

Leksah: An Integrated Development Environment for Haskell

Jürgen Nicklisch-Franken

Hamish Mackenzie

edited for v. 0.8: Andrew U. Frank and Christian Gruber

March 27, 2010

Contents

1	Introduction	4
1.1	Further Information	4
1.2	Release Notes	5
1.2.1	Version 0.8 Release March 2010	5
1.2.2	Version 0.6 Beta Release Juli 2009	5
1.2.3	Version 0.4 Beta Release February/March 2009	6
1.2.4	Version 0.1 Alpha Release February 2008	6
2	Installing Leksah	7
2.1	How to Install: Brief Instructions	7
2.2	Microsoft Windows	7
2.3	Mac OS X	7
2.4	Linux from Distro Packages	7
2.5	Install from Hackage	8
2.6	Post Installation steps	8
2.7	First start of Leksah	8
2.8	First start dialog	9
3	Hello World example	12
4	The Editor	14
4.1	Find and Replace in the current folder	15
4.1.1	Search in the package: Grep	15
4.2	Source Candy	15
4.3	Completion	17
4.4	Using the Flipper to Switch Between Editors	18

4.5	Change Your Preferences for the Editor	18
4.6	Further info	19
5	Working with Projects: Workspaces and Packages	20
5.1	Cross package build	21
5.2	File Organization with Workspaces	21
5.3	Workspace Operations	21
5.3.1	New workspace	21
5.3.2	Add packages to the workspace	21
5.3.3	Open workspace	22
5.3.4	Clean and make workspace	22
5.3.5	Jump between errors	22
5.3.6	Add all imports	22
5.3.7	The Notion of active package	22
5.4	Packages	22
5.4.1	Opening and closing a package	22
5.4.2	New package	23
5.4.3	Package editor	23
5.4.4	The most important parts of cabal files	23
5.4.5	Initializing a package: Clean and configure operations	25
5.4.6	Building	25
5.4.7	Run	25
5.4.8	Background build	26
5.4.9	Build system flags	26
5.5	Import Helper	27
6	Module Browser and Metadata	29
6.1	The Module Browser	29
6.2	The Search Pane	32
7	Debugger and Interpreter mode	34
8	Metadata collection	36
8.1	Background infos	36
9	Configuration	38
9.1	Layout	38
9.1.1	Advanced layout: Group panes	39
9.1.2	Using Leksah with multiple displays: Detached windows	39
9.2	Session handling	39
9.3	Shortcuts	40
9.4	Configuration files	40
10	The Leksah Project	41

11 Appendix	42
11.1 Command line arguments	42
11.2 The Candy file	42
11.3 The Keymap file	43

List of Figures

1 First-Start dialog	9
2 Leksah after first start	11
3 File menu	14
4 Edit menu	14
5 Find bar	15
6 Grep pane	16
7 Source candy example	17
8 Completion	17
9 Editor Preferences	18
10 Workspace Menu	21
11 Package Menu	23
12 PackageEditor 1	24
13 Error Pane	26
14 Package Flags	27
15 Module browser	30
16 Construct module dialog	31
17 Search pane	32
18 Debug & Buffer menu	34
19 Debug Pane	35
20 Metadata Preferences	37
21 View and panes menus	38

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.TX in the distribution.

1 Introduction

Leksah is an IDE (Integrated Development Environment) for the programming language Haskell. It is written in Haskell and integrates various tools available for writing programs in Haskell: the GHC compiler and interpreter, the CABAL package management system (the Common Architecture for Building Applications and Libraries), Haddock for producing documentation, etc. in one, single, comprehensive and easy to use environment. It allows the developer to concentrate on writing the program and Leksah gives him easy access to all information she needs and helps with the necessary housekeeping for compiling, linking and package management.

A unified focus for translating source code to executable programs: Leksah introduces the notion of a workspace that can include several packages transparently: to the programmer it appears as if there were a single program with a simple “make” command. Leksah manages rebuilding and installing packages as far as desired automatically.

Support for writing source code: Leksah supports debugging with GHCi, evaluation of expressions, gathering type information, setting breakpoints, displays values at breakpoints, etc. is all possible from within Leksah.

Last, but not least, Leksah collects information about installed packages, helps to find function names and their type and offers an auto-completion feature while you type new code.

The features of Leksah often reflect directly features of the Haskell tools used; therefore, to understand behavior in special cases needs sometimes reading the specific documentation of GHC, Cabal or Haddock, (and this manual, to a degree repeats what is found, with more detail and authority, in the respective tool documentation).

Leksah is written in Haskell, which means the Leksah developers use Leksah to develop Leksah and users of Leksah can read the code and contribute improvements. Leksah uses GTK+ as GUI Toolkit with the gtk2hs binding. It is platform independent and runs on any platform where GTK+, gtk2hs and GHC can be installed. It is used on Linux, Windows and Mac.

This document is a reference to the functionality of Leksah, it is not intended to be a tutorial. Since Leksah is still under development the information may be incomplete or superseded.

The current version is 0.8.

1.1 Further Information

The home page for Leksah is leksah.org. Stable version of Leksah can be installed from Hackage hackage.haskell.org/package/leksah using Cabal install. The source code for Leksah is hosted under code.haskell.org/leksah and code.haskell.org/leksah-head. The Leksah user Wiki is haskell.org/haskellwiki/Leksah. The Leksah forum can be accessed at groups.google.com/group/leksah/topics. The current version of this manual can be found at leksah.org/leksah_manual.pdf. An issue tracker to collect bug reports and suggestions for improvements is at code.google.com/p/leksah/issues/list. You can contact the developers at [info \(at\) leksah.org](mailto:info@leksah.org).

For information about the Programming language Haskell go to www.haskell.org. The GHC compiler is found at www.haskell.org/ghc. For information about gtk2hs www.haskell.org/gtk2hs/. For information about GTK+ go to www.gtk.org.

1.2 Release Notes

1.2.1 Version 0.8 Release March 2010

The 0.8 release adds the notion of workspaces to allow develop comfortably projects, where part of the code is in separate packages. This changes the handling of packages to a degree, which has been improved with introducing suitable defaults: a simple, single-shot program can be started with very few clicks and entering not much more than the name of the program; Leksah becomes usable even for just quickly testing an idea. Other changes include:

- Better metadata with non exported definitions for workspace packages
- Support for prebuild metadata packages
- Better completion (keywords, language extensions, module name, non exported definitions)
- Splitted in a client and server part (Client part doesn't import ghc-api)
- Added support for Ghc 6.12
- Prepared for Yi (Abstract TextEditor interface, not ready for use)
- Various other changes improve usability and stability of the platform.

A large number of bugs has been fixed, but there remain, probably a large number of, bugs - some old and not yet fixed and some new ones. We expect also to improve and streamline the user interface in the next minor release to achieve more consistency and make Leksah easier to learn. You may see comments to this effect in this document, suggesting possible changes in the interface. Your opinion on these and other possible improvements you see will be highly appreciated!

Version 0.8 works with GHC 6.10 and 6.12. The installation is described in Section 2 for the standard case, more up to date information on installation may be found on the Wiki haskell.org/haskellwiki/Leksah. If you have any trouble installing, please check the Wiki, the forum or contact the developers to find a solution. A smooth implementation is a priority for us and we like to hear about difficulties you encounter to fix them; please report them on the bug and issues tracker code.google.com/p/leksah/issues/list.

1.2.2 Version 0.6 Beta Release Juli 2009

The 0.6 version introduces an interpreter/debugger mode. This mode can be switched on and off from the toolbar. In interpreter/debugger mode expressions can be evaluated and the type of expressions can be dynamically shown. The GHCi debugger is integrated, so

that breakpoints can be set, it is possible to step through the code, observe the values of variables and trace the execution history.

The other features of Leksah like building in the background and reporting errors on the fly work in debugger mode as in compiler mode (but not configuring, installing, etc. of packages).

Another new feature is integration of grep and text search with regular expression. This can be accessed from the findbar.

The GUI framework has been enhanced, so that layouts can be nested in so called group panes. This feature is used for the debugger pane. Furthermore notebooks can be detached, so that Leksah can be used on multiple screens.

A lot of little enhancements has been made and numerous bugs has been fixed.

Known bugs and problems:

- The package editor works only for cabal files without configurations.
- MS Windows: The check for external modifications of source files does not work.
- MS Windows: Interruption of a background build does not work.
- GUI History still not working.
- Traces pane of the Debugger does not work appropriately.

1.2.3 Version 0.4 Beta Release February/March 2009

The 0.4 Release is the first beta release of Leksah. It should be usable for practical work for the ones that wants to engage with it.

It depends on $\text{GHC} \geq 6.10.1$ and $\text{gtk2hs} \geq 0.10.0$.

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

1.2.4 Version 0.1 Alpha Release February 2008

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

2 Installing Leksah

2.1 How to Install: Brief Instructions

You can install from

- a binary installer for your operating system, which is typically Windows or Macintosh.
- a package for your platform, which is currently Arch Linux and Fedora Linux, and Debian(Ubuntu) is in preparation.
- install from sources from Hackage via `cabal install leksah`
- leksah or leksah-head development repositories. (If you want the very last or want to help with Leksah development).

You can consult the Download page for up-to date information and try the user Wiki for further help.

2.2 Microsoft Windows

1. Install Haskell Platform with an installer for Windows (alternatively install Ghc directly)
2. Make sure wget and grep are on the path of your Windows shell
3. Install Leksah from the most recent binary installer for Windows.
4. Go to the post installation section.

