Leksah: An Integrated Development Environment for Haskell

Jürgen Nicklisch-Franken
Hamish Mackenzie

July 21, 2009

Contents

List of Figures

License

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 is tested on Linux, Windows and Mac. 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 of 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 leksah.org. The source code for Leksah is hosted under code.haskell.org/leksah. The Leksah user Wiki is haskell.org/haskellwiki/Leksah. The Leksah mailing list can be accessed at projects.haskell.org/cgi-bin/mailman/listinfo/leksah. The current version of this manual can be found at leksah.org/leksah_manual.pdf. An issue tracker is at code.google.com/p/leksah/issues/list. You can contact the developers at info (at) leksah.org.

For the Programming language Haskell go to www.haskell.org. 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 0.6 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.

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.

1.2.2 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 GHC $\geq$6.10.1 and gtk2hs $\geq$ 0.10.0.

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

1.2.3 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

It is a good idea to install sources for your Haskell libraries when using Leksah. Leksah can then give you much better information from Metadata. Installing with *cabal install* will do that, and you can download the sources for GHC even after you installed it from a binary package. However compiling from sources is necessary for special packages, which perform some special magic, like generating header files or such. With sources installed Leksah will show you source comments for functions while auto-completing and will be navigate you to the sources of functions.

If you have any trouble installing and this manual will not help, please check the Wiki, the mailing list or contact the developers to find a solution. If it is a Leksah problem, then it is important to us and needs to be fixed.

We work on installers for Windows and Mac and packages for Debian/Ubuntu are in preparation. Further help needed!

2.1 Generic Installation Instructions

2.1.1 Install GHC (Glasgow Haskell Compiler)

At the time of writing Leksah supports GHC in the versions (6.10.*). For information about installing GHC go to `http://haskell.org/ghc`.

2.1.2 Install Cabal

At the time of writing Leksah requires at least version 1.6.0.1. For information about installing Gtk2Hs got to `http://haskell.org/cabal`.

On operating systems with package managers you may find that a Cabal package is available. Once again if you want good Metadata for the Cabal functions, try to build binaries on your machine.

2.1.3 Install Gtk2Hs

At the time of writing Leksah supports only the most current version (0.10.*). For information about installing Gtk2Hs got to `http://haskell.org/gtk2hs`.

If you are planning on installing a newer version, then you will need to make sure it is compatible with the version of GHC you have installed first.

On operating systems with a package manager you may wish to use a source OS package and build a binary OS package to install. You should be able to prevent the working directory from being cleaned, so it can be used for Leksah's Metadata. Doing this means that other OS packages depending on Gtk2Hs can still easily be installed.

If you compile Gtk2Hs manually, check the output of ./configure. At the end it will list the Haskell packages that are going to be build. Leksah needs GTK, glib and gtk-sourceview2.

Gtk2Hs conditionally compiles some code dependent of the version of the underlying GTK libraries. This can cause strange compiler errors when compiling Leksah later.

2.1.4 Install Leksah

EITHER: Install cabal-install from `http://haskell.org/cabal`. Then run “cabal install leksah –user”. This is the preferred way.

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.

2.1.5 Where Things Are Installed

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.

2.1.6 Post installation steps

If you *upgrade* to version 0.6 you should clean the Current.session and maybe the Default.prefs file from your .leksah configuration folder. This means that you loose your personal settings. As well you should clean the IDE.session files from the project folders you are working on (or at least avoid to open them when prompted).

If you use a customized keymap in your .leksah folder, you need to append new key-bindings.

New With version 0.6 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.

2.2 OS X (using MacPorts)

We have just finished building an Intel Mac dmg and we would like to know if it works ok. If you have a chance, could you install it and let us know if it runs ok.

`http://leksah.org/Leksah.dmg`

Before starting with the installer, this is our recommended procedure for installing on OS X at present.

2.2.1 Install MacPorts

Download and install MacPorts by following the instructions on `http://www.macports.org/install.php`.

2.2.2 Set up ~/.profile

Add the following to your ~/.profile then open a new terminal window and type "set" to make sure it has worked

```
export PATH=~/.cabal/bin:/opt/local/bin:$PATH
export XDG_DATA_DIRS=/opt/local/share
```

XDG_DATA_DIRS is only needed at run time, so if you have already built without it you can just add it now and it should help GTK find the files it needs.

2.2.3 Update MacPorts

Run "sudo port selfupdate". To make sure you have the latest Portfiles.

2.2.4 Set Variants To Use Quartz (Optional)

If you want to use the Quartz version of GTK+ (instead of the X11 version) then add the following to /opt/local/etc/macports/variants.conf.

```
+no_x11
-x11
+quartz
```

Warning 1 doing this will disable OpenGL support in GTK+ or Gtk2Hs (gtkglext).

Warning 2 for some reason GTK applications when they start will not be in the foreground, instead they will be hidden all your other running applications.

Warning 3 if you already have GTK MacPorts packages installed then you need to uninstall and reinstall the packages that support these variants

2.2.5 Install

Run the following to install leksah. The -k is important it keeps the source and .hi files for use in the Leksah metadata. Do not include gtkglext in the first line if you are building the Quartz version (as it is only supported in X11 builds).

```
sudo port install gtk2 cairo librsvg libglade2 gtksourceview2 gtk-chtheme
gtk2-clearlooks gtkglext
pkg-config --modversion gtksourceview-2.0
```

The `pkg-config` should output version `2.4.2` or greater. If it does not check your `PATH` has `/op/local/bin` before any other folders with `pkg-config`. Once this is sorted you can move on to running...

```
sudo port -k install ghc gtk2hs hs-cabal
cabal install leksah
```

If you have errors at this point it is a good idea to check "`ghc-pkg list`"

2.2.6 Make It Look Nice

Run gtk-chtheme and choose one of the Clearlooks themes.

2.2.7 Point Leksah At The Source

Run "leksah" and when it asks for "paths under which haskell source packages may be found" add the following to the list.

```
/opt/local/var/macports/build/_opt_local_var_macports_sources_rsync.macports.org_release_ports_lang_ghc/work/ghc-6.10
/opt/local/var/macports/build/_opt_local_var_macports_sources_rsync.macports.org_release_ports_devel_gtk2hs/work/gtk2
```

2.3 Ubuntu

2.3.1 Install Prerequisites

Open up the package manager and make shure the following packages are installed:

- glib-devel
- gtksourceview2-devel
- make
- gcc
- g++
- libgmp3-dev

2.3.2 Install GHC (Once 6.10.1 is in the universe repository)

At the time of writing only 6.8.2 was available in the Ubuntu universe repository.

If you have ghc 6.8 installed then we recomend you uninstall it to avoid conflicts.

```
sudo apt-get remove ghc6
sudo apt-get build-dep ghc6
wget http://www.haskell.org/ghc/dist/6.10.1/ghc-6.10.*-src.tar.bz2
wget http://www.haskell.org/ghc/dist/6.10.1/ghc-6.10.*-src-extralibs.tar.bz2
tar xjf ghc-6.10.1-src.tar.bz2
tar xjf ghc-6.10.1-src-extralibs.tar.bz2
cd ghc-6.10.1
./configure --prefix=/home/username/ghc
make
make install
export $PATH=$PATH:/home/username/ghc/bin
```

2.3.3 Install Cabal

```
wget http://www.haskell.org/cabal/release/cabal-install-0.6.2/cabal-install-0.6.2.tar.gz
tar -xvjpf cabal-install-0.6.2.tar.gz
cd cabal-install-0.6.2
./bootstrap
```

2.3.4 Install Gtk2Hs

Check the output of ./configure, at the end it should list the haskell packages it is going to build. Leksah needs gtksourcevie, gtk and glib (if you have installed glib-devel and gtksourceview2-devel then these should be enabled). If there are any that are not going to be installed that you would like, then you most likely need to install the corresponding -devel ubuntu package.

```
  wget http://downloads.sourceforge.net/gtk2hs/gtk2hs-0.10.0.tar.gz
  tar -xvjpf gtk2hs-0.10.0.tar.gz
  cd gtk2hs-0.10.0.tar.gz
  ./configure
  make
  sudo make install
```

2.3.5 Add Cabal To Your PATH

Cabal can install packages globally or local to the current user (use –user and –global to select which). We recommend you add the following to your ~/.bash_profile

```
  PATH=$PATH:/home/sean/.cabal/bin
  export PATH
```

If you change your path you will have to restart X to make it apply automatically or you can type

```
  source ~/.bash_profile
```

each time you open a new terminal until you get a chance to restart X.

