Camelia IDE User Guide

Nathan Gaylinn (Program)
David Baelde (Indentation)
Colin Gordon (Mac OS X port, Testing, Documentation)
Patrick Ramsey (Testing, Documentation)
Matt Cannizzaro (Mac OS X port)
Itay Neeman (Testing, Windows port)
Dave Pacheco (Windows Installer)
Alexandra Heredea (Windows Installer)

October 27, 2005

Contents

1 What is Camelia?

Camelia is an IDE[1] for OCaml. It was developed specifically for CS017 at Brown University, placed under the GPL[2] (so as to comply with the QT[3] License) and put on Sourceforge so that others can continue to expand and improve Camelia. We had previously used an add-on (called Tuareg Mode) for the text editor XEmacs, which, while similarly featured, was very difficult to use[4]. Camelia is modeled after DrScheme.

2 Installing Camelia

2.1 Linux

1. Download camelia.tgz from the website to your home directory.
2. Open a terminal (which should start in your home directory, and type the following commands to unzip, compile, and install Camelia:

```
tar -zxvf camelia.tgz
cd camelia
./config
make
make install
```

If you like, you can use `./config --help` to see more options, so you can specify the place Camelia will initially look for OCaml, a custom installation prefix, etc. None of that is required though, the instructions above will install it with all the defaults, which should usually be fine.

2.2 Windows

1. First, follow the instructions for installing Cygwin, which can be found in the "Documentation" section of the CS017 web site. Make sure you install OCaml.
2. Now download `camelia.msi` from the "Documentation" section of the CS017 web site. Save it to the `cs17setup` folder on your desktop.
3. Open this folder, and double-click on the file you just downloaded.
4. You will see a window appear briefly saying "Preparing to install...", followed by a window with "Welcome to Camelia Setup" in large text. Click "Next".

[1] Integrated Development Environment: This refers to an editor, compiler, and debugging tools all together, or integrated with one another.

[2] GNU Public License; see http://www.gnu.org/licenses/gpl.html

[3] QT is the graphics toolkit that we used to do all the buttons and boxes in Camelia.

[4] You *can* try it yourself, but don't. Really, we developed Camelia for a reason.

5. The screen that appears now gives the option to install Camelia to a different folder than the default. Unless you really have a strong preference that it be put elsewhere, you can leave this alone and click "Next".

6. Now click the button labelled "Install".

7. A progress bar will show up, and indicate how far along the install is. When it is finished, another screen will appear that says "Completing the Camelia Setup Wizard". Click the "Finish" button.

8. Camelia is now installed, but it must be configured a bit.

9. On the desktop, right click "My Computer" and click "Properties."

10. Now click on the "Advanced" tab in the System Properties window.

11. In this section, click the "Environment Variables" button.

12. Now in the window that appears, clik on the "Path" variable in the "System Variables" section, and click "Edit."

13. At the end of the line, add `;C:\cygwin\bin`, so the whole line looks similar to `C:\Program Files;C:\Winnt;C:\Winnt\System32;C:\cygwin\bin`. If you installed Cygwin in a different directory, you should add the Cygwin `bin` directory here after a semicolon. This puts an important file Camelia needs to run in the list of places Camelia will look.

14. Now Camelia is on the Start Menu under Start -> All Programs -> Camelia (OCaml IDE), and you're ready to go![5]

2.3 Mac OS X

1. Download the file "Camelia.dmg" from the website.

2. Double-click on it to mount the disk image. A new Finder window should open with the contents.

3. Double-click on the file "ocaml.pkg" to run the OCaml installer.

4. After reading the introduction, click the "Continue" button.

5. This screen tells you that you need to have X11 and XCode installed. For some functions of OCaml which are rarely used, and not used at all in introductory courses, (compiling a new top level environment) this is necessary. Since we don't need it, go ahead and click "Continue".

[5] If when you open Camelia, the lower of the two larger windows does not include some text saying something like "`Objective Caml version 3.08.3`", Camelia probably does not have the right path to your copy of OCaml. This normally only happens if you installed Cygwin in a directory other than its default, C:\cygwin. If you did, you must configure Camelia to find OCaml by going to `OCaml -> Configure Camelia` from within Camelia.

