Leksah: An Integrated Development Environment for Haskell

Jürgen Nicklisch-Franken

January 30, 2009

Contents

List of Figures

License

Leksah has been put under the GNU GENERAL PUBLIC LICENSE Version 2. The full license text can be found in the file data/gpl.txt in the distribution.

1 Introduction

Leksah is an IDE (Integrated Development Environment) for the programming language Haskell. It is written in Haskell. Leksah is intended as a practical tool to support the Haskell development process.

Leksah uses GTK+ as GUI Toolkit with the gtk2hs binding. It is platform independent and should run on any platform where GTK+, gtk2hs and GHC can be installed. It uses the Cabal package management and build system for Package Management. It needs the Glasgow Haskell Compiler for full functionality (GHC).

This document is a reference to the functionality you can find in Leksah, it is not intended to be a tutorial. Since Leksah is in the state of development the information may be incomplete or even wrong.

1.1 Further Information

The home page for Leksah is `http://leksah.org/`.The source code for Leksah is hosted under `code.haskell.org/leksah`. For the Programming language Haskell go to `www.haskell.org`. For information about GTK+ go to `www.gtk.org`. You can contact the developer at info (at) leksah.org.

1.2 Release Notes

1.2.1 Version 0.4 Beta Release February 2009

The 0.4 Release will become the first beta when it is stable enough. It should be usable for practical work for the ones that wants to engage with it.

It depends on GHC $\geq$6.10.1 and gtk2hs $\geq$ 0.9.14.

The class browser and the history features are not quite ready, so we propose not to use it yet.

1.2.2 Version 0.1 Alpha Release February 2008

This is a pre-release of Leksah. The editor for Cabal Files is not yet ready, so we propose not to use it yet.

2 Installing Leksah

1. You need an installed GHC (Glasgow Haskell Compiler). For information about GHC go to `www.haskell.org/ghc`. It is a good idea to install everything with sources, specially when using Leksah!

2. Install GTK+ and gtk2hs in a version compatible with the version of GHC you just installed. This should be easy on Linux and it is easy on Windows, if you have a fitting installer (`http://haskell.org/gtk2hs`). The packages from gtk2hs you need are gtk, glib and gtksourceview2.

3. EITHER: Download, and build Leksah with cabal install leksah. (You have to install the package cabal-install before)
   OR: Download, configure, build and install the prerequisite packages: binary ≥0.4.1, bytestring ≥0.9.0.1, utf8-string ≥0.3.1.1, regex-posix ≥0.39.1 which is available from HackageDB `hackage.haskell.org` with typical Cabal procedure. (Go to the root folder of the package. Then do *runhaskell configure, runhaskell build, sudo runhaskell install.* The other packages needed should have been installed with GHC anyway. (I'm nut sure if GHC-extralibs is needed). Then get the leksah package via hackage and do the same.

Inside: Leksah installs a an executable in a folder that should be in the search path, and a couple of data files in a data folder. These places are chosen by the Cabal package management system and depend on the target platform and the way you install. On Linux the data folder may be */usr/share/leksah-0.4/data.* For storing preferences, sessions and collected meta-data Leksah constructs a .leksah directory in your home folder. If you want to change or add configuration files for keymaps, source candy, etc, you can put them in this place.

If you have any trouble installing Cabal please check the Wiki, the mailing list or contact the developers at (info at leksah.org) to find out if it is a Leksah problem.

In the future we would like to have packages/installers for Linux distributions, Windows and Mac. Please contact us if you can offer help.

3 First start of Leksah

1. When you start Leksah for the first time, the first start dialog appears (Figure 1) You have to specify folders, under which Haskell source code for installed packages can be found. This can be any folder above the source directories. So figure out what this will be on your system. You have to click the Add Button after selecting the folder.

> Later you can change this settings in the preferences pane in Leksah and you can rebuild the metadata at any time.

> If you want to see the first start dialog again, delete or rename the .leksah folder in your home folder.

2. Now Leksah collects information about all installed packages on your system. So it may take a long time, but be patient, at further starts it will only collect information for fresh installed packages. Their will eventually be a bunch of errors and warnings on your command line, but don't worry, it only means that Leksah has not succeeded to extract the source locations and comments in a certain file.

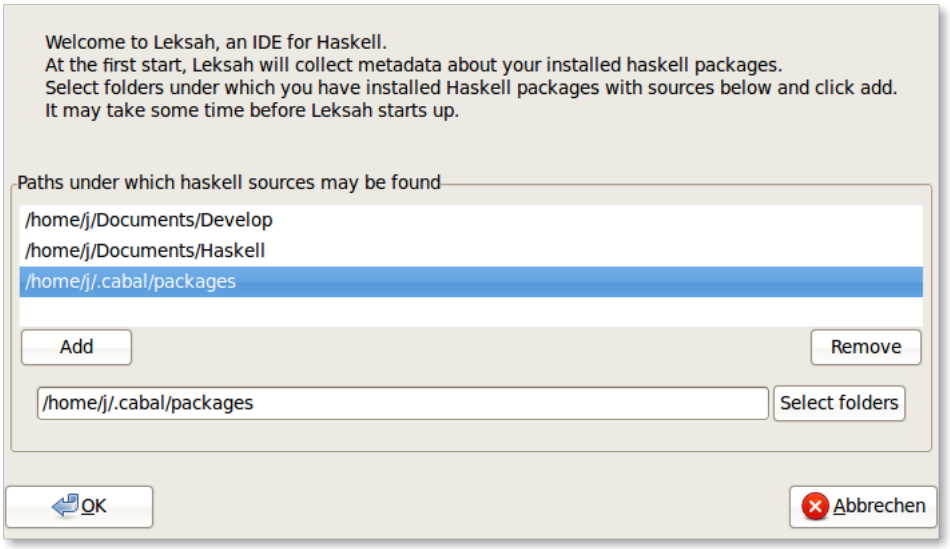

Figure 1: FirstStart dialog

3. After starting up, Leksah will open its Main window in a standard configuration (Figure 2).

4. The best way to start up will be to open an existing project. So select Package/OpenPackage from the menu and open a Cabal file of some project. Alternatively you can construct a new project selecting the Package/NewPackage menu option. A typical Leksah window may then look like Figure 3.