2.3.6 Install Leksah

```
cabal update
cabal install leksah
```

2.4 MS Windows

In the future I hope we can provide a Windows installer. For now the situation is difficult, because the latest gtk2hs Windows installer that works for Leksah is 0.10.0, which only works with GHC 6.10.2. Build 0.10.1 of gtk2hs has an issue with the the GTKSourceView2 component. A windows installer for GHC 6.10.4 is missing. Please write to the gtk2hs users mailing list if you want this situation to improve.

This is what you can do to get leksah running on Vista:

1. Install Cygwin with the online installer from http://www.cygwin.com/ Select packages wget, curl and gcc-core in adition to the standard selection. I used the 1.7.0 version of Cygwin setup, but it is still beta.

2. Install your GHC from http://www.haskell.org/ghc.

3. Install gtk2hs from http://www.haskell.org/gtk2hs/ or Sourceforge.net.

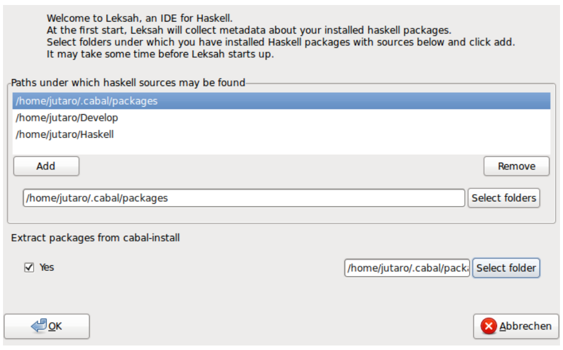
4. Download the GHC sources from http://www.haskell.org/ghc. Open a Cygwin shell, make a directory Haskell, copy the source tarballs here and unpack them. Take care that the directory name has no blanks in its path!

5. cabal install leksah. (Guess it is already installed with 6.10.2?)

6. You may wish to install a full Unicode monospace font if you want to use the source candy feature of leksah. e.g. Everson Mono or Deja Vu Sans Mono or perhaps the GNU FreeFonts which are aesthetically pleasing.

7. Start leksah and give your Haskell directory as source root. Select the right font from Help/Prefs.

8. Have fun with leksah, but remember it is a beta version!


Welcome to Leksah, an IDE for Haskell.
At the first start, Leksah will collect metadata about your installed haskell packages.
Select folders under which you have installed Haskell packages with sources below and click add.
It may take some time before Leksah starts up.

Paths under which haskell sources may be found
/home/jutaro/.cabal/packages
/home/jutaro/Develop
/home/jutaro/Haskell

/home/jutaro/.cabal/packages

Extract packages from cabal-install
☑ Yes
/home/jutaro/.cabal/pack

Figure 1: FirstStart dialog

3 First start of Leksah

1. When you start Leksah for the first time, the first start dialog appears (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.

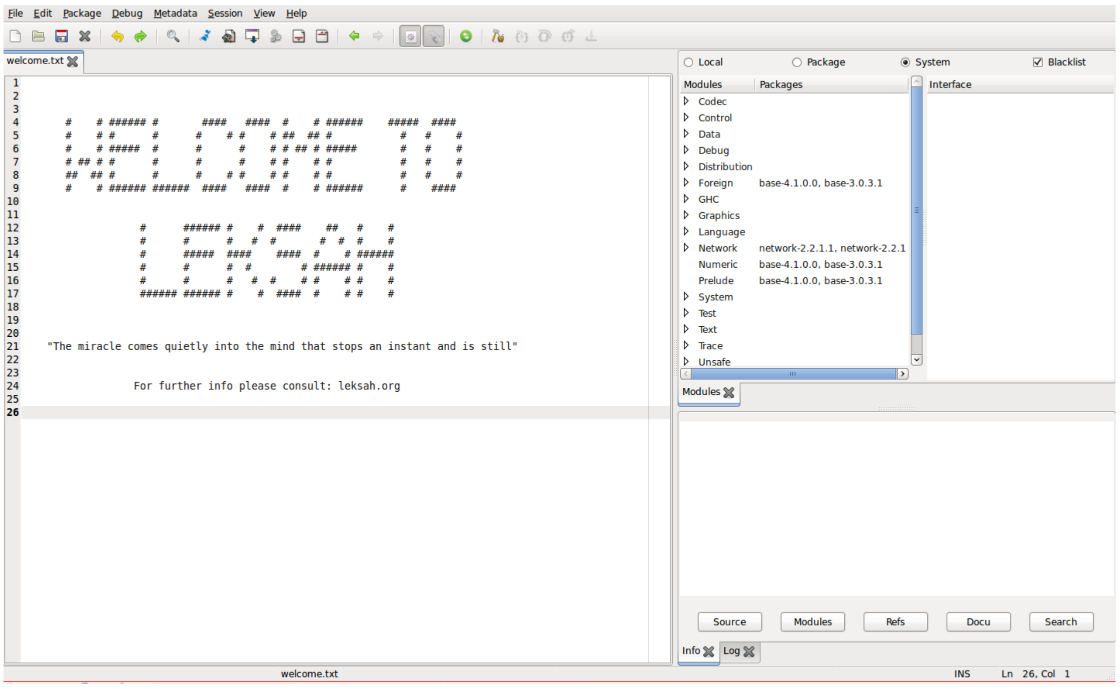
2. If you use cabal install add the cabal install package directory to the “extract packages from cabal-install” section. This means that leksah decompresses and untars the sources, so they can be scanned by Leksah. This is kind of a hack currently.

> Later you can change this settings in the preferences pane in Leksah and you can rebuild the metadata at any time. Details about metadata collection can be found here: 6.5.

> 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.

1. Now Leksah collects information about all installed packages on your system. So it may take some time, but at further starts only information for new packages will be installed. Their will eventually be a bunch of errors on your command line, but don’t worry, it only means that Leksah has not succeeded to extract the source locations and comments for certain modules or packages.