2.3 Mac OS X

1. Install Haskell Platform with an installer for Mac OS X (alternatively install Ghc directly)
2. Make sure wget and grep are on the path.
3. Install Leksah from the most recent binary installer for Mac
4. Go to the post installation section.

2.4 Linux from Distro Packages

1. Install Leksah with the package management system of your Linux platform, which should pull all prerequisites automatically.
2. Go to the post installation section.

2.5 Install from Hackage

1. Install Haskell Platform (alternatively install Ghc directly, install Cabal and cabal-install)
2. Install gtk2hs in a version compatible with the installed Ghc compiler (Currently gtk2hs can't be installed via Hackage, but this should be possible in the near future, so that you don't have to care about this step any more). Make sure the gtk2hs gtksourceview2 package gets built and installed.

3. open a Console and do:

```
cabal update
cabal install leksah
```

4. Go to the post installation section.

2.6 Post Installation steps

1. Until the next release of gtk2hs, for a pleasant visual appearance, you have to copy or append the .gtkrc-2.0 file from the Leksah data folder or from the data folder in Leksah sources to your home folder. If you miss the step, the cross [x] buttons on tabs are almost invisible (or don't fit in tabs). This step may become obsolete during the 0.8 release cycle.

```
cd ~
wget http://code.haskell.org/leksah/leksah/data/.gtkrc-2.0 -O
    .gtkrc-2.0-leksah
echo -e '\ninclude ".gtkrc-2.0-leksah"' >> .gtkrc-2.0
```

2. Before you start Leksah for the first time, do a:

```
ghc-pkg recache
```

It has been observed, that a package recache is often necessary after installation. The symptom is an empty Module Browser, if you select the System scope.

2.7 First start of Leksah

The first time you start Leksah it will take you through the follow steps:

1. You are asked to fill in a form telling Leksah where your Haskell sources are (if you are not sure or just want to test, you can accept the defaults and correct them later in the "metadata" preferences)
2. Leksah collects "metadata", i.e. exported symbols and their type, comments explaining them etc. for all installed packages on your machine. This step may take a while and may give no feedback or a lot of strange errors and warnings, don't worry but be patient.

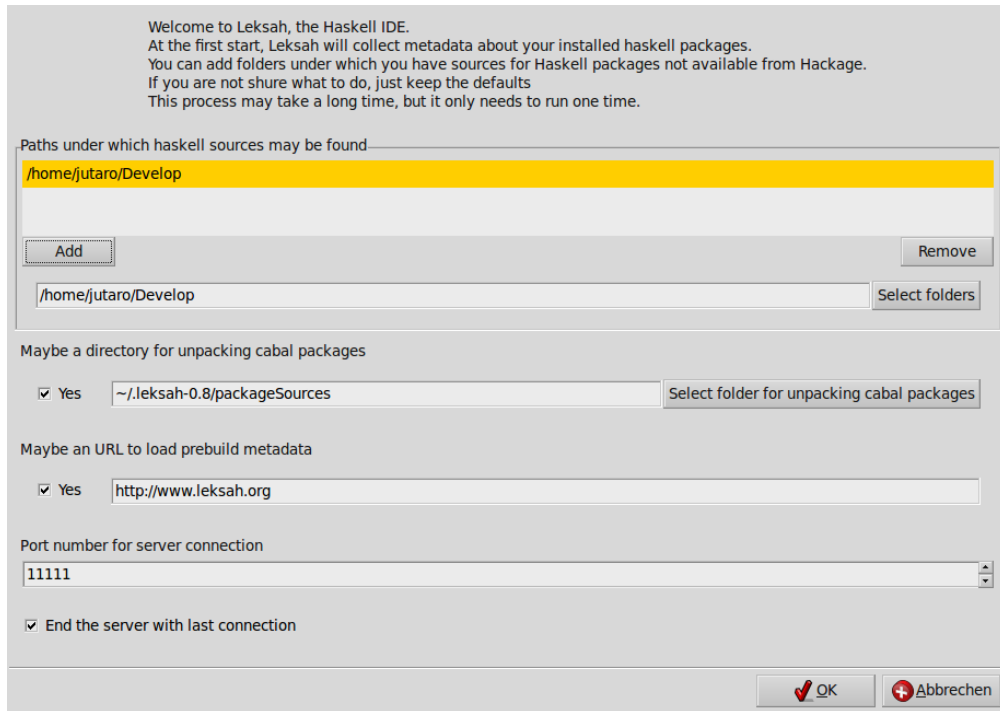


Figure 1: First-Start dialog

3. The Leksah IDE starts and you can start working.

Later starts will read in the previously collected metadata and check only for changes. After starting up, Leksah will open its Main window in a standard configuration.

Progress on your first contact with Leksah:

1. Start with the, infamous, “Hello World” example. The next section gives you a step by step description.
2. Then it might be the best to construct a workspace and add an existing project and explore Leksah while you work on it.

2.8 First start dialog

When you start Leksah for the first time it must collect the information about the packages you have on your computer and may use in your projects. The first start dialog let you enter settings about this process. Leksah then collects information about exported symbols, their type and possible comments (collectively called metadata) to support your work, e.g. by suggesting auto-completion and type information about functions you may use while you edit your source.

Later you can change this settings in the preferences pane in Leksah and you can rebuild the metadata at any time. `leksah-server -sbo +RTS -N2` from the console. Details about metadata collection can be found here: 8.

If you want to start from scratch again delete or rename the `.leksah-*.*` folder in your home folder. Then you will see the first start dialog again.

In the first start dialog you are asked for:

1. The location of folders, where Haskell source code for installed packages can be found. This is important for packages which can't be found on Hackage.
2. Maybe a directory, where Leksah will unpack source files for packages. If you give no directory here, Leksah will not try to unpack the sources.
3. Some packages are difficult to process with Haddock. So we provide some prebuild metadata. If you allow this, Leksah will look for prebuild metadata, if sources are available, but Haddock fails to process.
4. The port number used for the local connection to the Leksah server.
5. By default the Leksah server terminates with the last connection. You can change this setting here.

Leksah collects information about all installed packages on your system that will take some time (minutes to half an hour) the first time. Errors occuring in this metadata collection step indicate only that Leksah has not succeeded to extract the source locations and comments from a module or package; they are not consequential, except that some metainformation may be missing. The metadata is cached and future starts only scan newly installed packages, starts only information for new packages will be installed.

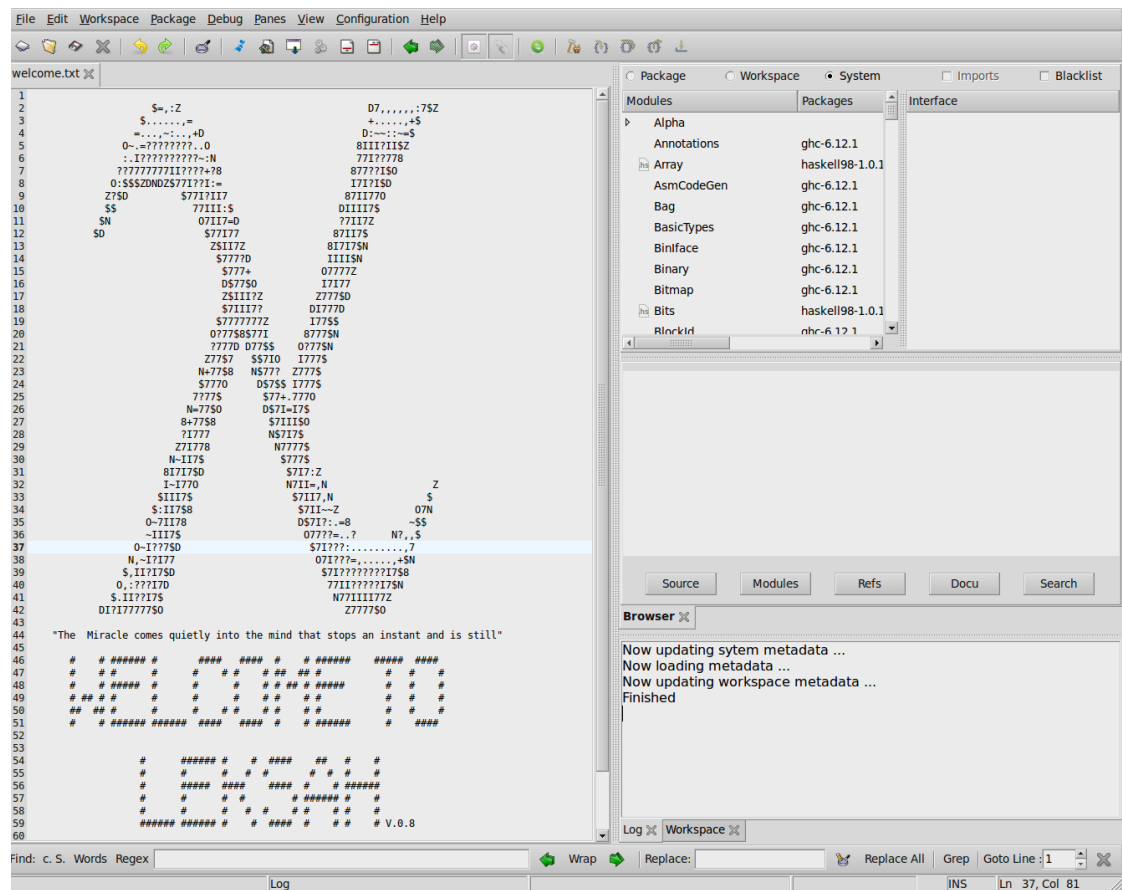


Figure 2: Leksah after first start

3 Hello World example

- Workspace -> New. Constructs a new workspace in a selected folder and give it a name (e.g. “Hello”). This produces a file Hello.lkshw.
- Package -> New and use the "Create Folder" button to make a new folder for the package. Make sure to be in this folder when you click “Open”. An editor opens up, which let you edit cabal files. The name proposed for your package is the name of the folder you just constructed. That is the convention with Cabal packages. The defaults are set for creating a simple executable. The base package is specified as build dependencies, and an executable with the name of the package will be constructed. The main module resides in a file “Main.hs”. The sources are in a “src” subdirectory of the packages.
- Click Save to write the .cabal file.
- The main module gets automatically constructed and opens.
- Now add your code to the module

```
main = putStrLn "Hello World" .
```

