The Oz Programming Interface

Leif Kornstaedt
Denys Duchier

Version 1.3.2
June 15, 2006

Abstract

The Oz Programming Interface (OPI) is the primary tool for interaction with the Mozart development system. It offers special support for editing Oz code, running Mozart as a sub-process, and interacting with Mozart's development tools. This document is a reference manual for the complete functionality of the OPI.

Credits

Mozart logo by Christian Lindig

License Agreement

Contents

1

Introduction

The Mozart Programming System provides a powerful environment for the development of software systems, called the 'Oz Programming Interface' (OPI). The OPI is built around the extensible Emacs editor and runs (at least) under GNU Emacs, Version 19.24 or greater, and XEmacs, Version 19.14 or greater. Its main features are:

Features

Editing Oz code. The OPI automatically indents program lines and colorizes Oz source code to ease reading and writing of Oz programs. Due to its awareness of the syntactical structure of Oz, one can work with programs by applying commands to whole constructs such as procedure or class definitions.

Running Mozart as a sub-process. The OPI handles input to and output from a Mozart sub-process, providing a convenient interface for the interactive use of the Mozart system and for explorative programming.

Starting Mozart's development tools. The OPI provides menus and shortcuts to interact with the development system's graphical tools, e.g., setting breakpoints for the thread debugger or displaying the current position in the source file being debugged.

The Manual's Structure This manual is structured as follows. Chapter 2 gives an overview of the OPI's general integration into the standard framework provided by Emacsen[1]. Chapter 3, Chapter 4, and Chapter 5 are dedicated to the three main features mentioned above respectively. Chapter 6 describes how to manage multiple Oz mode settings using profiles. Appendix A summarizes all Oz-specific key bindings.

The last three appendices provide information for advanced users. Appendix B details how to test Mozart system components locally and how to run Mozart under gdb. Appendix C documents some functions of the OPI that might be useful for users who want to write their own editing commands. Finally, Appendix D lists the known limitations of the OPI with workarounds.

[1] 'Emacsen' is the plural of 'Emacs'. In this manual, we use the term when the feature being described applies to both GNU Emacs and XEmacs.

Learning Emacs This manual assumes some familiarity with the general editing commands of Emacsen and uses standard Emacs terminology. If you want to exploit the full power of the OPI you should get some acquaintance with Emacs. A good place to start is the Emacs on-line tutorial [1], available from the Emacs Help menu; this is also the place to check if you are confused by the terminology used in this manual. You might especially want to look up the following words in the Emacs manual's glossary: point, mark, region, buffer, window, frame, mode line, killing, command, user option, prefix argument.

Acknowledgements

The Oz Programming Interface of the Mozart system is an extension and partial redesign of the Oz Programming Interface of DFKI Oz, Versions 1.1 and 2.0. Credit has to go to the following people:

- Michael Mehl[2], for initially providing editing support (indentation, fontification),
- Ralf Scheidhauer[3], for running Oz as a sub-process,
- Benjamin Lorenz[4], for the interaction with the Oz debugger,
- Jochen Dörre, for initially providing Oz expression editing commands and jumping to compiler error messages.

Leif Kornstaedt is responsible for most of the current Mozart OPI. Denys Duchier contributed the concept of *profiles*.

[2] `http://www.ps.uni-sb.de/~mehl/`
[3] `http://www.ps.uni-sb.de/~scheidhr/`
[4] `http://www.ps.uni-sb.de/~lorenz/`

2

Invoking the OPI

This chapter describes how to invoke the OPI, i.e., how to access its functionality.

2.1 Invoking the OPI in the Unix Environment

The easiest way to start the OPI is to type the following command at the shell prompt[1]:

```
% oz ⟨emacs args⟩
```

After setting up the necessary environment variables, this starts up an Emacs process, passing to it all arguments given on the command line, creates a new buffer named `Oz`, and starts a Mozart sub-process.

Which Emacs to Use The command used to invoke Emacs is determined through the following steps:

1. If the environment variable `OZEMACS` is set, its contents is used.
2. Else, if a command named `emacs` is found in the `PATH`, this is used.
3. Else, if a command named `xemacs` is found in the `PATH`, this is used.
4. Else, if a command named `lemacs` is found in the `PATH`, this is used.

2.2 Invoking the OPI Under Windows

The installation procedure will have created a program group for the Mozart system. The OPI is started by launching the Mozart item. This item is a shortcut to the `oz.exe` program within the `bin` subfolder of the installation folder; as under Unix, any arguments given to it are passed on to the invoked Emacs.

[1] The percent sign (%) represents the shell prompt; it is not part of the command.

Which Emacs to Use The command used to invoke Emacs is determined through the following steps:

1. If the environment variable `OZEMACS` is set, its contents is used.
2. Else, if the registry indicates where GNU Emacs is installed, this is used.
3. Else, if the registry indicates where XEmacs is installed, this is used.

2.3 Invoking the OPI From Within Emacs

You can also configure your Emacs so that you can use all of the OPI's functionality without using the `oz` script. Here's what you would typically add to your Emacs startup file (usually called `~/.emacs` under Unix and `C:\_emacs` under Windows 95; under Windows NT, it is located in your home directory):

```
(or (getenv "OZHOME")
    (setenv "OZHOME"
            "/usr/local/oz"))   ; or wherever Mozart is installed
(setenv "PATH" (concat (getenv "OZHOME") "/bin:" (getenv "PATH")))

(setq load-path (cons (concat (getenv "OZHOME") "/share/elisp")
                      load-path))

(setq auto-mode-alist
      (append '(("\\.oz\\'" . oz-mode)
                ("\\.ozg\\'" . oz-gump-mode))
              auto-mode-alist))

(autoload 'run-oz "oz" "" t)
(autoload 'oz-mode "oz" "" t)
(autoload 'oz-gump-mode "oz" "" t)
(autoload 'oz-new-buffer "oz" "" t)
```

Don't worry if you don't understand all of this (yet).

2.4 The Oz Major Modes

All of the OPI's functions are accessible in the following two major modes:

command **oz-mode**

This is the major mode for editing Oz code. Loading a file with extension `.oz` automatically puts a buffer into Oz mode. You can tell a buffer is in Oz mode by the string `Oz` in its mode line.

command **oz-gump-mode**

This is the major mode for editing Oz code with embedded Gump specifications (see *"Gump–A Front-End Generator for Oz"*). Loading a file with extension `.ozg` automatically puts a buffer in Oz-Gump mode. You can tell a buffer is in Oz-Gump mode by the string `Oz-Gump` in its mode line.

Oz Mode Hook To both of these, the following hook applies.