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



```



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


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



    "The miracle comes quietly into the mind that stops an instant and is still"


                  For further info please consult: leksah.org

```

( ) Local ( ) Package [✓ System] [✓ Blacklist]

Modules	Packages
Codec	
Control	
Data	
Debug	
Distribution	
Foreign	base-4.1.0.0, base-3.0.3.1
GHC	
Graphics	
Language	
Network	network-2.2.1.1, network-2.2.1
Numeric	base-4.1.0.0, base-3.0.3.1
Prelude	base-4.1.0.0, base-3.0.3.1
System	
Test	
Text	
Trace	
Unsafe	

Interface

welcome.txt Ln 26, Col 1

Figure 2: After start

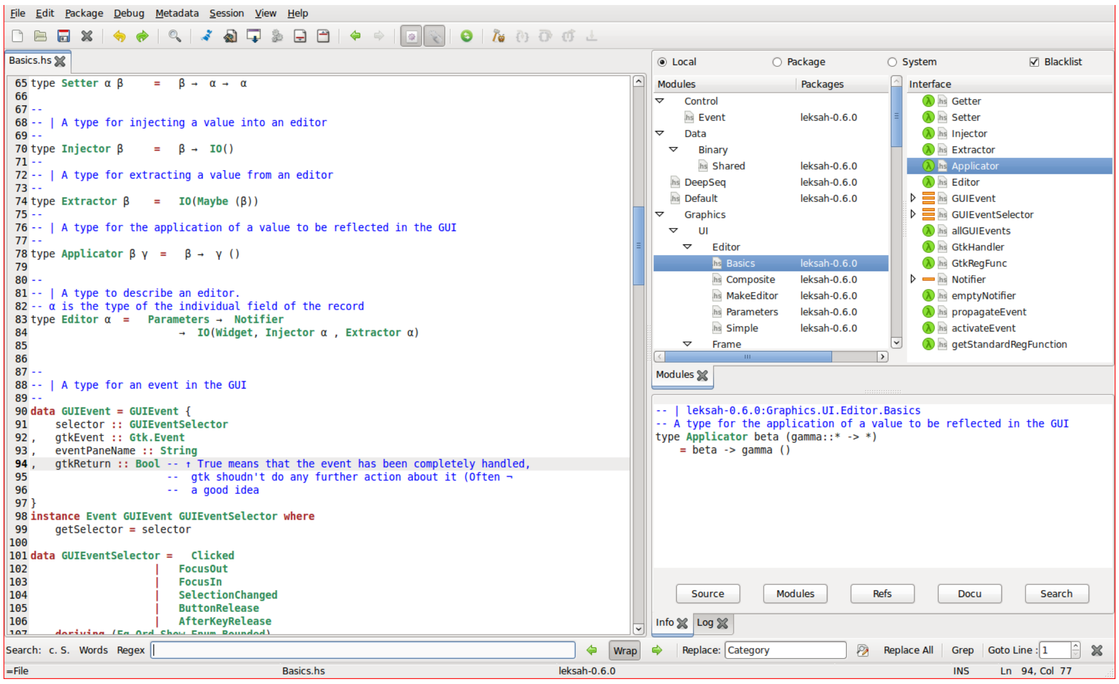
3. One way to start up, and probably the best way if you already have experience with Haskell, will be to open an existing project. So select Package/OpenPackage from the menu and open a Cabal file of some project. A typical Leksah window may then look like 3.

Alternatively you can start with a

3.1 Hello World example

Create a Package with one Executable (name of both can be the same)

- Package -> New and use the "Create Folder" button to make a new folder for the package. Fill in your package identifier including a version number.
- Click on the Dependencies tab (in the package editor). Type in "base" and click "Add" (this is like the references in visual studio projects)
- Click on the Executables tab (in the package editor). Fill in an executable name and put “Main.hs” in the “file with main function” and click "Add".
- Click on Build “1 Build Info”. Add “src” to the list. This way your source files should all end up in the src sub directory.



```
type Setter α β     =   β →  α →  α

--
-- | A type for injecting a value into an editor
--
type Injector β     =   β →  IO()
--
-- | A type for extracting a value from an editor
--
type Extractor β    =   IO(Maybe (β))
--
-- | A type for the application of a value to be reflected in the GUI
--
type Applicator β γ  =   β →  γ ()

--
-- | A type to describe an editor.
-- α is the type of the individual field of the record
type Editor α  =   Parameters →  Notifier
                        →  IO(Widget, Injector α , Extractor α)


--
-- | A type for an event in the GUI
--
data GUIEvent = GUIEvent {
    selector :: GUIEventSelector
,   gtkEvent :: Gtk.Event
,   eventPaneName :: String
,   gtkReturn :: Bool -- ↑ True means that the event has been completely handled,
                      --  gtk shoudn't do any further action about it (Often ¬
                      --  a good idea
}
instance Event GUIEvent GUIEventSelector where
    getSelector = selector

data GUIEventSelector =   Clicked
                      |   FocusOut
                      |   FocusIn
                      |   SelectionChanged
                      |   ButtonRelease
                      |   AfterKeyRelease
    deriving (Eq,Ord,Show,Enum,Bounded)
```

Modules	Packages
Control	
Event	leksah-0.6.0
Data	
Binary	
Shared	leksah-0.6.0
DeepSeq	leksah-0.6.0
Default	leksah-0.6.0
Graphics	
UI	
Editor	
Basics	leksah-0.6.0
Composite	leksah-0.6.0
MakeEditor	leksah-0.6.0
Parameters	leksah-0.6.0
Simple	leksah-0.6.0
Frame	

Interface
- Getter
- Setter
- Injector
- Extractor
- Applicator
- Editor
- GUIEvent
- GUIEventSelector
- allGUIEvents
- GtkHandler
- GtkRegFunc
- Notifier
- emptyNotifier
- propagateEvent
- activateEvent
- getStandardRegFunction

```
-- | leksah-0.6.0:Graphics.UI.Editor.Basics
-- A type for the application of a value to be reflected in the GUI
type Applicator beta (gamma::* -> *)
    = beta -> gamma ()
```

Replace: Category   Goto Line : 1

Figure 3: Leksah with open project

- Click Save and Close

Add the Main module

- Click on the Modules tab (this is like the class browser in visual studio). Right click and select Add Module. Put “Main” in the New Module box.
- Add *main* = *putStrLn "Hello World"*

Run it

- Package -> Configure
- Package -> Build
- Package -> Run
- Output will be in the Log window

Debug it

- Switch debugger Mode on.
- Debug -> Show Debugger

- Select the word “main” in your code
- Right click and choose “Eval”

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.

For a further info refer to: `http://en.wikibooks.org/wiki/Haskell`.

4 The Editor

The central functionality needed for development is to edit Haskell source files. Leksah uses the GtkSourceView2 widget for this. It provides editing, undo/redo, syntax highlighting and other features. In the file menu (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 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).

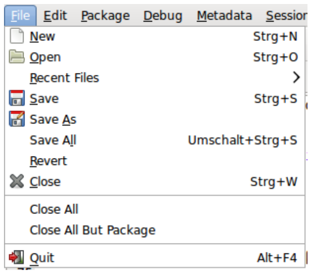


Figure 4: File menu

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. (See 4.4 for a better way to switch between source files.

On the right side in the status bar 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 (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.

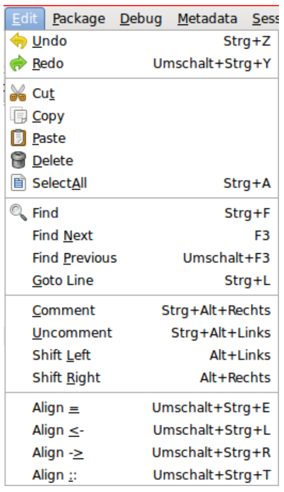


Figure 5: Edit menu

4.1 Find and Replace

Leksah supports searching in text files. When you select Edit/Find from the menu the find bar will open (6) and you can type in a text string. Alternatively you can hit Ctrl-F or select a text and hit Ctrl-F (This is a standard keystrokes. Keystrokes can be configures, see 8.3). Hitting the up and down arrow will bring you to the next/previous occurrence