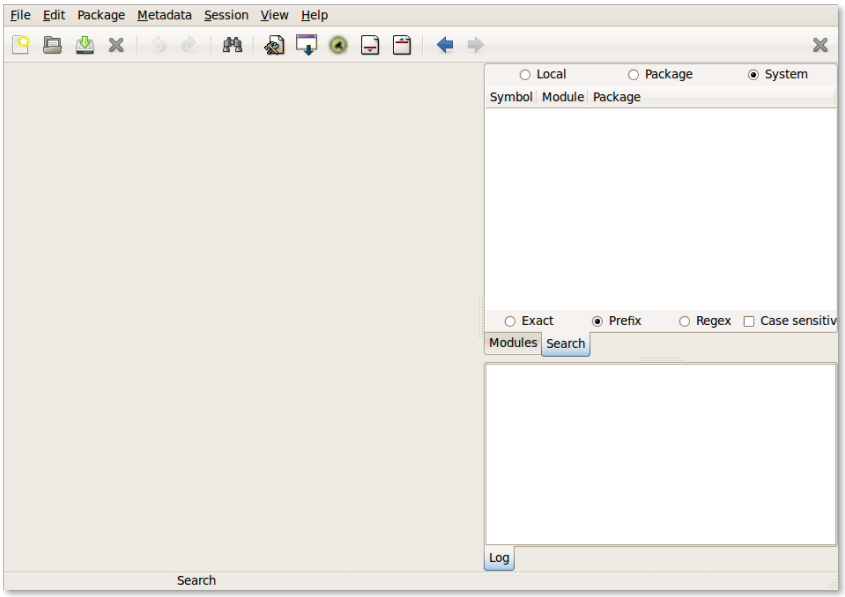

Figure 2: After start

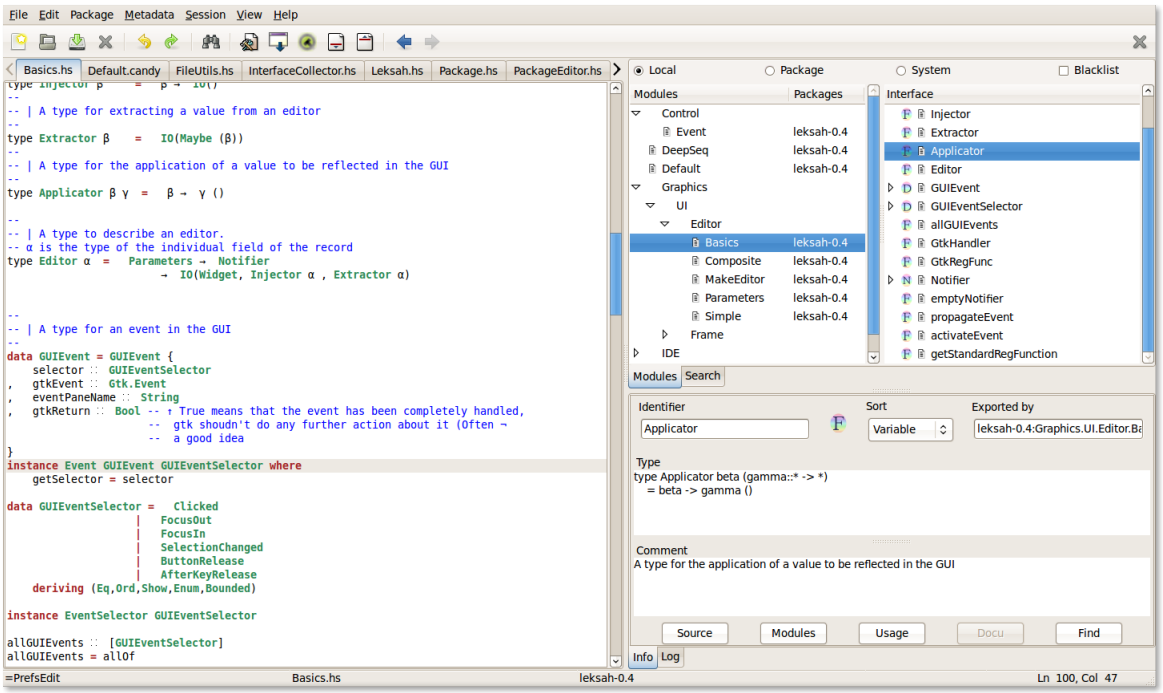

Figure 3: Leksah with open project

4 The Editor

The central functionality needed for development is to edit Haskell source files. Leksah uses the GtkSourceView2 widget for this. It provides syntax highlighting, undo/redo and other features. In the file menu (Figure 4) you have the usual functionality to open, save, close and revert files. You can as well close all files, and all files which are not stored in or below the top folder of the current project (this is the folder where the .cabal file resides). Leksah does not store backup files. Leksah detects if a file has changed which is currently edited and queries the user if a reload is desired. When you open a file which is already open, leksah queries if you want to make the currently open file active, instead of opening it a second time (Leksah currently does not support multiple views on a file, but if you open a file a second time, its like editing the file two times, which makes little sense).

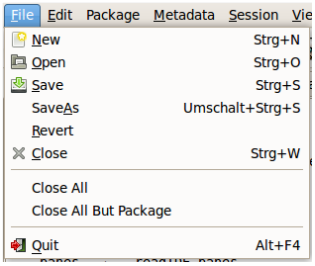

Figure 4: File menu

When a file has changed compared to the stored version, the file name is shown in red in the notebook tab. If you want to change to a different buffer you can open a list of all open buffers by pressing the right mouse button, while the mouse is over a notebook tab. You can then select an entry in this list to select this file.

On the right side in the status bar you can see the line and column, in which the cursor is and if overwrite mode is switched on. In the second compartment from the left you can see the currently active pane, which is helpful if you want to be sure that you have selected the right pane for some operation.

In the edit menu (Figure 5) you find the usual operations: undo, redo, cut, copy, paste and select all. In addition you can comment and un-comment selected lines in a per line style (–). Furthermore you can align some special characters (=,<-,->,::,|) in selected lines. The characters are never moved to the left, but the operation is very simple and takes the rightmost position of the special character in all lines, and inserts spaces before the first occurrence of this special characters in the other lines for alignment.

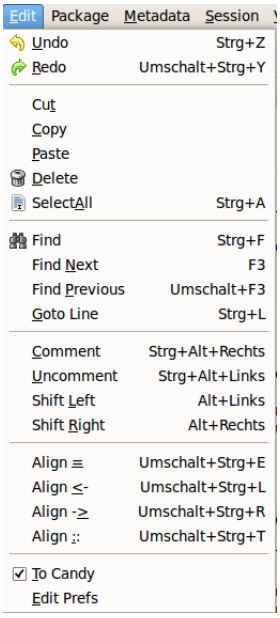

Figure 5: Edit menu

4.1 Search and Replace

Leksah supports basic functionality for searching in text files. When you select Edit/Find from the menu the find bar will open (Figure 6) and you can type in a text string. Hitting the up and down arrow will bring you to the next/previous occurrence of the search string.