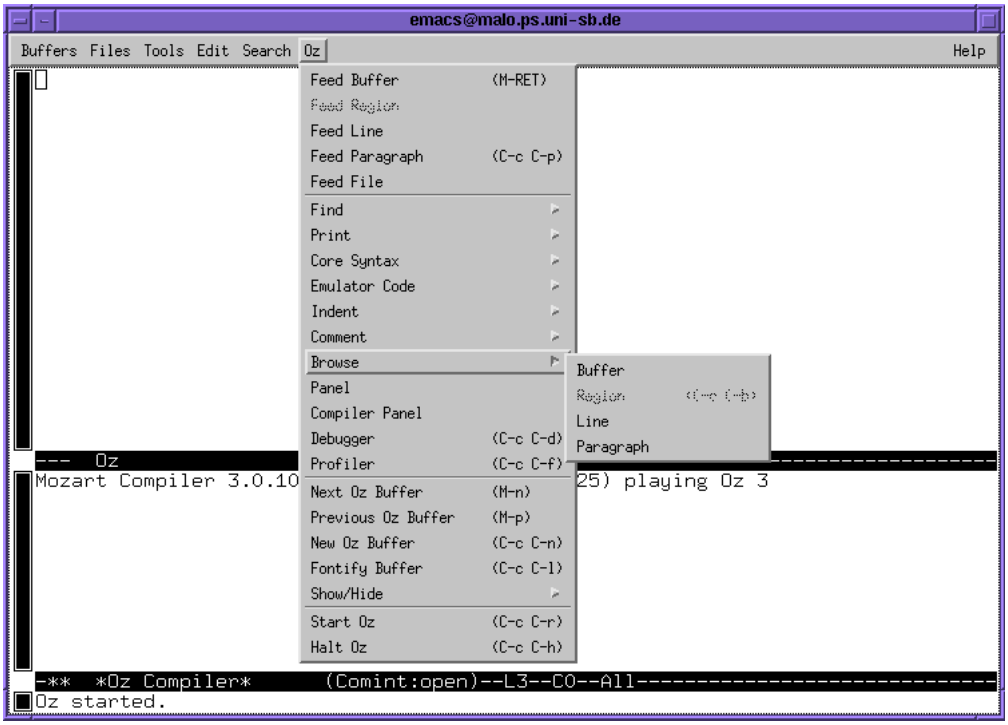
user option **oz-mode-hook**

A list of functions to be run when one of the Oz modes is activated. These functions are applied without arguments. Change using Emacs functions `add-hook` and `remove-hook`.

2.5 Inspecting the OPI's Commands and User Options

The Oz Menu The Oz major modes add a menu called `Oz` to the menu bar (see Figure 2.1); this menu is also accessible by pressing the right mouse button in an Oz buffer. Many of the commands described in the next chapters are accessible through this menu.

Figure 2.1 The Oz Menu.



Emacs Conventions The Oz modes conform to the following Emacs conventions:

- Nearly all functions and variables start with `oz-...`
- If the documentation string of a variable starts with an asterisk, then its value is meant for the user to modify at will (a so-called user option). The documentation string of a variable can be inspected with `M-x describe-variable` (`C-h v`).

- If a function has a documentation string, then it is meant for the user to use directly if she so wishes. Inspect the documentation string of a function with `M-x describe-function` (`C-h f`); if a command is bound to a key, you can examine its documentation string with `M-x describe-key` (`C-h k`).
- The OPI provides the feature `oz`. See Emacs' `require` function for more details.

Customization New Emacsen offer a feature called customization, which serves the purpose of setting the user options pertaining to a mode in a structured way. You can access this feature by `M-x customize`; look at the group `Programming/Languages/Oz`. You can also access this group directly via `M-x customize-group RET oz`.

Key Bindings A short description of the current major mode and its key bindings can be obtained through Emacs' `M-x describe-mode` (`C-h m`). In this manual, the key sequences a command is bound to by default will always be shown in parenthesis following the command name.

user option **`oz-mode-map`**

Keymap used in the Oz modes.

Generally, Oz-specific commands are made available both with `C-.` and `C-c .` as prefix. This manual always lists only the first of these. However, some terminals may not be able to generate `C-.`; this is why the second one is provided.

3

Editing Oz Code

The commands in this chapter assist in editing Oz code. To achieve this, many of these are aware of the lexical or syntactical structure of Oz programs.

3.1 Managing Oz Buffers

The Oz modes offer commands for creating new interactive buffers and quickly switching between Oz buffers:

command **oz-new-buffer** (**C-. n**)

Create a new buffer using the Oz major mode. Note that this buffer has no associated file name, so quitting Emacs will kill it without warning.

command **oz-next-buffer** (**M-n**)
command **oz-previous-buffer** (**M-p**)

Switch to the previous resp. next buffer in the buffer list that runs in an Oz mode. If no such buffer exists, an error is signalled.

3.2 Indentation

The preferred indentation style can currently be customized through the following user option:

user option **oz-indent-chars** (**default: 3**)

Number of columns that statements are indented wrt. the block containing them.

Several commands assist in formatting existing Oz code.

command **oz-indent-line** **&optional** COUNT (**TAB**)

Reindent the current line. If COUNT is given, reindent that many lines above and below point as well.

command **oz-indent-region**

Reindent all lines at least partly covered by the current region.

command **oz-indent-buffer**

Reindent every line in the buffer.

command **indent-oz-expr (M-C-q)**

Reindent all lines at least partly covered by the Oz expression following point. For a description of what constitutes an Oz expression, see Section 3.5.

The following command assists in authoring Oz code.

command **oz-electric-terminate-line (RET)**

Terminate the current line, i.e., delete all whitespace around point and break the line. If the user option `oz-auto-indent` is non-`nil`, indent both lines.

user option **oz-auto-indent (default: t)**

See `oz-electric-terminate-line`.

Additionally, `DEL` is bound to the Emacs command `backward-delete-char-untabify`.

3.3 Fontification

Fontification is the term used in Emacs for displaying text in different font faces, depending on its syntactical form and context, to ease reading of code. For example, comments and strings may be displayed in different colours.

Many major modes in Emacs provide several levels of fontification with increasing use of faces, but also increasing resource consumption. In the Oz modes, there are three levels. You can select one using the `font-lock-maximum-decoration` user option, e.g., add the following line to your `.emacs`:

```
(setq font-lock-maximum-decoration 3)
```

The default level depends on your version of Emacs.

The following user option controls automatic fontification in the OPI.

user option **oz-want-font-lock (default: t)**

If non-`nil`, automatically invoke `font-lock-mode` when any of the Oz modes is activated. If you prefer to control this via `global-font-lock-mode`, you can set this to `nil`.

You might like the following user option and command if you care about superfluous (usually invisible) spaces:

user option **oz-pedantic-spaces (default: nil)**

If non-nil, highlight ill-placed whitespace. Note that this user option must be set before the `oz` library is loaded.

face **oz-space-face**

The face in which ill-placed whitespace is highlighted.

command **oz-remove-annoying-spaces**

Remove all ill-placed whitespace from the current buffer. This is all the whitespace that is highlighted in `oz-space-face`.