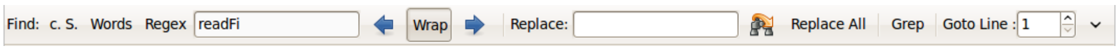
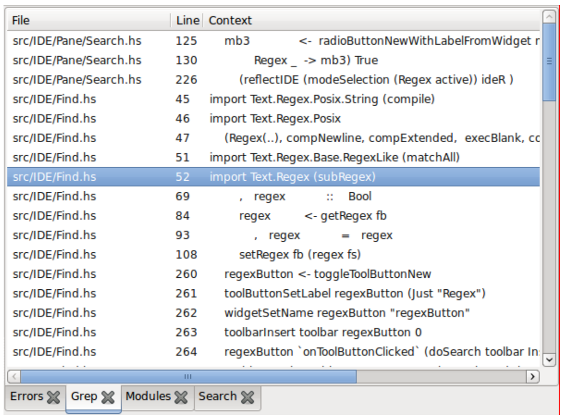


Figure 6: Find bar



File	Line	Context
src/IDE/Pane/Search.hs	125	mb3 <- radioButtonNewWithLabelFromWidget r
src/IDE/Pane/Search.hs	130	Regex _ -> mb3) True
src/IDE/Pane/Search.hs	226	(reflectIDE (modeSelection (Regex active)) ideR)
src/IDE/Find.hs	45	import Text.Regex.Posix.String (compile)
src/IDE/Find.hs	46	import Text.Regex.Posix
src/IDE/Find.hs	47	(Regex(..), compNewline, compExtended, execBlank, cc
src/IDE/Find.hs	51	import Text.Regex.Base.RegexLike (matchAll)
src/IDE/Find.hs	52	import Text.Regex (subRegex)
src/IDE/Find.hs	69	, regex :: Bool
src/IDE/Find.hs	84	regex <- getRegex fb
src/IDE/Find.hs	93	, regex = regex
src/IDE/Find.hs	108	setRegex fb (regex fs)
src/IDE/Find.hs	260	regexButton <- toggleToolButtonNew
src/IDE/Find.hs	261	toolButtonSetLabel regexButton (Just "Regex")
src/IDE/Find.hs	262	widgetSetName regexButton "regexButton"
src/IDE/Find.hs	263	toolbarInsert toolbar regexButton 0
src/IDE/Find.hs	264	regexButton `onToolButtonClicked` (doSearch toolbar In

Figure 7: Grep pane

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(c.S.), for searching only whole worlds (Words) and for wrapping around (Wrap), 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.

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.

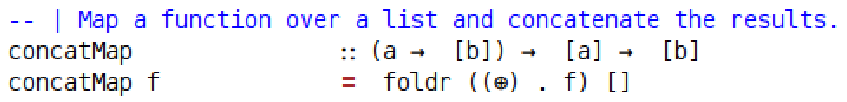
To replace a text enter the new text in the replace entry and select replace or replace all.

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

4.1.1 Grep

For this feature the grep program must be on your path. You can then enter a search string in the find bar and hit the Grep button. This will search for all occurrences for the string in the folder and subfolder of the current package. Greps supports the search for regular expressions. A pane will open (7), and in every displayed line the expression was found. By clicking on the line, an editor is opened or brought to front and the focus is set to the selected line. You can navigate between lines with the up and down keys.

4.2 Source Candy



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

Figure 8: 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!

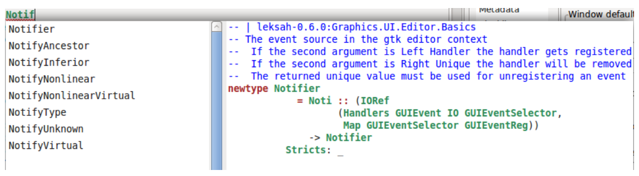
Leksah reads and writes files encoded in UTF-8. So you can edit Unicode Haskell source files. When you want to do this, switch of source candy, because otherwise Unicode characters may be converted to ASCII when saving the file.

4.3 Completion

Leksah has the ability to auto complete identifiers in text you type. Additionally the Package, Module and Type of the id gets displayed if selected.

This completion mode can either be always on, or only be activated on pressing Ctrl+Space (or a user defined keystroke). You can choose between these two possibilities in the Preferences.

Leksah currently uses all names provided by the package scope for completion. So it has no context sensitiveness, and doesn't provide locally defined names.



```
Notif
Notifier
NotifyAncestor
NotifyInferior
NotifyNonlinear
NotifyNonlinearVirtual
NotifyType
NotifyUnknown
NotifyVirtual
```

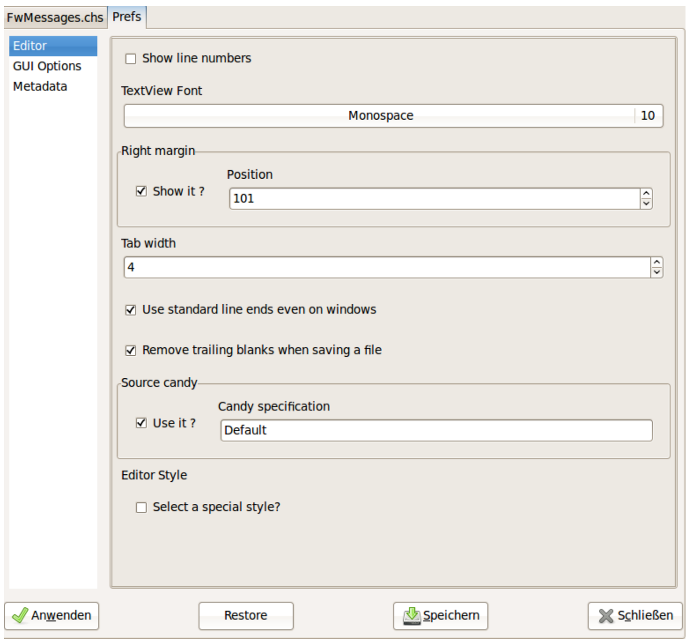
```
-- | leksah-0.6.0:Graphics.UI.Editor.Basics
-- The event source in the gtk editor context
--  If the second argument is Left Handler the handler gets registered
--  If the second argument is Right Unique the handler will be removed
--  The returned unique value must be used for unregistering an event
newtype Notifier
            = Noti :: (IORef
                   (Handlers GUIEvent IO GUIEventSelector,
                    Map GUIEventSelector GUIEventReg))
              -> Notifier
          Stricts: _
```

Figure 9: Completion

4.4 Using the Flipper to Switch 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.

4.5 Editor Preferences



☐ Show line numbers

TextView Font: Monospace 10

Right margin
☑ Show it ?
Position: 101

Tab width: 4

☑ Use standard line ends even on windows

☑ Remove trailing blanks when saving a file

Source candy
☑ Use it ?
Candy specification: Default

Editor Style
☐ Select a special style?

Figure 10: Editor Preferences

When selecting Configuration / 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).
[1]

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.

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 5.3.
- For information about editor preferences go to 4.5.

[1]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. This may not work anymore since switching to unicode sources?

5 Packages

Leksah does not only support editing Haskell source files, but as well building the application or program you are developing. The concept of a package is used to handle a unit of work for the development of some library or executable. One instance of Leksah can only open one package at a time. (It is a wish of many users to make this more flexible in the way of e.g. Eclipse Workspaces). 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 currently active 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, this is by convention 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.

