XCORAL Text Editor

For Unix Users

Release 3.45

May 30, 2006

Lionel Fournigault, Bruno Pagès and Dominique Lévêque

Contents

Part I

Xcoral

Chapter 1

Introduction

1.1 Overview

xcoral is a multiwindow mouse-based text editor for the X Window System[1]. A built-in browser enables you to navigate through C functions, C++ classes, methods and files. A SMall Ansi C Interpreter (*smac*) is also built-in to dynamically extend the editor's possibilities (user functions, key bindings, modes etc). Syntax highlighting (a la volee) and auto-indent are available. *xcoral* provides variable width fonts, search, regions, kill-buffers, macros and unlimited undo. An on-line manual box, with a table of contents and an index helps you to use and customize the editor. Commands are accessible from menus or key bindings. *xcoral* is a direct Xlib client.

1.2 Copyright

xcoral is a free software distributed on the *Internet*.

[1] X Window System is a registered trademark of The Massachusetts Institute of Technology

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The complete text of the GNU General Public License can be found in this document appendix.

Chapter 2

Editing

If you want to create, read/write files, just type 'xcoral' or 'xcoral filename [filename1, filename2, etc]'. The main editor commands, available from menus, allow you to easily carry out standard files operations and many other facilities.

2.1 Enter text

At start-up, the application provides an edition area with text window, menubar at the top, scrollbar at the right side and control-panel at the bottom (see figure §2.1 page 12). If the pointer is inside the text window (in X terminology this window has the focus), the characters typed are inserted at the current position, which is the cursor position. You can change the cursor position within the text with keyboard commands or with a mouse left button click after moving the pointer to the desired point.

2.2 Control-panel

The control-panel is divided into four parts. From left to right, a status window, a message window (this is also the mini-buffer location), a mode-name window (which indicates the current mode) and buttons (see figure §2.1 page 12).

In the status window, an 'S' is displayed if the buffer has been modified and not saved.

The six buttons are used to move through the buffer (top, bottom, smooth scroll up and down, previous and next page).

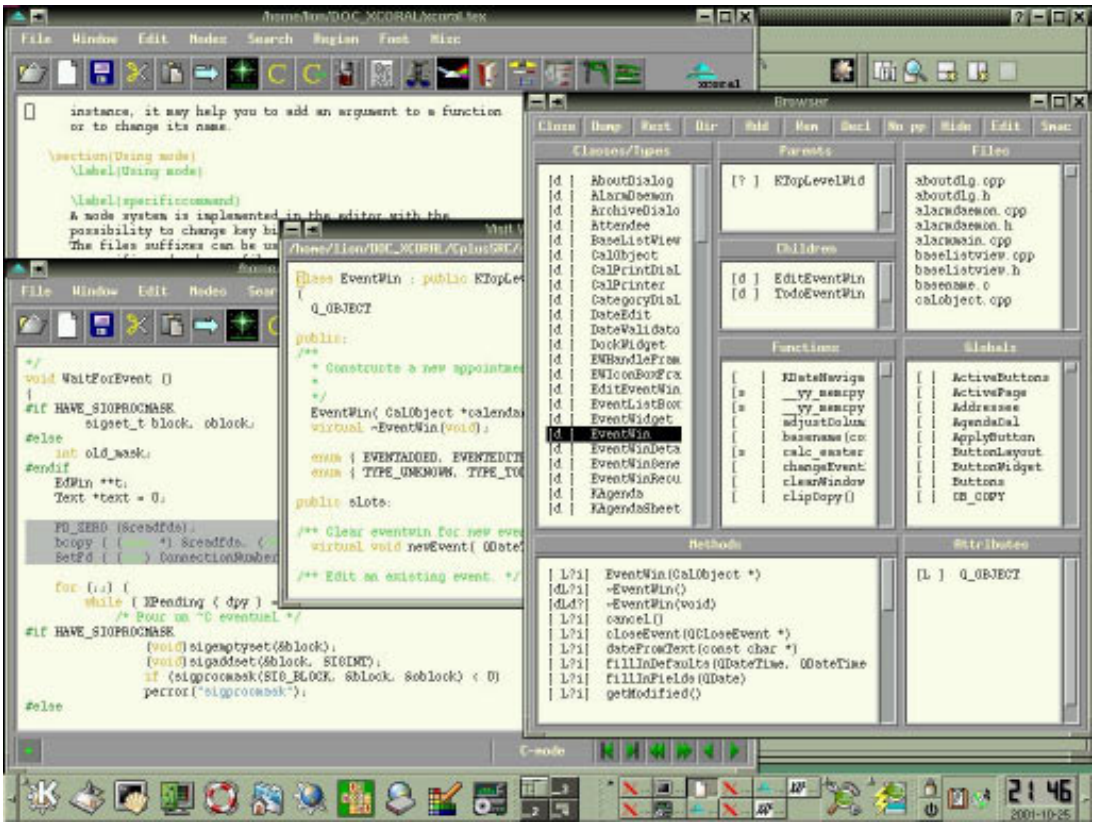


Figure 2.1: Xcoral session.

2.3 Multiple windows

Within a session, several edition areas can be used (Max: 32). A new edition area is opened with *New text window* in the **Window** menu. Each of them inherits its current directory from its parent. The first window has the current working directory. The directory of a text window is that of the loaded file.

2.4 Mouse

Each of the three mouse buttons can be used to manage XCORAL menus, buttons scrollbars and selection.

- **LeftButton click**

  Changes cursor position within a page. This means that the new position will be the pointer position, with two exceptions: a click beyond the end of a line (resp. of the buffer) moves the cursor to the end of that line (resp. of the buffer).

  If you grab the pointer when the left button is pressed, the standard X selection mechanism runs until the button is released.

  A new click releases the selection.

- **RightButton click**

  Selects region. The cursor moves at the selected point.

  A region is selected and highlighted from the previous location. If a selection already exists, it is extended to the current position.

- **MiddleButton click**

  In standard text window, restores the last selection at the current position. The selected text is inserted before the current position.

  In the browser visit window, colors the current buffer.

- **Control key + MiddleButton click**

  If a portion of text has been selected, this action erases it. The pointer position is unimportant for this sequence.

  When a text region has been erased it can be restored at any position by a MiddleButton click.

2.5 Commands

Most commands available from menus have their corresponding predefined key bindings which can be one of the following key sequences:

- **Control with key**

  For instance *Ctrl-s* which stands for 'forward search', means that the 's' key must be pressed while the control key is active (pressed).

- **Control with key then key**

  For instance *Ctrl-x i* which stands for 'insert file', means that the 'i' key must be pressed alone after the sequence *Ctrl-x*. Another example could be *Ctrl-x Ctrl-f* which stands for 'read file', it means that the 'x' key then the 'r' key must be pressed while the control key stays active.

- **Escape then key**

  For example *Esc q* which stands for 'query replace' means that the escape key must be pressed and released and then that the 'q' key must be pressed.

Important: In all cases the sequence *Ctrl-g* aborts the current sequence (i.e. the internal automaton comes back to its default state).

Using key bindings to drive the editor is more efficient and quicker. For users who are not familiarized with this mechanism, this means that the control and escape keys become the main keys. The advantage is that you can enter text and commands without having to move your hands to track the mouse, then open menus, then move pointer again, then release button, and finally come back to the keyboard.

Here is the complete default command list for all modes (see also the specific modes command list §4.3 page 31):

- **Move commands**

Ctrl-a	Move to beginning of line.
Ctrl-b	Move to backward char.
Ctrl-e	Move to end of line.
Ctrl-f	Move to forward char.
Ctrl-n	Move to next line.
Ctrl-p	Move to previous line.
Ctrl-v	Move to next page.
Ctrl-x l	Move to line number.
Ctrl-x m	Move to mark.
Ctrl-x Ctrl-x	Exchange cursor/mark.
Esc v	Move to previous page.
Esc <	Move to first page.
Esc >	Move to last page.
Down arrow	Move to next line.
Left arrow	Move to backward char.
Right arrow	Move to forward char.
Up arrow	Move to previous line.
R7	Move to first page.
R8	Move to previous line.
R9	Move to previous page.
R10	Move to backward char.
R12	Move to forward char.
R13	Move to last page.
R14	Move to next line.
R15	Move to next page.

- **Modify commands**

Crtl-d	Delete current character.
Crtl-h	Delete previous character.
Ctrl-i	Insert Tab.
Ctrl-j	Insert Return.
Crtl-k	Delete end of line.
Ctrl-m	Insert Return.
Ctrl-o	Open space.

- **Mark and region commands**

Ctrl-space	Set mark.
Ctrl-x Ctrl-x	Exchange cursor/mark.
Ctrl-y	Paste previous kill.
Ctrl-w	Cut region.
Esc w	Copy region.
Esc e	Eval region.

- **Search commands**

Esc q	Query replace.
Esc r	Global replace.
Ctrl-s	Forward search.
Ctrl-r	Backward search.

- **Files commands**

Ctrl-x b	Display open files.
Ctrl-x i	Insert File.
Ctrl-x Ctrl-s	Save file.
Ctrl-x Ctrl-f	Read File.
Ctrl-x Ctrl-w	Write file as.
Ctrl-x d	Dump browser database.
Ctrl-x r	Restore browser database.

- **Windows commands**

Ctrl-l	Refresh current page.
Ctrl-x n	Open new edition area.
Ctrl-x Ctrl-c	Delete window.
Ctrl-x e	Open edir window.
Ctrl-x k	Clear current buffer.

- **Misc commands**

Ctrl-c	Play macro.
Crtl-g	Abort current command.
Ctrl-q	Quote char.
Ctrl-t	Redo.
Ctrl-u	Undo.
Ctrl-x (	Learn macro.
Ctrl-x)	End macro.
Ctrl-x Ctrl-e	Eval expression.
Esc =	Display current line number.
Esc digits	Get repeat number.

2.6 Mini-buffer

A few commands are interactive commands, for which the user must reply to a question. From menus, these commands display a dialog box with an input field string, an *Ok* button and a *Cancel* button. The input field string is called a mini-buffer.

Some commands are available in the mini-buffer to edit a line of text:

Ctrl-a	Go to beginning of line.
Ctrl-b	Backward char.
Crtl-d	Delete current character.
Ctrl-e	Go to end of line.
Ctrl-f	Forward char.
Crtl-g	Abort.
Crtl-h	Delete previous character.
Crtl-k	Delete end of line.
Crtl-p	Restore previous user answer.
Tab	Expand file, tilde or Smac identifier.

Two of them, *Ctrl-p* and *Tab* in the mini-buffer context, do not have the same function as in default mode.

Tab provides an expansion mechanism for file names. The beginning of a filename being entered in the mini-buffer, *Tab* key will complete it. However, if several filenames match the current pattern, the *Message box* is displayed with the possible choices. **Note:** the tilde character is expanded to the homedir path of the user.

In the same way, *Tab* provides an expansion mechanism for SMAC names during an eval expression command. SMAC names can be functions or variables names.If the expansion is not possible, a bell rings.

Ctrl-p restores the last string entered in the mini-buffer. This is useful when you make a mistake.

Cut and paste facilities (see §2.4 page 13) can be used in the mini-buffer (one line only). The mini-buffer is shared by all buffers of a session and is used in dialog-boxes, file-selectors and control-panels.

2.7 Erasing text

Erasing text can be handled in five ways.

- With the mouse and the standard selection mecanism. (see §2.4 page 13).
- With a mark and the 'cut region' command (*Ctrl-w*). The text between the mark and cursor is removed (see §2.9 page 19).
- With the 'kill command' (*Ctrl-k*) which deletes the end of the current line. The *Esc digits* command (you type *Esc* then enter a number), allows you to set a number which must be used with *Ctrl-k*. In this case, 'num' lines from the cursor are removed.

  Note that, the 'Esc digits' command is used in two cases only: to delete several lines and to play a macro several times (see §2.10 page 19).
- With the *delete* or *backspace* keys which erase the character before the cursor.
- With the *Ctrl-d* command which erases the character under the cursor.

When deleted, text is stored in an unlimited linked kill-buffer list (each deleted text can be restored). This kill-buffer can be mapped with *Display Kill Buffer* in the **Window** menu. It is viewed as a scrolling list with the beginning of each entry. The first entry corresponds to the last remove operation. When selected, an entry is restored at the current position. The kill-buffer is shared by all buffers of a session.

2.8 Searching

Searching operations use the famous Boyer-Moore algorithm, with global buffers to store strings to search for or replace. These commands prompt a mini-buffer or a dialog box to enter strings whether you use key bindings or menus.

Ctrl-g resets the searching commands and new strings can be entered.

- **Forward and backward search**

  The *Ctrl-s* or *Ctrl-r* commands can be repeated in the current text window as well as in any other one.

- **Query replace**

  The *Esc q* command prompts the mini-buffer, first to set the old and new strings, then to replace (y), skip (n) or quit (q). *Query replace* starts from the current position.

- **Global replace**

  The *Esc r* command also prompts the mini-buffer to set strings, but replaces all silently from the current position to the end of the buffer.

2.9 Mark and region

A mark allows you to save a cursor position and to come back to it later. You can set only one mark per text window.

A region is a portion of text between the mark and the cursor. A region can be copied, deleted, restored, indented (mode C, C++, Latex, etc) or evaluated (the region must be a valid SMAC expression of definition).

- **Ctrl-space** sets the mark at the current position. When the text is modified, the mark is updated when allowed.

- **Ctrl-x m** moves to the mark.

- **Ctrl-x Ctrl-x** exchanges current position with mark when it is set.

- **Esc w** copies the region between mark and cursor in the kill-buffer. It can be pasted at any current position with the command *Ctrl-y* or with the kill buffer box.

- **Ctrl-w** deletes the region between mark and cursor. The corresponding portion of text is saved in the kill-buffer and can be restored at any current position with the command *Ctrl-y* or with the kill-buffer box.

2.10 Macros

A **macro** memorizes a sequence of commands.

- **Ctrl-x (** starts a macro definition in the current window. The following commands are saved during their execution. Simultaneously the message *Learn macro* is displayed.

  ⚠ During macro definition you can switch to another window but in this case the macro definition is suspended until you move the pointer back to the window where you started the macro.

- **Ctrl-x)** ends the macro definition. It is available from any text window.

- **Ctrl-c** plays the macro. All commands previously recorded are executed in order. With the commands *Esc digits* then *Ctrl-c*, the macro is played *num* times (see §2.7 page 18).

Example:

In a buffer of 100 lines, you want to add the string 'debut:' at the beginning of each line and the string ':fin' at the end of each line. The following macro does this work: at the beginning of the first line type the following sequence: *Ctrl-x (debut: Ctrl-e :fin Ctrl-f Ctrl-x)*. Then *Esc 99* and *Ctrl-c*.

⚠ A macro can include only one search command. If you want to use several search commands or more generally, if you want to implement a complex macro, it is better to write a *smac* function (see §4.9 page 43).

2.11 Undo redo

In each text window, every modification of the buffer is saved in an unlimited linked LIFO list. The modifications can be undone in order with the undo command *Ctrl-u*, up to the first modification. When the buffer is saved as a file, the undo list is reseted. It is possible to cancel one or several undo commands with *Ctrl-t* (i.e. redo).

The undo mechanism implemented in *xcoral* is simple. It does not take into account context or semantics, but only modifications. This means that several *Ctrl-u* commands show you exactly what you have done in the buffer (i.e. inserted/deleted characters, copied/deleted regions, cut and paste with mouse, etc).

2.12 Scrolling

In all text windows, a scrollbar is controllable by any mouse button to move the text vertically in the current window. A click inside the bottom or top part of the scroll

area moves the text to the next or previous page.

In every text window, an horizontal scroll may be handled with the left mouse button click on the last visible character at right or first visible character at left.

Chapter 3

Browser

A browser is available, in C and C++ mode, to navigate through C functions and C++ types (included typedef, class and struct) hierarchies, methods, attributes, functions and globals. The names of files, functions, types, parents, children, attributes, methods and globals are listed in subwindows of the control browser window. An object selected by one mouse click in one of these subwindows is viewed in a 'Visit Window'. A double click opens an edition area and loads the current selected object (see figure §3.1 page 26).

The browser can also be used to navigate through Java classes hierarchies when the Java mode is selected. In fact a simple filter is applied to the Java sources and the C++ browser parser is used (see java.sc).

3.1 Browser database

The *xCORAL* browser builds its database by parsing one or more source files or full directories. It collects useful information about functions, globals, types, methods, attributes and inheritance in C/C++ environments only. The browser parses files with the following default suffixes: *.c,.h,.cc,.cxx,.hxx and.C*. Of course it is possible to add other suffixes (See Using *SMAC* §4.3 page 31).

During *xCORAL* start-up, the browser does not parse any directory to reduce the start-up time. However an environment variable XCORAL_PARSE_PATH can be set to parse directories, like the classic PATH, with the standard separator ':'.


In this case the directories are parsed recursively.

The browser database is automatically updated after all write/save file operations.

To reduce information displayed on the browser window, you can hide some of them, see *Hide* browser button and SMAC functions (§4.7 page 39, §7.3.12 page 133).

To take into account macros and conditional compilations, it is possible to ask for a pre-processing before file parsing, see *Prepr/No pp* browser toggle button and SMAC functions (§4.7 page 39, §7.3.12 page 123).

Because pre-processing and parsing many files may take a long time, you can dump and restore the browser data base, see *Dump* and *Rest* browser buttons and SMAC functions (§4.7 page 39, §7.3.12 page 124).  Dumping is not made automatically after write/save file operations.

3.2 Browser control

The browser is useful only for C, C++ and Java programmers. When used in C mode it is displayed with 5 subwindows (functions, classes/types, attributes, files, and globals). In C++ mode (default), all subwindows appear (classes/types, parents, children, files, methods, attributes, functions and globals).
In Java mode, functions and globals windows are not displayed.

Loading files with C, C++ or java suffixes does not change the browser control mode. To change it, you must select the desired mode using the **Modes** menu.

In addition, a toolbar is available.

- The 'Dir/Rec' button is a toggle button to set a 'recursive' flag used during parsing directories. At start-up, if the environment variable XCORAL_PARSE_PATH is used, the recursive flag is true. In that case (Rec), the selected directories and all their subdirectories are parsed, otherwise only the files of the current directory are parsed (default).
- The 'Add' button is used to parse a selected file or directory via the file selector window, and load information in the browser database.
- The 'Rem' button is used to remove browser data for the selected file. One file or all files can be selected.
- The 'Decl-Impl' button is a toggle button to see Declaration or Implementation of selected methods.
- The 'Edit' button opens an edition area and loads the current selected object (same as a double click on that objet).

- The 'Close' button unmaps the browser control window and the visit window.
- The 'No pp-Prepr' button is a toggle button to indicate if the files are pre-processed before parsing. When the toggle button becomes 'Prepr' the pre-processor options are asked. See *browser_set_pp()* §7.3.12 page 123 SMAC function which indicates where the pre-processor is (default /lib/cpp).
- The 'Hide' button is used to hide/show some browser information.
- The 'Dump' button is used to save browser database in a file. The dump contains all showed or hiden information, including hide/show control.
- The 'Rest' button is used to restore browser database from a file.
- The 'Smac' button is used to parse the SMAC directorie and load information in the browser database.