3.4 Comments

command **oz-fill-paragraph** &optional JUSTIFY

Like the fill-paragraph command, but handles Oz comments. If any of the current line is a comment, fill the comment or the paragraph of it that point is in, preserving the comment's indentation and initial percent signs. The buffer-local variable fill-paragraph-function is bound to this command, so it will also be invoked by M-x fill-paragraph (M-q).

command **oz-comment-region** START END &optional ARG

Comment or uncomment each line in the region. With just C-u as prefix argument, uncomment each line in region. A numeric prefix argument ARG means use ARG comment characters. If ARG is negative, delete that many comment characters instead. Blank lines do not get comments.

command **oz-uncomment-region** START END &optional ARG

Comment or uncomment each line in the region. See the oz-comment-region command for more information; note that the prefix argument is negated though.

3.5 Expression-Level Commands

In this section, we use the term Oz definition to stand for the text from a **proc**, **fun**, **class** or **meth** keyword up to its matching **end**. Also, we use the term Oz expression to stand for the text corresponding to either a bracketed Oz construct (such as **proc** ... **end** or **local** ... **end**) or a single word.

command **forward-oz-expr** &optional COUNT **(M-C-f)**

Move point forward by one balanced Oz expression. With COUNT, do it that many times. Negative COUNT means backwards.

command **backward-oz-expr** &optional COUNT **(M-C-b)**

Move point backward by one balanced Oz expression. With COUNT, do it that many times. COUNT must be positive.

command **mark-oz-expr** COUNT **(M-C-@, M-C-SPC)**

Set mark COUNT balanced Oz expressions from point. The place mark goes to is the same place the forward-oz-expr command would move to with the same argument.

command **transpose-oz-exprs** ARG **(M-C-t)**

Like the transpose-words command (M-t) but applies to balanced Oz expressions. Caveat: This might not produce nice results in all cases.

command **kill-oz-expr** COUNT **(M-C-k)**

Kill the balanced Oz expression following point. With COUNT, kill that many Oz expressions after point. Negative COUNT means kill -COUNT Oz expressions before point.

command **backward-kill-oz-expr** COUNT (**M-C-DEL**[1])

Kill the balanced Oz expression preceding point. With COUNT, kill that many Oz expressions before point. Negative COUNT means kill -COUNT Oz expressions after point.

command **oz-beginning-of-defun** (**M-C-a**)

Move point to the start of the Oz definition it is in. If point is not inside an Oz definition, move to start of buffer. Returns t unless search stops due to beginning or end of buffer.

command **oz-end-of-defun** (**M-C-e**)

Move point to the end of the Oz definition it is in. If point is not inside an Oz definition, move to end of buffer.

[1] Note that under some configurations, this key combination kills the X server.

4

Running Mozart from the OPI

The OPI allows to run Mozart directly from the OPI. A sub-process is started that executes `ozengine` with a single root functor argument, by default called `OPI.ozf`. In particular, `{Property.get argv}` will always return `nil`.

Emulator and Compiler The output of the process is redirected into an Emacs buffer called `*Oz Emulator*`. For instance, all output done via `System.show` etc. will appear in this buffer. Additionally, the `OPI.ozf` program instantiates an Oz compiler and attaches its input and output to an Emacs buffer called `*Oz Compiler*`; communication, in this case, is done via a socket. The compiler might also create a new buffer for output of source code, called `*Oz Temp*`.

When we speak of the 'Oz Emulator' and 'Oz Compiler' buffers in this manual, we mean the buffers called `*Oz Emulator*` and `*Oz Compiler*` respectively.

In order to run the Mozart system, the OPI has to know its installation path. This is normally found through the environment variable `OZHOME`; it will have been set by the `oz` shell script if you started the OPI with it. If it is not set, the value of the following variable will be used instead.

user option **`OZ-HOME`** **(default: `/usr/local/oz`[1])**

Directory where Oz is installed. Only used as fallback when the environment variable `OZHOME` is not set.

4.1 Running and Halting

The following commands are used to start and halt the Mozart sub-process.

command **`run-oz`** **(`C-. r`)**

Start Mozart as a sub-process if it is not already running. Handle input and output via the Oz Emulator buffer. If the current buffer is not running in an Oz mode, create a new buffer in Oz mode.

user option **`oz-change-title`** **(default: `nil`)**

If non-`nil`, change the Emacs frame's title while a Mozart sub-process is running.

[1] This default is actually fixed at the time the Mozart system is configured and built, so it might vary on your system.

user option **oz-frame-title (default: "Oz Programming Interface (...)")**

String to use as Emacs frame title while a Mozart sub-process is running. In the default shown above, the old frame title will be inserted in place of the ellipsis.

command **oz-halt FORCE (C-. h)**

Halt the Mozart sub-process. With no prefix argument, feed an `/{Application.exit 0}` statement and wait for the process to terminate. Waiting time is limited by the user option `oz-halt-timeout`; after this delay, the process is sent a SIGHUP if still living.

With `C-u` as prefix argument, send the process a SIGHUP without delay. With `C-u C-u` as prefix argument, send it a SIGKILL instead.

user option **oz-halt-timeout (default: 30)**

Number of seconds to wait for shutdown in command `oz-halt`.

4.2 Mozart's Output Buffers

Several commands make inspecting the Oz Emulator and Oz Compiler buffers easier.

command **oz-toggle-emulator (C-. e)**
command **oz-toggle-compiler (C-. c)**
command **oz-toggle-temp (C-. t)**

Toggle visibility of the Oz Emulator, Compiler or Temporary window respectively. If the buffer is not visible in any window, then display it. If it is, then delete the corresponding window.

user option **oz-other-buffer-size (default: 35)**

Percentage of screen to use for Oz Compiler, Emulator or Temp window.

4.3 Feeding Code to the Compiler

Feedable Regions The commands that send regions of the current buffer to the Oz Compiler for compilation come in four flavors:

- Feeding the whole buffer. More specifically, the region the buffer has been narrowed to is fed.
- Feeding the currently marked region, i.e., the text contained between point and mark.
- Feeding the line point is in. If a numeric prefix argument is given, that many lines are fed; if the prefix argument is negative, that many preceding lines as well as the current line are fed.
- Feeding the paragraph point is in (or after, if it is not inside any paragraph). A paragraph is a region of text delimited by empty lines, i.e., lines not even containing whitespace. If a numeric prefix argument is given, that many paragraphs are fed; if the prefix argument is negative, that many preceding paragraphs as well as the current paragraph are fed.

command **oz-feed-buffer** (**C-. C-b**)
command **oz-feed-region** START END (**C-. C-r**)
command **oz-feed-line** COUNT (**C-. C-l**)
command **oz-feed-paragraph** COUNT (**C-. C-p**, **M-C-x**)

The corresponding text region is fed to the compiler and processed with its currently active switches.

command **oz-show-buffer** (**C-. s C-b**)
command **oz-show-region** START END (**C-. s C-r**)
command **oz-show-line** COUNT (**C-. s C-l**)
command **oz-show-paragraph** COUNT (**C-. s C-p**)

Feed the corresponding text region to the Oz Compiler. Assuming it to contain an expression, enclose it by an application of the procedure `Show`.

command **oz-to-coresyntax-buffer**
command **oz-to-coresyntax-region** START END
command **oz-to-coresyntax-line** COUNT
command **oz-to-coresyntax-paragraph** COUNT

The corresponding text region is prefixed by

```
\localSwitches
\switch +core -codegen
```

and fed to the Oz Compiler. If compilation succeeds, the resulting source file will be displayed in the Oz Temporary buffer.

command **oz-send-string** STRING **&optional** SYSTEM

Feed `STRING` to the Oz Compiler, restarting it if it died. If `SYSTEM` is non-`nil`, it is a command for the system and is prefixed by

```
\localSwitches
\switch +threadedqueries -verbose -expression -runwithdebugger
```

user option **oz-prepend-line** (**default:** `t`)

If non-`nil`, prepend a `\line` directive to all Oz queries, specifying the file name (or buffer name, if there's no associated file) and the line number. This information is used by the compiler to output meaningful error messages and to include debugging information in the generated machine code.