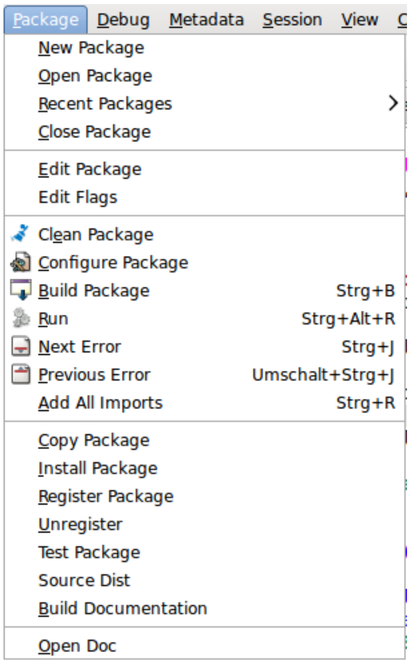


Figure 11: Package Menu

5.1 Package Editor

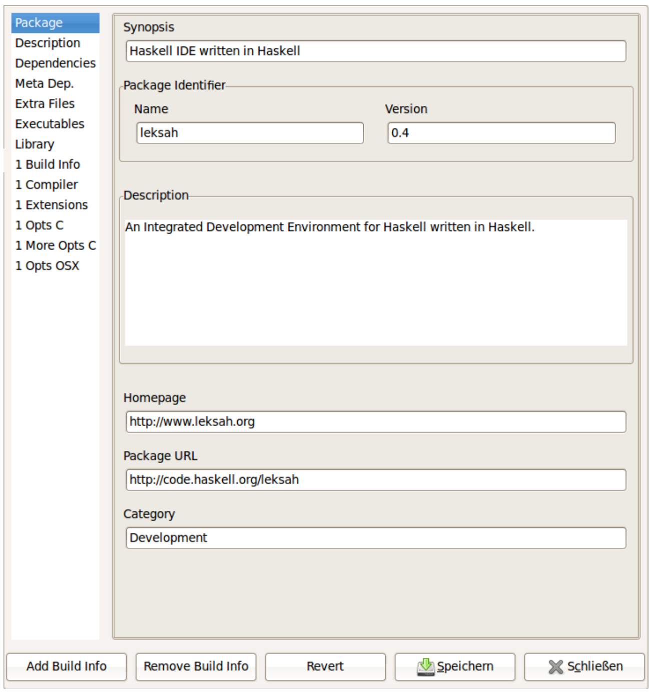
The package editor (12) is an editor for cabal files. Since cabal files offer complex options the editor is quite complex. For a complete description of all options see the Cabal User's Guide. The package editor does currently not support the cabal configurations feature. If you need cabal configurations, 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. (We plan to enhance the editor to support configurations in the future).

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 Executable 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:



Synopsis: Haskell IDE written in Haskell

Package Identifier

Name: leksah

Version: 0.4

Description: An Integrated Development Environment for Haskell written in Haskell.

Homepage: http://www.leksah.org

Package URL: http://code.haskell.org/leksah

Category: Development

Figure 12: PackageEditor 1

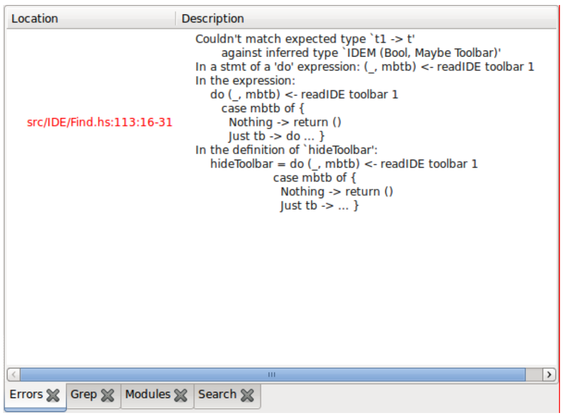
- where the sources can be found (relative to the root folder of the project, which is the one with the cabal file).
- what additional non-exposed or non main modules your project includes
- compiler flags
- used language extensions in addition to Haskell 98 (These can also be specified in the source 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.

5.2 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, 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



```
Location                    Description
                              Couldn't match expected type `t1 -> t'
                                   against inferred type `IDEM (Bool, Maybe Toolbar)'
                              In a stmt of a 'do' expression: (_, mbtb) <- readIDE toolbar 1
                              In the expression:
                                 do (_, mbtb) <- readIDE toolbar 1
                                   case mbtb of {
 src/IDE/Find.hs:113:16-31           Nothing -> return ()
                                     Just tb -> do ... }
                              In the definition of `hideToolbar':
                                 hideToolbar = do (_, mbtb) <- readIDE toolbar 1
                                                  case mbtb of {
                                                    Nothing -> return ()
                                                    Just tb -> ... }
```

Figure 13: Error Pane

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).

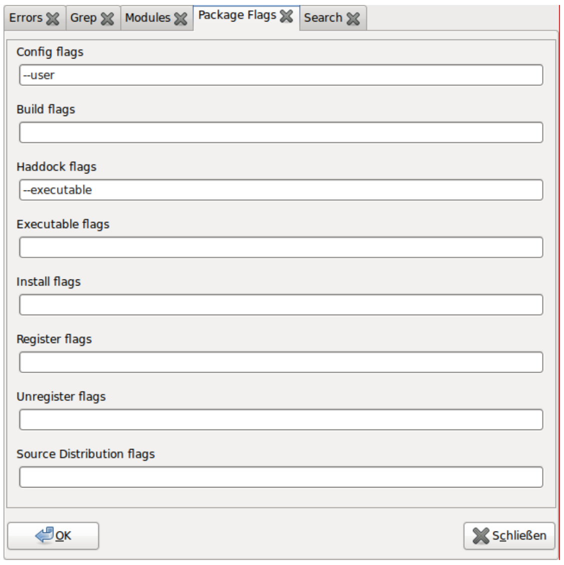
The error pane (13) shows the errors in the form of a table and provides the same functionality you find in the log, but it may be more convenient to use.

5.3 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 (sadly this step is OSX and Linux only at this point)
- waits for any running build processes to finish
- saves all the modified files
- starts a new build

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.



Config flags: --user

Build flags:

Haddock flags: --executable

Executable flags:

Install flags:

Register flags:

Unregister flags:

Source Distribution flags:

Figure 14: Package Flags

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

LINKING with background build on or off.

5.4 Build system flags

As you can see in the package menu (11) 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.

If you want to link with the locally installed libraries (ghc-pkg) Haskell packages locally, set: config flags –user

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 you need to add an import, you can choose *Add*

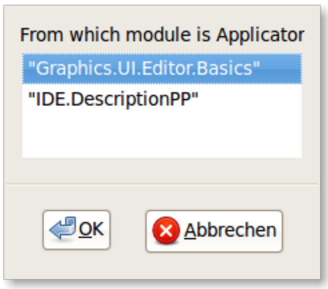


Figure 15: Import dialog

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 that exports this identifier, a dialog will appear which queries you about the module you want to import from (15).

Leksah then adds a line or an entry to the import list of the affected module with the compiler error. It adds as well an entry in the lop pane. 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

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.

6 Module Browser and Metadata

Leksah collects data about the exported modules of all installed Haskell packages on your system. It does this by reading the Haskell interface files (from GHC). 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 sort of the exported identifier?

If the source was found:

- What is the comment for this identifier?
- Where is the source with 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 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. 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.