Some additional information appear in types, methods, attributes and globals windows.

- **[c/d]** Constructor or destructor (method).
- **[v]** Virtual definition from parents (method).
- **[V]** First virtual definition (method).
- **[L]** Locally defined in current class (method, attribute).
- **[d]** Declared (type, method).
- **[i]** Implemented not inline (type, method).
- **[I]** Inline (method, function).
- **[?]** Not implemented (type, method).
- **[s]** Static (attribute, function, global).
- **[n]** Inheritance level (methods, because only one character is used, 9 is the greater displayed value of n).
- **[T]** Template (type, function)

In each subwindow of the browser control panel, you can type one character to move quickly to the first item starting with this character (case-sensitive).

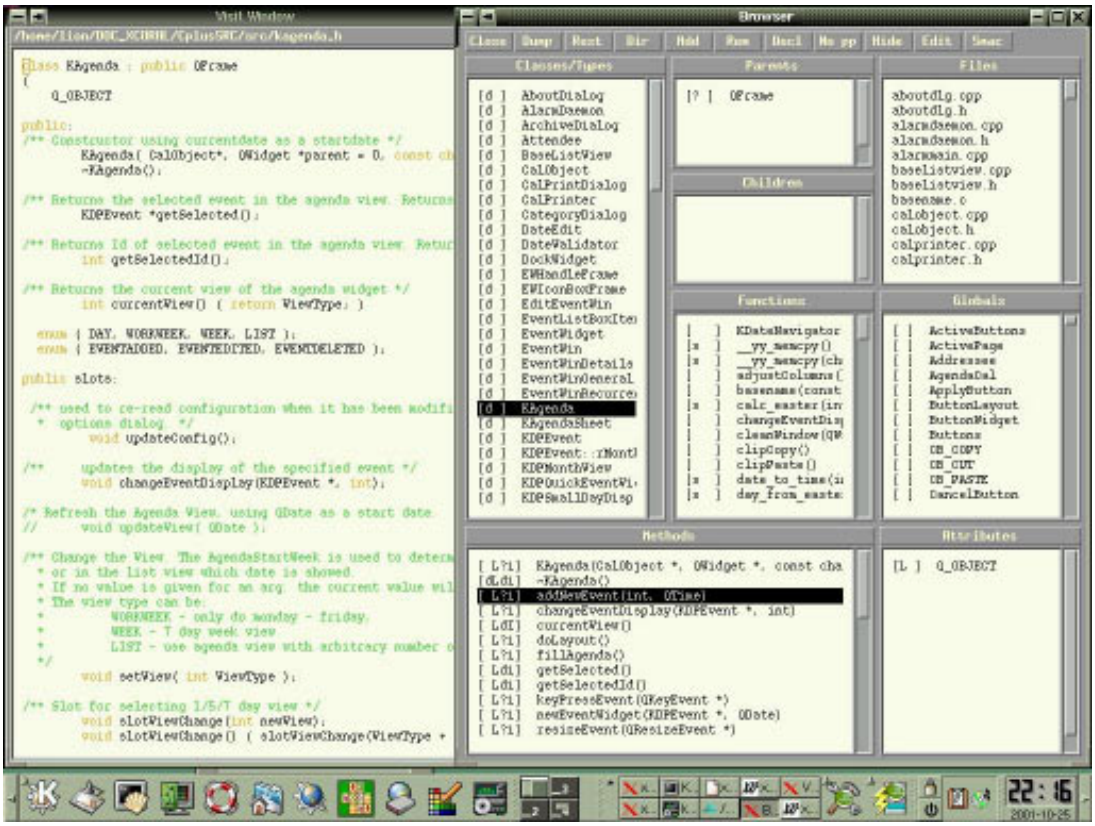


Figure 3.1: Xcoral browser, C++ mode.

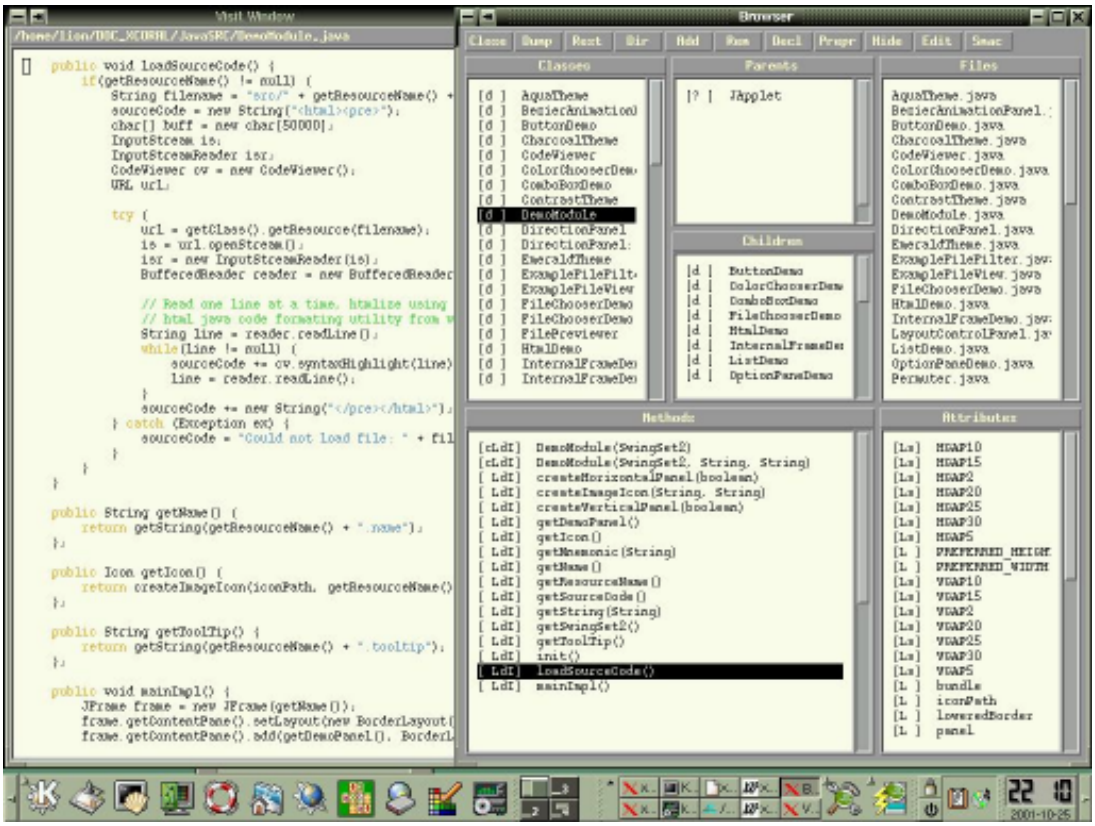


Figure 3.2: Xcoral browser, Java mode.

Chapter 4

Using Smac

This chapter describes, with simple examples, how to use SMAC within XCORAL. For more details about SMAC refer to the third part of this document.

Note 1: In SMAC, characters and lines are numbered from 0.

Note 2: Current position is the cursor position in the current buffer.

4.1 Eval expression

First type the *Ctrl-x Ctrl-e* 'eval expression' command. In the mini-buffer you can enter expressions which are **C** expressions. Typically the following forms are valid:

```
func1();
func1(arg1,...);func2(func3(),arg2,...);
```

In this context, the *Tab* key can be used to expand the SMAC functions, user functions and variables names. If the mini-buffer is empty, a complete list of these is displayed in the message box.

⚠ Don't forget the semicolon ';' at the end of the C expression you want to evaluate, otherwise you will get a *parser error line 0: syntax error* message.

- **insert_char (190);**

  Inserts the argument at the current position. The function *insert_char* is one of the built-in functions available in SMAC to drive the editor. This example shows how to insert a special character (via its *ASCII* code).

- **set_mode_font ("Latex", "9x15bold");**

  This example shows how to change font in Latex mode.

- **cmd_shell ("date | awk '{print $4}' ");**

  cmd_shell is a built-in editor function. The argument is executed in a sub-shell and the result is inserted at the current position.

- **select_window(new_window());read_file(file_select());**

  Opens a new text window, selects it, selects a file from a file-selector and loads it. Obviously it is easier to use the **Window** menu, but this example shows the SMAC facilities to interpret built-in editor functions through an eval expression command.

- **key_def ("C-mode", "^xe", "goto_end_of_file");**

  In C-mode, binds goto_end_of_file function to *Ctrl-x e*. The second argument syntax (key_binding), can be:

```
"c"     --> key.
"^[c"   --> Esc key.
"^[^c"  --> Esc Ctrl key.
"^c"    --> Ctrl-key.
"^xc"   --> Ctrl-x key.
"^x^c"  --> Ctrl-x Ctrl-key.
```

  where `'c'` is any character except `'^'`. Furthermore `'^^'` appoints the character `'^'` (so `^x^^` is `Ctrl-x ^`), `'^ '` and `'^@'` appoint the character null (code 0).

  It is also possible to bind *keysym* keys such as *Home*:

```
"k"    --> key.
"^[k"  --> Esc key.
"^xk"  --> Ctrl-x key.
```

  where `'k'` is a *keysym* name (see §7.3.11 page 122).

4.2 Compiling and searching within Xcoral

Evaluating **make();** runs *make -k* in a sub-shell and displays its results in a new *Shell* mode window. When *make* is finished the contents of this window can be:

```
bm_search.c:128:syntax error
get_file.c:97: syntax error
scroll.c:234:syntax error
```

You can then use the *Ctrl-x g* command (see *go_next* §5.10 page 65) which reads the lines in order (from top to bottom), opens a new text window if needed, loads the erroneous file and goes to the error location. After each *Ctrl-x g*, the cursor moves to the next line in the error list. The first *Ctrl-x g* you type goes to the first line of the current result.

In the same way evaluating **grep ("GetBuffer *.c");** searches the string *GetBuffer* through all files with .c suffix. With the *Ctrl-x g* command in the result window, you can easily navigate through the corresponding files. It is useful when you want to modify a string which appears in many files. For instance, it may help you to add an argument to a function or to change its name.

4.3 Using mode

A mode system is implemented in the editor with the possibility to change key bindings and fonts. The files suffixes can be used to automatically select a specific mode when a file is loaded. Some modes are built-in (default, C-mode, C++mode, Latex, shell, etc).

For more details, see *mode.sc* (§5.29 page 74).

4.3.1 Default mode

This mode provides the standard built-in key-bindings and some escape sequences bound to SMAC functions defined in the *mode.sc* file:

- **Esc f** goes to the end of the current or next word.
- **Esc b** goes to the beginning of the current or previous word.
- **Esc d** deletes the end of the current word or the next word.
- **Esc delete/backspace** deletes the beginning of the current word or the previous word.

- **Esc u** changes the end of the current word or the following word to upper case, and moves after it.
- **Esc l** changes the end of the current word or the following word to lower case, and moves after it.
- **Esc c** capitalizes the end of the current word or the following word, and moves after it.

4.3.2 C-C++ mode

The main advantage of this mode is the automatic indentation when you hit *return*. A blinking mechanism for parenthesis, brakets and braces is also implemented. Some key bindings have been defined for functions, forms and regions. For more details see *mode.sc* in the second part of the document (§5.29 page 74);

Auto-indentation

Indentation is done by the *return* key which inserts a *newline* character and indents the new line. The *Tab key* is used to reindent the current line if needed.

Moreover, some SMAC global variables are available to customize indentation in comment, string, parenthesis, bracket, brace, statement and function arguments.

Blinking

Blinking is used to visualize matching parenthesis, brackets or braces. This means that, for instance, if you insert a ')' character, the cursor moves to the corresponding '(' character and comes back to the current position. Blinking is confined to the current page else a message is displayed (i.e. match at line X pos Y).

Definition, form and region

The following key bindings are also available:

- **Esc ?** on a word acts as select (one click) in the browser window.
- **Esc Esc** on a word is like editing (double click) an object from the browser window.
- **Esc a** goes to the beginning of the current or previous definition.
- **Esc f** goes to the end of the current or next expression.
- **Esc b** goes back to the beginning of the current expression or previous expression.

- **Esc d** deletes the end of the current expression or the next expression.
- **Esc delete/backspace** deletes the beginning of the current expression or previous expression.
- **Esc i** indents region between mark (see §2.9 page 19) and current position.

4.3.3 Java mode

Java mode is an extension of C++ mode. Auto-indentation, blinking and colors are available (see .xcoralrc and java.sc).

4.3.4 Latex mode

This mode (named 'Latex') is a simple mode that wraps lines, colors and indents them following LaTeX keywords.

A fixed font should be used to have the same width for all characters. Therefore, the number of characters in a line depends only on the window width and on the font used. However you can fix the number of characters in a line by replacing *window_width('a')* with the desired value (see *latex.sc* file). Thus, when the current line becomes too long, its last space is changed to \n and the rest of the line is indented to the next line. Consequently, it is not possible to insert a space or a tab at the beginning of a line. To insert *spaces* or *tabs* it is necessary to press *Ctrl-q* before entering them.

To define your own indentation, you can set *latex_indent_step* (default 3) and *latex_indent_chapter* (default 0) to the desired values (see the file *latex.sc*). Bindings available in Latex mode are default mode bindings plus the followings:

- **Esc a** moves to the beginning of the first line containing the previous latex *begin, part, chapter,* [*sub*][*sub*]*section* or [*sub*][*sub*]*paragraph* keyword.
- **Tab** indents the current line.
- **Return** inserts a newline and indents.
- **Esc i** indents the current region.

To automatically set the Latex mode when a file is loaded, use the suffixes **.tex** or **.latex**.

Si vous voulez en plus quelques accents, il suffit d'ajouter dans votre fichier **.xcoralrc** les lignes suivantes:

```
{
key_def("Latex", "'", "french_accent");
key_def("Latex", "`", "french_accent");
key_def("Latex", "^^", "french_accent");
key_def("Latex", "~", "french_accent");
}
```

4.3.5 Shell-Script mode

This mode allows you to edit shell script with auto-indentation and colors highlighting.

4.3.6 Sub-Shell mode

Connects the current text window to a sub-shell. XCORAL_SHELL environment variable is used if defined, else SHELL environment variable is used if defined. If none of these variables is set, `/bin/csh` is used. A prompt is displayed and you can enter UNIX commands. The result of the commands is inserted in the buffer at the current position.

In this mode, all the default key bindings can be used but two of them have been redefined:

- **Ctrl-d** quits the sub-shell mode.
- **Ctrl-c** interrupts a sub-shell command.

⚠, if your *shell* mode does not work correctly (i.e. the prompt is not displayed), use the environment variable XCORAL_SHELL to set a basic shell (sh or csh).

4.3.7 Edit directory mode (Edir)

This mode helps you to navigate through directories and to do some operations on files (i.e. load, copy, change mode, compress etc). The *Edir()* SMAC function which is bound to *Ctrl-x e*, prompts a dialog box and waits for a target directory. A text window is then opened which contains the result of a *ls -alg*. In this window, the *Edir* commands will act on the file or the directory indicated by the cursor (the cursor only need to be on the line of the file or directory chosen).

If you hit the '?' key, the list of the available commands is viewed in the message box.

- **?** shows this list.
- **f** lists the contents of a directory or edits a file in the current window.

- **n** opens a new window and lists the contents of a directory or edits a file.
- < lists the contents of the parent directory in the current window.
- > lists the contents of the parent directory in another window.
- **q** quits the current Edir window.
- **r** rereads the current directory.
- **s** issues a shell command.
- **C** copies the pointed file.
- **G** changes the group of the pointed directory or file.
- **H** creates a hard link to a target directory or file.
- **M** changes the mode of the pointed directory or file.
- **P** prints the pointed file.
- **R** renames the pointed directory or file.
- **S** creates a symbolic link to a target directory or file.
- **Z** compresses or uncompresses the pointed file.
- = calls a *diff* command.

4.3.8 Html mode

This mode helps you to write *Html* pages (see the file *html.sc*). Through a menu, it provides most common Html 3.0 tags (links, forms, lists, styles, etc). Html tags are automatically colored.

4.3.9 Other modes

Some other modes are predefined for standard languages (Perl, Ada, Fortran and shell-script). These modes, available from the **Modes** menu, only offer color facilities

There is no auto-indentation as with *C, C++, Java, Shell* or *Latex* modes. To define your own indentation for theses modes, look at the file *mode.sc* as example.

4.4 C-C++ headers

For C-C++ and Java programmers, the *head.sc* file provides the following functions:

cplus_class_header(), java_class_header(), method_header(), function_header() and include_header().

Of course, these functions can be modified as you want. For instance, the first of them, function_header() prompts a dialog box to enter a C function name (fname) and inserts at the current position the following header:

```
/*
 * Function name: fname
 *
 * Description:
 * Input:
 * Output:
 */
fname()
{

}
```

4.5 Regular expressions

A regular expression (regexp) is a text string that describes some set of strings.

Functions that handle regular expressions, based on GNU **regexp-0.12**, have been implemented (for more details, see the GNU documentation about regexp rules).

The functions available from **Search** menu provide search forward or backward and replace. Each of them prompts a dialog box to get the target regexp.

Regular expressions are composed of characters and operators that match one or more characters. Here is an abstract of commons operators:

- **.** matches any single character.
- ***** repeats the smallest possible preceding regular expression as many times as necessary (including zero).
- **+** is similar to the previous operator except that it repeats the preceding regular expression at least once.

- **?** is similar to the * operator except that it repeats the preceding regular expression once or not at all.
- `|` matches one of a choice of regular expressions.
- `[...]` matches one item of a list.
- `[^...]` matches a single character not represented by one of the list items.
- **-** recognizes characters that fall between two elements.
- `(...)` treats any number of other operators (i.e. subexpressions) as a unit.
- **\digit** matches a specified preceding group.
- `^` matches the beginning of line.
- `$` matches the end of line.

SMAC provides the following functions:

- **int re_forward_search(char *regexp);**

  returns the position of the next regular expression *regexp*, or -1 if *regexp* has not been found, or -2 if *regexp* is not valid.

- **int re_backward_search(char *regexp);**

  returns the position of the previous regular expression *regexp*, or -1 if *regexp* has not been found, or -2 if *regexp* is not valid.

- **int re_match_beginning(int n);**

  returns the beginning position of the substring n of the regexp found by the previous search call to a regexp.

- **int re_match_end(int n);**

  returns the end position of the substring n of the regexp found by the previous search call to a regexp.

- **int re_replace(char *newstring [,char *regexp]);**

  replaces the regular expression *regexp* with the string *newstring*. If the argument *regexp* is omited, the previous search call to a regexp is used. It returns 1 on success else 0.

Example:

```
/* look for C++ line comment */
re_forward_search ("//.*$");
/* move to the beginning of the searched expression */
goto_char (re_match_beginning(0));
```

4.6 Color syntax highlighting

In every mode (i.e. built-in modes and user modes), keywords or expressions can be set off by colors with the following SMAC function:

- **void color_area(int start, int end, char *colorname)**;

  colors text region from positions *start* to *end* with the color *colorname* (see the standard X11 file rgb.txt).