4.4 Running the Command-Line Tools

command **oz-compile-file**

Compile an Oz program non-interactively.

user option **oz-compile-command** (**default:** `"ozc -c %s"`)

Default shell command to do a compilation. This may contain at most one occurrence of `%s`, which is replaced by the current buffer's file name. Used by `oz-compile-file`.

command **oz-debug-application**

Invoke `ozd`.

user option **oz-application-command (default: "%s")**

Default shell command to do execute an Oz application. This may contain at most one occurrence of `%s`, which is replaced by the current buffer's file name, minus the `.oz` or `.ozg` extension. Used by `oz-debug-application`.

4.5 Dealing With Errors

Error Messages An error message is either an error or warning message issued by the Oz Compiler or an exception displayed by the Emulator.

Error Coordinates Where available, error coordinates are associated with error messages, consisting of the file name (or buffer name) and line number of the corresponding Oz source code.

user option **oz-popup-on-error (default: t)**

If non-nil, pop up Compiler resp. Emulator buffer upon an error message.

command **next-error &optional** ARG **(C-x ')**

Visit next compilation error message and corresponding source code.

A prefix arg specifies how many error messages to move; negative means move back to previous error messages. Just `C-u` as a prefix means reparse the error message buffer and start at the first error.

This normally uses the most recently started compilation. To specify use of a particular buffer for error messages, type `C-x '` in that buffer.

4.6 Seeing the OPI from Mozart

Startup When the `OPI.ozf` file is applied, a startup file is searched and loaded as follows:

1. It is first checked whether the environment variable `OZRC` is set. If it is, its contents is interpreted as a file name that is fed to the OPI compiler.
2. Else, if the file `~/.oz/ozrc` exists and is readable, it is fed to the compiler.
3. Else, if the file `~/.ozrc` exists and is readable, it is fed to the compiler.

Compiler Environment The environment available when running Mozart from the OPI is an enriched base environment (see *"The Oz Base Environment"*). All of Mozarts system modules and tools are available under variables named like the corresponding modules, e.g., the functionality of the open programming component is available as `Open`. Additionally, the following aliases are introduced:

Alias	Long Form
`Show`	`System.show`
`Print`	`System.print`
`Browse`	`Browser.browse`
`Inspect`	`Inspector.inspect`
`Load`	`Pickle.load`
`Save`	`Pickle.save`
`SearchOne`	`Search.base.one`
`SearchAll`	`Search.base.all`
`SearchBest`	`Search.base.best`
`ExploreOne`	`Explorer.one`
`ExploreAll`	`Explorer.all`
`ExploreBest`	`Explorer.best`

Compiler Interface When Mozart is started from the OPI, an instance of the Mozart compiler is created that listens for queries from the interactive development environment. This interaction is handled via a compiler interface called `Emacs.interface` (see Section *Compiler Interfaces, (The Mozart Compiler)*).

System Properties It is possible to test whether Mozart is currently running under the OPI or as a standalone system via the following system property:

```
{Property.get 'oz.standalone' ?B}
```

This returns `false` when Mozart has been started from the OPI. When this is the case, a reference to the compiler interface via which the interaction with the Emacs development environment takes place can be obtained via

```
{Property.get 'opi.compiler' ?O}
```

5

Interacting With the Development Tools

This section briefly documents how Mozart's development tools are integrated into the OPI; several of these commands are available from the `Oz` menu. For more details about the tools themselves, see the individual user manuals. For a description of the feedable regions, see Section 4.3.

The following command is useful for several of the tools.

command **`oz-bar-remove`**

Remove any coloured bar marking an Oz source line. Such bars are used by the Compiler Panel, the Debugger and the Profiler.

Browser

command **`oz-browse-buffer`** **(`C-. b C-b`)**
command **`oz-browse-region`** `START END` **(`C-. b C-r`)**
command **`oz-browse-line`** `COUNT` **(`C-. b C-l`)**
command **`oz-browse-paragraph`** `COUNT` **(`C-. b C-p`)**

Feed the corresponding text region to the Oz Compiler. Assuming it to contain an expression, enclose it by an application of the procedure `Browse`.

Inspector

command **`oz-inspect-buffer`** **(`C-. i C-b`)**
command **`oz-inspect-region`** `START END` **(`C-. i C-r`)**
command **`oz-inspect-line`** `COUNT` **(`C-. i C-l`)**
command **`oz-inspect-paragraph`** `COUNT` **(`C-. i C-p`)**

Feed the corresponding text region to the Oz Compiler. Assuming it to contain an expression, enclose it by an application of the procedure `Inspect`.

System Panel

command **`oz-open-panel`** **(`C-. C-. s`)**

Open the System Panel by feeding the statement `{Panel.open}` to the Oz Compiler.

Compiler Panel

command **oz-open-compiler-panel** (**C-. C-. c**)

Open the Compiler Panel by feeding the statement `{New CompilerPanel.'class' init(OPI.compi` to the Oz Compiler.

Debugger

command **oz-debugger** ARG (**C-. C-. d**)

Open the Oz Debugger by feeding the statement `{Ozcar.open}` to the Oz Compiler. With ARG, close it instead by `{Ozcar.close}`.

command **oz-breakpoint-at-point** ARG (**C-x SPC**)

Set a dynamic breakpoint for the Oz Debugger in any code carrying the current source file name (or buffer name) and line number as debugging information. With ARG, delete any breakpoints at these coordinates instead.

Profiler

command **oz-profiler** ARG (**C-. C-. p**)

Open the Oz Profiler by feeding the statement `{Profiler.open}` to the Oz Compiler. With ARG, close it instead by `{Profiler.close}`.

6

Using Profiles

A profile defines all the settings necessary to run a particular version or installation of Oz. You can have an number of profiles and you can switch between them while remaining in the same Emacs session. The advantages for regular users are:

- You don't need to invoke `oz` to start an Oz session. Rather, you start your regular Emacs and whenever you want to start interacting with a certain version of Oz installed on your system, you select the appropriate profile and all appropriate settings are automagically installed.
- You can thus easily switch between several installations of Mozart.
- It is easy to customize the settings for a profile using Emacs's `customize` interface.

