode name and its configuration options.*)
let mode_name = "latex"
(* mode_rc_file is a convenient function to create a configuration filename
   in the user's ~/.cameleon2 directory, following the naming convention of Chamo. *)
let rc_file = Ed_sourceview_rc.mode_rc_file mode_name
let group = new Config_file.group
let default_key_bindings  = [
        [[`MOD1], GdkKeysyms._p], "pdflatex" ;
      ]
let key_bindings =
  new Config_file.list_cp Ed_config.binding_wrappers ~group
    ["key_bindings"] default_key_bindings "Key bindings"
(* we create functions read and write our mode's configuration file *)
let read () = group#read rc_file
let write () = group#write rc_file
```

The configuration file is

`~/.cameleon2/chamo.sourceview.mode.latex`

and looks like :

```
(* Key bindings *)
key_bindings =
  [(["A-p"], pdflatex)]
```

```
(* The class for our mode. *)
class latex_mode =
  object
    inherit Ed_sourceview.empty_mode
    method name = mode_name
    method key_bindings : (Okey.keyhit_state * string) list =
      key_bindings#get
    method menus : (string * GToolbox.menu_entry list) list = []

    initializer
      read() ; write()
  end
let latex_mode = new latex_mode
let _ = Ed_sourceview.register_mode latex_mode
let (add_sourceview_mode_latex_key_binding,
     add_sourceview_mode_latex_key_binding_string) =
       Ed_sourceview_rc.create_add_sourceview_mode_binding_commands
       key_bindings latex_mode#name
end ; ;
```

What's next ?

- Cameleon2 :
  - Complete integration of Chamo into Cameleon2,
  - Add a kind of minibuffer to Cameleon2,
- Chamo :
  - improve the ocaml code indenter,
  - a Oug view, currently in development,
  - more modes,
  - interface to other development tools (OCamlwizard, cmigrep, ...) to provide standard IDE features (idents completion based on types, refactoring, ...),
  - a standard way to indicate includes and other compilation directives, so that some external programs car be used easier (ocamlbuild, oug, ...)
  - interface to ocamldebug (when ocamldebug functionalities will be available through a library),
- Other tools/libraries :
  - OCaml-SVN/darcs/git ?

What's next ?

- Cameleon2 :
  - Complete integration of Chamo into Cameleon2,
  - Add a kind of minibuffer to Cameleon2,
- Chamo :
  - improve the ocaml code indenter,
  - a Oug view, currently in development,
  - more modes,
  - interface to other development tools (OCamlwizard, cmigrep, ...) to provide standard IDE features (idents completion based on types, refactoring, ...),
  - a standard way to indicate includes and other compilation directives, so that some external programs car be used easier (ocamlbuild, oug, ...)
  - interface to ocamldebug (when ocamldebug functionalities will be available through a library),
- Other tools/libraries :
  - OCaml-SVN/darcs/git ?

What's next ?

- Cameleon2 :
  - Complete integration of Chamo into Cameleon2,
  - Add a kind of minibuffer to Cameleon2,
- Chamo :
  - improve the ocaml code indenter,
  - a Oug view, currently in development,
  - more modes,
  - interface to other development tools (OCamlwizard, cmigrep, ...) to provide standard IDE features (idents completion based on types, refactoring, ...),
  - a standard way to indicate includes and other compilation directives, so that some external programs car be used easier (ocamlbuild, oug, ...)
  - interface to ocamldebug (when ocamldebug functionalities will be available through a library),
- Other tools/libraries :
  - OCaml-SVN/darcs/git ?