Leksah: An Integrated Development Environment for Haskell

Jürgen Nicklisch-Franken

February 3, 2008

Contents

License

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

1 Introduction

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

Leksah uses GTK+ as GUI Toolkit with the gtk2hs binding. It is platform independent and should run on any platform where GTK+, gtk2hs and ghc can be installed. (It is currently been tested on Windows and Linux. Please tell us if it runs on the Mac). It uses the Cabal package management and build system for Package Management. It will use Haddock for source code documentation. It currently only supports the Glasgow Haskell Compiler (ghc). It could possibly be integrated with other compilers, but their are no plans to do this.

1.1 Further Information

The source code for Leksah is hosted under `code.haskell.org/leksah`. For the Programming language Haskell go to `www.haskell.org`. For information about GTK+ go to `www.gtk.org`. For information about Ghc go to `www.haskell.org/ghc`. You can contact the developer at jnf AT arcor.de.

1.2 Release Notes

1.2.1 Version 0.1

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

2 Installing Leksah

1. Install GHC 6.8.x On Linux the best choice is currently 6.8.2 but 6.8.1 should work. On Windows use 6.8.1., until a gtk2hs installer for 6.8.2 is available. It is a good idea to install everything with sources, specially when using Leksah.

2. Install GTK+ and gtk2hs in a version compatible with the version of GHC you just installed. This should be easy on Linux and it is easy on Windows, if you use an installer for Windows (`http://haskell.org/ght2hs`). At the time of writing it

is the 0.9.12.1 version which is compatible with 6.8.1 on Windows. The Packages you need are gtk >=0.9.12, glib >=0.9.12 and sourceview >=0.9.12.

3. Download, configure, build and install the binary package (version >= 0.4.1) 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, runhaskell install.* The other packages needed should have been installed with GHC anyway. (I'm nut sure if GHC-extralibs is needed).

4. Download, and build Leksah with typical Cabal procedure.
   If you want to run Leksah without installing, which is interesting for development, you have to copy the /data folder with its contents to a place where Leksah will find it. This place depends on the target platform and will be chosen from Cabal. On my Windows machine it will work to *mkdir dist/build/leksah-0.1, cp -R data dist/build/leksah-0.1/,* on Linux it is a sudo *mkdir /usr/share/leksah-0.1, cp -R data usr/share/leksah-0.1/.*

3 Quick start

1. When you start Leksah for the first time 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.

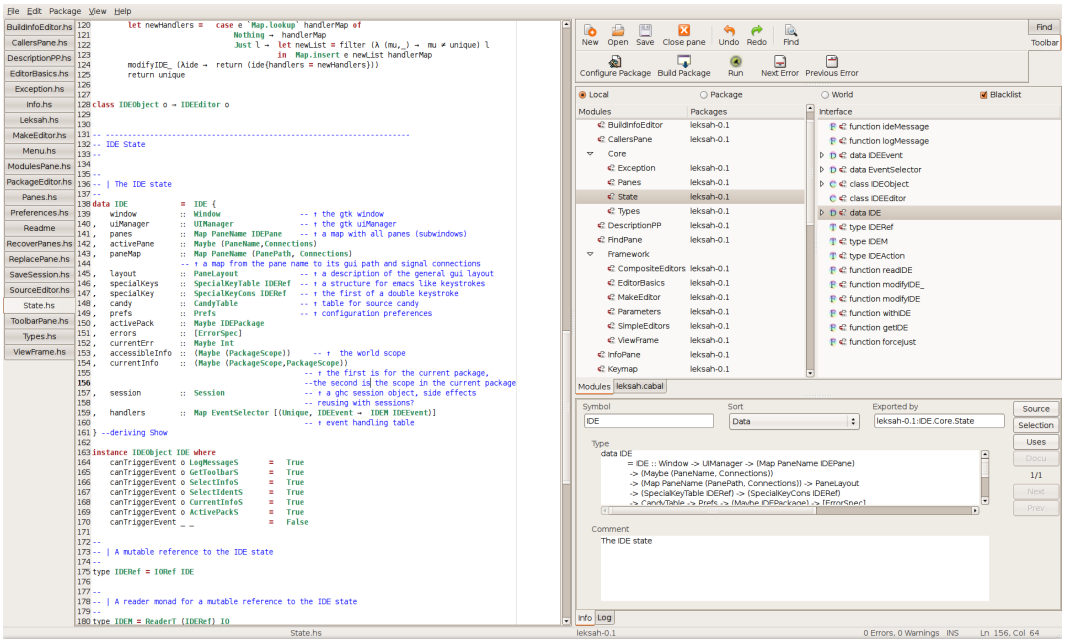
2. Start Leksah. Enter the folder/folders in the first start dialog. You have to click the Add Button after selecting the folder.