Additional advantages for Mozart developers are that you can easily define profiles

- to run Oz from a build directory.
- to run a debug emulator under GDB.

user option **`oz-profiles`** **(default: `nil`)**

An alist of profiles for different Oz mode configurations. The keys of this alist are the names of the profiles.

command **`oz-set-profile`** NAME

Select profile `NAME` from those defined in `oz-profiles`. If Oz is currently running, the user is prompted to decide whether to kill it. A new profile can only be installed when Oz is not currently running in the OPI.

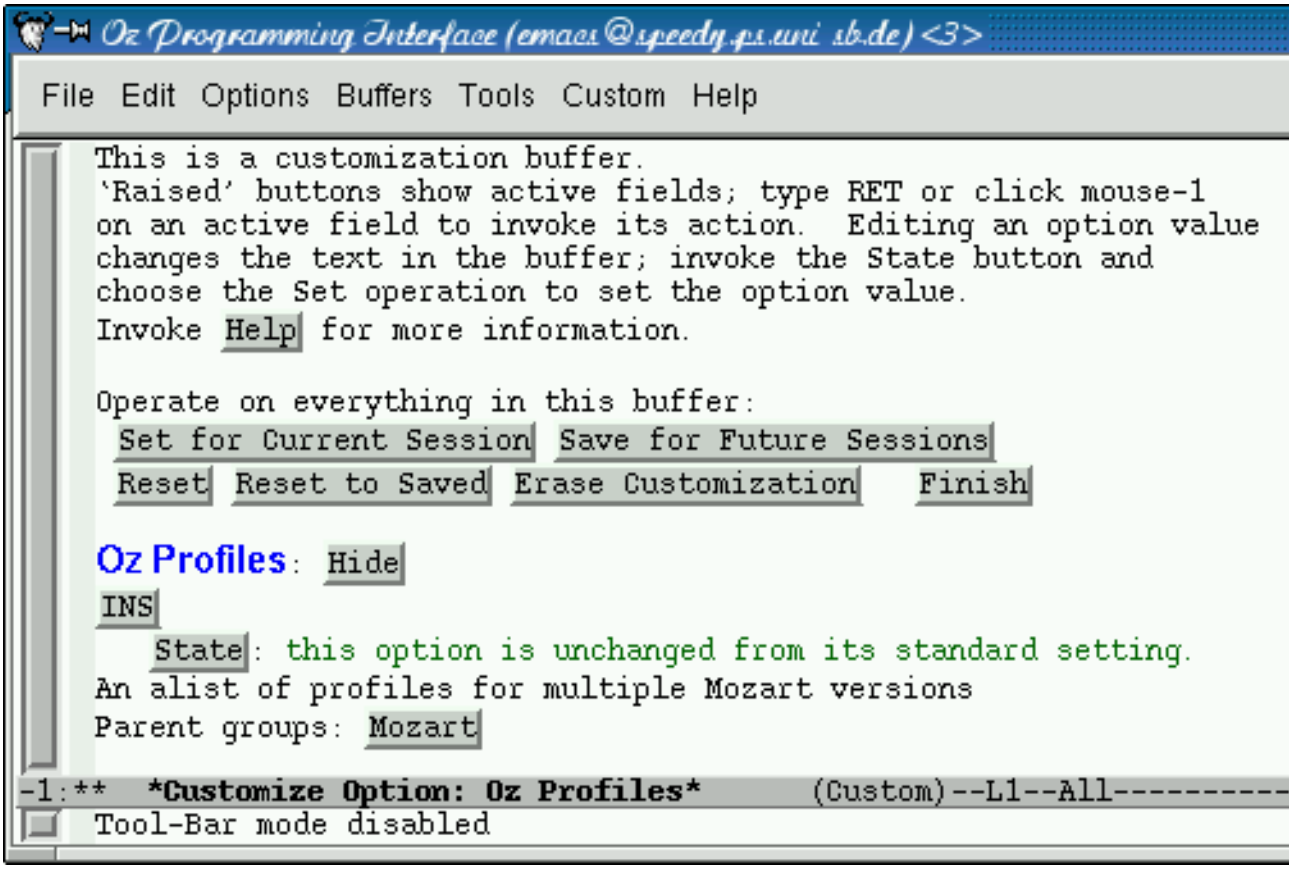
command **`oz-profile-undo`**

Undo the bindings established by the current profile. This is rarely useful because `oz-set-profile` automatically invokes `oz-profile-undo` in order to undo existing settings and replace them with new ones.

6.1 Creating and Customizing Profiles

A profile is an alist providing values for Oz mode variables and environment variables. The best way to create and customize profiles is through Emacs's `customize` interface. For example `M-x customize-variable RET oz-profiles` will provide you with an interactive customization sheet as shown in Figure 6.1. We create

Figure 6.1 Initial Customization Sheet



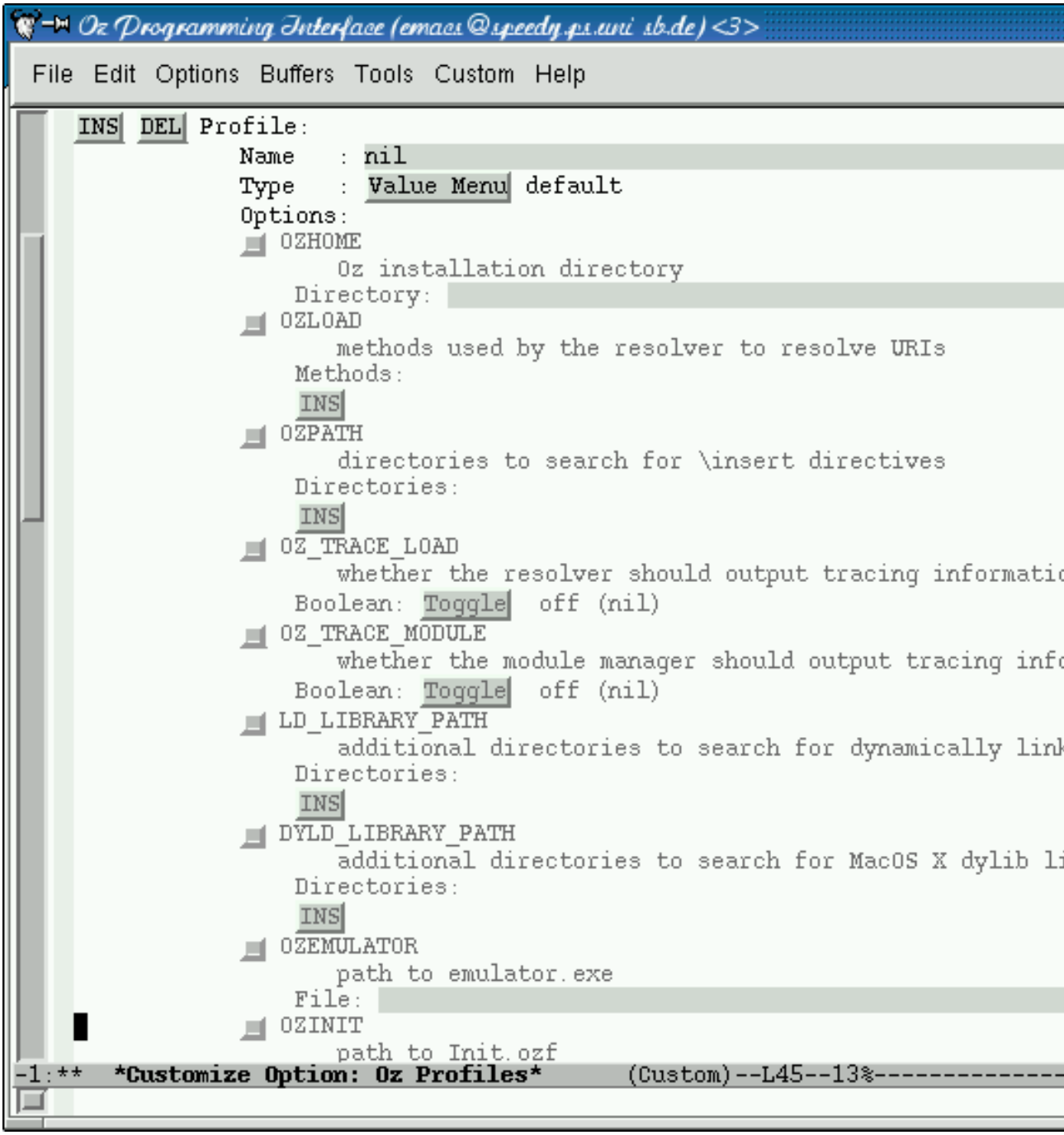
a new profile by clicking the `INS` (insert) button, and obtain a new entry as shown in Figure 6.2 with many possible settable parameters. Fortunately, you never need to specify many of them. All the others take appropriate default values.