The current Leksah metadata does not contain definitions local to a module. So names which are not exported will not be found in the metadata. We plan to change this for the current project for future versions. [2]

You can see the sort of expression by the icon before the identifier. The following sorts of identifiers are currently known in the Metadata:

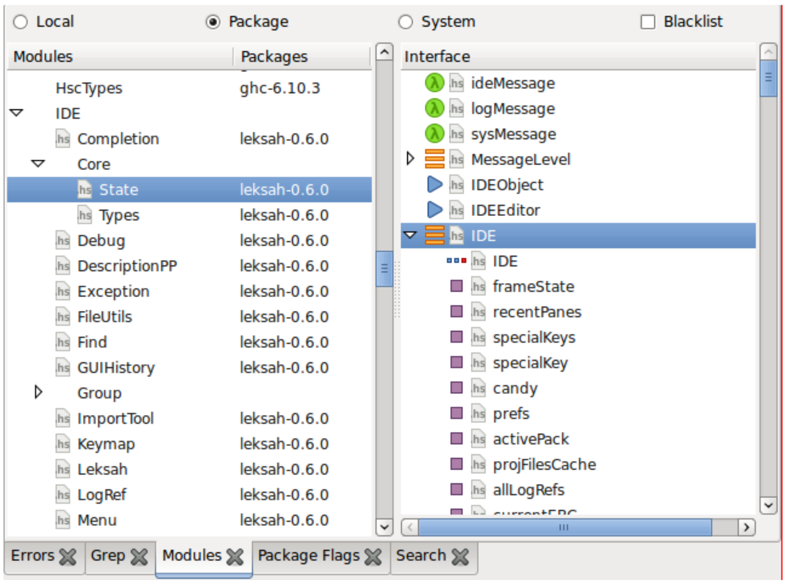
sort	symbol
function	🟢
data	☰
constructor	▪▪▪
slot	🟪
type	➡
newtype	▬
class	▶
member	🔵
instance	▸
rule	△

Table 1: Sorts of identifiers

6.1 The Modules Pane

In the modules pane (16) 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, it's name is displayed in the third subdivision from the left of the status bar.)

[2]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.


Local [✓ Package] System ☐ Blacklist

Modules	Packages
HscTypes	ghc-6.10.3
IDE	
Completion	leksah-0.6.0
Core	
State	leksah-0.6.0
Types	leksah-0.6.0
Debug	leksah-0.6.0
DescriptionPP	leksah-0.6.0
Exception	leksah-0.6.0
FileUtils	leksah-0.6.0
Find	leksah-0.6.0
GUIHistory	leksah-0.6.0
Group	
ImportTool	leksah-0.6.0
Keymap	leksah-0.6.0
Leksah	leksah-0.6.0
LogRef	leksah-0.6.0
Menu	leksah-0.6.0

Interface
- ideMessage
- logMessage
- sysMessage
- MessageLevel
- IDEObject
- IDEEditor
- IDE
  - IDE
  - frameState
  - recentPanes
  - specialKeys
  - specialKey
  - candy
  - prefs
  - activePack
  - projFilesCache
  - allLogRefs

Figure 16: Modules pane

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 current project. The *Package* scope shows all modules of the package and all packages which the current package depends on. The *System* scope shows all modules of installed packages of the system. (You can get the list of all installed packages with *ghc-pkg* list. Leksah scans the user and the system package database, when both are present).

If the Blacklist toggle button is selected, the packages in the blacklist are not displayed. The information is accessible, but the modules are excluded from the modules browser. [3]. The Blacklist can be edited in the preferences pane.

If you select a module in the modules list, its interface is displa
yed 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 can be done with selecting *Go to definition* from the context menu.

This is REEXPORTED from another module.

[3]I invented the blacklist mainly for the GHC package, when will they use hierarchical module names?

By selecting an element in the Interface List the so called Info Pane is shown with additional information.

The quickest way to edit some project file is to go to the modules pane, select local scope and find the module by entering text, and double click for editing the file.

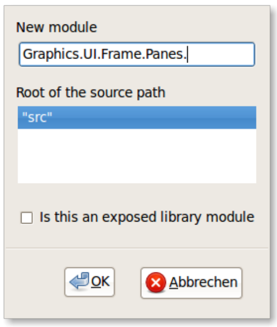


Figure 17: Construct module dialog

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 (17). You have to enter the name of the module, the source path to use if alternatives exist and, when the project is a library, if the module is exposed. Leksah will construct the directory, modify the cabal file and construct an empty module file from a template.

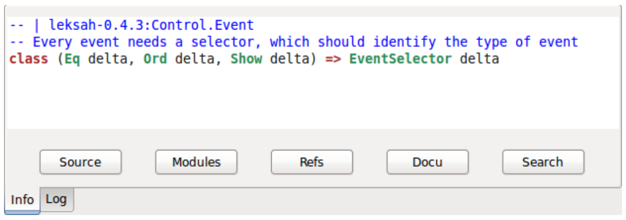
The modification of the cabal file will currently only happen, if it does not contain configurations.

6.2 The Info Pane

The Info Pane (18) shows information about an interface element, which may be a function, a class, a data definition, a type .... It shows the identifier, the package and module that it is exported by, it's Haskell type and if found a comment.

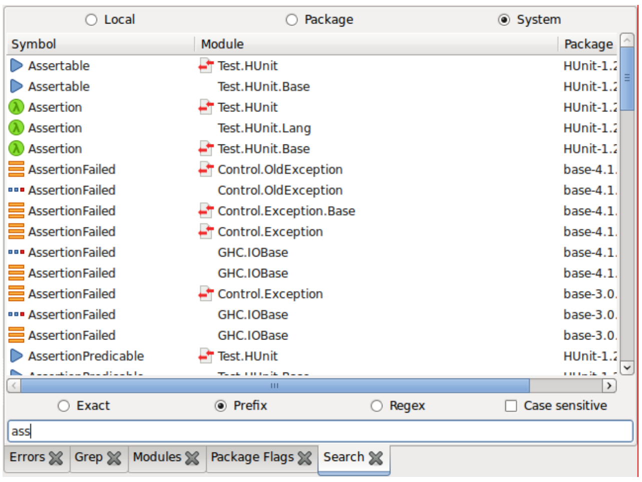
If you select initiate an identifier search in an editor, and information about this identifier is available in the package scope, it is automatically displayed in the info pane. Again, the easiest way to do this is to double click on an identifier while pressing Ctrl.

Remember that only statically collected information is available this way, and only about items which are exported by some module.



```
-- | leksah-0.4.3:Control.Event
-- Every event needs a selector, which should identify the type of event
class (Eq delta, Ord delta, Show delta) => EventSelector delta
```

Figure 18: Info pane



Local Package [✓ System]

Symbol	Module	Package
Assertable	Test.HUnit	HUnit-1.
Assertable	Test.HUnit.Base	HUnit-1.
Assertion	Test.HUnit	HUnit-1.
Assertion	Test.HUnit.Lang	HUnit-1.
Assertion	Test.HUnit.Base	HUnit-1.
AssertionFailed	Control.OldException	base-4.1.
AssertionFailed	Control.OldException	base-4.1.
AssertionFailed	Control.Exception.Base	base-4.1.
AssertionFailed	Control.Exception	base-4.1.
AssertionFailed	GHC.IOBase	base-4.1.
AssertionFailed	GHC.IOBase	base-4.1.
AssertionFailed	Control.Exception	base-3.0.
AssertionFailed	GHC.IOBase	base-3.0.
AssertionFailed	GHC.IOBase	base-3.0.
AssertionPredicable	Test.HUnit	HUnit-1.

Exact [✓ Prefix] Regex Case sensitive

ass

Figure 19: 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 Hoogle, depending on the configuration in the Preferences.

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

6.3 The Search Pane

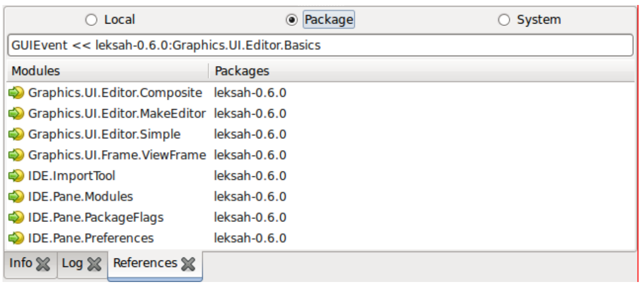
You can search for an identifier in the metadata by typing in characters in the entry at the bottom of the pane. The search result depends on the settings in the search pane (19). You can choose:

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 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. It does not depend on the selection in the search pane.