- By default, auto build is on and you can see that the file will be compiled in the Log pane.
- Choose Package -> Run or ctrl-alt-r, and you will see Hello World in the Log pane.
- Or: Choose Package -> Install, open a Shell and try out your newly created executable

Congratulations ! you have now entered, compiled, linked and run your first Haskell program with Leksah. It is as easy as: Create workspace / New package / enter your code/ Run. Remember:


- the project folder is the folder in which your .cabal file for the project is stored.
- A workspace is just a file, which contain information about included packages.
- You see what Leksah is doing by observing the output from the Log window.

Furthermore:

- You can add Packages with the context menu in the workspace pane. You can construct new packages with Package -> New, from the menubar.
- You can add other modules by selecting “Add module” from the context menu of the modules pane of the browser group.
- You can open panes you need by selecting Panes -> Browser | Log | ... from the menubar.

- You can editing modules by selecting them in the Browser. You can search in the modules pane of the browser by typing text.

You may as easily debug it

-  Switch debugger Mode on.
- Pane -> Debugger
- Select the word “main” in your code
- Right click and choose “Eval” from the pop-up menu or press ctrl-enter.
- Switch of debugger Mode if you want to compile an executable.

It is probably counter productive for new users to use Candy mode (converts some common ASCII based operators to Unicode alternatives) because all the tutorials use ASCII. Switch it off when you get irritated. Deselect Configuration -> To Candy from the menubar.

4 The Editor

Most of the time programming is editing source code. To edit Haskell source files Leksah uses the GtkSourceView2 widget. It provides editing, undo/redo, syntax highlighting and similar features. In the file menu (3) you find the customary functionality to open, save, close and revert files. To avoid confusion, it is useful to be able to close all files, or 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) at once - this helps you focus on your project. This way it is as well possible to close all files, which don't belong to a workspace.

Leksah does not store backup files. Leksah detects if a file which is currently edited has changed on disk and queries the user if a reload is desired. (Attention: This don't currently work for Windows, so take care). When you open a file which is already open, a dialog pops up to inquire if you want to make the currently open file active, instead of opening it a second time (Leksah does not support multiple views on a file, but if you open a file a second time, it's like editing the file two times, which makes little sense). The list of files is shown as notebook tabs (on top or left of the files - as you prefer (Menu -> View -> Tabs Left)).

When a file has changed compared to the stored version, the file name is shown in red in the notebook tab, reminding you that it needs to be saved before compilation.

If you want to change to a different file editor buffer you can open a list of all open files 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. (See 4.4 for a better way to switch between source files).

On the right side in the status bar at the bottom you can see the line and column, in which the cursor currently 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 (4) 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 (-); however, the comment symbol must start in the first column (beware of illegal sequences like `--#` which may be automatically produced by inserting a comment in front of some symbol).

Selected blocks of code can be shifted left or right using the tab or Tab/Shift-Tab keys. 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

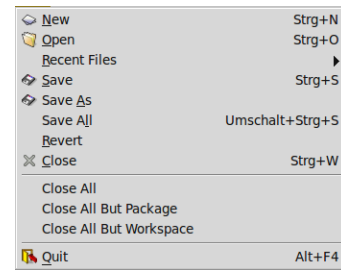


Figure 3: File menu

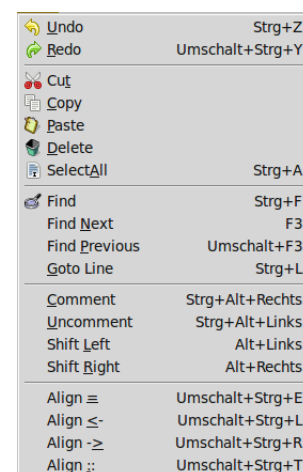


Figure 4: Edit menu

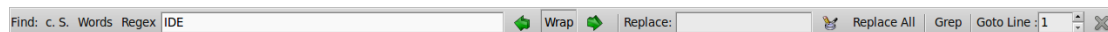


Figure 5: Find bar

occurrence of this special characters in the other lines for alignment.

4.1 Find and Replace in the current folder

Leksah supports searching in text files. When you select Edit/Find from the menu the find bar will open (5) and you can type in a text string. Alternatively you can hit Ctrl-F or select a text and hit Ctrl-F (a standard keystrokes binding, which can be configured, see 9.3). Pressing the up and down arrow will bring you to the next/previous occurrence of the search string. Hitting Enter has a similar effect as the down arrow. Hitting Escape will closes the find bar and sets the cursor to the current find position. You have options for case sensitive search (labeled “c.S.”), for searching only whole words (toggle Words) and for wrapping around (button Wrap), which means that the search will start at the beginning/end of the file, when the end/beginning is reached. If there is no occurrence of the search string in the currently open file the entry turns red.

You can search for regular expressions by switching on the Regex option. Leksah supports regular expressions in the Posix style (by using the regex-posix package). When the syntax of regular expressions is not legal, the background of the find pane turns orange.

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

The last field on the line gives you a mean to jump to a certain line number in the current text buffer.

4.1.1 Search in the package: Grep

Searching for text in all files in a package is often useful. For this feature the grep program must be on your path. You can then enter a search string in the find bar and search for all occurrences for the string in the folder and sub-folder of the current package with pressing the Grep button. A pane will open (6), and in every line show where the expression was found (with context). By clicking on the line, the file is opened in an editor and the focus is set to the selected line. You can navigate between lines with the up and down keys.

Grep supports the search for regular expressions.

4.2 Source Candy

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

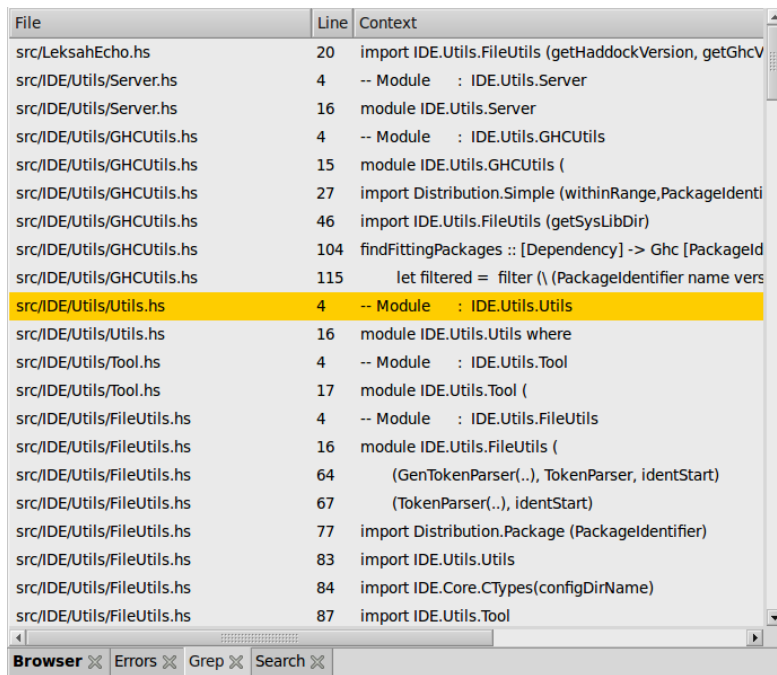


Figure 6: Grep pane

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

4.4 Using the Flipper to Switch Between Editors

You can change the active pane using a keyboard shortcut to bring up the flipper. It lists the most recently used panes first so they are easier to get to. The default shortcuts for the flipper are Ctrl+Tab and Ctrl+Shift+Tab or Ctrl+Page Down and Ctrl+Page Up.

The approach in Leksah is comparable to the Alt+Tab and Alt+Shift+Tab used to switch between programs in the OS (Ubuntu, Windows).