6.1.1 Global Profile

For example, let's define a profile to use the globally installed Mozart in the usual fashion. Here are the steps to follow:

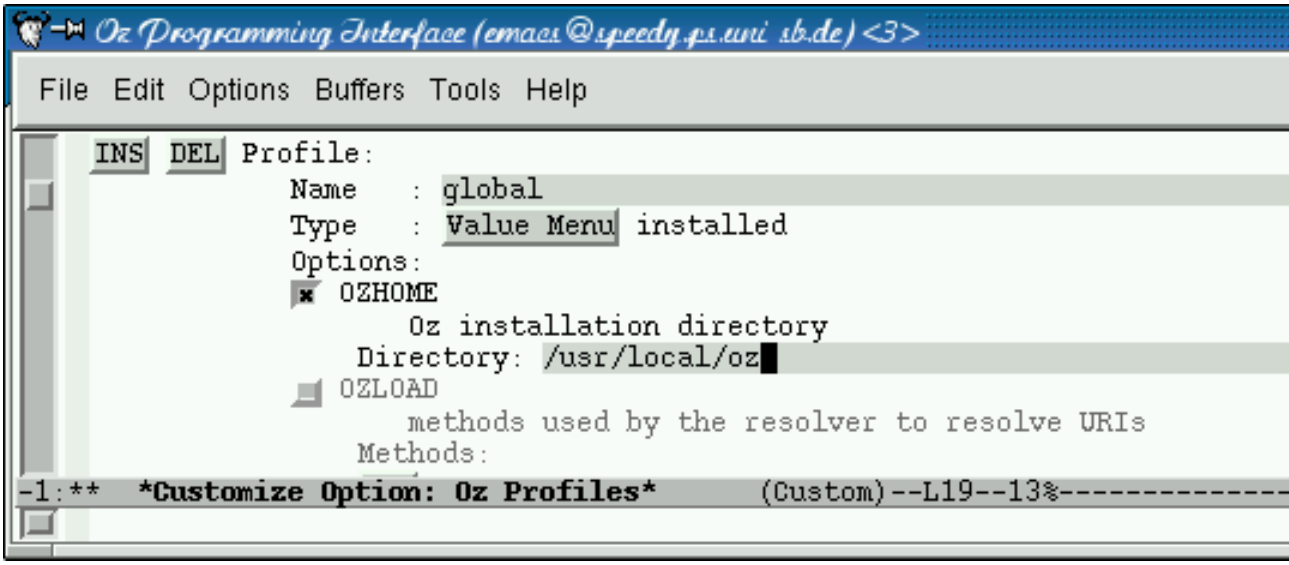
- Give a name to the profile, e.g. `global`.
- Select type `installed` from the value menu for the `Type` parameter. This type information allows other important settings to be automatically computed.
- Select the `OZHOME` parameter by clicking in the box, and then set its value to the directory of an installed Mozart system. For example `/usr/local/oz`.

Figure 6.2 After Clicking INS



We arrive at the profile customization shown in Figure 6.3. Don't forget to save it by clicking the `Save For Future Sessions` button.

Figure 6.3 Global Profile



6.1.2 Default Profile

A so-called default profile consists of whatever settings were present in your environment when you started Emacs. This is for proper operation when you actually choose to invoke `oz` rather than plain `emacs`: in that case, the default profile consists of the settings established by the `oz` script. You can also create a default profile:

- Give a name to the profile, e.g. `default`
- Select type `default`

The only reason to have a named default profile is so that you can revert to the original settings in force when you started `oz` by switching to this profile.

6.1.3 Build Profile

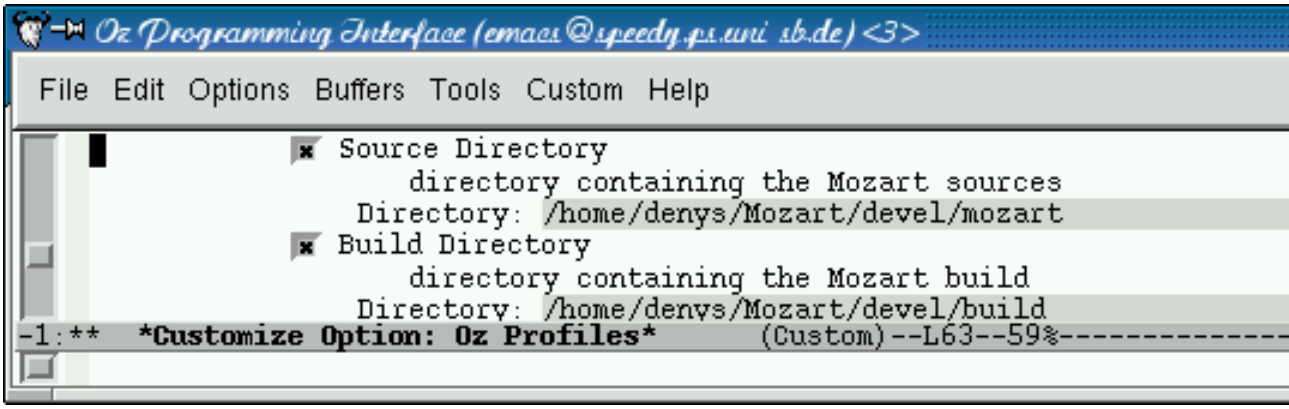
Developers may find it marginally convenient to be able to define a profile where the emulator and all modules are looked up directly in the build tree. Here are the steps to follow:

- Give a name to the profile, e.g. `build`
- Select type `build`
- Set the source directory: this is the top directory in which you checked out the Mozart sources

- Set the build directory: this is the top directory in which you built the Mozart system

For example, Figure 6.4 shows typical build settings on my laptop.