○ Local ● Package ○ System

GUIEvent << leksah-0.6.0:Graphics.UI.Editor.Basics

Modules	Packages
Graphics.UI.Editor.Composite	leksah-0.6.0
Graphics.UI.Editor.MakeEditor	leksah-0.6.0
Graphics.UI.Editor.Simple	leksah-0.6.0
Graphics.UI.Frame.ViewFrame	leksah-0.6.0
IDE.ImportTool	leksah-0.6.0
IDE.Pane.Modules	leksah-0.6.0
IDE.Pane.PackageFlags	leksah-0.6.0
IDE.Pane.Preferences	leksah-0.6.0

Figure 20: References Pane

6.4 The References Pane

As said in the end of the last section, this pane shows which modules uses 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 a text search on the selected element is initiated.

6.5 Metadata collection

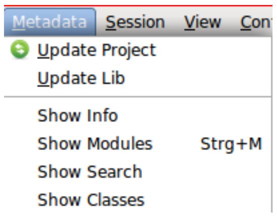


Figure 21: Metadata menu

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

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.

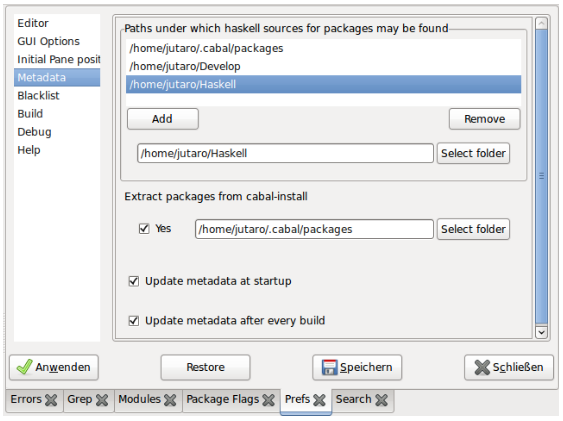
If you select Metadata / Update Lib Leksah checks if a new library has installed and if this is the case collects metadata for it.

In the Metadata part of the preferences (22) you can edit the settings concerning metadata collection.

Metadata collection is a difficult point, because it depends on the installation, environment, the installed packages, etc. However, to make good use of leksah it is highly desirable to have metadata with sources available for the packages you really need. So here we explain how metadata collection works:

1. Since cabal install has sources only as compressed tar archives on your machine, leksah needs to unpack this. You can see if this works by browsing the folders. You can initiate this step by calling leksah -x - n. If only root has write access to some cabal packages do a: sudo leksah -x -n (–Extract –NoGui).

2. Now leksah looks for all cabal files it can find below the source folders. 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



Paths under which haskell sources for packages may be found
/home/jutaro/.cabal/packages
/home/jutaro/Develop
/home/jutaro/Haskell

/home/jutaro/Haskell

Extract packages from cabal-install
☑ Yes /home/jutaro/.cabal/packages

☑ Update metadata at startup

☑ Update metadata after every build

Figure 22: Metadata Preferences

package has been correctly found. You can run this step by: leksah -s -n (–Sources –NoGui).

3. The metadata collection itself proceeds in two steps: A) Extract info from .hi files, which is usually no problem. B) Add source locations and comments by parsing the sources. This can be a problem because of preprocessing, header files, language extensions, etc. The result 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 for every package a metadata file 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 -r -n option (–Rebuild –NoGui).
You can update the metadata with: leksah -c -n (–Collect –NoGui)
If you have a problem with a certain package, remove its metadata file , e.g. rm ~/.leksah/ghc-.../binary*. And then do a collect: leksah -c -n. Look for error messages in the Console to see if some problem is reported.

4. For the current package a slightly different procedure is used, because leksah not only looks for exported library entities, but for all exports of every module. The written metadata file has the extension .packw.

7 Debugger

You can switch Debugger mode on from the toolbar:

Switch debugger Mode on or off.

After switching debugger mode on, you can see that packages and modules for your current project are loaded into GHCi.

When debugger mode is on, you see that the menu entries from the Debug menu are no longer disabled, and the context menu of source buffers have entries enabled as well, that were disabled when not in debugger mode.

You can open the debugger group pane by choosing Debug / Show Debugger.

You can now:

- EVALUATE expressions in the interpreter and observe the result.
  Select the expression in a source buffer. Select eval from the context menu. The result of the evaluation is shown in the log window and as *it* in the variables pane.
  The debugger has a pane which is for writing down evaluation expressions. The pane is a Haskell source buffer, which has the reserved name _Eval. Its contents is saved with the session.
  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. Select Type from the context menu.

- Get INFO about an identifier: Select Info from the context menu.

- Get THE KIND of a type.: Select Kind

- STEP through code: Select the expression in a source buffer. Select step from the context menu.

- or STEP through code: Set BREAKPOINTS by putting the cursor at the breakpoint and select *set breakpoint* from the context menu. Run your application or testcases and start stepping at the breakpoint.

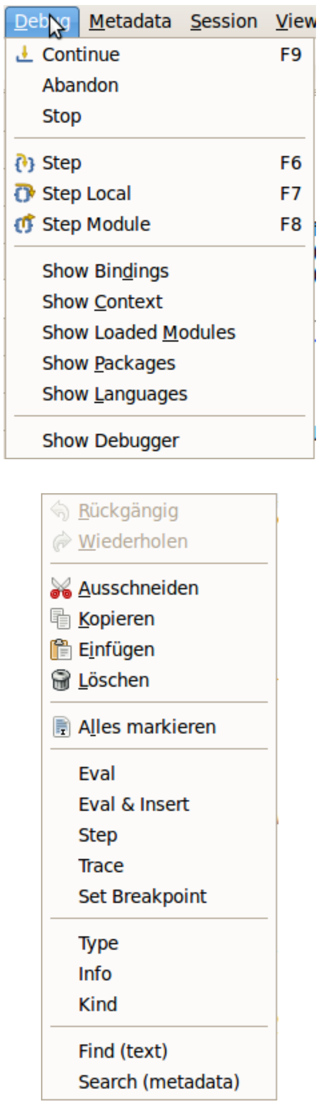
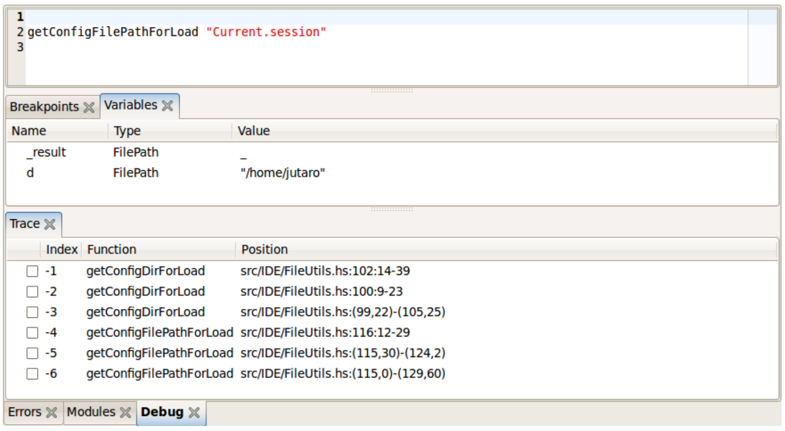


Figure 23: Debug & Buffer menu



```
getConfigFilePathForLoad "Current.session"
```

Name	Type	Value
_result	FilePath	_
d	FilePath	"/home/jutaro"

	Index	Function	Position
☐	-1	getConfigDirForLoad	src/IDE/FileUtils.hs:102:14-39
☐	-2	getConfigDirForLoad	src/IDE/FileUtils.hs:100:9-23
☐	-3	getConfigDirForLoad	src/IDE/FileUtils.hs:(99,22)-(105,25)
☐	-4	getConfigFilePathForLoad	src/IDE/FileUtils.hs:116:12-29
☐	-5	getConfigFilePathForLoad	src/IDE/FileUtils.hs:(115,30)-(124,2)
☐	-6	getConfigFilePathForLoad	src/IDE/FileUtils.hs:(115,0)-(129,60)

Figure 24: Debug Pane

- **Attention:** If you select a text, the breakpoint will be set for the implementation of this identifier

- You can observe the current breakpoints in the breakpoints pane. You can remove breakpoints from this pane.