4.5 Change Your Preferences for the Editor

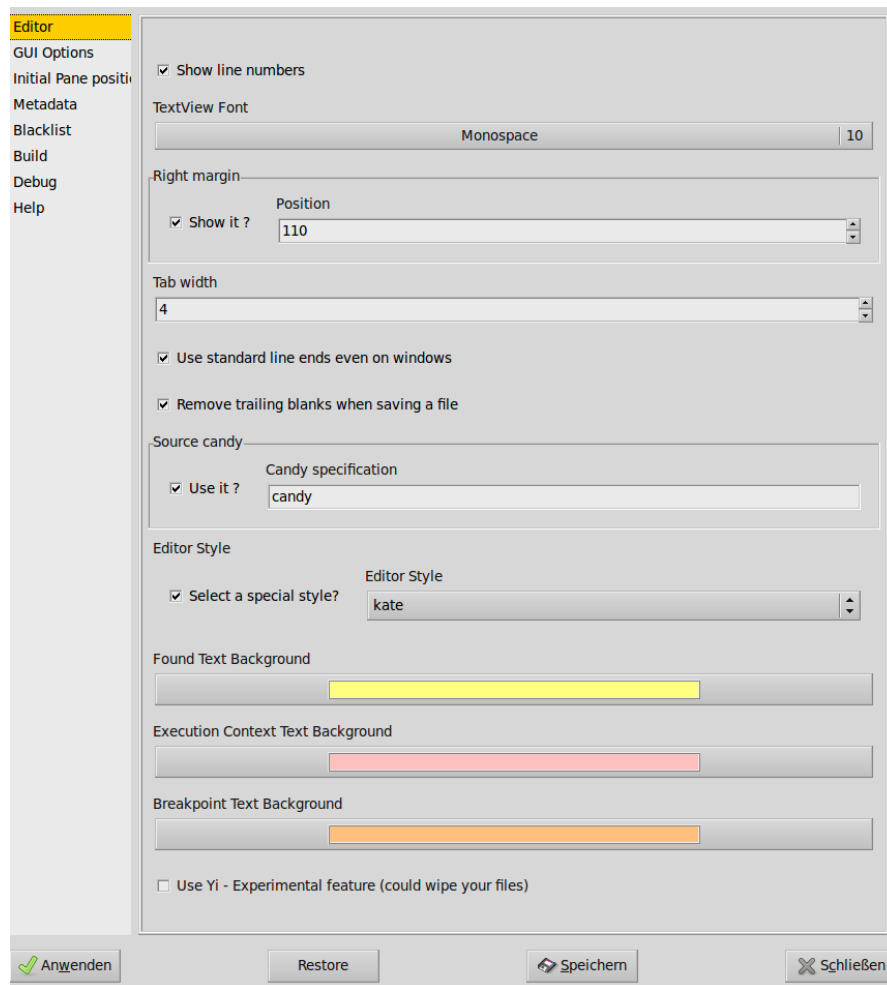


Figure 9: Editor Preferences

When selecting Configuration / Edit Prefs the preferences pane opens, which has a selection called Editor (Figure 8), where 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. Using hard tabs is not recommended for Haskell and the Haskell compilers allow only tab size of 8.

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.

Leksah dialogs offer mostly a APPLY, SAVE and a CLOSE button. Be aware that the close button does not save your changes; press apply, save and then close! We will change this for the next version.

In dialogs, where you select something, there is typically an ADD button, selecting does not add, neither! Therefore: adding an item means select item, add, save, close.

4.6 Further info

The work with the editor is influenced by other features

- For background building, which may save your files automatically after every change refer to 5.4.8.
- For information about editor preferences go to 4.5.

5 Working with Projects: Workspaces and Packages



Haskell organizes software projects in packages, which are managed independently. A package is compiled and linked as a unit to produce one or more executables or/and a library. It is installed with the package manager Cabal. A package can be uploaded to Hackage. And provided packages are downloaded from Hackage and installed. Packages have version numbers and specify version ranges for dependencies. Cabal assures that, if a packages is compiled, correct versions of other packages are selected.

The difficulties, when working with a project where source code under development is spread over several packages, are overcome in Leksah with the concept of a workspace. It combines several packages and allows smooth working with files from all packages.

Leksah always works in a workspace and always needs at least one package, to do anything useful. This seems overkill for very simple projects, where the workspace contains just one package and this package just one source module producing one executable, but Leksah provides defaults that reduce the effort to a minimum. The principle to always to work in a workspace and in a package is beneficial in the long run, because it gives a smooth transition from a one-shot idea to a complex projects and integrates the widely used cabal system fully.

Leksah, in addition, saves the state of your work environment with a workspace: so you can switch between workspaces and get exactly back to where you stopped working when leaving the workspace: the same files open in editors and the cursor in the same file and position. When you open a workspace, and a session is attached to it, Leksah prompts you, if you want to switch to the session associated with the workspace. Leksah silently always saves the session for the workspace you are closing.


If you have auto-build on and you change a file, Leksah detects if this file belongs to any project in your Workspace. If this is the case, it builds the package. So the source file you are working on, doesn't need to belong to the active package. Leksah will detect and compile the package for you. The active project is important for the menu items of the package menu, because they always work on the active package.

- Background build can be permanently set in Prefs -> Build -> Background build. You can temporarily enable and disable it from the toolbar with this button: .
- You can set, if Leksah automatically save all files before building, by the setting Prefs -> Build -> Automatically save all files before building.
- Linking can take a long time, and on Windows we can't interrupt the build process in the moment, so it may be an advantage to switch of linking. This can be done by: Prefs -> Build -> Include linking in background builds. You can temporarily enable and disable it from the toolbar with this button: .
- The same option (disable linking) can be used to disable cross package build temporarily.

5.1 Cross package build

The following is valid in compiler mode: Depending on your settings the following may happen. After a library has been successfully build, it will be installed if:

- “Include linking in background builds” is on and either
 - “Install always after a successful build” is selected or
 - “Install if it’s a library with depended packages in the workspace” is selected, and it has dependent packages

After this, dependend packages will be build. If you want a background build, but only for the one package you’re working on, you can temporarily disable this mechanism, by deselecting .

5.2 File Organization with Workspaces

A workspace is represented by a file (*workspace_name.lkshws*) in a directory. You may choose a hierarchical folder structure with a workspace file at the top and the projects in sub-folders for complex projects, but you can as well put all *workspaces* in one directory and put all packages flat. You have to use care when you create a new workspace file.

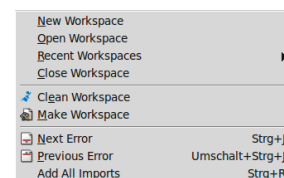
Each package directory contains at least a cabal file (*packagename.cabal*) and typically folders for the source files, following the usual Haskell ghc convention of hierarchical module names.

Cabal controls the compilation and linking of packages with GHC and puts the results in a dist folder in the package; this folder is reconstructed often and may be deleted without loss. In the folder `package_name/dist/build/executable_name` you find the executable, but it is also installed in the folder `~/.cabal/bin` (for a build with the `-user` flag. You may wish to add this folder to your search path).

5.3 Workspace Operations

5.3.1 New workspace

Under the menu workspace you find commands to create a new workspace with a specific name and select the folder in which it should reside. The windows title and the 3. compartment of the status bar informs you always about the currently open workspace.



5.3.2 Add packages to the workspace

Open the workspace pane (from menu (Panes → Workspace) and do a right-click to get the pop-up menu to add a package by selecting the corresponding cabal file.

Figure 10: Workspace Menu

5.3.3 Open workspace

When starting, Leksah opens the last workspace used. You can change to another workspace by opening the corresponding workspace file, or by choosing from the list of recently used workspaces. When opening a Workspace you can choose to

5.3.4 Clean and make workspace

A workspace can be cleaned, meaning all packages gets cleaned and must be recompiled from scratch. Make builds all the packages in a meaningful order, and installs libraries if needed. It only stops if an error occurs.

5.3.5 Jump between errors

There are menu items to move to the next or the previous error the compiler found. You can as well use keyboard shortcuts for this: CTR-J and SHIFT-CTRL-J. It is as well possible to move by pointing to the error messages in the log pane or error pane.

5.3.6 Add all imports

If you miss imports (given error messages (“xx is not in scope”) CTRL-R is adding them automatically to your import list. Limitation: it does so only, if the modules they export are in a workspace package or are in a package listed in the build dependencies of the package.

5.3.7 The Notion of active package

In the package pane a package can be marked as active either with the pop-up menu or by double clicking. The active package is the one that the commands in the Package menu refers to. (e.g. configure, build, install).

5.4 Packages

The concept of a package is used to handle a unit of work for the development of some library or executable. It is, in the first place, the unit Cabal deals with and is a standard in the Haskell community.

Leksah stores data for packages in the standard cabal files. The same files can be used outside of leksah: for example, you can issue the command `cabal install` in the folder that contains the cabal file and cabal will (as it would inside leksah) configures, compiles and links and install or register the library or executable produced.

5.4.1 Opening and closing a package

Leksah uses Cabal for package management, and opening a package is done by opening the corresponding *.cabal* file. So when you select Package / Open Package from the menu, select the *.cabal file of the desired package. You must not have more than one *.cabal file in a folder!

Leksah shows the currently active package in the third compartment in the status bar and in the window title. The package file contains appropriate defaults and for a small program, you may just save and close it.

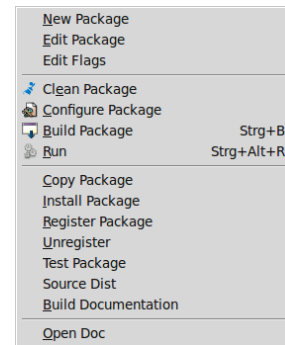


Figure 11: Package Menu

5.4.2 New package

To start with a new package select Package / NewPackage from the menu. Then you have to select a folder for the project, this is by GHC convention the same name you will give to your package in the package editor (see 5.4.3). Then the package editor will open to collect the package details.