This function can be used with the SMAC regexp interface to color strings of your own modes (see color.sc §5.11 page 67).

Example:

```
/* look for a regular expression */
re_forward_search (regexp);
/* color text region */
color_area (re_match_beginning(0),re_match_end(0), colorname);
```

The colornames and the regular expressions are predefined in the file *color.sc* for the following modes: **C, C++, Java, Latex, Html, Perl, shell-script** and **Edir**. The standards *Makefile, makefile, and Imakefile* can also be colored. You can modify them or add colors for your own mode.

Default key bindings are:

- **Ctrl-x a** colors buffer.

4.7 Browser

SMAC allows to customize the browser and to access its database. Here are some functions:

- **void browser_set_pp(char * mode, char * exec)**
- **void browser_set_pp_options(char * mode, char *options)**

Your **.xcoralrc** file is a good place to indicate where the pre-processors are (see §7.3.12 page 123), and their options. *browser_set_pp_options* function (see §7.3.12 page 124) forces browser toggle button to *Prepr* if its second argument is not 0, even if the pre-processor option list is empty (i.e. the second argument is the empty string "").

```
browser_set_pp("C-mode", "/lib/cpp"); /* default */
browser_set_pp("C++mode", "/lib/cpp"); /* default */

browser_set_pp_options("C-mode", "-DDEBUG");
browser_set_pp_options("C++mode", 0); /* no pre-processing in c++ */
```

- **void browser_add(char * file_or_dir [, int rec])**

  parses the given file or directory, forces the browser toggle button to *Rec* if the second argument is given and not equal to 0.

- **int browser_dump(char * file)**

  returns 1 if the dump can be realized, else 0 (file is write protected or other error).

- **int browser_restore(char * file)**

  returns 1 on success, else 0 (file doesn't not exist or other error).

- **void browser_show_all()**

  to see all browser information.

- **char ** browser_functions(char * prefix)**

  returns the functions list which name begins with *prefix*

```
void write_functions_description(char * prefix)
{
   char ** p, ** all = browser_functions(prefix);

   if (! all) return;

   for (p = all; *p; p += 3)
      printf("%s is defined in %s line %u\n",
             p[0]+7, p[1], *((int *)&p[2]));

   free(all);
}

{ write_functions_description("a");}
{ write_functions_description("" /* !! */);}
```

4.8 Writing new functions

The following simple examples can be written and tested on-line. First you must open a new window to write these functions (see *example.sc*). Before evaluating one or all functions with *Esc e*, you must select it or them in a region (see §2.9 page 19). To execute a function use *Ctrl-x Ctrl-e* then enter function_name(); and hit return.

- **Using Xcoral built-in functions**

```
HiddenMessage()
{
    int c;
    select_window(new_window());
    insert_string("Ydpsbm ju hppe gps zpv");
    while( current_position() != 0) {
        goto_previous_char();
        redisplay ();
    }
    while(current_position() != end_of_line() ) {
        c = current_char();
        delete_char();
        if ( c != ' ' )
            c -= 1;
        insert_char(c);
        redisplay();
    }
    wprintf("\nYes\n");
}
```

- **Using Xcoral boxes**

```
SomeBoxes()
{
    char *str;
    display_message("\nDialog box test: ");
    /* Message box */
    str = gets ("write something ");
    /* Dialog box */
    if ( str != 0) {
        display_message(str);
        free(str);
    }
    display_message("\nFile selector test: ");
    str = file_select(); /* File selector */
    if ( str != 0) {
        display_message(str);
        free(str);
    }
    display_message("\nList box test: ");
    clear_list(); /* List select */
    add_list_item ( "Choice 1" );
    add_list_item ( "Choice 2" );
    add_list_item ( "Choice 3" );
    str = select_from_list("My list");
    if ( str != 0 && strlen(str) > 1 )
        display_message(str);
    }
}
```

4.9 Built-in editor functions

These functions are the lowest level interface with the editor. With them you can build your own SMAC functions during a session and save them in a file.

- **Get information about position and current buffer**

int	at_end_of_file ()
int	beginning_of_line()
char	current_char ()
int	current_line()
int	re_match_beginning(int subs)
int	re_match_end(int subs)
char	next_char ()
char	previous_char ()
int	end_of_file ()
int	current_position()
int	end_of_line ()
char	the_char (int pos)
int	line_count ()
char	last_key ()

- **Search**

int	backward_search (char *str)
int	forward_search (char *str)
void	replace_char (int c)
void	blink (int pos)
int	msearch (char *chars, int end, int direction)
int	global_replace (char *old, char*new)
int	re_forward_search (char *regexp)
int	re_backward_search (char *regexp)
int	re_replace (char* newstring, [char *regexp])
int	re_match_beginning (int n)
int	re_match_end (int n)

- **Move**

void	current_line_to_top ()
void	goto_beginning_of_line ()
void	goto_char (int pos)
void	goto_next_char ()
void	goto_end_of_line ()
void	goto_end_of_file ()
void	goto_previous_char ()
void	goto_mark()

- **Modify current buffer**

void	delete_char ()
void	insert_char(int c)
void	read_file (char *file_name)
int	insert_file (char *file_name)
void	insert_string (char *string)
void	kill_current_buffer ()

- **Window**

int	new_window ()
int	kill_window (int num)
void	lower_window ()
void	raise_window ()
int	current_window ()
int	window_width (int c)
int	window_height ()
void	select_window (int win_id)

- **Mode**

void	set_mode (char *name)
void	create_mode (char *mode_name)
char *	current_mode()
void	key_def (char *m_name, char*keys, char *f_name)
void	set_mode_font (char *mode_name, char *font_name)
void	set_mode_suffixes (char *mode_name, char *suf)

- **Mark**

void	set_mark (int pos)
void	goto_mark()
void	reset_mark()
int	mark_position()
void	copy_region()
void	paste_region()
void	cut_region()

- **browser-1**

void	browser_set_pp(char * mode, char * exec)
void	browser_set_pp_options(char * mode, char * options)
void	browser_add(char * path [, int rec])
void	browser_del([char * name])
char *	browser_class_file(char * class_name [, int * pline])
char **	browser_class_parents(char * class_name)
char **	browser_class_children(char * class_name)
char **	browser_class_methods(char * class_name)
char **	browser_class_attributes(char * class_name)
char *	browser_class_flags(char * class_name)
char **	browser_functions(char * prefix)
char **	browser_globals(char * prefix)
int	browser_select_class(char * prefix)
int	browser_select_method(char * prefix)
int	browser_select_attribute(char * prefix)
int	browser_select_function(char * prefix)
int	browser_select_global(char * prefix)
void	browser_edit()
int	browser_dump(char * filename)
int	browser_restore(char * filename)

- **browser-2**

int	browser_class_entry(char * prefix)
int	browser_function_entry(char * prefix)
int	browser_global_entry(char * prefix)
int	browser_file_entry(char * prefix)
char *	browser_class(int n)
char *	browser_function(int n)
char *	browser_global(int n)
char *	browser_file(int n)
void	browser_show_all()
void	browser_hide_private_members()
void	browser_hide_protected_and_private_members()
void	browser_show_protected_and_private_members()
void	browser_hide_inherited_members()
void	browser_show_inherited_members()
void	browser_hide_internal_types()
void	browser_show_internal_types()
void	browser_hide_static_functions()
void	browser_show_static_functions()
void	browser_hide_static_globals()
void	browser_show_static_globals()
void	browser_hide_children_of(char * parent)
void	browser_show_children_of(char * parent)
void	browser_hide_class(char * name)
void	browser_show_class(char * name)
void	browser_hide_global(char * name)
void	browser_show_global(char * name)
void	browser_hide_function(char * name)
void	browser_show_function(char * name)

- **Miscellaneous**

int	cmd_shell (char *str)
char *	shell_to_string (char *str)
void	redisplay ()
char *	filename ()
char *	file_select ()
char *	getchar (char *prompt)
char *	gets (char *prompt)
void	save_file ()
void	set_font (char *fontname)
void	color_area (int start, int end, char *colorname)
void	watch_on ()
void	watch_off ()
int	monochrome()
void	usleep (int us)

Chapter 5

Environment

5.1 File configuration

Here is the configuration file **.xcoralrc** provided in this distribution.

```
/* ----------------------------------------------------------------
   Load standard libraries
   ---------------------------------------------------------------- */
{
/* ----------------------------------------------------------------------
   Load standard libraries
   ---------------------------------------------------------------------- */
{
  /* general SMAC programmer and XCORAL user utilities */
  load_file("utilities.sc");

  /* C and C++ mode, auto indent, reindent, indent region etc */
  load_file("mode.sc" );

  /* provide class method and function profile */
  load_file("head.sc");

  /* extract logical pathname from automount pathname */
  load_file("hack-filename.sc");

  /* command shell utilities, grep make user interface */
  load_file("cmd.sc" );
```

```
/* for us frenchies */
load_file("french.sc");

/* SMAC functions writer and/or user help */
load_file("describe.sc");

/* extensions of C and C++ modes */
load_file("mode-ext.sc");

/* mouse customization */
load_file("mouse.sc");

/* C C++ comments facilities */
load_file("comments.sc");

/* file title and custom organization title */
load_file("title.sc");

/* save file utilities */
load_file("save.sc");

/* global set key utilities */
load_file("keydef-ext.sc");

/* word completion */
load_file("complete-word.sc");

/* window-utilities */
load_file("window-utilities.sc");

/* compare two windows */
load_file("compare-win.sc");

/* latex indent */
load_file("latex.sc");

/* color region buffer */
load_file("color.sc");
```

```
  /* edit directory */
  load_file("edir.sc");

  /* rcs interface */
  load_file("rcs.sc");

  /* Version control */
  load_file("version.sc");

  /* Latex utilities */
  load_file("latex-macros.sc");

  /* Misc utilities */
  load_file("misc-commands.sc");

 /* html utilities */
  load_file("html.sc");

  /* java mode */
  load_file("java.sc");

  /* hanoi demo */
  load_file("hanoi.sc");

  /* shell mode */
  load_file("shell-script.sc");
}

/* ---------------------------------------------------------------------
   define fonts
   --------------------------------------------------------------------- */
{
  set_mode_font("default",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("C-mode",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("C++mode",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Java",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
```

```
  set_mode_font("Latex",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Html",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Ada",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Perl",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Fortran",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("shell",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Shell",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font("Edir",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
  set_mode_font ("french",
  "-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1");
}

/* ----------------------------------------------------------------------
   define colors
   Generics colors for all modes with background set to ivory.
   ---------------------------------------------------------------------- */
{
  gen_comment_color = "limegreen";
  gen_include_color = "goldenrod";
  gen_define_color = "forestgreen";
  gen_keyword_color = "goldenrod";
  gen_string_color = "steelblue";

  gen_varfunc_color = "tomato";
  gen_decl_color = "forestgreen";
  gen_class_color = "tomato";
  gen_del_new_color = "red";
  gen_return_color = "lightsteelblue";
  gen_link_color = "tan";
  gen_makefile_color = "plum";

  /* C,C++ colors */
```

```
cpp_comment_color = gen_comment_color;
cpp_keyword_color = gen_keyword_color;
cpp_define_color = gen_define_color;
cpp_string_color = gen_string_color;

/* Java colors */
java_comment_color = gen_comment_color;
java_category_color = gen_del_new_color;
java_result_color = gen_return_color;
java_class_color = gen_class_color;
java_modifier_color = gen_keyword_color;
java_keyword_color = gen_keyword_color;
java_import_color = gen_define_color;
java_string_color = gen_string_color;
java_package_color = gen_include_color;

/* latex colors */
latex_comment_color = gen_comment_color;
latex_keyword_color = gen_keyword_color;
latex_defun_color = gen_decl_color;
latex_define_color = gen_define_color;
latex_decl_color = gen_decl_color;
latex_label_color = gen_keyword_color;
latex_include_color = gen_include_color;
latex_italic_color = gen_decl_color;
latex_bold_color = gen_varfunc_color;
latex_ref_color = gen_comment_color;

/* html colors */
html_title_color = gen_comment_color;
html_ibtt_color = gen_keyword_color;
html_pre_color = gen_define_color;
html_img_color = gen_decl_color;
html_ref_color = gen_varfunc_color;
html_list_color = gen_define_color;
html_forms_color = gen_include_color;
html_hds_color = gen_keyword_color;
hltml_string_color = gen_string_color;

/* Edir colors */
```

```
  edir_directory_color = gen_comment_color;
  edir_link_color = gen_link_color;
  edir_c_file_color = gen_decl_color;
  edir_tex_file_color = gen_decl_color;
  edir_sc_file_color = gen_decl_color;
  edir_h_file_color = gen_keyword_color;
  edir_makefile_color = gen_string_color;
  edir_readme_color = gen_del_new_color;

  /* Perl colors */
  perl_comment_color = gen_comment_color;
  perl_string_color = gen_string_color;
  perl_label_color = gen_define_color;
  perl_include_color = gen_include_color;
  perl_decl_color = gen_decl_color;
  perl_defun_color = gen_class_color;
  perl_keyword_color = gen_keyword_color;

  /* Fortran90 colors */
  f90_comment_color = gen_comment_color;
  f90_string_color  = gen_string_color;
  f90_unit_color    = gen_class_color;
  f90_include_color = gen_include_color;
  f90_type_color    = gen_decl_color;
  f90_decl_color    = gen_decl_color;
  f90_keyword_color = gen_keyword_color;

  /* Shell colors */
  shell_comment_color = gen_comment_color;
  shell_string_color = gen_string_color;
  shell_include_color = gen_include_color;
  shell_define_color = gen_define_color;
  shell_var_color = gen_varfunc_color;
  shell_keyword_color = gen_keyword_color;
}

/* ---------------------------------------------------------------------
   define indentation variables (mode C, C++ and Java)
   --------------------------------------------------------------------- */
{
```

```
  c_indent_in_comment = 3;
  c_indent_in_string = 0;
  c_indent_in_parenthesis = 1;
  c_indent_in_bracket = 1;
  c_indent_in_brace = 2;
  c_indent_in_statement = 2;
  c_arg_decl_indent = 4;
}

/* ------------------------------------------------------------------
   define suffixes
   ------------------------------------------------------------------ */
{
  set_mode_suffixes("C++mode",".c .cc .cpp .h .hh .sc .xcoralrc");
  set_mode_suffixes("Java",".java .sc .xcoralrc");
  set_mode_suffixes("C-mode",".c .h .sc .xcoralrc");
  set_mode_suffixes("Latex",".tex .latex .sty");
  set_mode_suffixes("Html",".html");
  set_mode_suffixes("Perl",".pl");
  set_mode_suffixes("Fortran",".f .f90");
  set_mode_suffixes("Ada",".a");

  set_mode_suffixes("french",".txt .texte");
}

/* ------------------------------------------------------------------
   define modes on which global_key_def operates
   ------------------------------------------------------------------ */
{
  globalize_mode("default");
  globalize_mode("C-mode");
  globalize_mode("C++mode");
  globalize_mode("Java");
  globalize_mode("Latex");
  globalize_mode("Html");
  globalize_mode("Perl");
  globalize_mode("Ada");
  globalize_mode("Fortran");
  globalize_mode("shell");
  globalize_mode("french");
```

```
  globalize_mode("Shell");
}

/* ----------------------------------------------------------------------
   Util Smac functions.
   ---------------------------------------------------------------------- */
{
  global_key_def("^xt", "transpose_chars");
  global_key_def("^xy", "transpose_forms");
  global_key_def("^[\\", "delete_line_blanks");
  global_key_def("^[ ",  "just_one_blank");
  global_key_def("^[k",  "delete_to_beginning_of_line");
  global_key_def("^xc",  "center_line");
  global_key_def("^[m", "recenter");

  global_key_def("^x#", "sharp_comment");
  global_key_def("^x+", "plus_comment");
  global_key_def("^x=", "equal_comment");
  global_key_def("^x-", "minus_comment");
  global_key_def("^x%", "percent_comment");
  global_key_def("^xz", "update_title_and_save_file");
  global_key_def("^xs", "update_title_backup_and_save_file");
  global_key_def("^[/", "complete_word");
  global_key_def("^xC", "CompareAgain");
  global_key_def("^xg", "go_next");
  global_key_def("^xa", "color_buffer");

  global_key_def("^xe", "edir");
  key_def("Edir", "^xe", "edir");

  key_def("Shell", "^[b", "backward_c_form");
  key_def("Shell", "^[f", "forward_c_form");
  key_def("Shell", "^[d", "delete_next_c_form");
  key_def("Shell", "^[\b", "delete_previous_c_form");
  key_def("Shell", "^[\177", "delete_previous_c_form"); /* esc delete */
  key_def("Shell", "^x^c", "quit_shell");

  key_def("Latex","^[x", "latex_back_indent");
}
```

```
/* ------------------------------------------------------------------------
   French accents in Latex mode.
   ------------------------------------------------------------------------ */
{
    key_def("Latex", "'", "french_accent");
    key_def("Latex", "`", "french_accent");
    key_def("Latex", "^^", "french_accent");
    key_def("Latex", "\"", "french_accent");
}

/* ------------------------------------------------------------------------
   User commands: call by Users commands item in Misc Menu or toolbar
   ------------------------------------------------------------------------ */
user_commands()
{
/****
    char *str;
    int win = current_window();

    clear_list();
    add_list_item("User smac function 1");
    add_list_item("User smac function 2");
    add_list_item("User smac function 3");

    str = select_from_list("User commands");
    redisplay();
    select_window(win);

    if(str==0 || strlen(str) < 2) {
return;
    }

    if (strcmp(str,"User smac function 1")==0) {
user_smac_function_1();
return;
    }
    ...
****/
}
```

```
}
```

The function *user_commands()* allows you to define your own functions that will appear in the item *User commands* of the menu Misc.

5.2 Environment variables

Some environment variables can be used:

- XCORAL_SMACLIB is used to define the *SmacLib PATH* directory.
- XCORALRC is used to define a full pathname of the configuration file *.xcoralrc*.
- XCORAL_VISIT_RAISE is used in browser to raise or not the visit window during a select operation. Default is True.
- XCORAL_PARSE_PATH is used by the browser to parse the specified directories at start-up (the separator must be ':'). These directories will be parsed recursively.
- XCORAL_MODIF_COLOR is a string colorname used to display the 'S' letter when the buffer has been modified. Default color is red.
- XCORAL_SAVE_COLOR is a string colorname used to display the 'S' letter when the buffer has been saved. No default color.
- XCORAL_SHELL is used to define a shell pathname used in a shell window (see shell mode).
- XCORAL_SELECTION is a string colorname used as background color of a selection (default gray).