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

4. Their are command line options for rebuilding the collected meta-data at any time you want or need it.

5. After starting up, Leksah will open its Main window in a standard configuration.

6. The best way to start up will be to open an existing project. So select Package/OpenPackage from the menu and open a Cabal file of some project.



4 Configuration

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

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

3. 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.

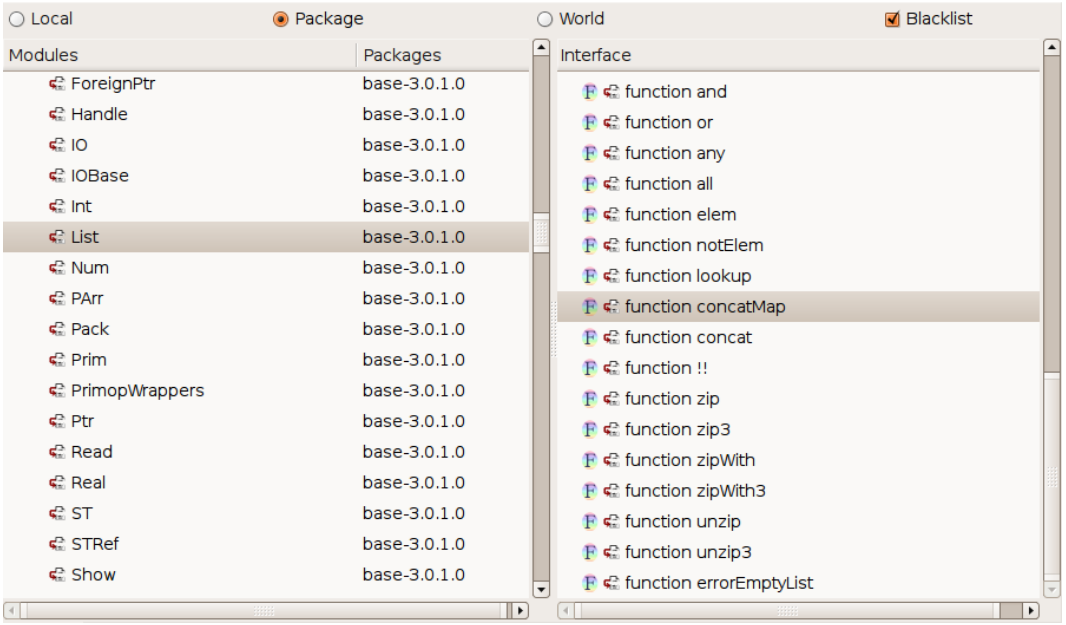
4. The source_packages.txt file stores source locations for installed packages. It can be rebuild by calling Leksah with the -s or –Sources argument . Do this after you moved your source or added sources for previous installed packages without sources.

5. 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.

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

5 Meta information and Navigation

5.1 The Modules Pane



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

2. We assume there is an open package. You can then select the scope of the displayed information with the radio button on top of the modules pane. The *Local* scope shows only modules which are part of the project. The *Package* scope shows all modules of the package and all packages the current package depends on. The *World* scope shows all modules of installed packages of the system. (You can get this list with *ghc-pkg list*).