This currently does not work, if an editor for a different package is open.

5.4.3 Package editor

The package editor (12) is an editor for cabal files and but you can edit the cabal files in your regular text editor as well. Leksah works (usually) with the cabal files you and others have already written, for example those you get when you install a package from Hackage. Since cabal files offer complex options the editor offers many separate sub-panes in a list on the right. For a complete description of all options see the Cabal User's Guide.

The package editor does currently not support the cabal configurations feature. If you need cabal configurations, you need to edit the cabal files as a text file separately. Leksah uses standard cabal files with no modifications this is no problem just the package editor will not work for you.

5.4.4 The most important parts of cabal files

A package has, as a minimal requirement, a name and a version (default is 0.0.1 – meaning something like “first idea”). If your code uses other packages then they must be listed in dependencies. This will be at least the *base* package (which is entered by default). This is independent whether you downloaded them, e.g., from Hackage produced them yourself. Version numbers are used to document (and enforce) that older versions of a program use the corresponding older versions of other packages with which it was developed originally.

The result of the packages can be an executable and you enter the name of the Haskell file that contains the main function in the executable pane and the name of the executable. The result of the package can be a library; in the corresponding pane you tick

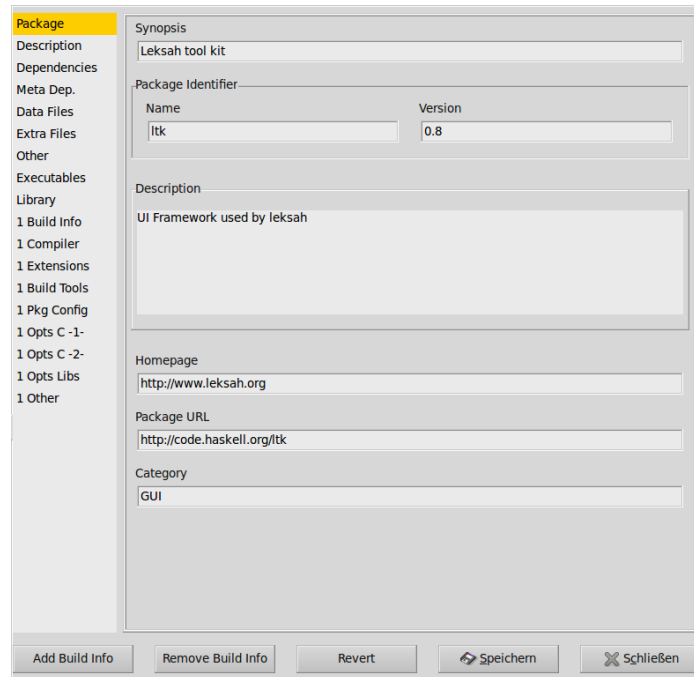


Figure 12: PackageEditor 1

off the modules which should be exposed (i.e., their exports can be used in other packages). Cabal gives the possibility to build more then one executable from one package and to build a library and executables from one package.

You have to specify 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
- 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.

5.4.5 Initializing a package: Clean and configure operations

Before a package can be acted on it must be configured; you may clean a package (i.e., delete its *dist* folder) to start afresh.

Configure checks that the packages the current packages depend on are installed in GHC package manager; it checks for name and version, if you specify them. If an Hackage package is missing, you can `CABAL INSTALL` it in a terminal window.

Two types of errors regarding packages may be reported:

While configuring, Cabal checks that the packages you have listed in the `depends on` section are installed on your computer. If one of your packages is missing (or missing the version that is needed) you can install it either – for packages you have the source on your computer, e.g. because you wrote them – by switching Leksah to the folder where this package is and configure, build and install them with the command *cabal install*. For packages that are on Hackage – use a console, go to the directory where you keep such sources and type `cabal install packageName` (possibly `package_name-version`); cabal then recursively installs the package and all packages it depends on.

Separate from this error message the case, where the compiler misses a module you want to import. Ghc provides an error message, indicating what package you have to add to the `depends on` section in the cabal file. Edit the package, add the dependency and do `configure/build`.

You have to take care as well, that there is a user and a global package db. Leksah uses the `-user` flag by default, to minimize errors.

5.4.6 Building

The most frequently used functionality with packages is to make a build. If the package was not configured before, Leksah does that step automatically. When you start a build, you can see the standard output of the Cabal build procedure in the Log pane.

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 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.



This is the symbol, which initiates a BUILD when clicked on the toolbar (Ctrl-b).

The error pane (13) shows the errors in the form of a table and provides the same functionality you find in the log, but it may be more convenient to use.

5.4.7 Run

You can run your program after the build operation has compiled and linked it. there is a convenient button  to start it!

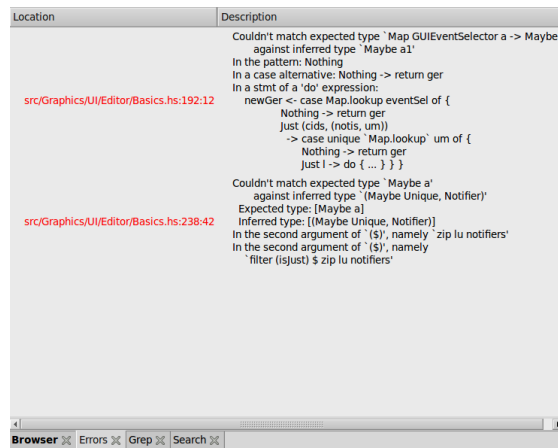


Figure 13: Error Pane

5.4.8 Background build

Leksah can run builds while you work and highlight errors as it finds them. This works with a timer that runs continuously in the background. If there are changes made to any open file it ...

- interrupts any running build by sending SIGINT (this step is OS X and Linux only at this point, it's not working on MS Windows)
- waits for any running build processes to finish
- saves all the modified files
- starts a new build

Current limitation: Because we can't interrupt the build on windows there is an option in the Leksah build preferences to have it skip the linking stage in background builds. This reduces the delay before a next build starts. Background build and linking can be configured in the preferences and as well switched on and off from the toolbar.



This is the toggle, which switches BACKGROUND BUILD on or off in the toolbar.



The LINKING toggle that switches background build on or off.

5.4.9 Build system flags

Cabal allows more operations than just build; for example producing documentation with Haddock (with The "Build documentation" item in the package menu). For each of these operations you can enter the specific flags they require for you special case. We give here two often examples of flags, others work similarly and we recommend that you consult the respective documentations.

(10) consult the Cabal User's Guide.

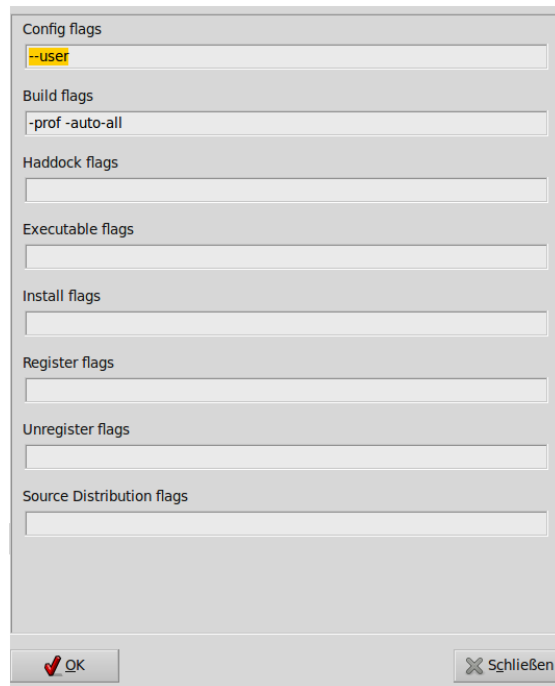


Figure 14: Package Flags

Cabal needs the `-user` flag (which is set by default in Leksah) to install the result of a built in the user package database (the alternative is `-global` to put the resulting files in global space in `ghc-pkg`).

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.

5.5 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 and to write import statements is a frequent and annoying task. In Leksah if the compiler informs about a missing import, 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 there is more then one module that exports this identifier, a dialog will appear which queries you about the module you want to import it from.

Leksah then adds a line or an entry to the import list of the affected module with the compiler error. 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 will 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 sequentially.

When Leksah does not find an identifier update the Leksah database.



UPDATE METADATA OR (CTRL-M)

The import helper just looks in imported packages, so if you miss a package import, you have to fix it manually.

Obviously some not in scope errors have other reasons, e.g. you have misspelled some identifier, which can't be resolved by adding imports. After adding all imports, you have to save the file and then start a new build.

6 Module Browser and Metadata

Leksah collects data about the modules of all installed Haskell packages on your system. It does this by reading the Haskell interface files `.hi` files (from GHC). It as well collects source positions and comments from sources. For this it looks in the source directories you specified in the preferences and downloads and unpacks sources from Hackage depending on your settings. Starting from the current version, Leksah can as well use prebuild metadata it might find on the web, to provide metadata for packages you have sources for, but the call to the Haddock library fails for some reason.

The packages in the workspace are treated differently, as not only external exported entities are collected, but all exports from all modules are collected. As well identifiers, which are not exported from a module get listed. The source symbol for them is shown in gray.

This metadata is used to answer questions like:


- Which packages and modules export this identifier?
- What is the type of the exported identifier?