- Use the toolbar (or shortcuts) for stepping:

Step (F6), Step local (F7)
Step in module (F8), Continue (F9)

- 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. E.g. ":set ..."

For further information about the GHCi debugger, please read the section 2.5. (The GHCi Debugger) of the GHC user manual.

The Debugger is the newest addition to Leksah and is still in an experimental stage. So please do not expect to much in the moment.

8 Configuration

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

8.1 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 the active one.

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.

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.

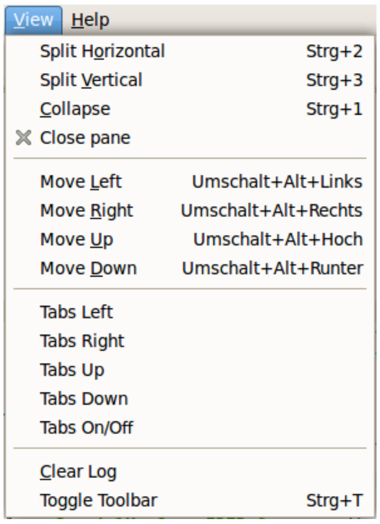


Figure 25: View menu

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 8.2.[4]

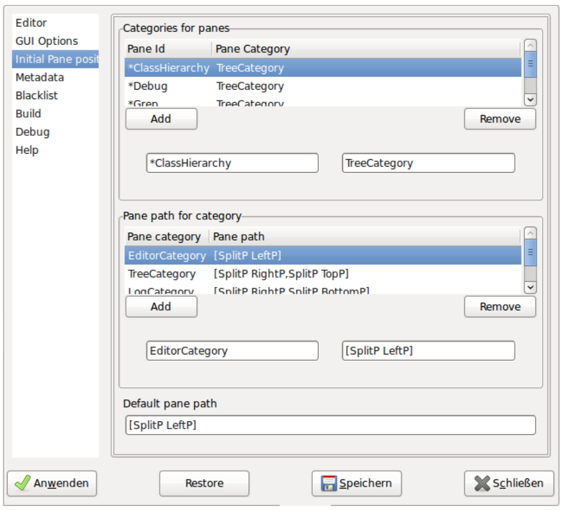
In the initial pane positions part of the Preferences, you can configure the placement of panes. Panes belongs to categories, and a category specify a path were a pane will open (26).

8.1.1 Group panes

Before we invented group panes, a notebook could only contain atomar panes. Now it can as well contain group panes, which have a layout on their own and may contain arbitary 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 best for you.

You can add a new group by selecting View / Group from the menubar. You have to give a unique name for any group. Then you can arrange panes in the group as you like.

[4] Currently there is no way to load different layouts independent of the other data stored in a sessions.



Categories for panes

Pane Id	Pane Category
*ClassHierarchy	TreeCategory
*Debug	TreeCategory
*Grep	TreeCategory

*ClassHierarchy TreeCategory

Pane path for category

Pane category	Pane path
EditorCategory	[SplitP LeftP]
TreeCategory	[SplitP RightP,SplitP TopP]
LogCategory	[SplitP RightP,SplitP BottomP]

EditorCategory [SplitP LeftP]

Default pane path

[SplitP LeftP]

Figure 26: Initial pane position

When closing a group, and the group is not empty, you have to confirm that this is what you really want.

8.1.2 Detached windows

This feature is useful if you want to use Leksah on multiple screens. You can select a notebook and choose View / Detach from the menubar. Then this notebook is opened in a new window, which you can then move to another screen.

If you close the detached window, the contents goes back to the place where it was before detaching.

When you close Leksah, the state of detached window is remembered, 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 menubar, toolbar and statusbar on their own. This may be a problem, because when you want to select a menu entry, the focus may change 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. However keystrokes works fine, and that is how we use this feature.

This is a new addition in this release, and the handling of detachement may be a bit unconveniant. Let us know what you need.

8.2 Session handling

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.

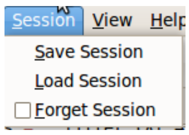


Figure 27: Session menu

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).

8.3 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.

8.4 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 Configuration / 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 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.

9 The Project

The development of an IDE is a big issue, so Leksah is intended to become more and more a community project. If you are a user or just test Leksah, we would appreciate to here from you. Everyone is invited to contribute. That can be development, supplying error reports, spreading the word, providing keymap and candy files, providing a tutorial, caring for a platform . . .

Possible extension and enhancements are:

- Workspaces
- Package Editor with configurations
- Context enriched completion
- Object browser
- Extension support with plugins by libraries
- Add traces to Debugger
- Collect local metadata for the projects worked on
- Version Control (Darcs, ...)
- Testing (Quick check,...)
- Coverage (HPC,...)
- Profiling (Ghc Profiler,...)
- Refactoring (HaRe,...)
- FAD (Functional Analysis and Design,...)
- . . .

Acknowledgment

Thanks to Ricardo Herrmann for making the new Leksah logo.
Thanks to Sean Chapel for help with Ubuntu installation instructions.
Thanks to Wolfgang Jeltsch for being our most helpful user.
Thanks to Fabian Emmes, who created the very nice icons for the module browser.
Thanks to Funa which gave its sheer existence for Leksah
And thanks to the constructors of Haskell, Miranda and GHC.
Thanks to Moses Schönfinkel and Alonzo Church
and to the four cardinal points

10 Appendix

10.1 Command line arguments

```
Usage: ide [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
-x[FILE]  --Extract[=FILE]   Extract tars from cabal install directory
-h        --Help             Display command line options
```

10.2 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
```

10.3 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 can be found at
-- http://gitweb.freedesktop.org/?p=xorg/proto/x11proto.git;a=blob_plain;f=keysymdef.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>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>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 lines"
<ctrl><alt>Left            ->          EditUncomment   "Remove a line style comment"
<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 horizontal"
<ctrl>3                    ->          ViewSplitVertical "Split the current vertical"
<ctrl>1                 ->          ViewCollapse    "Collapse the panes"
                        ->          ViewTabsLeft    "Shows the tabs  on the left"
                        ->          ViewTabsRight   "Shows the tabs on the right"
                        ->          ViewTabsDown    "Shows the tabs on the bottom"
                        ->          ViewSwitchTabs  "Switches tabs visible or invisible "
<ctrl>t                 ->          ToggleToolbar
<ctrl>b                 ->          BuildPackage
<ctrl>r                 ->          AddAllImports
<ctrl><alt>r            ->          RunPackage
<ctrl>j                 ->          NextError
<ctrl><shift>j          ->          PreviousError
<ctrl>m                 ->          ShowModules
<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 buffer in reverse recently used oder"
<ctrl>Page_Down         ->          FlipDown        "Switch to next buffer in recently used oder"
<ctrl>space             ->          StartComplete   "Initiate complete in a source buffer"
F6 -> DebugStep
F7 -> DebugStepLocal
F8 -> DebugStepModule
F9 -> DebugContinue
```