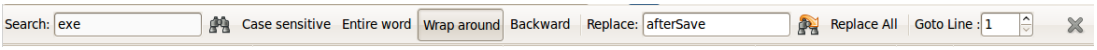

Figure 6: Find bar

Hitting Enter will close the find bar and place the cursor at the currently selected search position. Hitting Escape will close the find bar. You have options for case sensitive search, for searching only whole worlds and for wrapping around, which means that the search will start at the beginning/end of the file, when the end/beginning is reached. If their is no occurrence of the search string the entry turns red.

To replace a text enter the new text in the replace entry and select replace or replace all.

The find bar supports as well to jump to a certain line number in the current text buffer.

4.2 Source Candy

```
-- | Map a function over a list and concatenate the results.
concatMap              :: (a → [b]) → [a] → [b]
concatMap f            = foldr ((⊕) . f) []
```

Figure 7: Source candy example

When using Source Candy, Leksah reads and writes pure ASCII Code files, but can nevertheless show you nice symbols like λ.This is done by replacing certain character combinations by a Unicode character when loading a file or when typing, and replace it back when the file is saved.

The use of the candy feature can be switched on and off in the menu and the preferences dialog.

This feature can be configured by editing a .candy file in the .leksah folder or in the data folder. The name of the candy file to be used can be specified in the Preferences dialog.

Lines in the *.candy file looks like:

```
"\" 0x03bb --GREEK SMALL LETTER LAMBDA
"->" 0x2192 Trimming --RIGHTWARDS ARROW
```

The first entry in a line are the characters to replace. The second entry is the hexadecimal representation of the Unicode character to replace with. The third entry is an optional argument, which specifies, that the replacement should add and remove blanks to keep the number of characters. This is important because of the layout feature of Haskell. The last entry in the line is an optional comment, which is by convention the name of the Unicode character.

Using the source candy feature can give you problems with layout, because the alignment of characters with and without source candy may differ!

Leksah reads and writes files encoded in UTF-8. So you can edit Unicode Haskell source files. When you want to do this, switch of source candy, because otherwise Unicode characters may be converted to ASCII when saving the file.

4.3 Editor Preferences

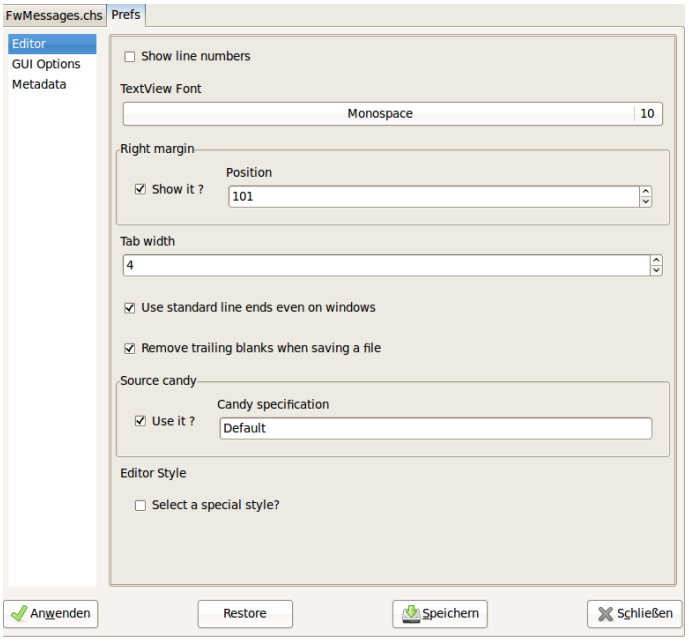

Figure 8: Editor Preferences

When selecting Edit/Edit Prefs the preferences pane opens, which has a selection called Editor (Figure 8), were you can edit preferences for the editor. Some of the options you find here refer to visual elements, like the display of line numbers, the font used, the display of a right margin and the use of a style file for colors and syntax highlighting.

You can set here the Tab size you want. Leksah always stores tabs as spaces to ease the use of layout. (As you may know, otherwise only a tab size of 8 can be digested by Haskell compilers).

Leksah has an option for storing the files with standard UNIX line ends even on Windows, and not using the infamous Cr/Lf combination. This is e.g. useful if Windows and other users commit to the same repository.

Leksah offers as well to remove trailing blanks in lines, which you may choose as default, because blanks at the end of lines make no sense in source code.

5 Packages (Cabal)

A central concept for any IDE is a package, which is a project for development of some library or executable you may work on. One instance of Leksah can only open one package at a time. Leksah can store configurations for packages separately (and does this by default), so that you can switch between packages and get exactly back to where you stopped when opening a different package.

Leksah uses Cabal for package management, and opening a package is done by opening a cabal file. So when you select Package / Open Package from the menu, select the cabal file of the desired package. Leksah shows the current package in the third compartment in the status bar!

To start with a new package select Package / NewPackage from the menu. Then you have to select a folder for the project, which you may give the same name you will give to your package. Then the package editor will open up, in which you have to supply information about your package.

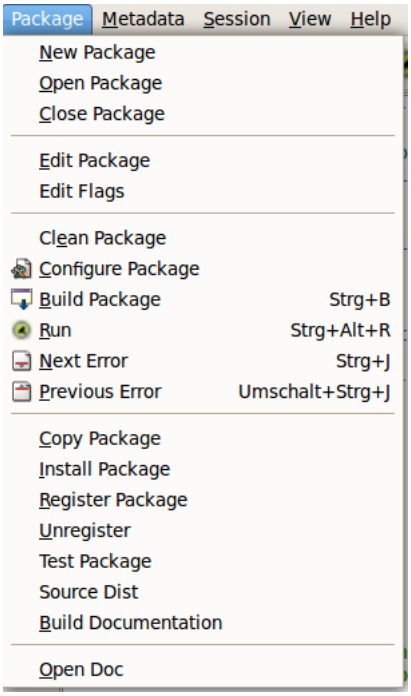

Figure 9: Package Menu

5.1 Package Editor

The package editor (Figure 10) is an editor for cabal files. Since cabal files offer complex options the editor is quite complex. For a complete description of all options see the Cabal User's Guide. The package editor does not support the cabal configurations feature. If you need cabal configurations, you need to edit the cabal files as a text file. Since Leksah uses standard cabal files with no modifications this is no problem, and you can use Leksah with such packages with no problem, just the package editor will not work for you.