6. Now you will be given the option to put Camelia on a specific drive. Most likely you will only have one. If not choose the main drive (usually "Macintosh HD") and click "Continue".

7. Now click "Install".

8. A progress bar with captions will run. When it says "The software was successfully installed" you may click the "Close" button.

9. Drag Camelia program to your Applications folder. If you receive a message involving permissions, simply click "Continue". Do **not** run Camelia from inside the disk image.

10. Click the eject icon to the right of the Camelia disk image in the Finder window to unmount the image.

11. If you open your Applications folder, you should see Camelia, which you can now double-click to run.

3 Using Camelia

3.1 Starting Camelia

The way you start Camelia, like installing it, depends on your operating system. For each operating system, there are two ways to start Camelia: one with no file to open, and one with a filename to open immediately.

Operating System	*Basic startup*
Linux	In a terminal window, run the command: `camelia`
Mac OS X	Double click the Camelia icon in your Applications folder.
Windows	Choose Camelia from the All Programs section of the Start Menu.

To immediately open a specific file on Linux, simply add the filename after the command to start it (for example, `camelia myfile.ml`). On Mac, you can drag and drop a file onto Camelia's icon. Windows users can just double-click any `.ml` files and Camelia will start up and open the file. You can also always start Camelia and then use the Open command on the File menu, under any operating system.

3.2 The Basic Interface

Camelia's interface is composed of three parts: the menus and tool bar, the code editor, and the interaction window.

3.2.1 Menus and Toolbar

The menus are fairly straightforward and standard. The toolbar has a few buttons. "Auto Format" goes through your code and automatically indents it. It uses the same auto-indent for OCaml code that the Vim[6] text editor uses. Don't be surprised if it runs through your code several times (you will see it visibly scrolling) before completing — this is normal, as it is rechecking if any changes it makes in one pass have tweaked anything.

"Save and Type Check" does exactly that — it saves your current file, and uses OCaml's type-check mode to locate type errors. When it finds one, a message box will pop up, and it will indicate the error with a red X to the left of that line of code. More on this comes in a later section.

"Save and Run" saves your active file and runs your program in the interaction window.

"Save and Debug" also saves your code. However instead of running, it starts the OCaml debugger (`ocamldebug`) in the interaction window. For more information on debugging with Camelia, see the Debugging section (Section 4.1).

3.2.2 Code Editor

The editor window is *tabbed*. Along the top of the editor area you'll see at least one tab at all times, and if you open additional files, more tabs will appear. Whichever tab is in front represents the active code, which is what you are editing, and this is the code that the toolbar buttons will affect. As you type, the syntax highlighter will work to make your code more readable. If you'd like to indent just the current line, hit the tab key, and Camelia will automatically place that line at the correct indentation from the left based on the lines of code above it (note that it does *not* simply insert a tab character). Line numbers are provided on the left for easy reference. If a file has been changed since it was last loaded or saved, an asterisk will appear at the end of the label for its tab.

3.2.3 Interaction Window

The interaction window has three parts. The status bar, along the top, just under the editor window tells you quite simply what's going on in the interaction window. It will usually display messages such as "Running ocaml with file <file>" or "Running ocamldebug with file <file>". If you edit the code in the editor while you are running or debugging a program, or just after, the bar will turn yellow and say "(OUT OF SYNC)" at the end of the line. This will go away as soon as the code is re-run.

[6] Vim is a command line text editor used by many programmers. It includes editing features for many many programming languages, similar to the features that Camelia provides. Vim however does not have an integrated debugger, and even running your code from Vim directly is a little tricky. And Vim itself is a bit tricky to learn initially.

The next frame is the display window, which displays the actual output of OCaml or the debugger depending on context. When you are debugging, the background is pink.

The final part is the one-line box at the bottom of the whole Camelia window, where you can type directly to OCaml or the debugger. If your program takes keyboard input, that is the box where you must type your input.

3.3 Features

3.3.1 Parentheses Matching

When you type a close-parenthesis, Camelia will indicate the matching opening parenthesis. If you place the cursor just after any parenthesis (either an open- or close-parenthesis), it will turn the background of the section enclosed by that set of parentheses a pale yellow.