5.3 Resources

The available resources are fonts (text, menu), foreground, background (text, menu, bottom dialog), displaywarning and geometry. Here is a simple .Xdefaults portion for XCORAL:

```
xcoral*geometry:    =600x500+200+100
xcoral*tbackground: midnightblue /*text window background*/
```

```
xcoral*tforeground: darkseagreen1 /*text window foreground*/
xcoral*font: 9x15bold /* text window font */
xcoral*mbg: lightslategray /*menu background*/
xcoral*cbg: lightslategray /*control panel background*/
xcoral*displaywarning: True
```

5.4 Options

- **=WxH+X+Y** Specifies the geometry in pixels for a text window (Width, Height and top-left position).
- **-browsercontrol =WxH+X+Y** Specifies the geometry for the browser control window.
- **-browservisit =WxH+X+Y** Specifies the geometry for the browser visit window.
- **-display host:0.0** Specifies an X Server connection.
- **-fn fontname** Specifies the text window font.
- **-bg colorname** Specifies text window background color.
- **-fg colorname** Specifies text window foreground color.
- **-mfn fontname** Specifies the menu font.
- **-mbg colorname** Specifies the menu background color.
- **-mfg colorname** Specifies the menu foreground color.
- **-cbg colorname** Specifies the control panel background color.
- **-cfg colorname** Specifies he control panel foreground color.
- **-dw** Displays browser warnings.
- **-mono** Forces xCORAL to be displayed using black and white colors.
- **-help** This causes xCORAL to print out a verbose message describing its options.
- **filename [filename ...]** Specifies the file(s) that have to be loaded during start-up.

5.5 Data and binaries

It is possible to display binary files or data, but the characters that are not drawable are viewed as a reverse question mark (191). This question mark is displayed only when the *Xlib* function *XTextWidth* returns 0. Unfortunately this is implementation dependent and in some cases, a simple blank character is viewed.

The *Ctrl-q* command (quoted char) allows you to insert a non drawable character. For instance to insert a ^L (new page), just type *Ctrl-q Ctrl-l*.

5.6 Colors

On color displays, *xcoral* uses some standard colornames from the RGB.TXT colors database, and also some colors used in the top and bottom shadow. This fails if the current colormap is full. This means that one or several running clients have used all the free color cells of the current colormap. *xcoral* does not handle virtual colormap and in this case the MONO option may be useful.

In *RESOURCE_MANAGER* property (see also *.Xdefaults*), the resource colors for *xcoral* must be defined as colorname (see also *rgb.txt*).

⚠: in some desktops (i.e. *CDE*) a wildcard for the foreground and background are sometimes predefined in *hexa* format. In this case, redefine these resources as a colorname else you will get a message like *Can't alloc named color*.

5.7 Memory resource

xcoral uses intensively the classic malloc/free calls without using huge portions of memory. However, if the size of free memory on your system draws near to zero, the editor and your buffers may remain only as a pleasant dream.

5.8 Xcoral man box

The on-line manual and the PS document are produced from a unique LaTeX source file which is not provided in this distribution.

In each subwindow of the manual box (see figure §5.1 page 61), you can enter one character to move quickly through the index window to the first item beginning with this character.

Figure 5.1: Xcoral man box.

Part II

SmacLib

5.9 SmacLib Overview

Some SMAC files are provided in this distribution and are known as *SmacLib*. These files contain many functions built on the top of the lower interface of the editor (see Built-in editor function §4.9 page 43 and Xcoral interface §7 page 105). These functions are **C** functions interpreted by SMAC. You can read them in the given files suffixed by *.sc*, for instance *cmd.sc* and *head.sc* files. They are good examples of SMAC and XCORAL use. I encourage you to read, use and perhaps adapt (correct ?) them.

cmd.sc	latex.sc
color.sc	latex-macros.sc
comments.sc	misc-commands.sc
compare-win.sc	mode.sc
complete-word.sc	mode-ext.sc
describe.sc	mouse.sc
edir.sc	rcs.sc
edt.sc	save.sc
example.sc	shell-script.sc
french.sc	sun-keydef.sc
hack-filename.sc	title.sc
hanoi.sc	top-ten.sc
head.sc	utilities.sc
html.sc	version.sc
java.sc	window-utilities.sc
keydef-ext.sc	

These files are generally loaded and evaluated during start-up. (see .xcoralrc file §5.1 page 49). See also the default xcoralrc.lf in the SmacLib directorie which can be used as your HOME/.xcoralrc. XCORALRC and XCORAL_SMACLIB environment variables are available at start-up to configure the editor. XCORALRC is used to specify the full pathname of a .xcoralrc file (default is $HOME/.xcoralrc) and XCORAL_SMACLIB the directory path where to find SMAC files (default is the directory specified at compile time: see XC_LIBDIR in Imakefile, typically /usr/local/lib/xcoral).

5.10 cmd.sc

This file contains some useful UNIX commands that you can call within the editor.

- **void cmd(char * cmdline);** executes *(echo ""; cd current-dir; cmdline; echo done)* in a subshell and displays its result in a Shell window. As *cmd* invokes

a sub-process, you can come back to the editor during its execution. Only one window is used for echoing your *cmd* calls. After that, if the *cmd* was *make* or *grep*, you can directly go to each indicated line (compile error line for *make* or matching line for *grep*) with the function *go_next* described bellow.

- **void go_next();** just after a *cmd* call, goes to the first line of its result then loads in a new window the corresponding file and places the cursor to the beginning of the indicated line. If this file is already loaded in a window, works in that window. Next call of this function will parse the next line in the Shell window. This function is bound to *Ctrl-x g* with *C/C++, Shell* and *default* modes or is global to all modes if you use the **.xcoralrc** provided in this distribution.

  You can also use the mouse to select a line in the Shell window.

  In the given file, four functions are defined to locate file pathnames and line numbers:

  – **void find_cmd_trace1();** locates lines starting by a string within double quotes, following by any characters, a number and again any characters. For instance:

  ```
  "myfile.c" syntax error line 13
  ```

  – **void find_cmd_trace2();** locates lines starting by a filename, following by a colon, a number and any characters. For instance:

  ```
  myfile.c:13 SearchMe
  ```

  – **void find_cmd_trace3();** locates lines starting by any characters following by a colon, a string within double quotes, a coma, the word line, a number, another coma and any characters. For instance:

  ```
  cc : "myfile.c", line 3, syntax error
  ```

  – **void find_cmd_trace4();** locates the same line as *find_cmd_trace3* except that the filename is not between double quotes. For instance:

  ```
  cc : myfile.c, line 3, syntax error
  ```

  Add other functions to parse desired patterns.

- **void make();** calls *cmd("make -k");*.
- **void Make(char *args);** calls *cmd("make args");* (see §4.2 page 31).

- **void grep(char * args);** calls *cmd("grep -n <args>");* (see §4.2 page 31).
- **void latex();** runs *latex* on the filename of the current buffer.
- **void xdvi ();** if the current filename suffix is *.tex* or *.latex* then it runs *xdvi* on the corresponding *.dvi* file.

5.11 color.sc

This file contains the colornames of the predefined modes and regular expressions.
The following functions are available from the **Region** menu.

- **void color_buffer();** colors the current buffer according to the current mode.
- **void color_region(int start, int end);** colors a region (i.e. from *start* to *end*) according to current mode.

  ⚠ *color_region* is not robust.

5.12 comments.sc

This file contains some functions to automatically insert C comment patterns.

- **void comment(char line_char, int width);** inserts a two line pattern displaying *width* times the character *line_char* at the current position.

  Example: **comment('x',50);** inserts the following pattern:

```
/* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx */
```

- **void equal_comment();** calls *comment('=', 80);* (default binding "^x=").
- **void minus_comment();** calls *comment('-', 80);* (default binding "^x-").
- **void percent_comment();** calls *comment('%', 80);* (default binding "^x%").
- **void plus_comment();** calls *comment('+', 80);* (default binding "^x+").
- **void sharp_comment();** calls *comment('#', 80);* (default binding "^x#").

5.13 compare-win.sc

This file contains functions that compare the contents of two windows. Position the cursor successively in the two windows to be compared, then execute 'Compare' from one. Execution stops at the first different character, or at end of file. Alter the windows or move cursor in either or both in order to execute 'CompareAgain'

- **void Compare();** starts comparing current window to another one.
- **void CompareAgain();** resumes comparing current window to another one.

5.14 complete-word.sc

This file contains functions that handle the completion of a word in the current buffer.

- **void complete_word();** searches backward for another word starting like the current incomplete word in the current window, memorizing last try and avoiding immediate repeat of same candidates. This function is bound to "^[/". Hit *Esc /* as many times as necessary to make adequate completion appear.

5.15 describe.sc

This file contains useful functions to get information about functions and SMAC objects.

- **void describe(void* f);** displays, in the message box, a SMAC function description.

  Example: **describe(set_mode_font);** displays:

  ```
  set_mode_font (builtin) : 2 params, void(*)(char*,char*)
  ```

- **void describe_all();** displays, in the message box, a description of all current SMAC functions.
- **void display_char(char value);** displays, in the message box, a global variable value of type *char*.
- **void display_int(int value);** displays, in the message box, a global variable value of type *int*.
- **void display_string(char* value);** displays, in the message box, a global variable value of type *char **.

5.16 edir.sc

This file contains functions that handle *edir* mode (see §4.3.7 page 34), which permits directory browsing and file operation.

5.17 edt.sc

This file contains an **EDT** emulation written by **Peter Chang**. This is not a complete emulation of DEC EDT keys. Emphasis has been made on the common keypad and cursor cluster functions. (No attempt at control keys.) No help currently available.

At the moment the problem is that there is no keysym for the Remove key (so I added a patch to handle_key.c). However keysyms for the top and right edges of the keypad are dependent on the keyboard design.

It defines:

edt_forward_word	edt_paste	edt_next_page	edt_copy
edt_previous_page	kp_gold	kp_help	kp_find
kp_page	kp_sect	kp_append	kp_dword
kp_back	kp_cut	kp_dchar	kp_word
kp_char	kp_bol	kp_select	kp_enter
cur_insert	cur_remove	cur_select	cur_prior
edt_cut	kp_dline	kp_forw	kp_eol
cur_find	cur_find	cur_next	

5.18 example.sc

This file contains two examples presented in the previous chapter.

- **HiddenMessage()**
- **SomeBoxes()**

5.19 french.sc

This file defines the *french mode*, which gives the possibility to accentuate characters. Lowercase and uppercase characters can be accentuated, the key sequences are:

a‘ ⇒ à a^ ⇒ â
e’ ⇒ é e‘ ⇒ è e^ ⇒ ê e~ ⇒ ë
i^ ⇒ î i~ ⇒ ï
o^ ⇒ ô
u‘ ⇒ ù u^ ⇒ û u~ ⇒ ü
c~ ⇒ ç c‘ ⇒ ç c^ ⇒ ç

5.20 hack-filename.sc

This file contains functions that restore the logical path from the automount path, either on text in a buffer or in a string argument.

- **void hack_file_name();** looks for the automount directory prefix */tmp_mnt* in the current line then deletes it.
- **char *hack_file_string(char *path);** returns a logical pathname.

Example: **hack_file_string ("/tmp_mnt/users/foo.c");** returns: */users/foo.c*. These two functions must be adapted to the site automount prefixes.

5.21 hanoi.sc

This file contains the SMAC source of the classic Hanoi tower demo.

5.22 head.sc

This file contains functions that automatically insert headers for classes, functions, methods and include files (see C-C++ forms).

- **void cplus_class_header();** prompts in a dialog box to get class name and parent class name then inserts the following form:

```
//
//    Class name: foo
//
//    Description:
//
class foo: public bar {
  public:
    foo();
    foo(const foo &);
    ~foo();

  protected:
  private:
};
```

- **void java_class_header();** prompts in a dialog box to get a java class name and its parent then inserts the following form:

```
//
//    Class name: foo
//
//    Description:
//
public class foo extends bar {

}
```

- **void method_header();** prompts in a dialog box to get a method name and its parent class name, then inserts the following form:

```
//
//      Method name: foo
//
//      Description:
//      Input:
//      Output:
//
bar::foo()
{

}
```

- **void function_header();** prompts in a dialog box to get a function name then inserts the following form:

```
/*
**      Function name: foo
**
**      Description:
**      Input:
**      Output:
*/
foo()
{

}
```

- **void include_header()** checks the current filename and its suffix *.h* then inserts the following form:

```
#ifndef _foo_h
#define _foo_h



#endif /* _foo_h */
```

5.23 html.sc

The *html macros* item in the *Misc* menu presents a scrolled list of html tags. Double click on an item to insert the corresponding tag. Some of these items can be used after having defined a region (Heading, Style, Form).

Example:

If you choose *New HTML document*, a dialog box will ask you to type your document title, then, the following lines will be inserted and colored.

```
<!-- This file has been generated with Xcoral html mode -->
<HTML>
<HEAD>
<! Insert title here ->
<TITLE>My title</TITLE>

</HEAD>

<BODY>
<!-- Insert text here -->
<H4>My title</H4>

</BODY>
</HTML>
```

5.24 java.sc

This file contains key bindings for the Java mode and the filter definition used by the browser.

5.25 keydef-ext.sc

This file contains functions that globalize key bindings, on a "global" mode list.

- **void globalize_mode(char* mode_name);** makes mode *mode_name* global. This means that the mode is added to the global list.
- **void global_key_def(char* key_name, char* command_name);** binds the key *key_name* to the command *command_name* in all modes that are in the global list. Beware that this "global" key definition is synchronous: *globalize_mode* does not bind any keys, and *global_key_def* only calls *key_def* on the modes currently in the list.

5.26 latex.sc

This file contains functions that handle the *Latex* mode indentation (see §4.3.4 page 33).

5.27 latex-macros.sc

Latex mode has been provided with a **latex** macros menu. This menu allows you to automatically insert **latex** macros in your current file.

For example, to insert:

```
\begin{minipage}[t]{5cm}

\end{minipage}
```

double click on the items *Other environments..* and *Minipage* in *Latex style and macros* from the menu *Misc*.

Macros inserted are automatically indented and colored.

Some macros like *Font style* can be used with a predefined region.

5.28 misc-commands.sc

This file contains the function linked to the item *Misc commands* of the menu *Misc*. It proposes four types of useful functions to insert titles, to update these titles, to insert comments and miscellaneous commands.

5.29 mode.sc

This file defines functions and key bindings for **C-mode**, **C++mode** and also **default** mode.

5.29.1 C-C++ mode

In *C-mode* and *C++mode* the key bindings defined by SMAC are:

- *tab* to reindent the current line. Made by the *c_indent_line* function.
- *return* to insert a *newline* and indents the new line according to previous lines. Made by the *c_indent_line* function. It may be necessary to type *tab* after some characters modifying the line context (for instance after *case*).

- '}', ')' and ']' to insert the character, blink the corresponding '{', '(' and '[', and last reindent the current line. Made by the function named:

  c_insert_blink_matched_char_and_indent.

- '{' and ':' to insert the character and reindent the current line. Made by the function *c_indent_line*.

- *Esc a* to go to the beginning of the current or previous definition. The beginning of a definition is supposed to be a line whose first character is not one of `#, }`, *space, tab* and *newline*, or whose first two characters are not `//` or `/*`, and whose previous line last char (before *newline*) is not \. Obviously the beginning of the buffer is also considered to be a definition beginning. Made by the function *goto_beginning_of_c_definition*.

- *Esc f* goes to the end of the current or next expression (skipping comments). Before a `{,(,[` or `"` it goes after the corresponding `},),]` or `"`, else it goes to the end of the following word. A word contains alphabetic characters, digits and underscore. Made by the *forward_c_form* function.

- *Esc b* goes to the beginning of the current or previous expression (skipping comments). Made by the *backward_c_form* function.

- *Esc d* deletes the end of the current expression, or the next expression (skipping comments). Made by the *delete_next_c_form* function.

- *Esc delete* or *Esc backspace* deletes the beginning of the current expression or the previous expression (including comments). Made by the function named:

  delete_previous_c_form.

- *Esc i* indents the current region. If you put a mark at the beginning of the buffer and go to the end before you type *Esc i*, all the buffer will be reindented (which is not immediate when the buffer is large). Made by the *c_indent_region* function.

Indentation is parameterized by the following global variables:

- *int c_indent_in_comment* (default value 3)

- *int c_indent_in_string* (default value 0)

- *int c_indent_in_parenthesis* (default value 1)

- *int c_indent_in_bracket* (default value 1)

- *int c_indent_in_brace* (default value 2)
- *int c_indent_in_statement* (default value 2)
- *int c_arg_decl_indent* (default value 4)

The indentation rules for an empty line, for instance when you type a *return* at the end of a line, are (take a deep breath):

- In a *comment*, indentation is set to the previous (not empty or only containing spaces and tabs) line indentation, except for the second line of comments, whose indentation is the '`/*`' position plus the absolute value of *c_indent_in_comment*. A comment begins with '`/*`' and ends with '`*/`', and a *C++* comment with '`//`' and ends at the end of line.
- In a *string*, if *c_indent_in_string* is less than 0 the indentation is the string beginning position minus *c_indent_in_string* (so plus its absolute value), otherwise indentation equals *c_indent_in_string*.
- In a *parenthesis*, indentation equals '(' position plus absolute value of the variable *c_indent_in_parenthesis*.
- In a *bracket*, indentation equals '[' position plus absolute value of the variable *c_indent_in_bracket*.
- In a *block* (between `{}`), all depends on the previous character (skipping *space, tab, newline and comments*):
  - If it is '{' (so you are at the beginning of the block), indentation is set to the '{' position plus the absolute value of *c_indent_in_brace*.
  - If it is '}', indentation is set to the indentation of the corresponding '{' or of the expression preceding it.
  - If it is '`.`' or '`;`', indentation is set to the previous expression indentation.
  - If it is ')' or ']', indentation is set to beginning expression indentation plus the absolute value of *c_indent_in_statement*.
  - Otherwise the character line indentation plus the absolute value of the variable *c_indent_in_statement*.
- At top level, all depends on the previous character (skipping *space, tab, newline and comments*):
  - If it is '}', indentation is set to 0.

- If it is ')', indentation is set to *c_arg_decl_indent* because it is probably the parameters of a function written with the old syntax.
- If it is `';'`, indentation is set to previous expression indentation.
- Otherwise, in an expression, indentation is set to the absolute value of *c_indent_in_statement*, elsewhere 0.

The indentation rules, when you indent a non empty line (for instance when you reindent a line with a *tab*) depend on the first character(s) (jumping space, *tab, newline* and comments) (cheer up `!`, we're almost there):

- If it is '`#`', indentation is set to 0.
- If it is '`*`' and the next character is '`/`', so you close a comment, indentation is set to the indentation of the '`/*`'
- If it is a '{' out of a block, indentation is set to 0.
- Otherwise in a block:
  - If it is '}', indentation is set to the indentation of the corresponding '{' or of the expression preceding it.
  - If it is '*case*' or '*default*', indentation is set to the corresponding *switch* indentation.
  - If it is `':'`, indentation is set to the indentation of the corresponding `'?'`.