The minimum requirements for any package is to give a name and a version. Then you will have to enter dependencies on other packages in the Dependencies part of the editor. This will be at least the base package.

Finally you have to specify an executable or a library that should be the result of your coding effort. You do this in the Executables and Library part of the editor. Cabal gives the possibility to build more then one executable from one package and to build a library and executables from one package.

For an executable you enter a name, the source file with the main function and a build info. For a library you enter the exposed modules and a build info.

With build information you give additional information, e.g:

- where the sources can be found (relative to the root folder of the project, which is the one with the cabal file).
- what additional non-exposed or non main modules your project includes

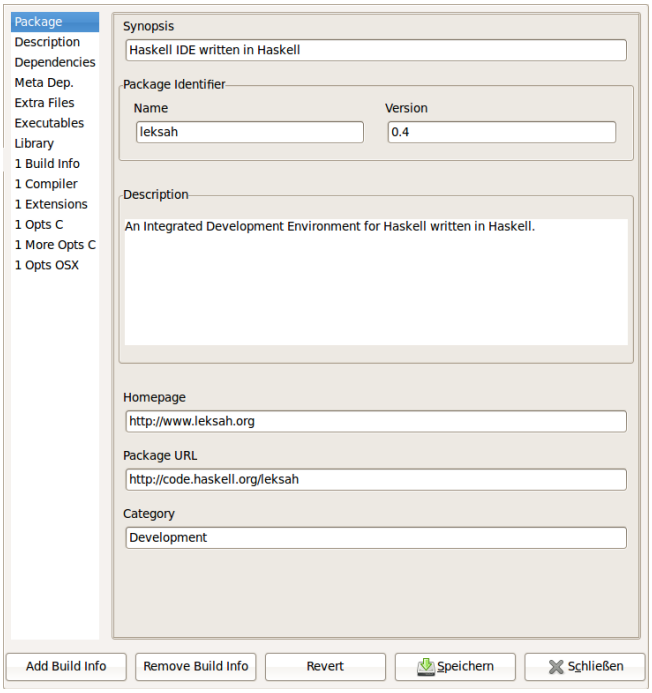

Figure 10: PackageEditor 1

- compiler flags
- used language extensions in addition to Haskell 98 (These can also be specified in the source files with pragmas)
- and many more ...

Because more then one executable and a library can be build from one package, it is possible to have cabal files with more then one build info. The package editor deals with this by the buttons Add / Remove Build Info. Every build info gets an index number, and for executables and a library you specify the index of the build info. (However, the usual case is to start with one build info).

5.2 Building

The most frequently used functionality with packages is to make a build, which is possible after a successful configure. When you start a build, the log window will be opened or displayed. In the Log window you can see the standard output the Cabal build produces, which comes from the GHC compiler.

A build may produce errors and warnings. If this is the case the focus is set to the first error/warning in the Log and the corresponding source file will open with the focus at the point where the compiler reports the error. You can navigate to the next or previous

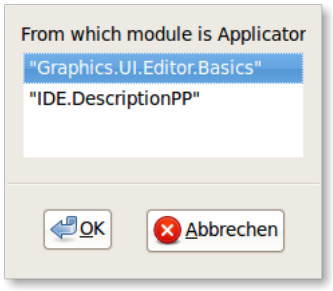

Figure 11: Import dialog

errors by clicking on the error or warning in the log window, or by using the menu, the toolbar or a keystroke.

In the statusbar the state regarding to the build is displayed in the third compartment from the right. It reads *Building* as long as a build is on the way and displays the numbers of errors and warnings after a build.

> Currently their is no way to cancel a build in progress, but this is on the list.

5.3 Import Helper

A frequent and annoying error is the *Not in scope* compiler error. In the majority of cases it means that an import statement is missing. If this is the case you can choose *Add import* from the context menu in the log pane. Leksah will then add an import statement to the import list. If their is more then one module the identifier can be imported from, a dialog will appear which queries you about the module you want to import from (Figure 11).

Leksah then adds a line or an entry to the import list of the affected module with the compiler error and adds a line in the Log window. Leksah imports individual elements, but imports all elements of a class or data structure if one of them is needed. The import helper can work with qualified identifiers and should add a correct import statement. You can as well select *add all imports* from the context menu, in which case all *Not in scope* errors will be treated at once. After providing the imports you have to save the file and recompile.

The import helper just looks in imported packages, so if you miss a package import it will not be treated automatically. If you find that an identifier is not exported by another module and you add it their and then run the import helper again, it will still not find the identifier, because the meta information the import mechanism depends on was not updated. So choose Metadata / Update project and choose add import again and it should work.

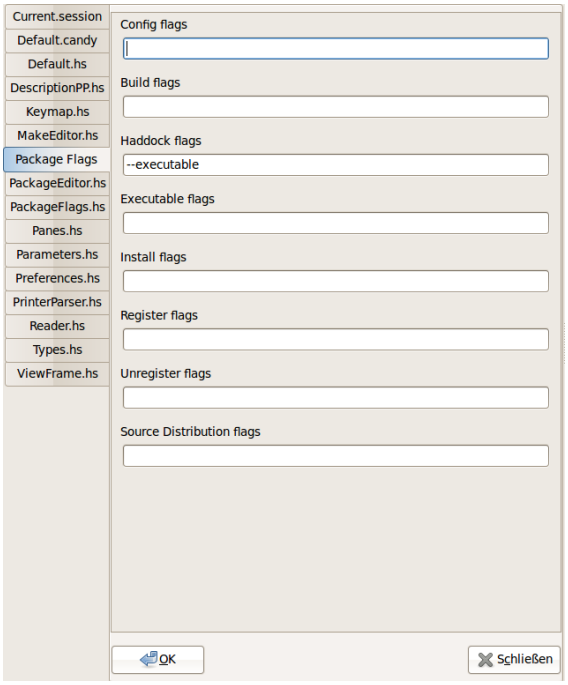

Figure 12: Package Flags

> This is a fresh feature which may still have some problems, please report them so that we can work on them.

Obviously some not in scope errors have other reasons, e.g. you have misspelled some identifier, which can't be resolved by adding imports.

5.4 Flags and other operations

As you can see in the package menu (Figure 9) you can do more operations with packages, which are mostly provided by the Cabal system. You can clean, configure, build and if you have build an executable run your program. And other operations like building a source distribution and building haddock documentation. For more details about these operations (as said before) consult the Cabal User's Guide. Since many of these operations can take additional flags you can enter these by selecting Package / Edit flags. Then the Flags pane opens up (Figure 12). For example haddock documentation for the leksah source will not be build, because it is not a library unless you pass the –executable flag. The flags are stored in a file called IDE.flags in the root folder of the project.