3.3.2 Syntax Highlighting

As you type, Camelia changes the typeface of small sections of code to accent different sections such as keywords. If a keyword is not highlighted, maybe you've left a comment open above it.

3.3.3 Typechecking

Pressing the "Save and Typecheck" button runs through your code from top to bottom using OCaml's typechecker to verify types. As things are checked, the area to the left of the checked lines turns blue. If a type error is detected, the area to the left of that line will turn pink and get a red X. Camelia will turn the text of the specific area causing the problem red, and a box will pop up with OCaml's error message. The sometimes odd English grammar comes directly from the error message returned by the OCaml compiler, not from Camelia.[7]

To resize the windows, look between the editor window and the status bar. You'll see a more noticeable visual divider. Mouse over that, press down the mouse button, and without releasing it, move it up and/or down. When it's where you want it to be, release the mouse button. That divider never goes away, so even if you drag it all the way to the top, or all the way to the bottom, so you can't see one of the windows, you can drag the divider away from the edge and the missing window will reappear.

Every line that gets highlighted in blue has type associated with its expressions. Hover your mouse over an expression or right click to see the type of that expression according to OCaml. If you hover your mouse or right click on selected text, Camelia will try to show you the type of the expression you have selected.

[7] We're glad *we* don't have to write this document in French, though.

3.4 Configuration Options

If you choose "Configure Camelia" from the "OCaml" menu, a window pops up with a couple options. Here you can set the path to the OCaml tools, in case installation did not set them correctly or you have multiple versions of OCaml, or if you move your OCaml installation, or for any other reason you might have for changing the command to run.

There is also a font section if you would like to choose a different font or font size for the editor. Only fixed-width fonts will work with Camelia.

4 Debugging with Camelia

4.1 Starting Debugging

When you enter debugging mode by clicking the "Save and Debug" button, two things happen. First, the display/interaction window turns pink, and a list of possible breakpoints will appear in it. The status bar will change to reflect that you are now using `ocamldebug` rather than `ocaml` in that window. Second, a new, free-floating window will appear with the title OCaml Debugger. All of your code will be type checked, and places for setting breakpoints will be marked by pale blue dots.

4.2 What should I know about debugging OCaml before I start?

Because OCaml is a functional language, it makes sense to track the flow of a program's execution by numbering the calls in order. The first call is the actual function call to start the program, and subsequent calls are the ones that are made inside each of those calls. Each of these things is called an *event*. Events in OCaml occur in three places:

1. Inside a function, meaning just after you enter that function's execution.
2. Inside anything with a body, for example inside an `if` statement or a `let` statement, or each case of a `match` statement.
3. After a function call returns.[1]

And at any of those places, you can set a breakpoint to pause execution. If you set a breakpoint at such a place and then start executing, when the execution reaches that point the execution will pause, and you can query the debugger for information about the current state of the program, for information like the values of variables.

[1] The exception to this is inside a tail-recursive function. Because tail-recursive functions are normally collapsed on the stack to save memory and time, in actual execution the tail-recursive call is not really a new function call.

4.2.1 Comparison to GNU Debugger for C (technical discussion)

If you have never programmed before, or never used an actual debugger program before, you can safely skip this small section. Whether you do or not, and particularly if you *have* programmed and/or used a debugger before, this section can provide some useful information.

`ocamldebug` is modelled after `gdb`, but with its program-flow model altered to better suit functional programming. `gdb` sets breakpoints by line number in the source code because that is how execution flows in C— line by line. `ocamldebug` is centered around events, as described in 4.2. This makes sense in a functional language since the program is not necessarily divided by lines.

When using `gdb`, you might set a breakpoint at the function named `foo`. Since in C every function must have a unique name, the debugger can provide this functionality. But functional languages, like Scheme and OCaml, can have anonymous procedures. Stopping at *every* procedure application wouldn't be particularity useful, so the OCaml debugger can't use procedure names as breakpoints, and instead uses events.