If the source was found, it lists as well :

- The comment for this identifier
- and can mark the item in the source file at the correct position

If you like to get information about some identifier in the code, the easiest way is to press `CTRL` and `DOUBLE CLICK` on it.

More precisely the operation starts with a release of the left mouse button with a selection with `CTRL` pressed; You can use this if the double click doesn't select the intended area. If the identifier 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.

The sorts of the identifiers shown are differentiated by the symbols you find in Table 1. Note as well the special symbol for identifiers exposed, but only indirectly, because the definition is imported from another module. 











sort	symbol
function	
data	
constructor	
slot	
type	
newtype	
class	
member	
instance	
rule	

Table 1: Sorts of identifiers

6.1 The Module Browser

The module browser (15) shows information about modules and their interface separated in scopes: package, workspace, and system. If no package or workspace is open only the system scope has information. (If a workspace/package is open, it's name(s) are displayed in the third subdivision from the left of the status and in the title bar.)

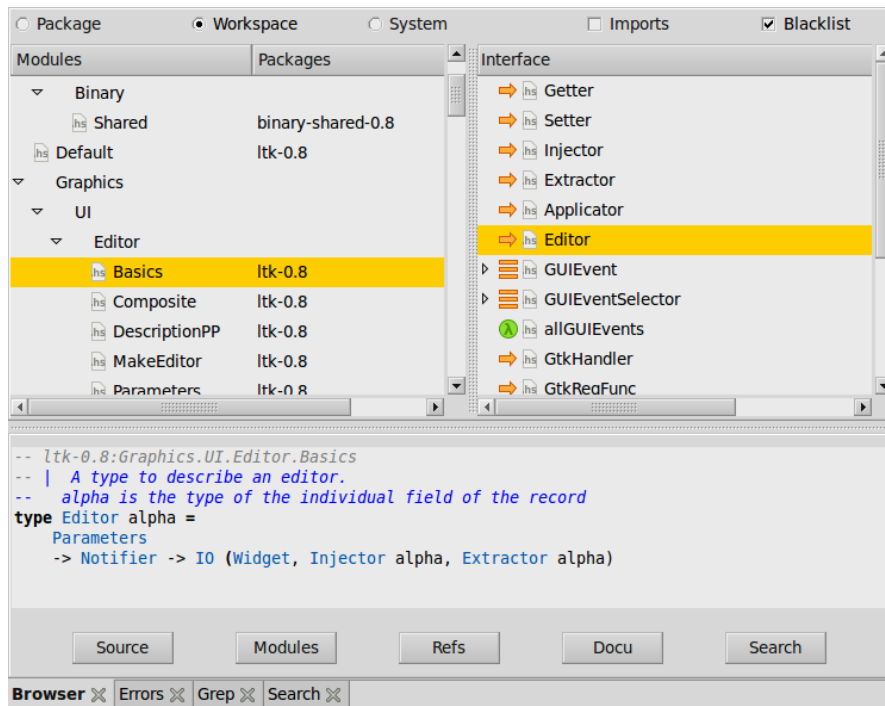



Figure 15: Module browser


The scope of the displayed information is selected with the radio button on top of the modules pane: The *Package* scope shows only modules which are part of the active project. The *Workspace* scope shows all modules of all packages in the workspace. The *System* scope shows all modules of installed packages of the system.


(It lists all modules of installed packages. These you would get with *ghc-pkg* list. Leksah scans the user and the global package database, when both are present).

The amount of information displayed may overwhelm you with details from packages that are not of interest to you (Like e.g. like Haskell-98, ghc, or base-3.0*). Such packages can be excluded, by blacklisting them. The packages you want to hide can be specified in the preferences and you can use the radio button at the right to hide them.

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 this icon shows up, Leksah has found a SOURCE file or source position for this element. You can open the 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 is achieved with selecting *Go to definition* from the context menu.

 This is the same as before, but is used for definitions not exported from the module.

 This icon indicates that the symbol is REEXPORTED from another module., because its long list is not much hierarchically structured.

By selecting an element in the Interface List the so called Info Pane is shown with detailed information (see next subsection).

The modules pane provides detailed information and are the quickest way to open a source file for edit. Go to the modules pane, select package or workspace scope, possibly find the module by entering some text, and double click on the module's name to open the file in the editor for editing the file.

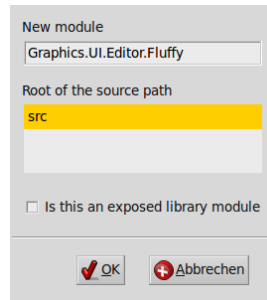


Figure 16: Construct module dialog

From the context menu of (right-click) the modules pane you can add a new module by selecting *Add module*. The Construct Module dialog will open (16). You have to enter the name of the module, the source path to use if alternatives exist. If the project is a library you have to specify if the module is exposed. Leksah will construct the directory, modify the cabal file and construct an empty module file from a template (The template is stored in the file `module.lksht` in the data folder of the project, and will be read from the `.leksah-*` folder if you want to provide a different template file their).

The modification of the cabal file will currently only happen, if it does not contain configurations.

The Info Pane

The Info Pane is the lower pane of the module browser and shows information about an interface element, which may be a function, a class, a data definition or a type (selected, for example, in the modules pane). It shows the identifier, the package and module that it is exported by, it's Haskell type and, if found, the Haddock documentation inserted in the source as a comment.

If you select and initiate an identifier search in an editor pane, the information about this identifier is automatically displayed in the info pane (maybe nothing!). The easiest way to do this is to double click on an identifier while pressing CTRL.

Only previously collected metadata is available this way. If the item has changed you could initiate an update of the information collected with update workspace metadata (menu configuration → update workspace data, or Ctrl-m).

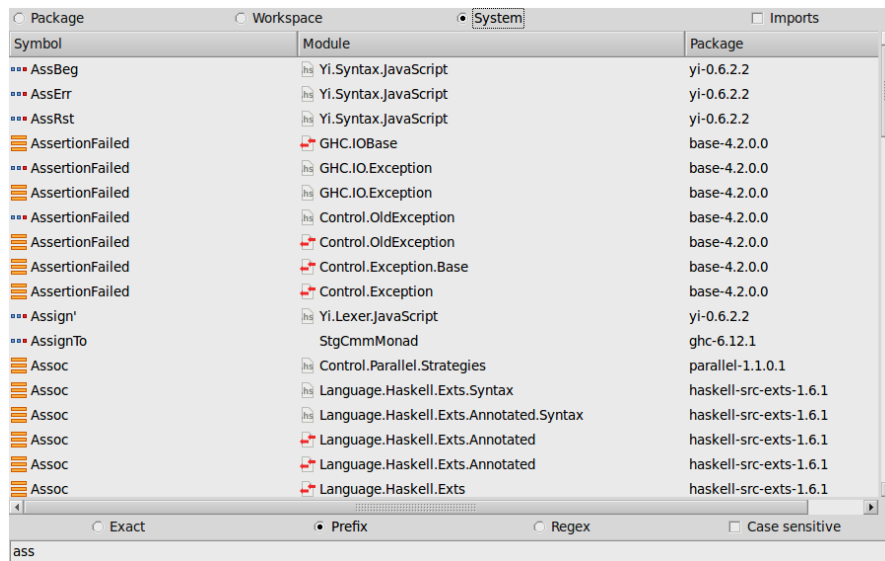


Figure 17: Search pane

If a source location is 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 *Refs* button a pane opens which displays modules which uses this element.

With the *Docu* button you can initiate an external search in a browser with e.g. Hayoo or Hoogole, depending on the configuration in the Preferences.

With the *Search* button you can initiate a metadata search for the identifier.

6.2 The Search Pane

You can search for an identifier in the metadata by typing in characters in the entry at the bottom of the pane (not the search entry at the bottom of the window!). The search result depends on the settings in the search pane (17). You can choose:

1. The scope in which to search, which can be Package, Workspace or System. For Package and Workspace scopes you can search with or without imports, which gives 5 different scopes.
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 if the module reexports the identifier, or if the source of the identifier is reachable. When you single click on a search result, the module browser 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 while pressing Ctrl in an editor pane, a case sensitive and exact search in the is started.

7 Debugger and Interpreter mode

You can switch Debugger mode on *only* from the toolbar with the:

 toggle, which switches debugger Mode on or off.



In debugger mode the packages and modules for your current project are loaded into GHCi.

In debugger mode, the menu entries from the Debug menu are no longer disabled (Fig 18), and the context menu of source buffers have entries that were not meaningful in the regular (GHC) mode. There is also a group of panes specifically used for debugging, allowing you to manage break-points, observe variables, etc.