6 Navigation and Metadata

Leksah collects data about all installed Haskell packages on your system. It does this by reading the Haskell interface files which GHC writes. In addition it adds source positions and comments of packages for which a cabal file with the corresponding source files can be found. The package you work on is treated differently, as not only external exported entities are collected, but all exports from all modules are collected. This makes it possible to get information about identifiers:

- Which packages and modules export this identifier?
- What is the type of the exported identifier?
- If the source is found: What is the comment for this identifier?
- If the source is found: What is the implementation?

If you like to get information about some identifier in the code, the easiest way is to press Ctrl and double click on it. If the id is known unambiguously the modules and info pane will show information about it. If more then one possibility exist the search pane will open and present the alternatives.

More precisely the operation is not triggered by the double click operation, but by the release of the left button. So if the double click does not select the right area for a special id like ++ you can select the desired characters with the left button and then release it while you hold down the Ctrl key.

> Currently Leksah only uses the collected "global" metadata, and does not know what the Haskell compiler knows about your code. So definitions which are local to a module will not be found, types of variables which are not exported will not be known, and Leksah does not known about which definition is the one you are looking for, because it is the only imported one.
>
> We will work on adding this information in the future, but we started with the "global" approach from the intuition, that it takes most of our time to find something that is not already imported and "known". A local definition can be easily find by a text search. Please try it out on your own.

6.1 The Modules Pane

In the modules pane (Figure 13) you get information about modules and their interface. The displayed information depends on the open package. If no package is open only the system scope has information. (If a package is open its name is displayed in the third subdivision from the left of the status bar.)

We assume there is an open package. You can then select the scope of the displayed information with the radio button on top of the modules pane. The *Local* scope shows only modules which are part of the project. The *Package* scope shows all modules of the

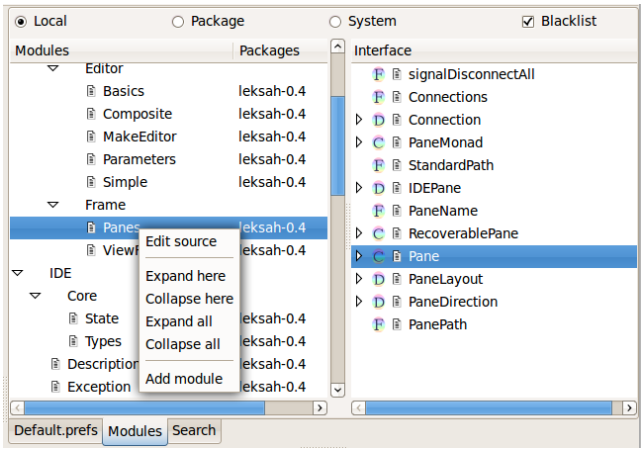

Figure 13: Modules pane

package and all packages the current package depends on. The *System* scope shows all modules of installed packages of the system. (You can get this list with *ghc-pkg list*).

If the Blacklist toggle button is selected, the packages in the blacklist are not displayed. This doesn't mean that the information of this packages is not loaded or otherwise accessible. (I invented the blacklist mainly for the GHC package, which is very big and does not use name-spaces and so pollutes the list). The Blacklist can be edited in the preferences dialog.

If you select a module in the modules list, its interface is displayed in the interface list on the right. You can search for a module or package by selecting the modules list and typing some text. With the up and down arrows you find the next/previous matching item. With the escape key or by selecting any other GUI element you leave the search mode. If there is a little icon with a text in front of a module, Leksah has found a source file for this module. You can open this source file, or bring it to the front if it is already open with a *double click* on the module. (the same can be done with selecting *Edit source* from the context menu.

By selecting an element in the Interface List the so called Info Pane is shown with additional information. If there is a little with a text in front of an identifier, Leksah has found a source location for this element. You can open this source file, or bring it to the front and display the source for the selected location with a *double click* on the element. (the same can be done with selecting *Go to definition* from the context menu. You can again search for an identifier by selecting the interface list and typing some text.

The easiest way if you want to edit some file is not to choose File open, but to select the modules pane with local scope, find the module by entering text, and double click for editing the file.

The easiest way to add a new module is by selecting *Add module* from the context menu of the modules pane. The Construct Module dialog will open (Figure 14). You have to enter the name of the module, the source path to use if their are alternatives and if the module is exposed, if it is a library. Leksah will construct the directory, modify

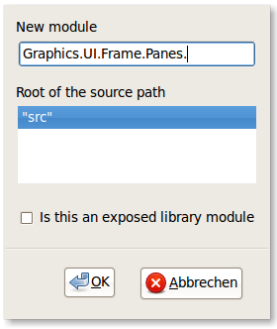

Figure 14: Construct module dialog

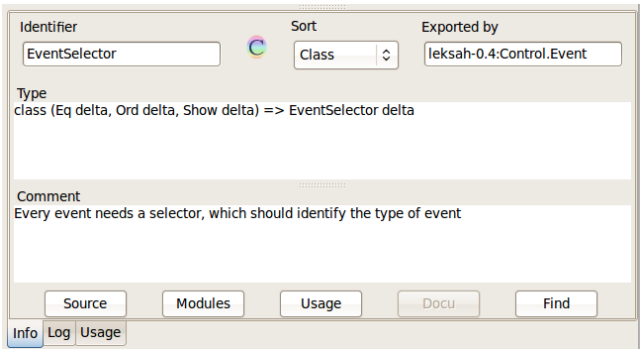

Figure 15: Info pane

the cabal file and construct an empty module file from a template.

6.2 The Info Pane

The Info Pane (Figure 15) shows information about an interface element, which may be a function, a class, a data definition .... It shows the identifier, of which sort it is, the package and module that it is exported by, it's Haskell type and if possible a comment.

If you select an identifier in an editor, and their is information about this identifier available in the package scope, it is automatically displayed in the info pane. The easiest way to do this is to double click on an identifier while pressing Ctrl. For special identifiers (e.g. a_) select the word and release the button, actually the search is initiated by the release of the button.

Remember that only statically collected information is available this way. So the meta data contains only information about items which are exported by some module.

If there is a source locating attached you can go to the definition by clicking the *Source* button.

You can select the module and the interface element in the modules pane by clicking the *Modules* button.