Figure 6.4 Build Profile



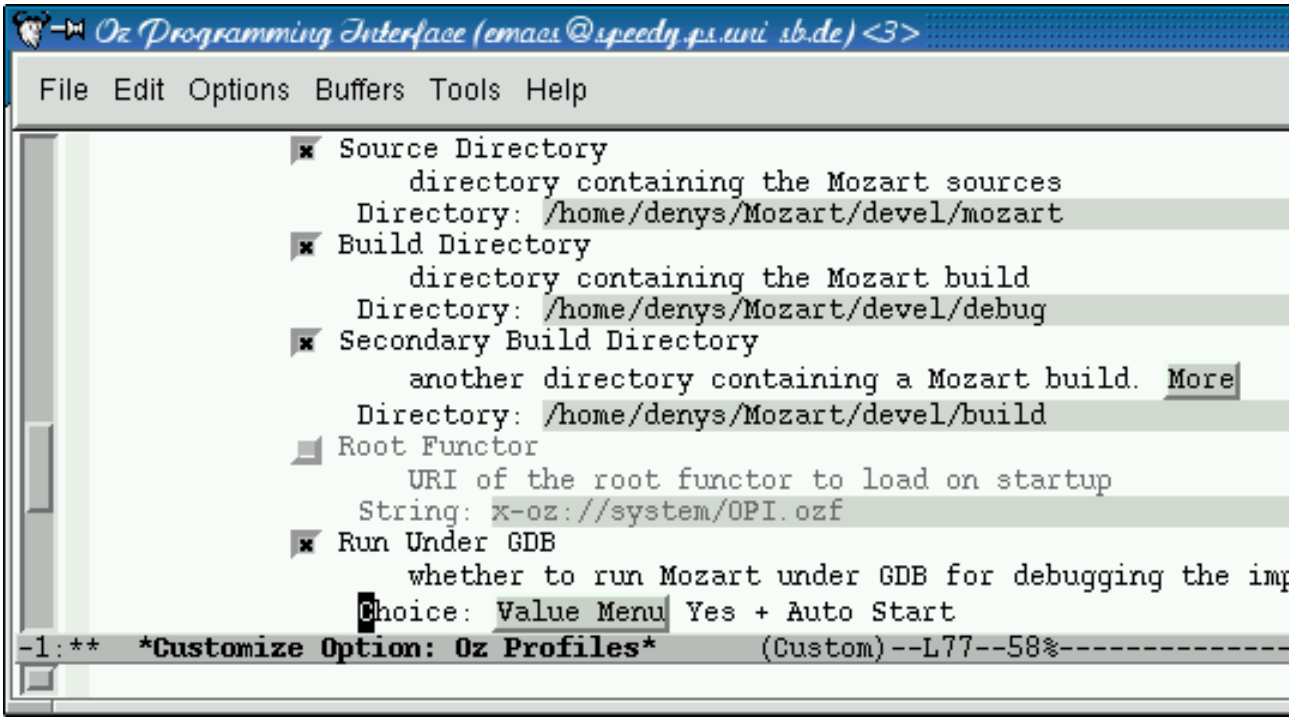
6.1.4 Debug Profile

Developers will find it particularly convenient to be able to define debug profiles, i.e. build profiles for builds configured with `-enable-opt=d`, and, in particular the ability to state that they should be run under control of GDB. Here are the steps to follow:

- Give a name to the profile, e.g. `debug`
- Select type `build`
- Set the source directory
- Set the build directory
- When compiling a debug emulator you really don't want to recompile the entire system, and in particular not the libraries because that is rather slow with a debug emulator. So, typically you will configure with `-enable-opt=d`, but then you will `cd platform` and invoke `make bootstrap` there. Yet, in order to run the system you need the rest too. This is why it is possible to specify a second build directory: this is supposed to be your usual build directory, not the one configured for debugging. By setting the second build directory, everything not found in the debug build directory is looked in the second build directory.
- You can optionally, state that the debug profile should be started under control of GDB: select item `Run under GDB`, then select e.g. `Yes + Auto Start` from its value menu.

For example, Figure 6.5 shows typical debug settings on my laptop. Note that any profile can be set to run under GDB, but this is rarely useful except when used with a debug emulator.

Figure 6.5 Build Profile



6.2 Profile Parameters

There are many parameters in a profile, but you almost never have to fiddle with them: the very few used in the earlier examples are typically all you'll ever need. They all correspond to parameters also documented elsewhere. However, here is the full list of them:

`Name`

The profile's name

`Type`

Its type: one of `default`, `installed`, or `build`

`OZHOME`

Oz installation directory

`OZLOAD`

Methods used by the resolver to resolve URIs

`OZPATH`

Directories searched for `\insert` directives

`OZ_TRACE_LOAD`

Whether the resolver should output tracing information

OZ_TRACE_MODULE

Whether the module manager should output tracing information

LD_LIBRARY_PATH

Additional directories to search for dynamically linked libraries

DYLD_LIBRARY_PATH

Additional directories to search for MacOS X dylib libraries

OZEMULATOR

Path to `emulator.exe`

OZINIT

Path to `Init.ozf`

PATH

Additional directories to search for executable programs

Change Title

Whether to change the Emacs frame title while Mozart is running

Frame Title

String to use as the Emacs frame title while Mozart is running

Prepend Line

Whether to prepend a `\line` directive to all Oz queries

Default Host

Name of host to use for creating socket connections

Source Directory

Directory of the Mozart sources

Build Directory

Directory in which Mozart was configured and built

Secondary Build Directory

Another such directory

Root Functor

URI of the root functor to load on startup

Run Under GDB

Whether to start the OPI under control of the GNU Debugger. Possible values are one of `No`, `Yes`, and `Yes + Auto Start`

Other Buffer Size

Percentage of frame to use for Oz Compiler/Emulator/Temp window

Popup on Error

Whether to popup the Compiler resp. Emulator buffer upon error

Halt Timeout

Number of seconds to wait for shutdown in `oz-halt`

Compile Command

default shell command to do a compilation This may contain at most one occurrence of '%s', which is replaced by the current buffer's file name

Application Command

default shell command to execute an Oz application This may contain at most one occurrence of '%s', which is replaced by the current buffer's file name, minus the '.oz' or '.ozg' extension.