You can open the debugger group pane by choosing Panes / Debugger. Commands using the debugger are given mostly in the source editor pane with a context menu: You select some text and right-click to get the context menu. it lets you:

- EVALUATE the selected expression in the interpreter and observe the result. If no text is selected the current line is taken as input. Select eval. The result of the evaluation is shown in the log window and as *it* in the variables pane. You can as well use the keystroke Ctrl-Enter.
Choose “Eval & Insert”, to insert a string representation of the result after the selected expression.
- Determine the TYPE of an expression: Select the expression in a source buffer and select Type from the context menu.
- Get INFO about an identifier select: Select Info from the context menu.
- Get THE KIND of a type select: Select Kind
- STEP through code: Select the expression in a source buffer. Select step from the context menu (or F7). Use the toolbar icons (or shortcuts) for stepping

 Step (F6),  Step local (F7)

 Step in module (F8),  Continue (F9)

- Set BREAKPOINTS by putting the cursor at the breakpoint and select *set breakpoint* from the context menu. Run your application or test cases and start stepping at

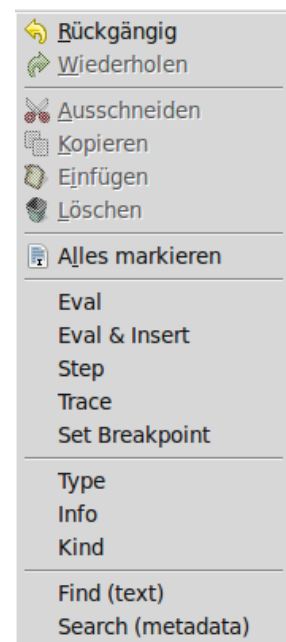
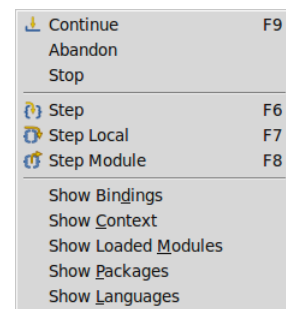


Figure 18: Debug & Buffer menu

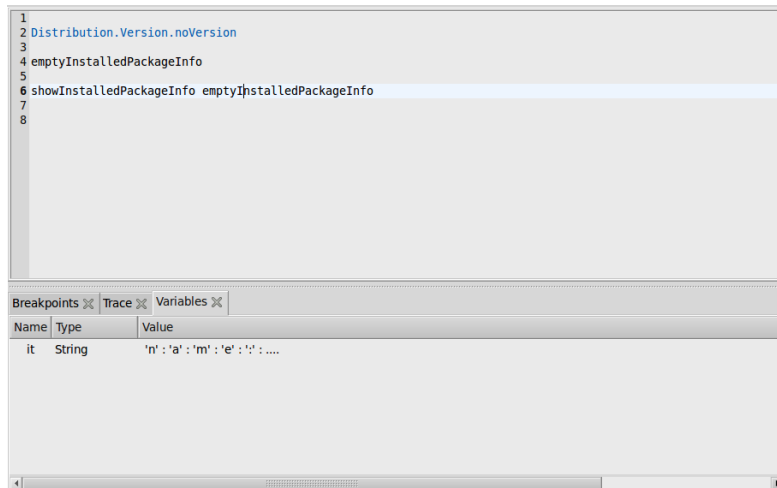


Figure 19: Debug Pane

the break point. After a break point is reached you use the operations of GHCi with convenient shortcuts.

The debugger has a pane in which you can enter expressions and have them evaluated. The pane is a Haskell source buffer, which has the reserved name `_Eval`. Its contents is saved with the session.

Note that:


- breakpoints are set on identifiers selected, not necessarily where you have found it in the source (e.g., used in an expression);
- current breakpoints are listed in the breakpoints pane; you can remove breakpoints from this pane
- While stepping through code, you can observe VARIABLES in the variables pane. You can print or force a variable from the context menu of the variables pane. You can update the pane from the context menu.
- You can observe an execution trace in the traces pane. Navigation in the traces pane is currently not supported (`:back`, `:forward`).
- You can query information about the current state of GHCi from the Debugger menu. E.g. *Show loaded modules*, *Show packages* and *Show languages*.
- You can directly communicate with GHCi by evaluating commands entered as text in the source editor and select it. E.g. “`set ...`”

For more information about debugging in GHCi read the GHCi section in the GHC manual.

8 Metadata collection

Remember, that metadata is the data Leksah has collected from all the Haskell code (including .hi files for installed packages) it could reach on your computer.

The initial scan may take a long time (some minutes); when Leksah starts later, it checks only for changes, but does not scan all files again. Metadata collection depends on the local configuration, especially the list of places where Haskell code may be found, which is entered in the preferences. Occasionally, you may find it useful to rebuild the metadata.

Metadata collection can be manually triggered: If you select *Configuration -> Update workspace data*, the metadata for the current project is collected. This brings the metadata of the current project up-to-date. You can as well press Ctrl-m or hit this symbol in the toolbar: 

If you select *Configuration -> Rebuild workspace data*, the metadata for the current project is rebuild.

If you select *Configuration -> Update system data*, Leksah checks, if a new library has been installed and then collects metadata for additions.

If you select *Configuration -> Rebuild system data*, Leksah rebuilds all metadata, which may take a long time. Currently the preferred way to do this is to call `leksah-server -rbo +RTS -N2` from the command line. The reason for this is that the server process may allocate a lot of memory during collection.

8.1 Background infos

The metadata collection itself proceeds different for workspace and system packages:

- For workspace packages Leksah just uses the parser without typechecking and maybe .hi files if available.
- For system packages Leksah uses .hi files and if sources are available Haddock as a library.

Collection for system packages works as follows:

1. Packages you installed with cabal from Hackage. If Leksah can't find sources, it does a *cabal unpack* in the source directory you specified for this in the preferences (By default `.leksah-*/packageSources`).
2. Source files in the folders listed as source folder in the preferences. Leksah looks for all .cabal files it can find below the source folders. Therefore, Leksah collects source information only from “Cabalized” projects (i.e., projects that have a .cabal file). From this information the file `source_packages.txt` in the .leksah folder is written. If you miss sources for a package in Leksah, consult this file if the source place of the package has been correctly found. You can run this step by typing in a terminal: `leksah-server -o` (or `-sources`).

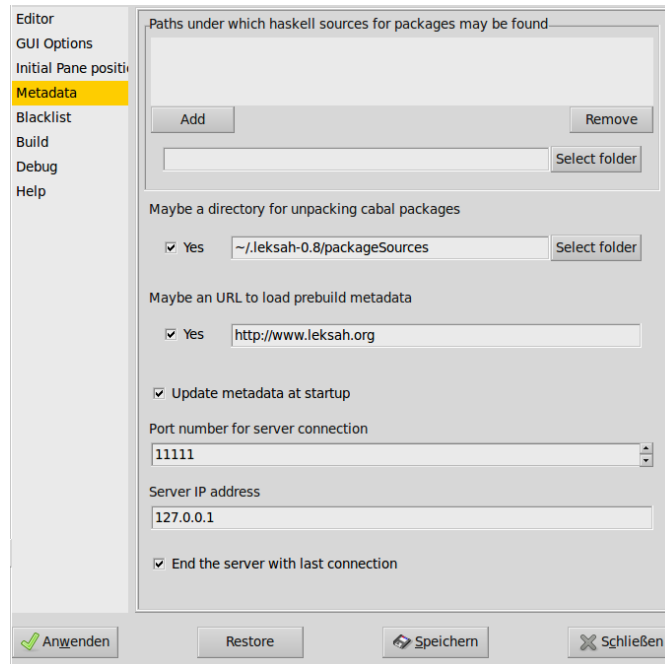


Figure 20: Metadata Preferences

3. Problems may occur due to preprocessing, header files, language extensions, etc. Error message produced while metadata collection indicate that not all information for a package was found. If the Haddock call doesn't succeed, Leksah looks if it can download a prebuild metadata package from the server, if this option is selected in the preferences.
4. The result of metadata collection is stored in the folder *.leksah-*/*/metadata* in files called **.lkshm*. In this folder for every package a metadata file is stored (e.g. *binary-0.4.1.lkshm*). These files are in binary format. If you want to rebuild just one package you can delete it here and update the system metadata. The files *.lkshe* specify the base path to sources, if the collection for sources was successfully or if a metadata file could be downloaded for this.

For the workspace packages a different procedure is used.

1. Metadata is collected from the source directories of the packages you are working on. The results are stored in a per module base in a folder with the package name (e.g. *.leksah-*/*/metadata/package-*.*.**).
2. Update happens on a per file base only for changed source files.

9 Configuration

Leksah is highly customizable and can be adapted to your specific needs and work organization. What follows here is not needed for initial use of Leksah (and need not be read on a first lecture of the manual). Leksah works well with the default settings and a desire to adapt better to your work habits comes only with extended use of Leksah. However, with time, you may use one or the other option to tailor Leksah to your personal preference. It is easy! Here it is explained how this works.

9.1 Layout

Leksah has always one special pane, which is called the active pane, and its name 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 the active one.

You can tailor the layout with the View menu to suit your work style better. Internally, the panes are arranged in a layout of a binary tree, where the leaves are horizontal or vertical splits. Every area can be split horizontally or vertically and panes can collapse. With the commands in the View menu you manipulate this tree to change the layout.

In the initial pane positions part of the Preferences, you can configure the placement of panes. Panes belongs to categories, and a category specifies a path where a pane will open.

The layout of the Leksah window contains areas which contain notebooks which contain panes. The division between the two areas is adjustable by the user by dragging a handle.

Panes can be moved between areas in the window. This can be done by dragging the notebook tab, and release it on the frame of another notebook. Alternatively you can use keystrokes (Shift Alt Arrow) to move panes around. 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 example quickly select one of many open source buffers.

The layout will be saved with sessions. The session mechanism will be explained in 9.2.¹

¹ Currently there is no way to load different layouts independent of the other data stored in a session.