With the *Usage* button a pane opens which displays the modules which imports this

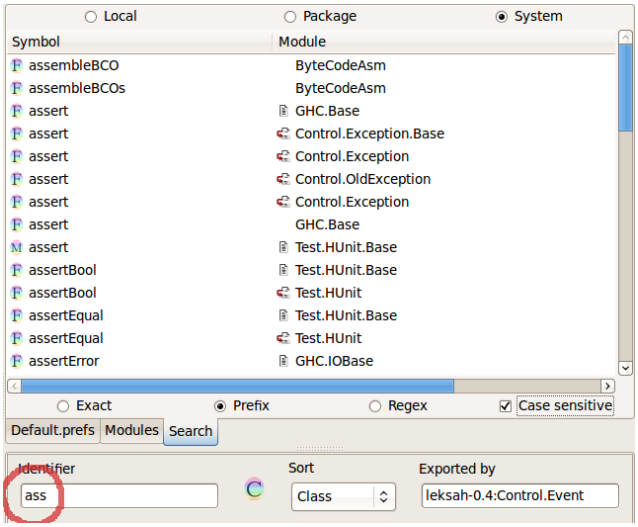

Figure 16: Search pane

element.

You can search for elements by typing text in the Identifier field. For details see the read the next section called Search Pane.

6.3 The Search Pane

You can search for a string by typing in characters in the Identifier field of the Info Pane. If the field contains less then 3 characters only exact matches are found. If more characters are given the search result depends on the settings in the search pane (Figure 16). You can choose:

1. The scope in which to search, which can be local, package or system.
2. The way the search is executed, which can be exact, prefix or as a regular expression.
3. You can choose if the search shall be case sensitive or not.

The result of the search is displayed in the list part of the Search pane. You can see the sort of expression by the icon before the identifier. You can see if the module reexports the identifier, or if the source of the identifier is reachable. When you single click on a search result, the info pane shows the corresponding information. If you double click on an entry, the modules and info pane shows the corresponding information.

If you double click on an identifier and press Ctrl in a source buffer, it is a case sensitive and exact search in the package scope. So this does not depend on the selection in the search pane, even if the result is displayed in the list box of the search pane.

In the info pane you can click the button usage, if you want to explore which modules import the selected element. This information is extracted from the Haskell Interface file.

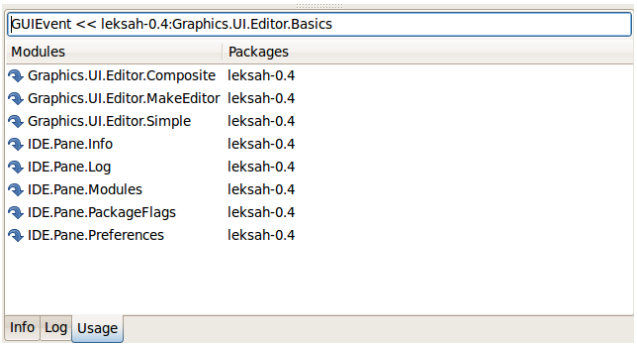

Figure 17: Usage Pane

6.4 The Usage Pane

As said in the end of the last section, this pane shows which modules import a certain element. The element is displayed in the top, and the modules which import it are displayed in the list box. If you double click on an entry in the list box, the corresponding source will be opened if possible. Then leksah tries a text search on the selected element.

6.5 Metadata collection

Metadata collection depends on the configuration and can be manually triggered.

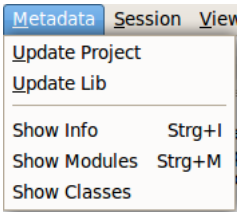

Figure 18: File menu

If you select Metadata / Update Project the metadata for the current project is collected from the .hi files and the source files. You should select this if the metadata of the current project is out of sync.

If you select Metadata / Update Lib Leksah checks if a new library has installed and if this is the case collects metadata for it.

The metadata is stored in a folder under the .leksah folder under your home folder. The folder will be named after the compiler version (e.g. ghc-6.8.1). In this folder collected information about installed packages for a compiler version is stored. (e.g. binary-0.4.1.pack). These files are in binary format. You can rebuild the whole metadata when you start Leksah with the -sr option (–Sources –Rebuild).

In the Metadata part of the preferences (Figure 9) you can edit the settings concerning metadata collection.

7 Configuration

Leksah is highly customizable. Here it is explained how this works.

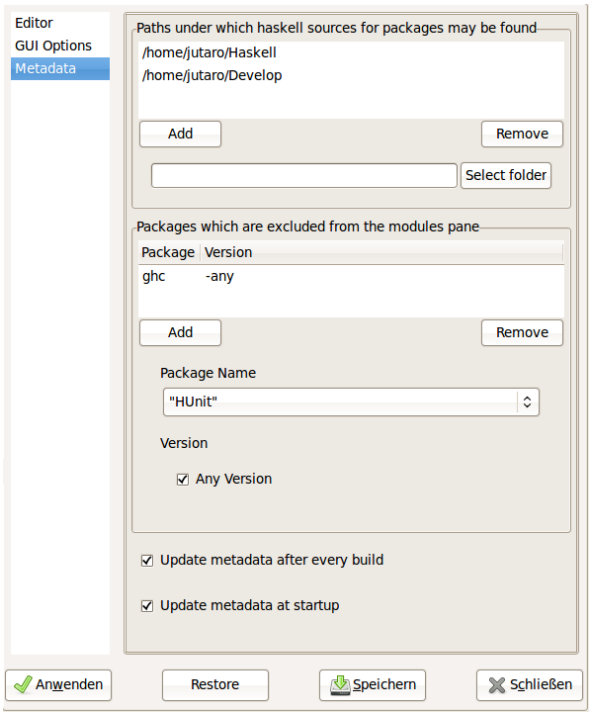

Figure 19: Metadata Preferences

7.1 Window Layout

In Leksah there may be an active pane. The name of this pane is displayed in the second compartment from the left side in the status bar. Some actions like moving, splitting, closing panes or finding or replacing items in a text buffer act on the current pane, so check the display in the status bar to see if the pane you want to act on is really active.

The layout of the Leksah window contains areas which contain notebooks which contain so called panes. The division between the two areas is adjustable by the user by dragging a handle. The areas form a binary tree, although this tree is not visible to the user. Every area can be split horizontally or vertically. Panes can collapsed, the effect of collapsing depends on the position of the pane in the binary layout tree.