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

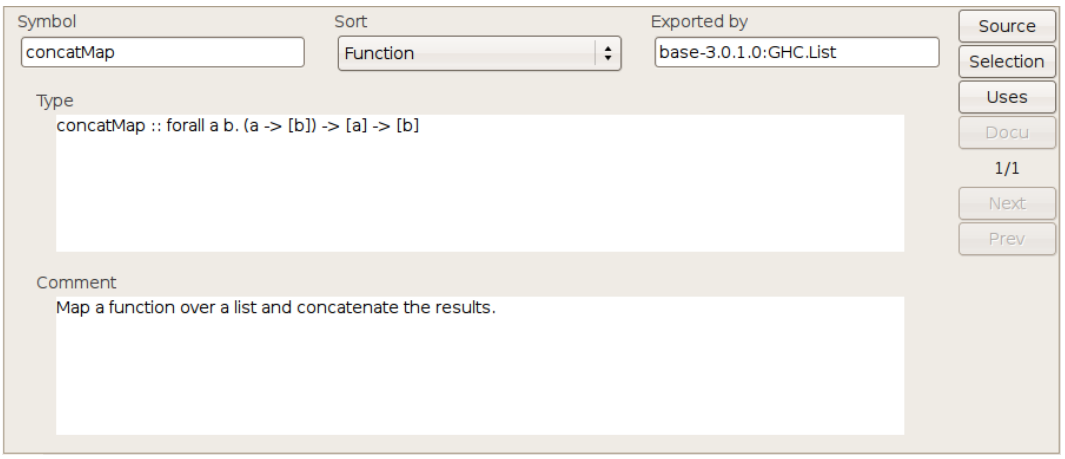
4. If you select a module in the modules list, its interface is displayed in the interface list on the right. You can search for a module or package by selecting the modules list and typing some text. With the up and down arrows you find the next/previous matching item. With the escape key or by selecting any other GUI element you leave the search mode.

5. If there is a little symbol with an arrow in front of a module, Leksah has found a source file for this module. You can open this source file, or bring it to the front if it is already open with a *double click* on the module. (the same can be done with selecting *Edit* from the context menu.

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

7. If there is a little symbol with an arrow in front of a module, Leksah has found a source location for this element. You can open this source file, or bring it to the front and display the source for the selected location with a *double click* on the element. (the same can be done with selecting *Edit* from the context menu. (Currently their is a bug, that the location is not selected for a fresh opened file. In this case repeat the double click).

5.2 The Info Pane



1. The Info Pane shows information about an interface element, which may be a function, a class, a data definition .... It shows the identifier, which sort it is of, its type and if possible a comment.

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

3. Remember that only statically collected information is available this way. So the meta data contains only information about items which are exported by some module. (Currently there is as well a bug with reexported items).

4. There may be many definitions of an identifier in the scope, in which case you see something like (3/1) on the right side of the pane and the Next button gets active so that you can navigate to the next/previous information.

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

6. If the definition can be found in the scope selected in the modules pane, you can select the module and the interface element in the modules pane by clicking the *Selection* button.

7. With the *Uses* button a pane opens which displays the modules which imports this element. (This feature is not ready yet).

8. The *Docu* button will bring you to the haddock Documentation when it is available (and the implementation of this is ready).

9. (I plan to add a search feature here in the near future).

5.3 The Collector

6 The View Frame

6.1 Special Keystrokes

1. 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"
```

2. 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"
```

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

4. The name of the action can be any one of the *ActionDescr's* given in the *action* function in the Module *IDE.Menu*. (All *IDEAction*'s can be added to this actions, which may not be the current situation).

5. 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.

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

7. 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.

6.2 Menus and Toolbars

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

6.3 Window Layout

1. 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 right pane is active. (This is specially important in the current state of development). As well text buffers by default open up on top of the current pane.

2. 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.

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

4. There are certain panes which can't be activated and thus can't be moved like the Toolbar, the Find and the Replace pane.

5. The layout will be saved when you leave Leksah and will be restored when you restart it. Currently there is no way to load different layouts, but this feature would be easy to implement.

7 The Editor

7.1 Source Candy

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

1. 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.

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

3. 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.

4. 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

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

8 Packages (Cabal)

8.1 Building

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

2. 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 using the menu, the toolbar or a keystroke. Another possibility is to click on the error in the Log pane, then the corresponding source buffer at the error place will be shown.

3. After a successful build, that is a build, which didn't produce any errors, the metainfo for the active package will be rebuild.

4. 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.

9 Planed Developments

9.1 Plans for Version 0.x

- Working Cabal Editor with Configurations (etc..)
- Completion
- Search
- Class Hierarchy pane
- Import helper
- (Maybe) Editor: Highlighting according to file type
- (Maybe) Properties of single editor (no highlight, no candy,...)
- Haddock integration
- Internal: Generic record editor
- Using find in sources without location
- Build only one time - Cancel build
- Source collector only one time - Make it run in background
- Preferences embed den
- Uses Pane (Currently called Callers)
- Hints for developers

9.2 Version x.y

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

9.3 Any version later 1.0

- Plugins

Postscript

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