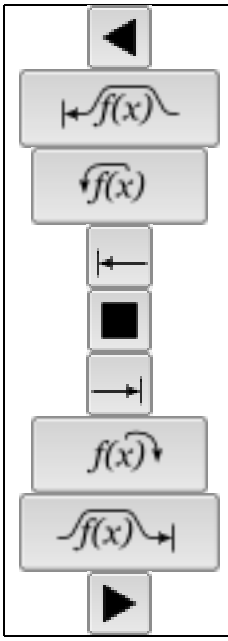
Anyway, breakpoints can only occur at events; however, you still set breakpoints by giving the line number or character offset of a position in a file where one of the three kinds of events will occur.

4.3 OCaml Debugger Control Window

The list on the left labeled "Stack Frames" is a list of stack frames. If you're not sure what that means, don't worry about it, it's not necessary for basic debugging.

You should at least be aware of the list on the right labeled "Observe Events in" however. This lists all the different modules involved in executing your program. Check boxes to the left of each module name allow you to control what events you want to pay attention to. By default, only the check boxes for your own code will be checked; that is enough for what you will be doing most of the time. The others are modules referenced by your code, or modules referenced by modules your code refers to, and so on. Two that you will always see are Pervasives, since that module is loaded for every OCaml program, and Std_exit, which is used to end your program when it is finished. Other commonly loaded modules are Char, List, and String. If your project uses your own custom modules, those will appear as well. In general, you don't care what's happening in modules you didn't write, which is why they are unchecked. If you *do* want to stop at events in some module, though, just check it.

Below those two boxes are buttons that control program execution, and below that is a display of the current event number, a box to choose a new event number, and a "Go!" button. If you enter an event number in that box, and click "Go!" execution will jump to that point. A particularly useful one is event 0, which is the beginning of your program. From left to right, these are the control buttons:



Runs program in reverse, until a breakpoint is encountered.

Steps the program backwards over the preceding procedure.

Steps the program backwards into the preceding procedure.

Steps the program backwards to the previous event.

Stops execution of the program.

Steps the program forward to the next event.

Steps the program forward into the next procedure.

Steps the program forward over the next procedure.

Runs the program until a breakpoint is encountered.

Note that an OCaml program can be run forwards *and* backwards, so when you notice something odd, you can step backwards to find out how it came to be.

4.4 Breakpoints

There are two ways you can set breakpoints for your program.

1. When you enter debug mode, a hollow circle will appear to the left of the line number on every line that has an event. Clicking this circle turns it blue, and sets a breakpoint there.

2. You will also notice that some of the first characters of keywords, and some parentheses turn blue in the code window during debugging. These show you specifically where on the line there are events. You can right-click on or near them, and you'll see in the pop-up menu a list of nearby events which you can click on to set breakpoints.

Removing a breakpoint is as easy as doing either of the above where a breakpoint is already set — only the blue circles will turn hollow instead of the other way around.

4.5 Examining Variables

Once you've reached a breakpoint, you may want to see the values of various variables. Which variables you can examine is limited by the type of event your breakpoint was set on. Basically, you can see the values of any value that has

been set in the function you are in *before* the breakpoint you have hit. At a minimum, you can see the values of global bindings and the parameters to the function you are inside.

To actually see the value of a variable during debugging, you can right-click on it and click "Try to inspect the value of <variable>" on the menu that pops up. You can also type "print <variable>" into the text box under the interaction display window — this is effectively what that menu item does. Another option is to use "display" instead of "print". It shows a bit less information (for example, if you display a list, it does not show the values of the contents of a list) but can be useful for trimming down excessive output.

4.6 Additional Capabilities

In short, anything `ocamldebug` can do, Camelia can do, since the input line at the bottom of the screen is just like typing directly to it in a terminal. Camelia makes some things easier, however, and a typical student will never need to type anything there.

5 Appendix: "Undocumented" Features

Obviously since this section is in the documentation, these features are not *really* undocumented. But this section is intended for capabilities the program has not because they were design decisions, but more for additional flexibilities that result from how the program was designed.

Currently there is only one feature which fits here. Because the programs are run by basically just tacking the filename onto the end of the OCaml command set in the "Configure Camelia" window, additional options can actually be set on that line in the configuration - in fact everything but a filename can be put right there. For certain things, like using the Str module, this unfortunately becomes necessary (which must be linked explicitly on the command line), since #use syntax is technically invalid.