Active panes can be moved between areas in the window. The tabs of notebooks can be positioned at any of the four directions, or the tabs can be switched off. Note that holding the mouse over the tabs and selecting the right button brings up a menu of all panes in this area, so that you can for

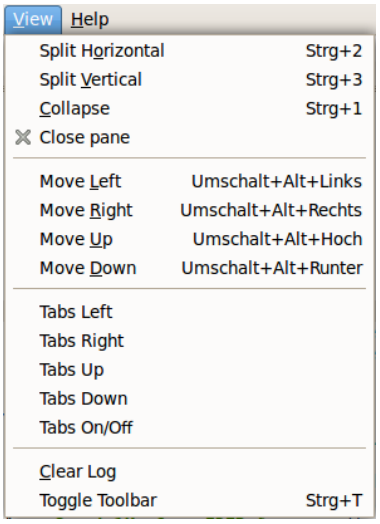

Figure 20: View menu

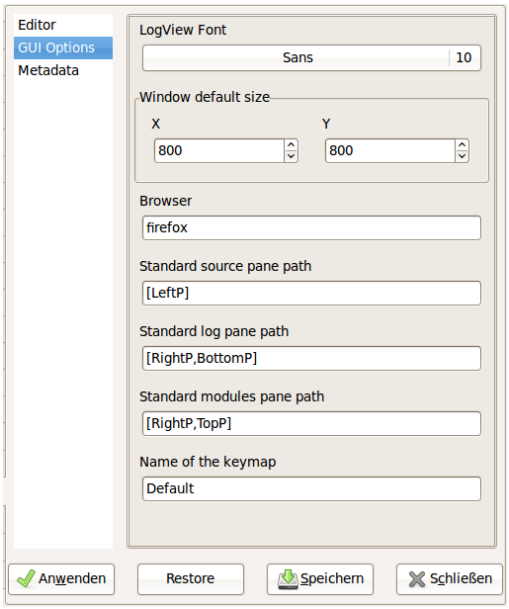

Figure 21: GUI Preferences

example quickly select one of many open source buffers.

The layout will be saved with sessions. The session mechanism will be explained in the next section. Currently there is no way to load different layouts independent of the other data stored in a sessions.

In the GUI Options part of the Preferences (Figure 21), you can configure options regarding the layout, namely were windows of certain types are opened.

7.2 Session handling

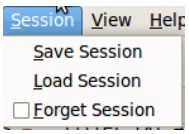

Figure 22: Session menu

When you close Leksah the current state is saved in the file Current.session in the ~/.leksah folder. A session contains the layout of the window, its population, the active package and some other state. When you restart Leksah it recovers the state from this information. When you close a package, the session is saved in the project folder in the file IDE.session. When you open a project and Leksah finds a IDE.session file in the folder of the project you are going to open, you get prompted if you want to open this session. This should help you to switch between different packages you are working on.

Beside of this you have the possibility to store and load named sessions manually by using the session menu. Actually you may live well without using this feature.

You can as well choose to mark Forget Session, if you don't want the current session to be stored. This can be useful, if something goes wrong (e.g. you hit accidentally Ctrl - 0 and the layout collapses completely).

7.3 Shortcuts

You can configure the keystrokes by providing a .keymap file, which can either be in the .leksah folder or in the data folder. The name of the key map file to be used can be specified in the Preferences dialog. A line in the .keymap file looks like:

<ctrl>o -> FileOpen "Opens an existing file"

Allowed Modifiers are <shift> <ctrl> <alt> <apple> <compose>. <apple> is the Windows key on PC keyboards. <compose> is often labeled Alt Gr. It is as well possible to specify Emacs like keystrokes in the following way:

<ctrl>x/<ctrl>f -> FileOpen "Opens an existing file"

The comment on the right will be displayed as tool tips on top of toolbar buttons, if such exist for this action.

The name of the action can be any one of the *ActionDescr's* given in the *action* function in the Module *IDE.Menu*.

Whenever you call an action, by a menu, a toolbar or a keystroke, the keystroke with its associated ActionsString is displayed in the Status bar in the leftmost compartment.

Every keystroke shall obviously only be associated with one action, and more important every action may only have one associated keystroke.

Simple keystrokes are shown in the menu, but Emacs like keystrokes are not. This is because simple keystrokes are delegated to the standard gtk mechanism, while other keystrokes are handled by Leksah.

7.4 Configuration files

Leksah stores its configuration in a directory called ~/.leksah under your home folder.

The file Default.prefs stores the general Preferences. These Preferences can be edited in a dialog by choosing Help/Edit Prefs from the menu. If this file is not available the Default.prefs file from the installed /data folder will be used.

The Current.session file stores the state of the last session, so that Leksah will recover the state from the last session. If this file is not available it will be taken from the installed /data folder.

The source_packages.txt file stores source locations for installed packages. It can be rebuild by calling Leksah with the -s or –Sources argument . Do this after you moved your source or added sources for previous installed packages without sources.

The folder will contain one or many other folders (e.g. ghc-6.8.1). In this folder collected information about installed packages for a compiler version is stored. (e.g. binary-0.4.1.pack). These files are in binary format. If you start Leksah with the -r or –Rebuild argument, it cleans all .pack files and rebuilds everything.

Files for Keymaps and SourceCandy may be stored here and will be found according to the name selected in the Preferences Dialog. Leksah first searches in this folder and after this in the /data folder.

7.5 Menus and Toolbars

Menus and Toolbars can be customized by editing the file Default. menu. The format is a gtk+ xml format. Leksah requires the definition of one menu bar and one toolbars in this order. The names of the actions can be all in the *ActionDescr's* given in the *action* function in the Module *IDE.Menu*.

8 The Future

The development of an IDE is a big issue, so Leksah is intended to become a community project and everyone is invited to contribute. If you are a user or just test Leksah, we would appreciate to here from you and your problems with and wishes for Leksah.

I personally plan to develop up to version 1.0. So, if the community does not show enough interest, or some better alternative may appear, the features marked as Version x may never be implemented.

8.1 Version 0.6

- Completion
- Working Class pane
- Properties of single files (no highlight, no candy,...)
- Cancel build - Build only one time
- Source collector only one time
- Working history navigation

8.2 Version 1.0

- Interpreter
- Debugger

8.3 Version x

- Versioning support (Darcs,...)
- Test support (Quick check,...)
- Coverage (HPC,...)
- Profiling (Ghc Profiler,...)
- Refactoring (HaRe,...)
- FAD (Functional Analysis and Design,...)
- Plugins

9 Appendix

9.1 Command line arguments

```
Usage: leksah [OPTION...] files...
-r --Rebuild Cleans all .pack files and rebuild everything
-c --Collect Collects new information in .pack files
-u FILE --Uninstalled=FILE Gather info about an uninstalled package
-s --Sources Gather info about pathes to sources
-v --Version Show the version number of ide
-d --Debug Write ascii pack files
-l NAME --LoadSession=NAME Load session
-n --NoGUI Don't start the leksah GUI
```

