Customizing LyX: Features for the Advanced User

by the LyX Team*

Version 2.0.x

July 15, 2011

*If you have comments or error corrections, please send them to the LyX Documentation mailing list, lyx-docs@lists.lyx.org. Include "[Customization]" in the subject header, and please cc the current maintainer of this file, Richard Heck <rgheck@comcast.net>.

Contents

1 Introduction **1**

2 LyX configuration files **3**
2.1 What's in LyXDir? . . . 3
2.1.1 Automatically generated files . . . 3
2.1.2 Directories . . . 4
2.1.3 Files you don't want to modify . . . 4
2.1.4 Other files needing a line or two... . . . 5
2.2 Your local configuration directory . . . 5
2.3 Running LyX with multiple configurations . . . 5

3 The Preferences dialog **7**
3.1 Formats . . . 7
3.2 Copiers . . . 7
3.3 Converters . . . 8

4 Internationalizing LyX **11**
4.1 Translating LyX . . . 11
4.1.1 Translating the graphical user interface (text messages). . . . 11
4.1.1.1 Ambiguous messages . . . 12
4.1.2 Translating the documentation. . . . 12
4.2 International Keymap Stuff . . . 13
4.2.1 The .kmap File . . . 14
4.2.2 The .cdef File . . . 15
4.2.3 Dead Keys . . . 16
4.2.4 Saving your Language Configuration . . . 16

5 Installing New Document Classes **17**
5.1 Installing new LaTeX files . . . 18
5.2 Types of layout files . . . 19
5.2.1 Layout modules . . . 20
5.2.1.1 Local Layout . . . 21
5.2.2 Layout for .sty files . . . 21
5.2.3 Layout for .cls files . . . 23
5.2.4 Creating templates . . . 23
5.2.5 Upgrading old layout files . . . 24

5.3 The layout file format . . . 24
5.3.1 The document class declaration . . . 24
5.3.2 The Module declaration . . . 26
5.3.3 Format number . . . 27
5.3.4 General text class parameters . . . 27
5.3.5 `ClassOptions` section . . . 30
5.3.6 Paragraph styles . . . 31
5.3.7 Internationalization of Paragraph Styles . . . 37
5.3.8 Floats . . . 38
5.3.9 Flex insets and InsetLayout . . . 40
5.3.10 Counters . . . 42
5.3.11 Font description . . . 43
5.3.12 Citation format description . . . 44
5.4 Tags for XHTML output . . . 45
5.4.1 Paragraph styles . . . 46
5.4.2 InsetLayout XHTML . . . 48
5.4.3 Float XHTML . . . 49
5.4.4 Bibliography formatting . . . 49
5.4.5 LyX-generated CSS . . . 50

6 Including External Material **51**
6.1 How does it work? . . . 51
6.2 The external template configuration file . . . 52
6.2.1 The template header . . . 53
6.2.2 The Format section . . . 54
6.2.3 Preamble definitions . . . 55
6.3 The substitution mechanism . . . 55
6.4 Security discussion . . . 57

1 Introduction

This manual covers the customization features present in LyX. In it, we discuss issues like keyboard shortcuts, screen previewing options, printer options, sending commands to LyX via the LyX Server, internationalization, installing new LaTeX classes and LyX layouts, etc. We can't possibly hope to touch on everything you can change—our developers add new features faster than we can document them—but we will explain the most common customizations and hopefully point you in the right direction for some of the more obscure ones.

2 LyX configuration files

This chapter aims to help you to find your way through the LyX configuration files. Before continuing to read this chapter, you should find out where your LyX library and user directories are by using Help ▷ About LyX. The library directory is the place where LyX places its system-wide configuration files; the user directory is where you can place your modified versions. We will call the former `LyXDir` and the latter UserDir in the remainder of this document.

2.1 What's in `LyXDir`?

`LyXDir` and its sub-directories contain a number of files and that can be used to customize LyX's behavior. You can change many of these files from within LyX itself through the Tools ▷ Preferences dialog. Most customization that you will want to do in LyX is possible through this dialog. However, many other inner aspects of LyX can be customized by modifying the files in `LyXDir`. These files fall in different categories, described in the following subsections.

2.1.1 Automatically generated files

The files, which are to be found in UserDir, are generated when you configure LyX. They contain various default values that are guessed by inspection. In general, it is not a good idea to modify them, since they might be overwritten at any time.

`lyxrc.defaults` contains defaults for various commands.

`packages.lst` contains the list of packages that have been recognized by LyX. It is currently unused by the LyX program itself, but the information extracted, and more, is made available with Help ▷ LaTeX Configuration.

`textclass.lst` the list of text classes that have been found in your `layout/` directories, along with the associated LaTeX document class and their description.

`lyxmodules.lst` the list of layout modules found in your `layout/` directories

`*files.lst` lists of various sorts of LaTeX-related files found on your system

`doc/LaTeXConfig.lyx` is automatically generated during configuration from the file `LaTeXConfig.lyx.in`. It contains information on your LaTeX configuration.