Engine Program

Default `ozengine` to run the OPI

A

Summary of Oz-Specific Emacs Key Bindings

In this appendix, we present a summary table of all Oz-specific Emacs key bindings. This is intended as a convenient reference; more detailed explanations are given in previous chapters.

`C-.` is the short Oz-specific prefix; `C-. C-.` is the short Oz-specific tool prefix. Both have equivalent long prefixes: `C-c .` and `C-c . C-c .`; these are useful on terminals that cannot generate `C-.` (such as a VT100). The table below documents only the short prefix.

Editing Code	
`M-C-f` (page 9)	forward expression
`M-C-b` (page 9)	backward expression
`M-C-k` (page 9)	kill expression
`M-C-DEL` (page 10)	backward kill expression
`M-C-@` (page 9)	mark expression
`M-C-SPC` (page 9)	mark expression
`M-C-q` (page 8)	indent expression
`M-C-a` (page 10)	beginning of definition
`M-C-e` (page 10)	end of definition
`M-C-t` (page 9)	transpose expressions
`C-x ‘` (page 14)	next error
Managing Buffers	
`M-n` (page 7)	next Oz buffer
`M-p` (page 7)	previous Oz buffer
`C-. n` (page 7)	new Oz buffer
Interacting With a Mozart Sub-Process	
`C-. e` (page 12)	toggle emulator buffer
`C-. c` (page 12)	toggle compiler buffer
`C-. t` (page 12)	toggle temporary buffer
`C-. r` (page 11)	start Mozart sub-process
`C-. h` (page 12)	halt Mozart sub-process
`C-u C-. h` (page 12)	halt Mozart sub-process (forced)
Executing Code	
`C-. C-b` (page 13)	feed buffer
`C-. C-r` (page 13)	feed region
`C-. C-l` (page 13)	feed line
`C-. C-p` (page 13)	feed paragraph
`M-C-x` (page 13)	feed paragraph
Evaluating Expression and Browsing Result	
`C-. b C-b` (page 17)	browse buffer
`C-. b C-r` (page 17)	browse region
`C-. b C-l` (page 17)	browse line
`C-. b C-p` (page 17)	browse paragraph
Evaluating Expression and Showing Result	
`C-. s C-b` (page 13)	show buffer
`C-. s C-r` (page 13)	show region
`C-. s C-l` (page 13)	show line
`C-. s C-p` (page 13)	show paragraph
Interacting With Tools	
`C-. C-. s` (page 17)	open system panel
`C-. C-. c` (page 18)	open compiler panel
`C-. C-. p` (page 18)	start profiler
`C-. C-. d` (page 18)	start debugger
`C-x SPC` (page 18)	set breakpoint on current line

B

Mozart System Development Support

The commands and user options described in this section are probably only interesting for people developing or extending parts of Mozart and thus compiling their own system components. They provide for testing parts of the system locally before installing and for running them under the GNU Debugger gdb.

For completeness and as a reference for the developers themselves, they are described here nevertheless.

B.1 Viewing Emulator Bytecode

The bytecode produced by the compiler can be displayed conveniently in an Emacs buffer. See Section 4.3 for a description of the feedable regions.

command **oz-to-emulatorcode-buffer**
command **oz-to-emulatorcode-region** START END
command **oz-to-emulatorcode-line** COUNT
command **oz-to-emulatorcode-paragraph** COUNT

The corresponding text region is prefixed by

```
\localSwitches
\switch -core +codegen +outputcode -feedtoemulator
```

and fed to the Oz Compiler. If compilation succeeds, the resulting source file will be displayed in the Oz Temporary buffer.

command **ozm-mode**

This is the major mode for displaying (especially fontifying) bytecode. Loading a file with extension `.ozm` automatically puts a buffer into Oz-Machine mode. You can tell a buffer is in Oz-Machine mode by the string `Oz-Machine` in its mode line.

B.2 Testing Locally

One part of the support is concerned with testing system functors locallyetc. This part is now subsumed and replaced by profiles (see Chapter 6).

B.3 Running under gdb

This part is now also subsumed and replaced by profiles (see Chapter 6).

C

Application Programmer's Interface

This section documents some functions that might be useful to users wanting to write their own Oz-syntax-aware commands. All of these commands respect Oz syntax wrt. quoted elements.

function **oz-is-quoted**

Return non-`nil` iff point is inside a string, quoted atom, backquote variable, ampersand-denoted character or end-of-line comment. In this case, move the point to the beginning of the corresponding token. Else point is not moved.

function **oz-backward-keyword**
function **oz-forward-keyword**

Search backward resp. forward for the last resp. next keyword or parenthesis preceding resp. following point. Return non-`nil` iff such was found. Ignore quoted keywords. Point is left at the first character of the keyword.

function **oz-backward-begin**

Move to the last unmatched start of a bracketed Oz construct and return column of point.

function **oz-forward-end**

Move point to the next unmatched **end**.

function **oz-backward-paren**

Move to the last unmatched opening parenthesis and return column of point.

function **oz-forward-paren**

Move to the next unmatched closing parenthesis.

Please submit interesting commands you formulate using these functions to the author.

D

Limitations

Some features of Oz syntax are not handled correctly for purposes of fontification and indentation. These will be described in the following so that you can work around these limitations.

Fontification

- An ampersand as the last character in a string or before a backslash-escaped double quote in a string prevents this double quote from being recognized as a string delimiter. Workaround: Write `[&&]` or `"\&"` instead of `"&"`.
- A backslash character token `&\\` immediately followed by a lowercase letter is misinterpreted as a directive, e.g., in `C == &\\andthen ...` Workaround: Include a space character.
- At maximum fontification level, method names are coloured in `font-lock-funtion-name-face`. If one mistakenly uses a keyword as method name, as in `meth lock() ... end`, then one is not reminded of the fact that this constitutes a syntax error.
- The use of non-escaped double quotes in Gump regular expression tokens written with angle brackets confuses fontification. Workaround: Express the regular expression by a string.

Indentation

- If a keyword is immediately preceded by a number (without space), e.g., `10thread`, the keyword is not recognized as such. This also concerns fontification. Workaround: Write a space.
- Indentation does not know about `/* ... */` style comments, that is, their contents is indented like code and taken into account for computing the following indentation level. Workaround: Only use such comments to comment out properly nested code.
- Indentation does not know about conditional compilation. Workaround: Only use conditionals around properly nested code.

- Line breaks inside strings, quotes or backquote variables are reported as errors when computing the indentation level. Workaround: Write line breaks as `"\n"` and/or use virtual strings with `#` concatenation for multiline strings.

- Indentation is not aware of infix operators, e.g.:

```
feat
   f:
      5 +
   7
```

  The `7` should be underneath the `5`. Workaround: Enclose the expression in parentheses.

- The contents of Gump regular expression tokens in angle bracket notation are not ignored for purposes of indentation. Workaround: Express the regular expression by a string.

Bibliography

[1] Richard M. Stallman. *GNU Emacs Manual*, 7th edition, 1991.

Index