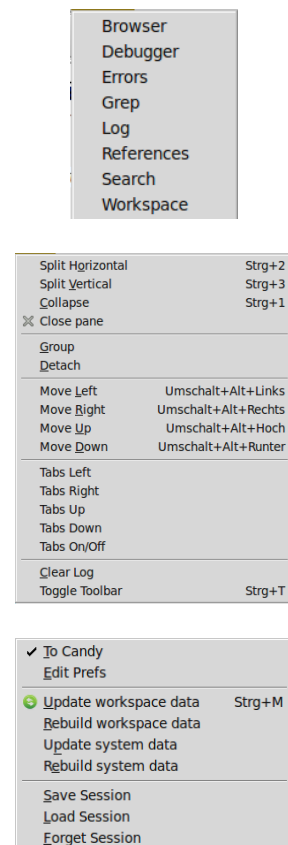


Figure 21: View and panes menus

9.1.1 Advanced layout: Group panes

A notebook cannot only contain single panes, but it can as well contain group panes, which have a layout on their own and may contain arbitrary other panes; the debug pane is an example for a group pane. This gives you the possibility to arrange the subpanes in a debugger pane as it fits you best.

A new group starts by selecting VIEW / GROUP from the menu. You have to give a unique name for the group. Then you can arrange panes in the group as you like. When closing a group, and the group is not empty, you have to confirm it.

9.1.2 Using Leksah with multiple displays: Detached windows

This feature allows you to move panes to a separate window on a separate display. This is as simple as: You select a notebook and choose VIEW / DETACH from the menu bar. Then the notebook is opened in a new window, which you can then move to another screen.

When you close the detached window, the pane goes back to the place where it was before detaching. The state of detached window is remembered, when you close Leksah, and they will be reopened when you restart Leksah.

It is possible to drag and drop panes between windows. But splitting and collapsing of panes is disabled for detached windows. So a recommended way to use this feature is to split a pane, arrange the panes that you want to detach in the area of the new notebook. Select the new notebook and detach.

The detached windows have no menu bar, toolbar and status bar on their own. This may be a problem, when you want to select a menu entry: the focus changes from a pane in the detached window to a pane in the main window, and you may not be able to do what you want. We recommend that you use keystrokes or context menus.

9.2 Session handling

When you close Leksah the current state is saved in the file *current.lksks* in the *~/leksah-*.** folder. A session contains the layout of the window, its content, the active package and some other state. When you restart Leksah it recovers the state from this information.

When you close a workspace, the session is saved in the folder of the workspace in a file named *workspacename.lksks*. When you open a workspace and Leksah finds a *workspacename.lksks* file together with the workspace file you are going to open, you get prompted if you want to open this session (this means mostly opening the files you had open before in the editor). This helps you to switch between different workspaces you are working on.

In addition, sessions can be stored and loaded with a name manually by using the session menu, but the need to use these features occurs rarely. The menu Configuration -> Forget Session is useful if you inadvertently changed the layout drastically and do not want the current session to be stored.

9.3 Shortcuts

You can configure the keystrokes by providing a keymap file, which should be in the `~/leksah-0.8` folder. The name of the key map file to be used can be specified in the Preferences dialog (without extension!).

A line in the `.keymap` file looks like:

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

Description of the key or key combination: Allowed modifiers are `<shift>` `<ctrl>` `<alt>` `<apple>` `<compose>`. `<apple>` is on a Microsoft keyboard the windows key (and on a Mac, obviously, the apple key!). `<compose>` is right ALT key, 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 name of the action can be any one of the *ActionDescr*'s given in the *action* function in the Module *IDE.Command*. The comment following will be displayed as tool tip for the toolbar button, if one exists for this action.

Every keystroke must at most be associated with one action, and 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 handled by the standard GTK mechanism, while other keystrokes are handled by Leksah.

Independently how you initiated an action, by a menu, a toolbar button or a keystroke, the keystroke with its associated ActionsString is displayed in the Status bar in the leftmost compartment.

9.4 Configuration files

Leksah stores its configuration in a directory called `~/leksah-*.*` under your home folder. Indexing the hidden Leksah directory with the version number avoids that changes to preferences file layout from version to version cause difficulties. Moving your preferences from a previous version can potentially be automatic.

The file *prefs.lkshp* stores the general preferences. It is a text file you could edit with a text editor, but more comfortable and safer is to do it in Leksah with the menu CONFIGURATION / EDIT PREFS from the menu.

If no preference file is found in your `.leksah-*.*` folder then the global *prefs.lkshp* in the installed data folder will be used. If a preference file get corrupted, which means Leksah does not start; it is then often sufficient to just delete the preference file.

The *source_packages.txt* file stores source locations for installed packages. It can be rebuild by calling *leksah-server* in a terminal with the `-o` or `-sources` argument. Do this after you moved your source or added sources for previous installed packages without sources.

Files for Keymaps (*keymap.lkshk*) and SourceCandy(*candy.lkshc*) may be stored in the `~/leksah-*.*` folder 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.

10 The Leksah Project

The development of an integrated Development Environment is a major undertaking and Leksah should become increasingly supported by the user community. If you are a user or just test Leksah, we would appreciate to here from you. Do not miss to report bugs, unclear or wrong information in the documentation and suggestion for improvements on the Leksah issue tracker.

Everyone is invited to contribute. Spreading the word, supplying error reports, providing keymap and candy files, providing a tutorial, caring for a platform will develop are all helpful and meaningful ways to contribute.

Leksah will advance over time and become more useful. Possible extension and enhancements are:

- Package editor with configurations
- Add traces to the Debugger
- Support for some kind of plug-ins or extensions
- Context enriched completion
- Version Control (Darcs, ...)
- Testing (Quick check,...)
- Object browser
- Coverage (HPC,...)
- Profiling (Ghc Profiler,...)
- Re-factoring (HaRe,...)
- FAD (Functional Analysis and Design,...)

Acknowledgment

Thanks to Ricardo Herrmann for making the new Leksah logo.

Thanks to Fabian Emmes, who created the icons for the module browser.

Thanks to Lakshmi Narasimhan for packaging for Fedora.

Thanks to all others who helped us with patches, bug reports and helpful feedback.

11 Appendix

11.1 Command line arguments

for leksah-server:

```
Leksah Haskell IDE (server) Usage: leksah-server [OPTION...] files...
-s                --system                Collects new information for installed packages
-r[Maybe Port]  --server[=Maybe Port]   Start as server.
-b              --rebuild                Modifier for -s and -p: Rebuild metadata
-o              --sources                Modifier for -s: Gather info about pathes to sources
-v              --version                Show the version number of ide
-h              --help                  Display command line options
-d              --debug                  Write ascii pack files
-e Verbosity     --verbosity=Verbosity   One of DEBUG, INFO, NOTICE, WARNING, ERROR, CRITICAL, ALERT, EMERGENCY
-l LogFile       --logfile=LogFile       File path for logging messages
-f              --forever                Don't end the server when last connection ends
-c              --endWithLast            End the server when last connection ends
```

for leksah:

```
Usage: leksah [OPTION...] files...
-v              --Version                Show the version number of ide
-l NAME         --LoadSession=NAME      Load session
-h              --Help                  Display command line options
-e Verbosity     --verbosity=Verbosity   One of DEBUG, INFO, NOTICE, WARNING,
                                         ERROR, CRITICAL, ALERT, EMERGENCY
```

11.2 The 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
-- ":@"         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   --ALPHA
"beta"         0x03b2   --BETA
"gamma"        0x03b3   --GAMMA
"delta"        0x03b4   --DELTA
"epsilon"      0x03b5   --EPSILON
"zeta"         0x03b6   --ZETA
"eta"          0x03b7   --ETA
"theta"        0x03b8   --THETA
```

11.3 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.
--The defined values for the keys can be found at
-- http://gitweb.freedesktop.org/?p=xorg/proto/x11proto.git;a=blob_plain;f=keydefs.h.
-- The names of the keys are the names of the macros without the prefix.
--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><shift>s ->      FileSaveAll  "Saves all modified buffers"
<ctrl>w      ->      FileClose    "Closes the current buffer"
<alt>F4      ->      Quit         "Quits this program"
--Edit
<ctrl>z      ->      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>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 occurrence of the text string"
<shift>F3     ->      EditFindPrevious "Find the previous occurrence 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 lines"
<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 current selected pane into one"
                                "Shows the tabs of the current notebook on the left"
                                "Shows the tabs of the current notebook on the right"
                                "Shows the tabs of the current notebook on the top"
                                "Shows the tabs of the current notebook on the bottom"
                                "Switches if tabs for the current notebook are visible"
<ctrl>t       ->      ToggleToolbar
                                "Shows the tabs of the current notebook on the bottom"
                                "Switches if tabs for the current notebook are visible"
--<ctrl>b       ->      BuildPackage
--<ctrl>r       ->      AddAllImports
--<ctrl><alt>r    ->      RunPackage
--<ctrl>j       ->      NextError
--<ctrl><shift>j ->      PreviousError
--<ctrl>o       ->      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
-- "For the next to entries the <ctrl> modifier is mandatory"
<ctrl>Page_Up ->      FlipUp       "Switch to next pane in reverse recently used order"
<ctrl>Page_Down ->      FlipDown    "Switch to next pane in recently used order"
<ctrl>space    ->      StartComplete "Initiate complete in a source buffer"
F6 -> DebugStep
F7 -> DebugStepLocal
F8 -> DebugStepModule
F9 -> DebugContinue
<ctrl>Return   ->      ExecuteSelection
<ctrl>m        ->      UpdateMetadataCurrent
```