As you can see, it is rather complicated, nevertheless some cases are not taken into account and need braces. For instance it is the case for nested *if* like:

```
if (t1)
  if (t2)
    f1();
  else
    f2();
else
  f3();
```

```
while (t1)
  if (t2)
    f2();
  else
    f3();
```

are poorly indented:

```
if (t1)
  if (t2)
    f1();
else
  f2();
else
  f3();
```

```
while (t1)
  if (t2)
    f2();
else
  f3();
```

but the indentation is correct with braces:

```
if (t1) {
  if (t2)
    f1();
  else
    f2();
}
else
  f3();
```

```
while (t1) {
  if (t2)
    f2();
  else
    f3();
}
```

Of course the quantity of code interpreted by *SMAC* when you type a simple *return* is rather disturbing, but don't be afraid, *SMAC* is a good boy.

5.29.2 default mode

In *default* mode the bindings defined by *SMAC* are the followings:

- *Esc f* goes to the end of the current or next word. A word can contain alphabetic characters, digits and underscore. Made by the *forward_word* function.
- *Esc b* goes to the beginning of the current or previous word. Made by the *backward_word* function.
- *Esc d* deletes the end of the current word or the next word. Made by the *delete_next_word* function.
- *Esc delete* and *Esc backspace* deletes the beginning of the current word or the previous word. Made by the *delete_previous_word* function.

5.30 mode-ext.sc

This file contains some functions to handle C/C++ forms (i.e. symbol, bracketed expression) in the current buffer. These functions call *mode.sc* functions.

- **char* current_form();** returns current form as string.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

- **void transpose_forms();** transposes previous form and current form.

- **char* next_form();** returns next form.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

- **char* next_form_and_delete();** returns next form and deletes it.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

5.31 mouse.sc

This file contains some useful functions bound to mouse buttons with modifier keys.

- **void left_button_control(int pos);** defines *Ctrl-Left* to delete clicked-on char.
- **void right_button_control(int pos);** defines *Ctrl-Right* to delete from clicked-on character to end of line.
- **void left_button_shift(int pos);** defines *Shift-Left* to transpose clicked-on form and preceding form.
- **void middle_button_shift(int pos);** defines *Shit-Middle* to insert clicked-on form at current position.
- **void right_button_shift(int pos);** defines *Shift-Right* to delete clicked-on form.

5.32 rcs.sc

This file contains interface functions for the GNU *Revision Control System*. *RCS* manages multiple revisions of a file and automates the storing, retrieval, logging and identification of revisions. The following functions work on the current file and are reachable by the **Version** menu (see §5.39 page 88. For more information, consult the **RCS** manual.

- **void rcs_initialize();** creates and initializes a new RCS file then deposits the 1.1 revision and checks out unlocked (i.e. read only) this revision. If the current buffer has no name, it prompts in a dialog box to get a filename. Then a message indicates that the revision 1.1 has been initialized.
- **int rcs_check_in();** checks existing, initializing and locking permissions then prompts for a *log message* and deposits a new version (the current buffer is cleared). Return 1 if success else 0.
- **void rcs_check_in_and_out_locked();** does the same as the previous function then checks out locked (i.e. writable) the desired revision (the current buffer is restored).
- **void rcs_check_in_and_out_unlocked();** does the same work as the function *rcs_check_in* then checks out unlocked the desired revision (the current buffer is restored but the file is read-only).
- **void rcs_check_out_locked();** verifies locking permissions and checks out locked the needed revision (default last).
- **void rcs_check_out_unlocked();** verifies locking permissions and checks out unlocked the needed revision (default last).
- **void rcs_diff();** displays the differences between the current revision and another specified revision.
- **void rcs_lock_revision();** prompts for a filename and a revision number then locks it.
- **void rcs_unlock_revision();** prompts for a filename and a nrevision number then unlocks it.
- **void rcs_log();** prints information about revisions of the current **RCS** file.
- **void rcs_repository();** displays the contents of the local RCS directory.

5.33 save.sc

This file contains two functions that update the title of the current file (see §5.36 page 81) before saving it. Filename, path, modification,author and date are updated if they are included in the title.

- **void update_title_and_save_file();** updates the title and saves the current file (default binding "^xz").

- **void update_title_backup_and_save_file();** does the same, but moves old version to *filename~* first (default binding "^xs").

5.34 shell-script.sc

This file contains functions that handle the *Shell-Script* mode indentation (see §4.3.5 page 34).

5.35 sun-keydef.sc

This file contains some **SUN** keyboards bindings.

- key *Stop/L1* is bound to "abort".
- key *Open/L7* is bound to "new_window".
- key *Find/L9* is bound to "forward_search".
- key *Undo/L4* is bound to "undo".
- key *Copy/L6* is bound to "copy_region".
- key *Paste/L8* is bound to "paste_region".
- key *Cut/L10* is bound to "kill_region".

5.36 title.sc

This file contains some functions to automatically insert titles (file headers). The followings fields are included: *File, Path, Description, Created, Author, Modified, Last maintained by, RCS revision, state, Note* and *copyright*.

The three variables *CopyrightFile, custom_organization,* and *custom_copyright* are used to define the path of your complete copyright file, your organization and the abbreviation of your copyright (Example custom_copyright = "(c) Copyright 1994 Xcoral Galactic Company";). Their default values may be modified in the *title.sc* file, or set in the *.xcoralrc* or another *.sc* file.

When *CopyrightFile* is set to zero (default), the introduction of the GNU **General Public License** is inserted by *title(), shell_title()* and *smac_title()* functions.

The fields *Created, Author, Modified, Last maintained by* and *copyright* are automatically filled in at the function call.

The fields *File, Path, Modified and Last maintained by* are updated by the function **update_title** or when the file is saved if *save.sc* is used (see §5.33 page 80).

The field *RCS Revision State* is updated with *SMAC RCS* functions .

- **void title(char* name);** with *foo* as argument, inserts the following title:

```
/* ################################################

                        foo

   File:
   Path:
   Description:
   Created: Sat Jan 14 22:06:51 MET 1995
   Author: Luke Skywalker
   Modified: Sat Jan 14 22:06:53 MET 1995
   Last maintained by: Luke Skywalker

   RCS $Revision$ $State$


   ################################################

   Note:

   ################################################

   Copyright (c): Luke Skywalker

   This program is free software; you can redistribute
   it and/or modify it under the terms of the
   GNU General Public License as published by
   the Free Software Foundation; either version 2,
   or (at your option) any later version.

   ...
   ...

   ################################################ */
```

- **void shell_title(char* name);** inserts the same title as the previous function but in *shell comments* (i.e. with a '#' at the beginning of each line).

- **void custom_title(char* str);** with *foo* as argument, inserts the following title:

```
/* ################################################

                        foo

                XCORAL GALACTIC COMPANY
                   99999 stars city
                     Orion nebula.

   File:
   Path:
   Description:
   Created: Sat Jan 14 22:50:46 MET 1995
   Author: Luke Skywalker
   Modified: Sat Jan 14 22:50:49 MET 1995
   Last maintained by: Luke Skywalker

   RCS $Revision$ $State$


                  (c) Copyright 1994
                Xcoral Galactic Company

   ################################################

   Note:

   ################################################ */
```

- **void custom_shell_title(char* str);** inserts the same title as the previous function but in *shell comments* (i.e. with a '#' at the beginning of each line).

- **void smac_title();** inserts the following title:

```
/* ################################################

                SMAC FILE USED BY XCORAL EDITOR

   File:
   Path:
   Description:
   Created: Sat Jan 14 23:27:53 MET 1995
   Author: Luke Skywalker
   Modified: Sat Jan 14 23:27:55 MET 1995
   Last maintained by: Luke Skywalker

   RCS $Revision$ $State$


   ################################################

   Note:

   Requires:

   Defines:

   Suggested bindings:

   Procedure:

   ################################################

   Copyright (c): Luke Skywalker

   This program ...

   ################################################ */
```

- **void update_title();** updates the fields *File, Path, Modified* and *Last maintained by.*

5.37 top-ten.sc

This file contains a function that displays runtime statistics of SMAC functions (see §6.6.6 page 101).

5.38 utilities.sc

This file contains some general functions frequently used in SMACLIB.

- **char *basename(char *path);** extracts a filename from a full pathname. This function uses the UNIX basename command.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

- **char *dirname(char *path);** extracts a dirname from a full pathname. This function uses the UNIX dirname command.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

- **void insert_chars(int n, char repeated_char);** inserts at the current position *n* times *repeated_char*.

- **void delete_chars(int ndel);** deletes forward *ndel* characters from the current position.

- **void delete_to_beginning_of_line();** deletes characters from the current position to the beginning of the line.

- **void delete_to_end_of_line();** deletes characters from the current position to the end of the line.

- **void delete_region(int start, int end);** deletes characters from *start* position to *end* position.

- **int does_file_exist(char *filename);** returns *True* if the file *filename* exist.

- **int does_unix_command_exist(char *unix_cmd);** returns True if the command *unix_cmd* exists.

- **void print_ascii();** prints the characters from 0 to 255 with the current font.

- **void change_return_to_newline();** replaces \r with \n from the current position to the end of file.
- **char * concat(char *s1, char *s2);** returns a string which is the concatenation of *s1* and *s2*.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.
- **char *substring (char *s, int n, int m);** returns m characters from position n in s (i.e. substring ("chapeau", 2, 3) = "ape").

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.
- **char* file_type(char* filename);** returns the type of the f ile *filename*. This function uses the UNIX *file* command.

  Examples:

  - file_type("smaclib.tex") returns the string *ascii text*
  - file_type("/bin/ls") returns the string *mc68020 pure dynamically linked executable* (on a SUN 3/60).
- **int is_dir(char* filename);** returns True if *filename* is a directory.
- **int is_file(char* filename);** returns True if *filename* is a regular file (i.e. not a directory nor a link, socket or special).
- **void wait_for_file(char *filename);** waits until the file *filename* be present.
- **void wait_for_nofile(char *filename);** waits until the file *filename* be removed.
- **void purge();** closes all text windows which content are already saved.
- **char* window_substring(int start, int end);** extracts a string in the current window from *start* position to *end* position.

  ⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.
- **void center_line();** centers the current line within a width of 80 characters.
- **void center_line_within(int width);** centers the current line within a width of *width* characters.

- **void delete_line_blanks();** deletes blanks and tabs just before current position and behind if current character is a blank or a tab.
- **void just_one_blank();** calls the previous function and inserts a blank.
- **void delete_previous_char();** deletes the previous character preceding the current position.
- **int gets_string_cancelled(char* str);** returns *True* if the string *str* is neither null nor equal to ^g (ASCII code 7). Useful after a call to the *gets functions*.
- **int is_num(char *str);** returns *True* if the string *str* is numeric (12.34 is valid).
- **int read_int_from_string(char* str);** reads an int from the string *str* and returns it.
- **void recenter();** recenters current position vertically in window.
- **void sleep(s);** waits *s* seconds.
- **void transpose_chars();** transpose current and preceding character.

5.39 version.sc

This file contains the upper level interface to a *Version Control System*. The default is **RCS**.

Example:

```
void cv_check_out_locked ()
{
  rcs_check_out_locked();
}
```

You can redefine these functions to set up the interface of your choice (for *SCCS*:

```
void cv_check_out_locked () {sccs_check_out_locked()}).
```

The following functions, which are reachable from **Version** menu call the *SMAC* **RCS** interface (see rcs.sc §5.32 page 79).

- **void cv_initialize ();**
- **void cv_check_in ();**

- **void cv_check_in_and_out_locked ();**
- **void cv_check_in_and_out_unlocked();**
- **void cv_check_out_locked ();**
- **void cv_check_out_unlocked ();**
- **void cv_diff ();**
- **void cv_lock_revision();**
- **void cv_log ();**
- **void cv_repository ();**
- **void cv_unlock_revision();**

5.40 window-utilities.sc

This file contains some window utilities, that are used by *compare-win.sc* (see §5.13 page 68) for example:

- **int choose_window(char* msg);** enables the user to select a window from a list of all windows.
- **int current_position_in(int win);** returns the current position in the window *win*.
- **int end_of_file_in(int win)** returns the end of file position in window *win*.
- **void goto_char_in(int win, int pos);** goes to the position *pos* in the window *win*.
- **char the_char_in(int win, int pos);** returns the character at the position *pos* in the window *win*.

Part III

Smac

Chapter 6

Smac definition

SMAC is a SMall Ansi C interpreter developed to extend XCORAL editor services, but it is also possible to use SMAC outside XCORAL (see §9 page 153).

This chapter gives definitions of SMAC functions independently of XCORAL, valid whether they are used within or without XCORAL. The next chapter will describe their interface with XCORAL.

SMAC implements a subset of the ANSI C standard, but to be fully functional it also implements some extensions of it:

- In traditional C, a program has a single entry point named *main()*. In SMAC, each function is an entry point, thus a *main()* definition is not required, besides SMAC does not know the name *main*.
- You can evaluate a sequence of expressions outside any function definition. These expressions are placed between braces as a function body, and are named *free expressions*. The only restriction is that you cannot put any declaration in a free expression, for instance to define a local variable.

  For example, you can directly write:

```
{ printf("Hello world\n");
  printf("I am SMAC, a SMall Ansi C interpreter\n"); }
```

- There is no programming time and run time (in the C usual sense) in SMAC. As in LISP you define functions and global variables, or execute free expressions when and where you want (providing that all necessary global definitions are made).

- Files are not separately considered, as soon as you declare or define functions and variables, you increase your global environment.

To manually interrupt a SMAC execution, hit *Ctrl-c* in the window where xCORAL or just SMAC is running (*kill -2* on xCORAL pid). To stop current execution by program (as with *exit()* UNIX *system function* but without xCORAL or SMAC ending) see **error()** §6.6.7 page 104.

According to the SMAC compiling flags (see §9 page 153) SMAC can check all your memory access/modification to avoid *core dump* or other things. In this case, all allocated objects are taken from a special memory area whose default size is `2^18` bytes under xCORAL, and `2^15` bytes otherwise. You can update the SMAC_MEMORY_SIZE environment variable to increase it, its value must be a power of two (default `2^18` or `2^15`).

In all cases the SMAC stack size is 1024 words by default. To increase it update the SMAC_STACK_SIZE environment variable; its value corresponds to the number of words it can contain (a word permits to memorize a *char*, an *int* or a pointer).

6.1 Function definition

You must declare functions before use, but of course, when you define a function, you declare it at the same time. You can both use an old style to declare a function arguments list:

```
return_type function_name(arg1, ... argn)
    type1 arg1;
       ...
       ...
    typen argn;
{
     ...
}
```

or the ANSI C syntax:

```
return_type function_name(type1 arg1, ... typen argn)
{
   ...
}
```

You cannot define a function with a variable number of arguments (ellipsis), although some predefined functions use this facility (e.g. **printf()**). Classically, by default, the function return value type and its parameters type are *int*.

You cannot redefine predefined functions, and when you redefine a user function or variable XCORAL displays a warning to avoid surprises. This default protection can be changed with the **debug_mode()** function (see §6.6.4 page 100).

6.2 Global variable definition

Global variable definitions are written in SMAC as in C, but their initialization is performed just after their definition reading (thus before reading the following).

By default global variables and arrays are initialized with 0, but local variables and arrays are not initialized.

I am sorry, but it is not (yet ?) possible to initialize an array within its declaration with values in braces, furthermore you must give every dimension values (it is not possible to use *[ ]*).

6.3 Preprocessor

SMAC has no *preprocessor*, thus you cannot use associated directives. For instance, you cannot use the #*include* directive, but SMAC knows all predefined functions profiles.

If you want to load a declaration file (probably a *.h* file), use **load_file()** (see §6.6.7 page 104).

6.4 Types

SMAC knows the types **void**, **char** and **int** and the derivated types **pointer** (for instance, a pointer to **function**) and **array**. You cannot define any **struct** or **enum**.

SMAC does not fix *int, char* and *pointer* characteristics such as size and alignment. They are the same ones as those used by the compiler to compile XCORAL and SMAC. For instance, if *chars* are signed for this compiler, they will also be signed in SMAC.

6.5 Keywords

The keywords taken into account are: `break`, `case`, `char`, `continue`, `default`, `do`, `else`, `for`, `if`, `int`, `return`, `sizeof`, `switch`, `void` and `while`. The followings are discarded: `auto`, `const`, `double`, `enum`, `extern`, `float`, `goto`, `long`, `register`, `short`, `signed`, `static`, `struct`, `typedef`, `union`, `unsigned` and `volatile`.

Obviously "." and "->" operators are also discarded, but every others are taken into account.

6.6 Predefined functions

Here we describe SMAC predefined functions not linked with XCORAL. Some of these functions could have been defined in SMAC but they are predefined for performance reasons.

6.6.1 Formatted output conversion

Two formatted output conversion functions are defined (see also **wprintf()** §7.3.4 page 112):

printf

```
int printf(char * format, ...)
```

printf places output on *stdout* and returns the number of output characters. The conversion characters taken into account are *%diopuxXcs* with associated flags and fields. Use your preferred *man* for more information.

sprintf

```
char * sprintf(char * str, char * format, ...)
```

sprintf sets output on *str* and returns *str*. The conversion characters taken into account are the same as *printf* ones.

6.6.2 Memory

Three memory allocation and free functions are predefined in SMAC (see also **strdup()** §6.6.3 page 98):

malloc

```
void * malloc(int size)
```

malloc returns a pointer on a block of at least *size* bytes correctly aligned, or 0 on failure (see §6 page 94).

calloc

```
void * calloc(int nelt, int eltsize)
```

calloc returns a pointer to a zero initialized block able to memorize *nelt* elements of size *eltsize*, or 0 on failure.

free

```
void free(void * ptr)
```

free releases a block allocated by *malloc* or *calloc*.
For performance reasons, a copy function has also been predefined:

memcpy

```
char * memcpy(char * to, char * from, int n)
```

memcpy copies *n* bytes from *from* to *to*.

6.6.3 Strings

SMAC defines classical strings functions (read your local *man* for more information):

strcat

```
char * strcat(char * s1, char * s2)
```

strchr

```
char * strchr(char * s, int c)
```

strcmp

```
int strcmp(char * s1, char * s2)
```

strcpy

```
char * strcpy(char * s1, char * s2)
```

strcspn

```
char * strcspn(char * s1, char * s2)
```

strdup

```
char * strdup(char * s)
```

strlen

```
int strlen(char * s)
```

strncat

```
char * strncat(char * s1, char * s2, int n)
```

strncmp

```
int strncmp(char * s1, char * s2, int n)
```

strncpy

```
char * strncpy(char * s1, char * s2, int n)
```

strpbrk

```
char * strpbrk(char * s1, char * s2)
```

strrchr

```
char * strrchr(char * s, int c)
```

strspn

```
int strspn(char * s1, char * s2)
```

strstr

```
char * strstr(char s1, char * s2)
```

strtok

```
char * strtok(char s1, char * s2)
```

index

```
char * index(char * s, int c)
```

rindex

```
char * rindex(char * s, int c)
```

6.6.4 Redefinition

It is possible to redefine functions and global variables (in this case new definitions step over old ones) under certain conditions, essentially for debug reasons, but it is not possible to change function or global variable profiles. Thus, you cannot change a function return value type, nor its parameters number and their types, nor the type of a global variable (dimensions included). Neither is it possible to redefine an active function, or to redefine predefined functions.

remove_function_definition

```
void remove_function_definition(void * pfunc)
```

The first way to redefine a function is to undefine it, and then to give it its new definition. After a *remove_function_definition* call, the function whose name is given as argument remains simply undeclared. However, it is not possible to undefine an active function (during its execution), nor to undefine a predefined function.