2.1.2 Directories

These directories are duplicated between `LyXDir` and `UserDir`. If a particular files exists in both places, the one in `UserDir` will be used.

`bind/` this directory contains files with the extension `.bind` that define the key-bindings used in LyX. If there exists an internationalized version of the bind file named `$LANG_xxx.bind`, that will be used first.

`clipart/` contains graphics files that can be included in documents.

`doc/` contains LyX documentation files (including the one you are currently reading). The file `LaTeXConfig.lyx` deserves special attention, as noted above. The internationalized help docs are in subdirectories `doc/xx` where "xx" is the ISO language code. See chapter 4 for details.

`examples/` contains example files that explain how to use some features. In the file browser, press the Examples button to get there.

`images/` contains image files that are used by the Document dialog. In addition, it also contains the individual icons used in the toolbar and the banners that can be shown when LyX is launched.

`kbd/` contains keyboard keymapping files. See Chapter 4.2 for details.

`layouts/` contains the text class and module files described in Chapter 5.

`lyx2lyx` contains the `lyx2lyx` Python scripts used to convert between LyX versions. These can be run from the command line if, say, you want to batch-convert files.

`scripts/` contains some files that demonstrate the capabilities of the External Template feature. Also contains some scripts used by LyX itself.

`templates/` contains the standard LyX template files described in Chapter 5.2.4.

`ui/` contains files with the extension `.ui` that define the user interface to LyX. That is, the files define which items appear in which menus and the items appearing on the toolbar.

2.1.3 Files you don't want to modify

These files are used internally by LyX and you generally do not need to modify them unless you are a developer.

`CREDITS` this file contains the list of LyX developers. The contents are displayed with the menu entry Help ▷ About LyX.

`chkconfig.ltx` this is a LaTeX script used during the configuration process. Do not run directly.

`configure.py` this is the script that is used to re-configure LyX. It creates configuration files in the directory it was run from.

2.1.4 Other files needing a line or two...

`encodings` this contains tables describing how different character encodings can be mapped to Unicode

`external_templates` this file contains the templates available to the new External Template feature.

`languages` this file contains a list of all the languages currently supported by LyX.

2.2 Your local configuration directory

Even if you are using LyX as an unprivileged user, you might want to change LyX configuration for your own use. The `UserDir` directory contains all your personal configuration files. This is the directory described as "user directory" in Help ▷ About LyX. This directory is used as a mirror of `LyXDir`, which means that every file in `UserDir` is a replacement for the corresponding file in `LyXDir`. Any configuration file described in the above sections can be placed either in the system-wide directory, in which case it will affect all users, or in your local directory for your own use.

To make things clearer, let's provide a few examples:

- The preferences set in the Tools ▷ Preferences dialog are saved to a file `preferences` in `UserDir`.
- When you reconfigure using Tools ▷ Reconfigure, LyX runs the `configure.py` script, and the resulting files are written in your local configuration directory. This means that any additional text class file that you might have added in `UserDir/layouts` will be added to the list of classes in the Document ▷ Settings dialog.
- If you get some updated documentation from LyX ftp site and cannot install it because you do not have sysadmin rights on your system, you can just copy the files in `UserDir/doc/` and the items in the Help menu will open them!

2.3 Running LyX with multiple configurations

The configuration freedom of the local configuration directory may not suffice if you want to have more than one configuration at your disposal. For example, you may want to be use different key bindings or printer settings at different times. You can

achieve this by having several such directories. You then specify which directory to use at run-time.

Invoking LyX with the command line switch `-userdir` <*some directory*> instructs the program to read the configuration from that directory, and not from the default directory. (You can determine the default directory by running LyX without the `-userdir` switch.) If the specified directory does not exist, LyX offers to create it for you, just like it does for the default directory on the first time you run the program. You can modify the configuration options in this additional user directory exactly as you would for the default directory. These directories are completely independent (but read on). Note that setting the environment variable `LYX_USERDIR_VER` to some value has exactly the same effect.

Having several configurations also requires more maintenance: if you want to add a new layout to `NewUserDir/layouts` which you want available from all your configurations, you must add it to each directory separately. You can avoid this with the following trick: after LyX creates the additional directory, most of the subdirectories (see above) are empty. If you want the new configuration to mirror an existing one, replace the empty subdirectory with a symbolic link to the matching subdirectory in the existing configuration. Take care with the `doc/` subdirectory, however, since it contains a file written by the configuration script (also accessible through Tools▷ Reconfigure) which is configuration-specific.

3 The Preferences dialog

All options of the preferences dialog are described in the Appendix *The Preferences Dialog* in the *User's Guide*. For some options you might find here more details.

3.1 Formats

The first step is to define your file formats if they are not already defined. To do so, open the Tools ▷ Preferences dialog. Under File Handling ▷ File formats press the New... button to define your new format. The Format field contains the name used to identify the format in the GUI. The Short Name is used to identify the format internally. You will also need to enter a file extension. These are all required. The optional Shortcut field is used to provide a keyboard shortcut on the menus. (For example, pressing Alt-V F D will View ▷ View (Other Formats) ▷ DVI.)