9.2 The Candy file

```
-- Candy file
"->" 0x2192 Trimming --RIGHTWARDS ARROW
"<-" 0x2190 Trimming --LEFTWARDS ARROW
"=>" 0x21d2 --RIGHTWARDS DOUBLE ARROW
">=" 0x2265 --GREATER-THAN OR EQUAL TO
"<=" 0x2264 --LESS-THAN OR EQUAL TO
"/=" 0x2260 --NOT EQUAL TO
"&&" 0x2227 --LOGICAL AND
"||" 0x2228 --LOGICAL OR
"++" 0x2295 --CIRCLED PLUS
--"::" 0x2551 Trimming --BAR
"::" 0x2237 Trimming --PROPORTION
".." 0x2025 --TWO DOT LEADER
"^" 0x2191 --UPWARDS ARROW
"==" 0x2261 --IDENTICAL TO
" . " 0x2218 --RING OPERATOR
"\" 0x03bb --GREEK SMALL LETTER LAMBDA
--"=<<" 0x291e --
">>=" 0x21a0
"$" 0x25ca
">>" 0x226b -- MUCH GREATER THEN
"forall" 0x2200 --FOR ALL
"exist" 0x2203 --THERE EXISTS
"not" 0x00ac --NOT SIGN
"alpha" 0x03b1
"beta" 0x03b2
"gamma" 0x03b3
"delta" 0x03b4
"epsilon" 0x03b5
```

9.3 The Keymap file

```
--Default Keymap file for Leksah
--Allowed Modifiers are <shift> <ctrl> <alt> <apple> <compose>
--<apple> is the Windows key on PC keyboards
--<compose> is often labelled Alt Gr.
--File
<ctrl>n -> FileNew "Opens a new empty buffer"
<ctrl>o -> FileOpen "Opens an existing file"
--<ctrl>x/<ctrl>f -> FileOpen "Opens an existing file"
<ctrl>s -> FileSave "Saves the current buffer"
--<ctrl>x/<ctrl>s -> FileSave "Saves the current buffer"
<ctrl><shift>s -> FileSaveAs "Saves the current buffer as a new file"
--<ctrl>x/<ctrl>w -> FileSaveAs "Saves the current buffer as a new file"
<ctrl>w -> FileClose "Closes the current buffer"
--<ctrl>x/k -> FileClose "Closes the current buffer"
<alt>F4 -> Quit "Quits this program"
--<ctrl>x/<ctrl>c -> Quit "Quits this program"
--Edit
<ctrl>z -> EditUndo "Undos the last user action"
--<ctrl>x/u -> EditUndo "Undos the last user action"
<shift><ctrl>y -> EditRedo "Redos the last user action"
--<ctrl>x/r -> EditRedo "Redos the last user action"
--<ctrl>x -> EditCut
--<ctrl>c -> EditCopy
--<ctrl>v -> EditPaste
-> EditDelete
<ctrl>a -> EditSelectAll "Select the whole text in the current buffer"
<ctrl>f -> EditFind "Search for a text string (Toggles the "
F3 -> EditFindNext "Find the next occurence of the text string"
<shift>F3 -> EditFindPrevious "Find the previous occurence of the text string"
<ctrl>l -> EditGotoLine "Go to line with a known index"
<ctrl><alt>Right -> EditComment "Add a line style comment to the selected lies"
<ctrl><alt>Left -> EditUncomment "Remove a line style comment"
<alt>Right -> EditShiftRight "Shift right"
<alt>Left -> EditShiftLeft "Shift Left"
--View
<alt><shift>Left -> ViewMoveLeft "Move the current pane left"
<alt><shift>Right -> ViewMoveRight "Move the current pane right"
<alt><shift>Up -> ViewMoveUp "Move the current pane up"
<alt><shift>Down -> ViewMoveDown "Move the current pane down"
<ctrl>2 -> ViewSplitHorizontal
"Split the current pane in horizontal direction"
<ctrl>3 -> ViewSplitVertical
```

```
"Split the current pane in vertical direction"
<ctrl>1 -> ViewCollapse "Collapse the panes around the currentla selected pane into o
-> ViewTabsLeft "Shows the tabs of the current notebook on the left"
-> ViewTabsRight "Shows the tabs of the current notebook on the right"
-> ViewTabsUp "Shows the tabs of the current notebook on the top"
-> ViewTabsDown "Shows the tabs of the current notebook on the bottom"
-> ViewSwitchTabs "Switches if tabs for the current notebook are visible"
<ctrl>t -> ToggleToolbar
-> HelpDebug
-> HelpAbout
<ctrl>b -> BuildPackage
<ctrl><alt>r -> RunPackage
<ctrl>j -> NextError
<ctrl><shift>j -> PreviousError
<ctrl>m -> ShowModules
<ctrl>i -> ShowInterface
<ctrl>i -> ShowInfo
<ctrl><shift>e -> EditAlignEqual
<ctrl><shift>l -> EditAlignLeftArrow
<ctrl><shift>r -> EditAlignRightArrow
<ctrl><shift>t -> EditAlignTypeSig
-- <alt>i -> AddOneImport
-- <alt><shift>i -> AddAllImports
```

9.4 Preferences file

```
Show line numbers:
               True
     --(True/False)
TextView Font: "Monospace 10"
Right margin:  101
     --Size or 0 for no right margin
Tab width:     4
Use standard line ends even on windows:
               True
Remove trailing blanks when saving a file:
               True
Source candy:  Default
     --Empty for do not use or the name of a candy file in a config dir
Name of the keymap:
               Default
     --The name of a keymap file in a config dir
Editor Style:  ""
LogView Font:  "Sans 10"
```

```
Window default size:
               (800,800)
     --Default size of the main ide window specified as pair (int,int)
Browser:       "firefox"
Standard source pane path:
               [LeftP]
Standard log pane path:
               [RightP,BottomP]
Standard modules pane path:
               [RightP,TopP]
Paths under which haskell sources for packages may be found:
               []
Packages which are excluded from the modules pane:
               [Dependency (PackageName "ghc") AnyVersion]
Update metadata after every build:
               True
Update metadata at startup:
               True
```

9.5 Session File

This file is only displayed partial to give you an idea of what it may contain

```
Time of storage:
               "Mon Jan 19 10:35:04 CET 2009"
Layout:        VerticalP (TerminalP (Just TopP) 0) ...
Population:    [(Just (BufferSt (BufferState ...
Window size:   (1440,850)
Active package:
               Just "/home/j/Documents/Develop/leksah/leksah.cabal"
Active pane:   Just "Log"
Toolbar visible:
               True
FindbarState:  (False,FindState ...
```