Obviously, this way is cumbersome when you want to redefine several functions, for instance when you want to reload a file. In this case you will probably prefer the following function:

debug_mode

```
void debug_mode(int mode)
```

After a *debug_mode* call with a non null argument, you can redefine your functions and global variables. If the argument is greater than 1, redefinitions are signaled by a warning. After *debug_mode* call with zero as argument you cannot make redefinitions anymore.

6.6.5 Function

Because SMAC is an interpreter, a *function* is a real object and it is possible to access to information about it at run time:

function_name

```
char * function_name(void * function)
```

Returns the function (built-in or not) whose name is *function_name* if this one exists, else returns 0. It is used for instance in the *edir.sc SmacLib* files to load *color.sc*, if this is not already done.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

function_arg_count

```
int function_arg_count(function)
```

Returns the function number of arguments. If the function accepts a variable number of arguments, it returns its minimal number, for instance 1 for **printf()**.

function_is_builtin

```
int function_is_builtin(function)
```

Returns 1 if the function is a built-in function, else 0.

function_type

```
char * function_type(function)
```

Returns a new allocated string containing the function type. For instance `function_type(function_type)` returns the following string: `"char*(*)(void*)"`.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

function_list init_function_list

`void init_function_list() void * function_list()`

These two functions are intended to access to all the defined functions (built-in or not). *init_function_list* initializes the iteration, and *function_list* returns each time a different function or 0 when all the functions have been accessed. For instance, the following function is defined in the *describe.sc* file):

```
void describe_functions()
{
  void * f;

  init_function_list();
  while (f = function_list()) {
    char * name = function_name(f);
    char * decl = function_type(f);

    printf("%s : %s%d params, %s\n",
           name, (function_is_builtin(f)) ? "(builtin) " : "",
           function_arg_count(f), decl);

    free(name);
    free(decl);
  }
}
```

6.6.6 Execution profile

Profiling helps you to approximately know the run time ratio taken by your functions. Built-in functions are not taken into account and their run time are included in their caller's one. To achieve this you must call *start_profile()*, execute your program, and last call *stop_profile()*. After that *function_percent()* may return run time percentages.

When profile is running, a timer is used to watch periodically which function is the current function, and to increase its counter. If SMAC is used from xCORAL, this timer is disarmed when you are under the xCORAL toplevel, but it remains active when you call an xCORAL interface function from SMAC.

To see the ten most greedy functions (the *top_ten* function is defined in the *top_ten.sc* file), you can run:

```
void top_ten()
{
  void * f;
  void * ttf[10];
  int ttc[10];
  int i;

  for (i = 0; i != 10; i += 1) { ttf[i] = 0; ttc[i] = 0; }

  init_function_list();
  while (f = function_list())
    for (i = 0; i != 10; i += 1)
      if (function_percent(f) > ttc[i]) {
        int j;

        for (j = 9; j != i; j -= 1) {
          ttf[j] = ttf[j - 1];
          ttc[j] = ttc[j - 1];
        }
        ttc[i] = function_percent(f);
        ttf[i] = f;
        break;
      }

  for (i = 0; (i != 10) && ttf[i]; i += 1) {
    char * name = function_name(ttf[i]);

    printf("%s : %d%%\n", name, ttc[i]);
    free(name);
  }
}
{
  start_profile();
  /* any execution */
  stop_profile();
  top_ten();
}
```

start_profile

```
int start_profile()
```

Resets counters, starts the profiling and returns the number of not built-in functions.

stop_profile

```
int stop_profile()
```

Stops profiling and returns the amount of time the timer has run.

function_percent

```
int function_percent(void *)
```

Takes a function as argument and returns its execution profile associated counter, and 0 if the timer has never run (run time was too small) or if the function is built-in.

6.6.7 Others

Here are some other useful functions.

load_file

```
void load_file(char * filename)
```

Loads the file specified by *filename* (the *sc* suffix used in the given files is not required, SMAC does not use default suffixes), all included definitions and declarations are added in the global environment, free expressions are executed.

error

```
void error(char * msg)
```

Aborts the current execution and prints *msg*. If SMAC was called from xCORAL, control is given back to xCORAL and *msg* is printed in a special window (see also display_message §7.3.13 page 138) and the stack of last called functions is given.

showed_stack_size

```
void showed_stack_size(int height)
```

Sets the number of last called functions given when an error occurred, default value is 10.

Chapter 7

Xcoral interface

The interface between SMAC and XCORAL is composed of a large list of built-in functions. With them you can customize your XCORAL environment by adding your own functions, for instance to define **modes** (like the *C-mode*, see §5.29 page 74), to automatically indent your code files, or other features (see also the *cmd.sc* file description §5.10 page 65).

7.1 Smac access

From XCORAL you have four ways to call SMAC to execute a program:

- The **Eval expression** menu entry or *Ctrl-x Ctrl-e* allows you to feed SMAC free expressions (see §6 page 93) on one line and execute them. In this case, it is not necessary to type the including braces, for instance to run *Makefile* and go to the first erroneous line just enter:

  ```
  make(); go_next();
  ```

  after eval expression menu selection or *Ctrl-x Ctrl-e* key sequence.

- The **Eval region** menu entry or *Esc e* makes SMAC evaluate the free expressions or definitions of the current region, as if the region was placed in a file loaded by the *load_file* function (§6.6.7 page 104).

- The **Load and eval file** menu entry prompts for a file name and loads it as with *load_file*.

- At last, you can bind a function with a **key** or a **sequence of keys**; thus when you hit this keys or sequence of keys the corresponding function is called.

Furthermore at *xcoral* start-up, the **.xcoralrc** file is loaded by *smac* as with *load_file*. In this file, you may load mode definitions, select fonts, ...

7.2 Conventions

The general conventions between *smac* and *xcoral* are the following:

- The **windows** are numbered from 0 to 31. Each window receives a constant number. When you kill a window its number will be reused by a future window creation. See **current_window()** function (see §7.3.9 page 119).

- The current window, is the window pointed at by the mouse, therefore the current edited window. All functions refer to the current window except those for selecting (see **select_window()** §7.3.9 page 118) or killing a window (see **kill_window()** §7.3.9 page 119).

- The first character in a window is numbered 0 (see **current_position()** §7.3.1 page 106, **goto_char()** §7.3.2 page 108 and **the_char()** §7.3.3 page 110). This number is incremented for each following character (including *newline*) up to the end of the buffer.

- The first line in a window is numbered 0 (see **goto_line()** §7.3.2 page 110 and **current_line()** §7.3.1 page 107).

- In most cases there is no window redisplay during *smac* execution. Necessary redisplays are made when *smac* execution ends and *xcoral* recovers control, or when you call the **redisplay** function (see §7.3.8 page 117).

7.3 Functions

7.3.1 About position

current_position

```
int current_position()
```

Returns the current position in the current window. When you are at the beginning of the buffer *current_position* returns 0.

end_of_file

```
int end_of_file();
```

Returns the end of file position, thus the number of characters in the buffer. For an empty file *end_of_file* returns 0.

at_end_of_file

```
int at_end_of_file();
```

Returns 1 if you are at the end of the current buffer, else returns 0. Its definition is equivalent to:

```
int at_end_of_file()
{
  return (current_position() == end_of_file());
}
```

current_line

```
int current_line();
```

Returns the current line number. Lines are terminated by a *newline* character. Here is an example of this function use:

```
{ goto_char(0);
  printf("%d\n", current_line()); }  /* writes 0 */
```

beginning_of_line

```
int beginning_of_line();
```

Returns the position of the beginning of the current line, therefore the current position at the beginning of the buffer or when the previous character is a *newline*.

end_of_line

```
int end_of_line()
```

Returns the position of the end of the current line, therefore the current position if the current character is *newline*, else the position of the next *newline* if it exists, else the end of file position (See also **mark_position()** §7.3.7 page 115).

7.3.2 Change position

goto_char

```
void goto_char(int nth);
```

Changes the current position in the current window to *nth* if it is possible, else does nothing.

```
{ goto_char(0);     /* go to the beginning of buffer */
  goto_char(-1); }  /* do nothing */
```

goto_next_char

```
void goto_next_char();
```

Increments the current position if it is not the end of file position, else does nothing.

```
{ goto_end_of_file();
  goto_next_char(); }  /* do nothing */
```

goto_previous_char

```
void goto_previous_char();
```

Decrements the current position if it is not the beginning of file position, else does nothing.

```
{ goto_char(0);
 goto_previous_char(); }  /* do nothing */
```

goto_end_of_file

```
void goto_end_of_file();
```

Changes the current position to the end of file position. Its definition is equivalent to:

```
void goto_end_of_file()
{
  goto_char(end_of_file());
}
```

goto_beginning_of_line

```
void goto_beginning_of_line();
```

Changes the current position to the beginning of the current line position.

goto_end_of_line

```
void goto_end_of_line();
```

Changes the current position to the end of the current line position.

goto_next_line

```
void goto_next_line();
```

Changes the current position to the same column of the next line when possible. If the current line is the last one, goes to the end of file. If the current position is the end of file position, does nothing. If the length of the next line is shorter than the length of the current line, goes to the end of the next line. The column is the *nth* character of the line (Note that a *tab* counts 1, not 8). Its definition is equivalent to:

```
void goto_next_line()
{
  int col = current_position();

  goto_beginning_of_line();
  col -= current_position();
  goto_end_of_line();
  goto_next_char();
  while (col-- && (current_char() != '\n'))
    goto_next_char();
}
```

goto_previous_line

```
void goto_previous_line();
```

Changes the current position to go to the same column of the previous line when possible. If the current line is the first, goes to the beginning of file. If the current position is the beginning of file position, does nothing. If the length of the previous line is shorter than the length of the current line, goes to the end of the previous line.

goto_line

```
void goto_line(int nth);
```

Changes the current position to go to the beginning of the *nth* line when possible, else (*nth* is too large or negative) does nothing.

See also **forward_search()** (§7.3.5 page 112), **backward_search()** (§7.3.5 page 113), **msearch()** (§7.3.5 page 113) and **read_file()** (§7.3.8 page 115).

7.3.3 Get buffer contents

current_char

```
int current_char();
```

Returns the current character *ascii* code, that means the code of the character at the current position. Returns 0 at end of file or when the buffer is empty.

the_char

```
int the_char(int nth);
```

Returns the *ascii* code of the *nth* character of the buffer, 0 for illegal position or at end of file. *the_char* definition is equivalent to:

```
int the_char(int nth)
{
  if ((nth >= 0) && (nth <= end_of_file())) {
    int here = current_position();
    int result;

    goto_char(nth);
    result = current_char();
    goto_char(here);
    return result;
  }
  return 0;
}
```

next_char

```
int next_char();
```

Returns the next character *ascii* code, 0 if the current position or the next one is the end of file, or if the buffer is empty. *next_char* definition is equivalent to:

```
int next_char()
{
  return(the_char(current_position() + 1));
}
```

previous_char

```
int previous_char();
```

Returns the previous character *ascii* code, or 0 at the beginning of the file or if the buffer is empty.

7.3.4 Change buffer contents

delete_char

```
void delete_char();
```

Removes the current character, does nothing at the end of file or if the buffer is empty. Obviously the buffer number of lines is updated if the current character was *newline*.

replace_char

```
void replace_char(int newchar);
```

Replaces the current character by *newchar*. At the end of file or if the file is empty the new character is inserted at the current position and the buffer size is incremented. Obviously the number of lines of the buffer is updated if one of the current or the new character is *newline*. Its definition is equivalent to:

```
void replace_char(int newchar)
{
  delete_char();
  insert_char(newchar);
  goto_previous_char();
}
```

insert_char

```
void insert_char(int newchar);
```

Inserts *newchar* before the current character and goes to the next position, therefore the current character remains the same. Obviously the number of lines of the buffer is updated if the inserted character is *newline*.

insert_string

```
void insert_string(char * str);
```

Inserts all the *str* chars, except the final 0, as each character being inserted by *insert_char*:

```
void insert_string(char * str)
{
  while(*str)
    insert_char(*str++);
}
```

wprintf

```
int wprintf(char * format, ...);
```

wprintf is equivalent to *printf* except that the output characters are inserted in the current window starting at the current position. See also *display_message* (§7.3.13 page 138).

See also the following functions: **global_replace()**, (§7.3.5 page 114), **insert_file()** (§7.3.8 page 116), **read_file()** (§7.3.8 page 115) and **kill_current_buffer()** (§7.3.8 page 116).

7.3.5 Search and substitution

forward_search

```
int forward_search(char * str);
```

Searches for *str* through the next characters, returns 1 and goes to the beginning of the string on success, else returns 0.

backward_search

```
int backward_search(char * str);
```

Searches for *str* through the previous characters, returns 1 and goes to the beginning of the string on success, else returns 0.

msearch

```
int msearch(char * string, int end, int direction);
```

Searches for a character among *str* (except the last 0) in the current buffer, forward if *direction* is positive or *null* and backward if *direction* is negative. Search stops at the *end* position. Returns 1 and goes to the corresponding position on success, otherwise returns 0 and goes to *end* position. Here is its definition:

```
int msearch(char * str, int end, int direction)
{
  if (direction >= 0) {
    if (end > end_of_file())
      end = end_of_file();
    while (current_position() < end)
      if (strchr(str, current_char()))
        return 1;
      else
        goto_next_char();
  }
  else {
    if (end < 0)
      end = 0;
    while (current_position() > end) {
      goto_previous_char();
      if (strchr(str, current_char()))
        return 1;
    }
  }
  return 0;
}
```

global_replace

```
int global_replace(char * str1, char * str2)
```

Searches for and replaces *str1* by *str2* in the next characters and returns the number of substitutions. The current position is not modified.

7.3.6 Regular expressions

re_forward_search

```
int re_forward_search(char * regexp)
```

Returns the position of the next regular expression *regexp* or -1 if *regexp* has not been found or -2 if *regexp* is not valid.

re_backward_search

```
int re_backward_search(char * regexp)
```

Returns the position of the previous regular expression *regexp* or -1 if *regexp* has not been found or -2 if *regexp* is not valid.

re_match_beginning

```
int re_match_beginning(int n)
```

Returns the beginning position of the substring *n* of the regexp found by the previous search call to a regexp else -1 if regexp is not valid. The first substring of a regexp is numbered 0. (see GNU regexp manual).

re_match_end

```
int re_match_end(int n)
```

Returns the end position of the substring *n* of the regexp found by the previous search call to a regexp, else -1 if regexp is not valid (see GNU regexp manual).

re_replace

```
int re_replace(char *newstring)
int re_replace(char *newstring, char * regexp)
```

Replaces the regular expression *regexp* with the string *newstring*. If the argument *regexp* is omited, the previous search call to a regexp is used. It returns 1 on success, else 0 (see GNU regexp manual).

7.3.7 Mark

mark_position

```
int mark_position()
```

Returns the position of the mark, else -1 if has been set no mark.

set_mark

```
void set_mark(int pos)
```

Sets the current buffer mark to pos.

[Note]: for a region defined by *mark_position()* and the current position, the current position may be the end or the beginning of the region. Anyway the mark must be defined first.

reset_mark

```
void reset_mark()
```

Removes the mark of the current buffer.

goto_mark

```
void goto_mark()
```

Changes the current position to the mark position.

7.3.8 Buffers and files

Some editing operations called by menus can be executed from SMAC. But in this case the information asked for by XCORAL menus have to be given as arguments of the corresponding functions (questions are not performed).

read_file

```
int read_file(char * filename)
```

Changes the current buffer to the file associated to *filename* like the **Read File** menu entry, except that if the current buffer is not saved, its contents is lost (see *current_buffer_is_modified()* §7.3.8 page 116). Returns 0 if there is no file matching *filename* and `-1` if the file is already loaded in an other buffer of the current session, leaving the buffer unchanged. Otherwise returns 1 and loads filename in the current buffer. The current position becomes the beginning of buffer.

save_file

```
void save_file()
```

Saves the buffer into the corresponding file in the same way as the **Save File** menu entry does.

write_file

```
void write_file(char * newfilename)
```

Saves the buffer into the file corresponding to *newfilename* like the **Write File** menu entry does.

insert_file

```
int insert_file(char * filename)
```

Inserts the contents of the file associated to *filename* at the current position. Current position stays unchanged (it is the beginning of the inserted file). Returns 1 on success, otherwise 0 (if *SMAC* cannot read *filename*).

kill_current_buffer

```
void kill_current_buffer()
```

Empties the current buffer. If it has not already been saved, its contents is lost.

current_buffer_is_modified

```
int current_buffer_is_modified()
```

Returns 1 if the current buffer is modified, else 0. Here is an example of this function use:

```
void save_buffers()
{
  int win;

  for (win = 0; win != 32; win += 1)
    if ((select_window(win) != -1) &&
        current_buffer_is_modified())
      save_file();
}
```

line_count

`int line_count()`

Returns the current buffer number of lines, a non terminated line is counted, therefore an empty buffer has 1 line. Its definition is equivalent to:

```
int line_count()
{
  int result = 0;
  int here = current_position();

  goto_beginning_of_file();
  do {
    goto_end_of_line();
    goto_next_char();
    result += 1;
  } until (at_end_of_file());
  goto_char(here);
  return result;
}
```

redisplay

`void redisplay()`

Updates the contents of windows needing redisplay. Of course this update is immediate, and not delayed until current SMAC execution ends.

filename

`char * filename()`

Returns a new allocated string containing the absolute *filename* corresponding to the current buffer, or 0 when the buffer is untitled.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

current_line_to_top

```
void current_line_to_top()
```

Scrolls to position the current line as the window first line. Note that the window will not be updated until the current *SMAC* execution ends.

⚠ Be careful, if you change the current position after this function call, the window appearance could be changed and the line designated might not become the first of the window.

set_font

```
void set_font(char * fontname)
```

Changes the current buffer font to *fontname*.

See also **current_window()** and **select_window()** below.

7.3.9 Windows

select_window

```
int select_window(int win)
```

Changes the current window to *win* and returns *win* if the corresponding window exists. Else returns -1 and the current window is unchanged. Here is an example of this function use:

```
void print_windows_name()
{
  int my = current_window();
  int win;

  for (win = 0; win != 32; win += 1)
    if (select_window(win) != -1) {
      char * name = filename();

      printf("window %d : %s\n", win, name);
      free(name);
    }
  select_window(my);
}
```

current_window

```
int current_window()
```

Returns the current window number.

new_window

```
int new_window()
```

On success, creates a new untitled window and returns its number, else returns -1.

kill_window

```
void kill_window(int win)
```

Kills the window *win*. If *win* is the last window, quits XCORAL. Else if *win* is the current window another window becomes the current window. Following is an example of this function use:

```
void the_end()
{
  int win;

  for (win = 0; ; win += 1)
    kill_window(win);
}
```

lower_window

```
void lower_window()
```

Puts the current window under all the others.

raise_window

```
void raise_window()
```

Puts the current window above all the others.

window_height

```
int window_height()
```

Returns the current window height (number of lines in current page).

window_width

```
int window_width(int character)
```

If *character* width is not null for the current window font, returns the number of times *character* can be written on the current window width; for a fixed font this value is the number of columns. Else, if *character* is not visible (it is the case for the null character in all fonts), returns the opposite of current window width in pixels.

See also **redisplay()** (§7.3.8 page 117), **filename()** (§7.3.8 page 117) and **set_font()** (§7.3.8 page 118).

7.3.10 Colors

color_area

```
void color_area(int start, int end, char *colorname)
```

Colors text region from position *start* to position *end* with color *colorname* (see the standard X11 file rgb.txt). This function can be used with the *SMAC* regexp interface to color strings of your own modes (see *color.sc* §5.11 page 67).

remove_colors

```
void remove_colors()
```

Removes all colors in the current buffer.

monochrome

```
int monochrome()
```

Returns 1 if your display is monochrome, else 0.

7.3.11 Modes

Modes permits to locally configure key bindings and default font. See also **C-mode** §5.29 page 74 and **french mode** §5.19 page 69.

current_mode

```
char * current_mode()
```

Returns the name of the current window mode in a new allocated string.

 Be careful, the string is allocated each time the function is called, use **free()** to release it.

create_mode

```
void create_mode(char * name)
void create_mode(char * name, char * suffixes)
void create_mode(char * name, char * suffixes, char * fontname)
```

Creates a new mode named *name* if it does not already exist. This mode will be created with default mode bindings. If *suffixes* is valuated, the next file having one of these suffixes will be loaded in *name* mode. This also goes for *fontname*.

suffixes is one or a sequence of words beginning by a dot, and separated by a space. For instance the *C_mode* associated suffixes are defined with `".h .c .C .H .sc"`.

```
create_mode("secret-mode", ".dontreadme", "5x8");
```

set_mode_suffixes

```
void set_mode_suffixes(char * modename, char * suffixes)
```

Changes the suffixes associated to the mode *modename*.

set_mode_font

```
void set_mode_font(char * modename, char * fontname)
```

Changes the font associated to the mode *modename*.

set_mode

```
void set_mode(char * modename)
```

Changes the current window mode to *modename*.

key_def

```
void key_def(char * modename, char * keys, char * funcname)
```

Changes or creates the given mode key binding *keys* if *funcname* is not 0, else removes key binding *keys*. The specified key binding can be (`?` is any character except `^`):

- `"?"` for instance `}` blinks the current `{` in *C_mode*,
- `"^?"` to bind a control key (except ^x), for instance `"^i"` re-indents the current line in *C_mode*(of course, in this case you can also give `"\t"`),
- `"^[?"` to bind an *escape key* sequence,
- `"^x?"` to bind an *ctrl−x key* sequence
- `"^x^?"` to bind an *ctrl−x ctrl−key* sequence.
- `"^[^?"` to bind an *escape ctrl−key* sequence.

Furthermore:

- `‘^^’` appoints the `‘^’` character
- `‘^’` and `‘^@’` appoint the null character (code 0).

It is also possible to bind *keysym* keys (`‘k’` is the keysym name):

- `"k"` to bind a *keysym*
- `"^[k"` to bind an *escape keysym*
- `"^xk"` to bind an *ctrl−x keysym*

where `'k'` is one of (case is not significant) *Multi_key*, *Home*, *Left*, *Up*, *Right*, *Down*, *Prior*, *Next*, *End*, *Begin*, *Select*, *Print*, *Execute*, *Insert*, *Undo*, *Redo*, *Menu*, *Find*, *Cancel*, *Help*, *Break*, *Mode_switch*, *script_switch*, *Num_Lock*, *KP_Space*, *KP_Tab*, *KP_Enter*, *KP_F1*, *KP_F2*, *KP_F3*, *KP_F4*, *KP_Equal*, *KP_Multiply*, *KP_Separator*, *KP_Add*, *KP_Subtract*, *KP_Decimal*, *KP_Divide*, *KP_0*, *KP_1*, *KP_2*, *KP_3*, *KP_4*, *KP_5*, *KP_6*, *KP_7*, *KP_8*, *KP_9*, *F1*, *F10*, *F11*, *F12*, *F13*, *F14*, *F15*, *F16*, *F17*, *F18*, *F19*, *F2*, *F20*, *F21*, *F22*, *F23*, *F24*, *F25*, *F26*, *F27*, *F28*, *F29*, *F3*, *F30*, *F31*, *F32*, *F33*, *F34*, *F35*, *F4*, *F5*, *F6*, *F7*, *F8*, *F9*, *L1*, *L10*, *L2*, *L3*, *L4*, *L5*, *L6*, *L7*, *L8*, *L9*, *R1*, *R10*, *R11*, *R12*, *R13*, *R14*, *R15*, *R2*, *R3*, *R4*, *R5*, *R6*, *R7*, *R8*, *R9*. The *KP_* names refer to the keypad numbers and functions. Sometimes a key can have several names.

funcname is the name of the function you want to be called when you hit the *keys* sequence in the mode *modename*. The name appoints a SMAC function (built-in or not, taking no arguments) or must be one of the following: *set_mark*, *abort*, *return*, *delete*, *tab*, *kill_line, cursor_down*, *cursor_up*, *open_space*, *undo*, *redo*, *next_page*, *kill_region*, *paste_region*, *eval_region*, *first_page*, *last_page*, *previous_page*, *copy_region*, *delete_window*, *goto_mark*, *backward_search*, *forward_search*, *global_replace*, *write_file*, *kill_current_buffer*, *query_replace*, *save_file*, *read_file*, *insert_file*, *exchange_cursor_mark*, *list_open_file*, *goto_line*, and *eval_expression*.

These functions are the functions called by the default key bindings, contrarily to SMAC functions they take no argument but asks for them if necessary. For instance the *goto_line()* SMAC function take one argument, but a key binding with *"goto_line"* prompts the line number in the mini-buffer. The *kill_current_buffer()* SMAC function empties a buffer without confirmation if the buffer is modified, but a key binding with *"kill_current_buffer"* asks if the buffer must be saved or not.

7.3.12 Browser interface

Here are the functions to customize browser and access browser database. The examples can be placed in a SmacLib file.

Cpp functions:

browser_set_pp

```
void browser_set_pp(char * mode, char * exec)
```

To indicate where the C and C++ pre-processor are (default pre-processor is */lib/cpp* for each mode). *mode* must be `"C-mode` or `"C++mode"`, and *exec* must be an absolute filename.

browser_set_pp_options

```
void browser_set_pp_options(char * mode, char *options)
```

mode must be `"C-mode` or `"C++mode"`. If *options* is 0, removes pre-processing before parsing, else indicates the pre-processor options and asks for a pre-processing before parsing.

File management:

browser_add

```
void browser_add(char * path [, int rec])
```

parses the given file or directory, momentary forces the browser toggle button to *Rec* if the second argument is given not equal to 0.

browser_del

```
void browser_del([char * name])
```

deletes the file *name* (or all files if the function is called without argument) from the browser database.

browser_dump

```
int browser_dump(char * filename)
```

Saves the current browser database in the specified file, Returns 1 if the dump can be made, else 0 (file is write protected or other error).

browser_restore

```
int browser_restore(char * filename)
```

Restores (quickly) the browser database information dumped in the specified file. Returns 1 on success, else 0 (file doesn't not exist or other error).

 An implicit *Del* is made before updating the browser database, for all the files referenced in the dump.

Access functions:

browser_class_file

`char * browser_class_file(char * type_name [, int * pline])`

Returns the file pathname where *type_name* is defined, if *pline* is given, it is modified to the line number. Returns 0 if *type_name* is not in the browser database or if SMAC cannot allocate memory to memorize the result.

Be careful, the string is allocated each time the function is called, use *free()* to release it.

browser_class_parents

`char ** browser_class_parents(char * class_name)`

Returns a vector containing the *class_name* parents names and the inheritance specifier, or 0 if *class_name* is not in the browser database or if SMAC cannot allocate memory to memorize the result.


Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_class_parent(char * name)
{
  char ** p, ** v = browser_class_parents(name);

  if (! v) return;

  printf("%s parents are :\n", name);
  for (p = v; *p; p += 2) {
    printf("\t%s : ", *p);
    switch (*((int *)(p + 1))) {
    case 1 : printf("private\n"); break;
    case 2 : printf("protected\n"); break;
    case 4 : printf("public\n"); break;
    }
  }
  free(v);
}
```

browser_class_children

`char ** browser_class_children(char * class_name)`

Returns a vector containing the *class_name* children names, or 0 if *class_name* is not in the browser database or if SMAC cannot allocate memory to memorize the result.


Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_class_children(char * name)
{
  char ** p, ** v = browser_class_children(name);

  if (! v) return;

  printf("%s childrens are : ", name);
  for (p = v; *p; p += 1)
    printf("%s ", *p);
  printf("\n");
  free(v);
}
```

browser_class_methods

`char ** browser_class_methods(char * class_name)`

Returns a vector containing the *class_name* methods declaration with its flags, their declaration and definition files, their lines, and information bits. Returns 0 if *class_name* is not in the browser database or if SMAC cannot allocate memory to memorize the result.


Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_class_methods(char * name)
{
  char ** p, ** v = browser_class_methods(name);

  if (! v) return;

  printf("%s is declared in %s, its methods are :\n", name, *v);

  for (p = v+1; *p; p += 5) {
    int bits = *((int *) (p + 2));

    printf("\t%s\n\t\tdeclared line %u",
           p[0] + 8,  /* remove flags */
           *((int *) (p + 1)));
    if (p[3])
      printf(", defined in %s line %u", p[3],
             *((int *) (p + 4)));
    printf("\n\t\t");
    if (bits & 1) printf("private ");
    else if (bits & 2) printf("protected ");
    else if (bits & 4) printf("public ");
    if (bits & 8) printf("inline ");
    if (bits & 16) printf("constructor ");
    else if (bits & 32) printf("destructor ");
    else if (bits & 64) printf("operator ");
    if (bits & 128) printf("const ");
    if (bits & 256) printf("virtual ");
    else if (bits & 512) printf("pure virtual ");
    if (bits & 1024) printf("friend");
    printf("\n");
    if (p[0][6] == 'H')
      printf("\t\thiden\n");
  }
  free(v);
}
```

browser_class_attributes

```
char ** browser_class_attributes(char * class_name)
```

Returns a vector containing the *class_name* attributes declaration with its flags, their declaration file, their line, and information bits. Returns 0 if *class_name* is not in the browser database or if *SMAC* cannot allocate memory to memorize the result.

 Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_class_attributes(char * name)
{
  char ** p, ** v = browser_class_attributes(name);

  if (! v) return;

  printf("%s is declared in %s, its attributes are :\n",
         name, *v);

  for (p = v+1; *p; p += 3) {
    int bits = *((int *) (p + 2));

    printf("\t%s\n\t\tdeclared line %u",
           p[0] + 6,  /* remove flags */
           *((int *) (p + 1)));
    printf("\n\t\t");
    if (bits & 1) printf("private ");
    else if (bits & 2) printf("protected ");
    else if (bits & 4) printf("public ");
    if (bits & 8) printf("static ");
    printf("\n");
    if (p[0][4] == 'H')
      printf("\t\thiden\n");
  }
  free(v);
}
```

browser_class_flags

```
char * browser_class_flags(char * class_name)
```

Returns the class flags and name, or 0 if *class_name* is not in the browser database or if *SMAC* cannot allocate memory to memorize the result.

⚠ Be careful, the string is allocated each time the function is called, use *free()* to release it.

```
void print_class_info(char * name)
{
  char ** v = browser_class_flags(name);

  if (! v) return;

  printf("class %s is know by browser %s\n", name,
         (v[4] == 'H') ? "but hidden" : "and visible");
  free(v);
}
```

browser_functions

```
char ** browser_functions(char * prefix)
```

Returns a vector containing the list of functions (with their flags, definition file and line) which name and eventually parameter type list begins with the given prefix.

Returns 0 if SMAC cannot allocate memory to memorize the result.

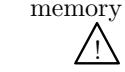
⚠ Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_functions(char * pfix)
{
  char ** p, ** v = browser_functions(pfix);

  if (! v) /* not enough memory */ return;

  printf("matched functions are :\n");

  for (p = v; *p; p += 3)
    printf("%s defined in %s line %d (%s)\n",
           p[0] + 7, p[1],
           *((int *) (p + 2)),
           (p[0][5] == 'H') ? "hidden" : "visible");

  free(v);
}
```

browser_globals

```
char ** browser_globals(char * prefix)
```

Returns a vector containing the list of globals (with their flags and definition file and line) which name begins with the given prefix. Returns 0 if *SMAC* cannot allocate memory to memorize the result.


Be careful, the result is allocated each time the function is called, use *free()* to release it. The vector and its included strings are placed in a unique allocated block.

```
void print_globals(char * pfix)
{
  char ** p, ** v = browser_globals(pfix);

  if (! v) /* not enough memory */ return;

  printf("matched globals are :\n");

  for (p = v; *p; p += 3)
    printf("%s defined in %s line %d (%s)\n",
           p[0] + 5, p[1],
           *((int *) (p + 2)),
           (p[0][3] == 'H') ? "hidden" : "visible");

  free(v);
}
```

browser_select_class

```
int browser_select_class(char * prefix)
```

Selects the first type in the browser subwindow *Types* which name matches *prefix* as you can do it with the mouse. Returns 1 if a type name is found, else 0 (an hidden class cannot be selected).

browser_select_method

```
int browser_select_method(char * prefix)
```

Selects the first method in the browser subwindow *Methods* which name (and optionally parameter type list) matches *prefix* as you can do it with the mouse. Returns 1 if a method is found, else 0 (an hidden method cannot be selected).

browser_select_attribute

```
int browser_select_attribute(char * prefix)
```

Selects the first attribute in the browser subwindow *Attribute* which name matches *prefix* as you can do it with the mouse. Returns 1 if an attribute is found, else 0 (an hidden attribute cannot be selected).

browser_select_function

```
int browser_select_function(char * prefix)
```

Selects the first function in the browser subwindow *Functions* which name (and optionally parameter type list) matches *prefix* as you can do it with the mouse. Returns 1 if a function is found, else 0 (an hidden function cannot be selected).

browser_select_global

```
int browser_select_global(char * prefix)
```

Selects the first global in the browser subwindow *Globals* which name matches *prefix* as you can do it with the mouse. Returns 1 if a global is found, else 0 (an hidden global cannot be selected).

browser_edit

```
void browser_edit()
```

Edits the last selected item, as a double click on the browser window.

```
void edit_object(char * pfix)
{
  if (browser_select_class(pfix) ||
      browser_select_function(pfix) ||
      browser_select_global(pfix))
    browser_edit();
}
```

browser_class_entry

```
int browser_class_entry(char * prefix)
```

Returns the rank of the first visible or hidden type in the browser *Types* list (alphabetically sorted) which name matches *prefix*, or −1 if a type cannot be found. See *browser_class()* described below.

browser_function_entry

```
int browser_function_entry(char * prefix)
```

Returns the rank of the first visible or hidden function in the browser *Functions* list (alphabetically sorted) which name (and optionally parameter type list) matches *prefix*, or −1 if a function cannot be found. See *browser_function()* described below.

browser_global_entry

```
int browser_global_entry(char * prefix)
```

Returns the rank of the first visible or hidden global in the browser *Globals* list (alphabetically sorted) which name matches *prefix*, or −1 if a global cannot be found. See *browser_global()* described below.

browser_file_entry

```
int browser_file_entry(char * prefix)
```

Returns the rank of the first file in the browser *Files* list (alphabetically sorted) with complete pathname matching *prefix*, or −1. See *browser_global()* described below.

browser_class

```
char * browser_class(int n)
```

Returns the name (without flags) of the nth type from the browser *Types* list (alphabetically sorted), or 0 for invalid index.

```
int count_types()
{
  int i = 0;

  while (browser_class(i))
    i += 1;

  return i;
}
```

browser_function

```
char * browser_function(int n)
```

Returns the name (without flags) and parameter type list of the nth function from the browser *Function* list (alphabetically sorted), or 0 for invalid index.

browser_global

```
char * browser_global(int n)
```

Returns the name (without flags) of the nth global from the browser *Global* list (alphabetically sorted), or 0 for invalid index.

browser_file

```
char * browser_file(int n)
```

Returns the pathname of the nth file from the browser *File* list (alphabetically sorted), or 0 for invalid index.

Visibility:

By default, all browser information are displayed on its window, but you can hide some of them with the following functions (or browser window *Hide* buttons §3.2 page 24).

browser_show_all

```
void browser_show_all()
```

All browser information become visible.

browser_hide_private_members

```
void browser_hide_private_members()
```

Hides all private class members (methods and attributes).

browser_hide_protected_and_private_members

```
void browser_hide_protected_and_private_members()
```

Hides all protected and private class members (methods and attributes). Only public members still visible.

browser_show_protected_and_private_members

```
void browser_show_protected_and_private_members()
```

All class members (methods and attributes) become visible.

browser_hide_inherited_members

```
void browser_hide_inherited_members()
```

Hides all inherited class members.

browser_show_inherited_members

```
void browser_show_inherited_members()
```

All class inherited members become visible

browser_hide_internal_types

```
void browser_hide_internal_types()
```

Hides types defined inside another type. For instance if the browser knows the following definitions:

```
struct S {
  struct SS {
    int i;
  } ss;
};
```

the types displayed on the browser window are *S* and `S::SS`, this last will be hidden after calling *browser_hide_internal_types*.

browser_show_internal_types

```
void browser_show_internal_types()
```

Types defined inside one type become visible.

browser_hide_static_functions

```
void browser_hide_static_functions()
```

Hides *functions* declared *static*.

browser_show_static_functions

```
void browser_show_static_functions()
```

Shows all *functions*

browser_hide_static_globals

```
void browser_hide_static_globals()
```

Hides *globals* declared *static*.

browser_show_static_globals

```
void browser_show_static_globals()
```

Shows all *globals*.

browser_hide_children_of

```
void browser_hide_children_of(char * parent)
```

Hides sub classes of the specified class. Re-browsing the specified class definition cancels this hidding.

browser_show_children_of

```
void browser_show_children_of(char * parent)
```

Show sub classes of the specified class (except if another hidding cause exists).

browser_hide_class

```
void browser_hide_class(char * name)
```

Hides the specified class, but the visibility of its sub classes is not affected. Re-browsing the specified class definition cancels this hidding.

browser_show_class

```
void browser_show_class(char * name)
```

The specified class becomes visible (except if another hidding cause exists).

browser_hide_global

```
void browser_hide_global(char * name)
```

Hides the specified global. Re-browsing the specified global definition cancels this hidding.

browser_show_global

```
void browser_show_global(char * name)
```

The specified global becomes visible (except if another hidding cause exists).

browser_hide_function

```
void browser_hide_function(char * name)
```

Hides the specified function (*name* may contain parameters type list). Re-browsing the specified function definition cancels this hidding.

browser_show_function

```
void browser_show_function(char * name)
```

The specified function becomes visible (except if another hidding cause exists).

7.3.13 Others

last_key

```
int last_key()
```

Returns the last character of the last keys sequence which called *SMAC*. For instance, only one function is defined to blink `{,(` and `[`, it calls *last_key* to know if you have typed a `}`, a `)`, or a `]`.

blink

```
void blink(int nth)
```

If *nth* is negative writes *No Match* in the mini buffer, else highlights the *nth* character if it is visible, else writes *Match '<the char>' at line <the line> pos <nth>*. This function is used by the *C−mode* (§5.29 page 74) when you press `},)` and `]` keys to indicate corresponding `{,(` and `[` position. Obviously its visibility effect is obtained when *blink* is called, not when the current *SMAC* execution is finished.

cmd_shell

```
void cmd_shell(char * cmd)
```

Executes *sh -c cmd* in a sub-shell and inserts the result in a Shell window. All the characters produced by the command *cmd* in the *stdout* are inserted starting at the current position. To get the characters produced in *stdout* and *stderr*:

```
cmd_shell("(the_command)2>&1");
```

cmd_shell_to_string

```
char * cmd_shell_to_string(char * cmd)
```

Executes *sh -c cmd* in a sub-shell and returns a new allocated string containing all the characters produced by the command *cmd* in the *stdout* or *stderr*.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

file_select

```
char * file_select()
```

Returns a new allocated string containing the absolute *filename* selected by a file selector, or 0 if you cancel the selection.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

select_from_list

```
char * select_from_list(char * title)
```

Displays the *Xcoral List Box* with given items (look at the next two functions) and the given *title*. Returns the item chosen with a mouse click, or 0 if cancelled.

add_list_item

```
void add_list_item(char * item)
```

Adds *item* to the items list. There is no check to verify that items are different.

clear_list

```
void clear_list()
```

Empties the items list.

display_message

```
void display_message(char * msg)
void display_message(char * msg, char * title)
void display_message(char * msg, char * title, int flag)
```

Writes *msg* in the *Xcoral Messages Box*. If *title* is given, it is written at the top of the window, otherwise previous title is cleared. If *flag* is not null, a separator is inserted before *msg* (i.e. the string '>>>' and a return are inserted , thus the message will be displayed on a new line); when *flag* is not given its value is 0.

getchar

```
char getchar()
char getchar(char * strprompt)
```

Displays the *Xcoral Dialog Box* to get a character. If *strprompt*is not given, it is replaced by *"getchar ? "*, which is written before the prompt. Only one character may be given, without line editing nor required *return*. If you choose the *ok* button, returns 0, if you choose *cancel* returns 7 (the bell).

gets

```
char * gets()
char * gets(char * strprompt)
```

Displays the *Xcoral Dialog Box* to get a string with the prompt *strprompt* (*"gets ? "* if *strprompt* is omitted). You have line editing, and the *return* or *ok* button is required to validate the input string. If you choose the *cancel* button, returns a string with one character (and the null terminator) of code 7. If you select the *ok* button without inputting a string, returns 0.

⚠ Be careful, the string is allocated each time the function is called, use **free()** to release it.

usleep

```
void usleep(int us)
```

Waits *us* micro-seconds.

watch_off

```
void watch_off()
```

Displays the default cursor.

watch_on

```
void watch_on()
```

Displays the cursor as a watch. It is sometimes useful to display a watch as cursor when SMAC works, typically when a function runs several seconds.

Chapter 8

Error messages

xcoral displays error messages in a special window, otherwise they are written on *stderr*. Errors can occur in three cases:

- When smac reads your programs, in which case, the line number and filename containing the error, is given.
- When smac *interprets* your programs and produces an internal form to memorize it. In this case and the next one, smac gives the function name or informs that it is a top level form. Unfortunately the line number is lost.
- During executions.

smac does not generate warnings, and when an error occurs, the execution or the reading stops. If you are under xcoral, this one recovers control.

The possible messages are:

8.1 Errors statically detected

These errors are detected by smac when it reads and interprets your programs.

8.1.1 Control structures

- **for (..; void; ..) ..** if the *for* end test form is not empty and returns no value.
- **for without test and body, I think it too dangerous !**
- **if (void) ..** when the if test form returns no value.
- **(void) ? .. : ..** if the test form returns no value.

- **while (void) ..** when the while test form returns no value.
- **do .. while(void)** when the do while test form returns no value.
- **more than one default: in switch**
- **duplicate case <case_entry>**
- **case <case_entry> out of switch**

```
{ case 1: ; }  /* case 1 out of switch */
```

- **illegal break** or **illegal continue** when you use a *break* or a *continue* out of a loop or a *switch*.

```
{ continue; }
```

- **return at top level** signals a *return* out of a function.

```
{ return; }
```

8.1.2 Function definitions and calls

- **Bad function definition** for an illegal form which looks like a function definition.

```
f{}
```

- **function initialized as a variable**.

```
f(){} = 1;
```

- **missing declared argument <var_name>** when an input parameter is not in the parameter list.

```
f(a) int a,b;{} /* missing declared argument b */
```

- **redeclaration of <var_name>**

```
f(a,a) int a; {}      /* redeclaration of a*/
f(a) int a; int a {}  /* redeclaration of a*/
```

- **the function returns no value** when a function as declared *void* attempts to return a value.

```
void novalue() { return 1; }
```

- **the function returns a value** when a function declared returning a value attempts to return no value.

```
int value() { return; }
```

- **incompatible return type** when the returned value type and declared returned type are inconsistent.

```
int badvaluetype() { return badvaluetype; }
```

- **illegal function call**

```
{ 1(); }
```

- **? arguments for <function_name>, ? required** and **? arguments for <function_name>, at least ? required** when you call a function with a wrong number of arguments.

```
{ error(); }   /* 0 arguments for error, 1 required */
{ printf(); }  /* 0 arguments for error, at least 1 required */
```

- **incompatible type for <function_name> argument ?, <type1> <> <type2>**

```
{ printf(1); }
```
(incompatible type for print argument 1, char * <> int)

- **incompatible type argument ?, <type1> <> <type2>**, as previously but in this case the function is executed.

```
int (*pf)(char);

{ pf("qwe"); }
```

- **? arguments at least ? required** when the number of argument of a computed function is insufficient. In this case SMAC does not test the equality of the number of arguments and the declared number at reading time, because the function has perhaps a variable number of arguments (as *printf*). Therefore check is performed at run time.

```
int (*pf)(char);

{ pf(); }
```

8.1.3 Assignment

- **incompatible types in assignment,** <**type1**> <> <**type2**> when the assigned value is incorrect.

`int i; { i = "123"; }` (incompatible types in assignment for i int <> char*)

- **incompatible types in array assignment,** <**type1**> <> <**type2**>
- **illegal assignment** <**var_name**> = **..** when the variable (local or global) is an array.

```
int tab[10];

{ tab = 1; }   /* illegal assignment tab = .. */
```

- **illegal assignment** <**var_name**>++ or **illegal assignment** <**var_name**>-/- when the variable (local or global) is an array.
- **illegal assignment** when you attempt to assign a constant:

```
{ "qwe" = 1; }
{ 1 = 1; }
```

- **illegal assignment (*&array = ...)** and
  illegal assignment (*(&array)++/-/-)

```
int mat[2][3];

{ *mat = 1; }
```

8.1.4 Operators

- **incompatible types in ?:,** <**type1**> <> <**type2**> the two values returned by an arithmetic if operator have different types.

```
{ (1) ? "1" : 2; } /*incompatible types in ?:, char* <> int */
```

- **pointer negation** and **void negation**

```
void f();

{ -"qwe"; }
{ -f(); }
```

- **pointer complement** and **void complement**
- **! void**
- <**operator**> **arguments have different types** arguments are pointers but not of the same type.

```
int t[10];
int f();

{ f - t; } /* - arguments have different types */
```

- **bad argument in** <**type1**> <**operator**> <**type2**>

```
{ 1 - "qwe"; }
```

- **illegal array reference** when you attempt to reference an array where there is no array or int (or char) index:

```
int tab[10], mat[2][3];

{ &tab[2]; }            /* legal */
{ &2[tab]; }            /* legal */
{ &mat[1]; }            /* legal */

{ &tab["2"]; }          /* illegal */
{ &error[2]; }          /* illegal */
{ &"2"[tab]; }          /* illegal */
{ &2[error]; }          /* illegal */
```

- **cast unacceptable operand of &** you cannot reference a *cast*. *SMAC* accepts cast at the left of an assignment, but it is probably too permissive.

- **illegal reference**

```
{ &1; }
```

- **illegal indirection (not a pointer)**, in an access or an assignment.

```
{ *1; }
```

8.1.5 Initialization

- **illegal array initializer for <var_name>**

```
int t1[10];
int t2[10] = t1; /* illegal array initializer for t2 */
```

- **incompatible initial value type for <var_name> <type1> <> <type2>** when the initialization value does not have a good type.

```
int i = "1";/*incompatible initial value type, i int <> char* */
```

- **array initialization not yet implemented**

```
int tab[2] = {1, 2};
```

- **{..} illegal initializer** when using brace list to initialize something else than an array.

```
int i = {123};
```

8.1.6 Array

- **? illegal array dimension** where ? is a negative integer

```
int tab[-1];
```

- **illegal array or index** when an array access or assignment is illegal because there is no array (or pointer) or int (or char) index, or when you attempt to affect an array:

```
int tab[10], mat[2][3];

{ tab[2]; }              /* legal */
{ 2[tab]; }              /* legal */

{ tab["2"]; }            /* illegal */
{ error[2]; }            /* illegal */
{ "2"[tab]; }            /* illegal */
{ 2[error]; }            /* illegal */
{ mat[1] = tab; }        /* illegal */
```

- [] **not yet implemented** linked with array initialization.

```
int tab[] = {1, 2};
```

8.1.7 Redefinition

- <**name**> **is already a function**

```
int i();
int i;    /* i is already a function */
```

- <**var_name**> **is already a global var** when you attempt to redefine a global variable, see *debug_mode* (§6.6.4 page 100).

- <**var_name**> **is already a global var with different definition** when you attempt to redefine a global variable with a new type or new dimension(s).

- <**var_name**> **already defined (global var)** when you reuse a global variable name as a function name, definition or declaration.

```
int i;
int i();  /* i already defined (global var) */
```

- <**function_name**> **already defined** when you attempt to redefine a function, see *debug_mode* (§6.6.4 page 100) and *remove_function_definition* (§6.6.4 page 99).

- **<function_name> already defined with different return type** when you redefine a function and change the return value type.
- **<function_name> already declared with different profile** when you redefine a function and change the number or the type of its arguments.
- **<function_name> is built-in, you cannot redefine it**

8.1.8 Others

- **Declaration in eval form** when you put a declaration in a free expression.
- **<a_name> unknown**

```
f() { not_a_variable_or_a_function_name; }
```

- **syntax error** the eternal message when the parser does not recognize a form.
- **I cannot understand the type description** theoretically this message never appears, else please send me the piece of program which generates it.

8.2 Errors detected at run time

8.2.1 Illegal memory access and sets

To get these messages, you must define RUNTIMECHECK *cpp* name when you compile SMAC (see §9 page 153).

- **char array access out of memory** to point out an illegal access through a character array (the array address plus the index is out of bounds). Here is an example generating this error message:

```
char * cp;

{ printf("Wait a moment please, I'm searching !\n");
  while (*cp++); }
```

⚠ Be careful, SMAC never checks if an index is out of bounds: For example this code won't produce any error message:

```
char surprise()
{
   char str[4];

   return str[11];
}
```

- **int array access out of memory** to point out an illegal access through an integer array (the array address plus the index is out of bounds).
- **illegal pointer alignment for int access** to point out an illegal access through an integer pointer with a bad alignment. Example:

```
int i;

{ *((int *) (((char *) &i) + 1)); } /* May be */
```

- **pointer array access out of memory** to point out an illegal access in an array of pointers or a multi dimensional array.
- **illegal pointer alignment for pointer access** to point out an illegal access through a pointer to a pointer with a bad alignment.
- **char access through pointer out of your memory area** to point out an illegal access through a character pointer out of boundaries.
- **int access through pointer out of your memory area** to point out an illegal access through an integer pointer out of bounds.
- **pointer access through pointer out of your memory area** to point out an illegal access through a pointer to a pointer out of bounds.
- **char assignment through pointer out of your memory area** to point out a character pointer out of memory in an assignment.
- **int assignment through pointer out of your memory area** to point out an integer pointer out of memory in an assignment.
- **illegal pointer alignment for int assignment** to point out an illegal assignment through an integer pointer with a bad alignment.
- **pointer assignment through pointer out of your memory area** to point out a pointer out of memory in an assignment.

- **illegal pointer alignment for pointer assignment** to point out an illegal assignment through a pointer with a bad alignment.
- **string access out of your memory area** to point out an illegal string access or assignment in a built-in function, of course the fault is yours.

```
{ strcmp((char *)error, "grrr..."); }
```

8.2.2 About functions

- **remove_function_definition argument is not a user function pointer**

```
remove_function_definition("123");
remove_function_definition(error);
```

- **you cannot redefine or undefine an active function**

```
void missed()
{
  remove_function_definition(missed);
}
```

- **function <function_name> not yet defined** when you call a function declared but not defined.

```
void i_am_undefined();

{ i_am_undefined(); }
```

- **computed function is not a function** when you attempt to call through a function pointer an illegal function, checked only if you have defined a RUNTIMECHECK *cpp* name. Here is an example generating this error message:

```
void (*pfunc)();

{ pfunc(); }           /* pfunc is null */
{ pfunc = (void(*)()) "crazy case";
  pfunc(); }
```

- **? arguments for <function_name>, ? required** when you call a function through a pointer with a bad number of arguments, checked only if you have defined a RUNTIMECHECK *cpp* name. Example:

```
((void(*)()) error)(); /* 0 arguments for error, 1 required */
```

- **not enough number of argument for <function_name> (? given)** when you call a built-in function with a variable number of arguments with not enough arguments.

```
{ printf("the missing argument is : %s"); }
```

8.3 Others

- **zero divide**, to get this message you must have defined a RUNTIMECHECK *cpp* name when you compiled *sMAc* (see §9 page 153). Example generating this message:

```
{ 1 % 0; }
```

- **cannot open <filename>** this error is generated by *load_file* (see §6.6.7 page 104) when *sMAc* cannot open or read the specified file.

```
{ load_file("/bin/abracadabra");
  error("What ?! who is your system engineer ?") }
```

- **function_percent argument type is <a type>, not a function**
- **function_name argument type is <a type>, not a function**
- **function_arg_count argument type is <a type>, not a function**
- **function_is_builtin argument type is <a type>, not a function**
- **function_type argument type is <a type>, not a function**
- **Error: stack is full**, perhaps the *sMAc* stack size is fixed too small, change it (in a new session), see §6 page 94. Terminal, crossed, wrap or other types of recursions are not recognized by *sMAc* to minimize stack size.

```
void catastrophe() { catastrophe(); }
```

- **Stop execution on** *ctrl-c* (*kill -2* on process *id*).
- **illegal free (address = 0x????)** only if SMAC is compiled with RUNTIMECHECK defined.

```
{ free("illegal"); }
```

Chapter 9

Compiling Smac

When you compile $\mathcal{SMAC}$ definition code, you can define or not the following *cpp* names:

- __STDC__ to have complete function profiles
- XCORAL to compile $\mathcal{SMAC}$ for $\mathcal{XCORAL}$, otherwise you must not compile and link *smacXcoral.c*, but just add *main.c*.
- RUNTIMECHECK to protect execution against *core dump* and other sweets. If you define it (recommended for $\mathcal{XCORAL}$) all your memory accesses and modifications are checked and an error occurs for each illegal addressing.

 The *word.h* file must not be present when you compile $\mathcal{SMAC}$. Then it can be produced by the *word.c* program to obtain objects sizes and alignments.

To compile $\mathcal{SMAC}$ for using it under $\mathcal{XCORAL}$ use *Imakefile* and *Makefile*. They define XCORAL and RUNTIMECHECK.

To compile $\mathcal{SMAC}$ alone, type *make −f MakeAlone*

Chapter 10

Miscellaneous

10.1 Bugs

Please send bug reports, fixes and suggestions to xcoral@free.fr

10.2 Xcoral home site

xcoral home page is **http://xcoral.free.fr**.

10.3 Authors

Lionel Fournigault is the founder of the *xcoral* editor. Bruno Pagès has written the Ansi C Interpreter SMAC. Dominique Lévêque has worked for the C/C++ browser.

10.4 Thanks

First, we would like to thank all beta-testers that tried intermediate versions before this 3.45.

Thanks to Thierry Emery who has written a part of **SmacLib** files.

Special thanks to Marie-Paule Kluth who has intensively tested the editor and who has helped me to review this manual.

Thanks also to the following folks who have sent comments, bug-reports and clever ideas.

```
Anthony Baxter
```

Austin G. Hastings
Bert Bos
Bert Gijsbers
Bertrand Zidler
Carsten Jerichow
Chris Sherman
Christophe Le Bars
DaviD W. Sanderson
Eckehard Stolz
Eric Sink
Erik Jan Lingen
Emmanuel Snyers
Frank Barnes
Fred J.R. Appelman
Fred R. Beck
Geraldo Veiga
George M. Menegakis
Grant McDorman
Heimir Thor Sverrisson
Jacques Tremblay
Jan Skibinski
Joerg Heuer
Joerg Stiller
Justin Seiferth
Jeffry R. Abramson
Jody Goldberg
Klamer Schutte
Laurent Duperval
Michael Andres
Marc Baudoin
Marie-Paule Kluth
Michael Baentsch
Mitch Baltuch
Olivier Marce
Pascal Perichon
Paul Hudson
Paul Sander
Peter Chang
Philippe Juhel
Richard Czech

```
Robert Nicholson
Roger Reynolds
Serge S.Maleyev
Thierry Emery
Todd Vernon
Todd C. Miller
Torsten Blum
Torsten Schlumm
```

Appendix A GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and

passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

1. (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE

DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the program's name and a brief idea of what it does.*
> Copyright (C) 19*yy* *name of the author*
>
> ```
> This program is free software; you can redistribute it and/or modify
> it under the terms of the GNU General Public License as published by
> the Free Software Foundation; either version 2 of the License,
> or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful,
> but WITHOUT ANY WARRANTY; without even the implied warranty of
> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
> GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License
> along with this program; if not, write to the Free Software
> Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
> ```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> ```
> Gnomovision version 69, Copyright (C) 19@var{yy} @var{name of author}
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
> This is free software, and you are welcome to redistribute it
> under certain conditions; type ‘show c’ for details.
> ```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> *signature of Ty Coon*, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index