A Format can have a Viewer and an Editor associated with it. For example, you might want to use Ghostview to view PostScript files. You can enter the command needed to start the program in the corresponding fields. In defining this command, you can use the four variables listed in the next section. The viewer is launched when you view an image in LyX or use the View menu. The editor is for example launched when you right-click on an image and choose Edit externally in the appearing context menu.

The Document format option tells LyX that a format is suitable for document export. If this is set and if a suitable conversion route exists (see sec. 3.3), the format will appear in the File ▷ Export menu. The format will also appear in the View menu if a viewer is specified for the format. Pure image formats, such as `png`, should not use this option. Formats that can both represent vector graphics and documents like `pdf` should use it.

The option Vector graphics format tells LyX that a format can contain vector graphics. This information is used to determine the target format of included graphics for pdflatex export. Included graphics may need to be converted to either pdf, png, or jpg, since pdflatex cannot handle other image formats. If an included graphic is not already in pdf, png, or jpg format, it is converted to pdf if the vector format option is set, and otherwise to png.

3.2 Copiers

Since all conversions from one format to another take place in LyX's temporary directory, it is sometimes necessary to modify a file before copying it to the temporary

directory in order that the conversion may be performed.[1] This is done by a Copier: It copies a file to (or from) the temporary directory and may modify it in the process.

The definitions of the copiers may use four variables:

`$$s` The LyX system directory (e. g. /usr/share/lyx).

`$$i` The input file

`$$o` The output file

`$$l` The 'LaTeX name'

The latter should be the filename as it would be used in a LaTeX's **\include** command. It is relevant only when exporting files suitable for such inclusion.

Copiers can be used to do almost anything with output files. For example, suppose you want generated pdf files to be copied to a special directory, `/home/you/pdf/`. Then you could write a shell script such as this one:

```
#!/bin/bash
FROMFILE=$1
TOFILE=`basename $2`
cp $FROMFILE /home/you/pdf/$TOFILE
```

Save it in your local LyX directory—say, `/home/you/.lyx/scripts/pdfcopier.sh`—and make it executable, if you need to do so on your platform. Then, in the Tools ▷ Preferences dialog, select under File Handling ▷ File formats the PDF(pdflatex) format—or one of the other pdf formats—and enter `pdfcopier.sh $$i $$o` into the Copier field.

Copiers are used by LyX in various of its own conversions. For example, if appropriate programs are found, LyX will automatically install copiers for the HTML and HTML (MS Word) formats. When these formats are exported, the copier sees that not just the main HTML file but various associated files (style files, images, etc.) are also copied. All these files are written to a subdirectory of the directory in which the original LyX file was found.[2]

3.3 Converters

You can define your own Converters to convert files between different formats. This is done in the Tools ▷ Preferences ▷ File Handling ▷ Converters dialog.

[1] For example, the file may refer to other files—images, for example—using relative file names, and these may become invalid when the file is copied to the temporary directory.

[2] This copier can be customized. The optional "-e" argument takes a comma-separated list of extensions to be copied; if it is omitted, all files will be copied. The "-t" argument determines the extension added to the generated directory. By default, it is "LyXconv", so HTML generated from /path/to/filename.lyx will end up in /path/to/filename.html.LyXconv.

To define a new converter, select the From format and To format from the drop-down lists, enter the command needed for the conversion, and then press the Add button. Several variables can be used in the definition of converters:

`$$s` The LyX system directory

`$$i` The input file

`$$o` The output file

`$$b` The base filename of the input file (i. g., without the extension)

`$$p` The path to the input file

`$$r` The path to the original input file (this is different from $$p when a chain of converters is called)

`$$e` The iconv name for the encoding of the document.

In the Extra Flag field you can enter the following flags, separated by commas:

`latex` This converter runs some form of LaTeX. This will make LyX's LaTeX error logs available.

`needaux` Needs the LaTeX .aux file for the conversion.

`xml` Output is XML.

The following three flags are not really flags at all because they take an argument in the key = value format:

`parselog` If set, the converter's standard error will be redirected to a file `infile.out`, and the script given as argument will be run as: `script < infile.out > infile.log`. The argument may contain `$$s`.

`resultdir` The name of the directory in which the converter will dump the generated files. LyX will not create this directory, and it does not copy anything into it, though it will copy this directory to the destination. The argument may contain `$$b`, which will be replaced by the base name of the input and output files, respectively, when the directory is copied.
Note that resultdir and usetempdir make no sense together. The latter will be ignored if the former is given.

`resultfile` Determines the output file name and may, contain `$$b`. Sensible only with resultdir and optional even then; if not given, it defaults to 'index'.

None of these last three are presently used in any of the converters that are installed with LyX.

You do not have to define converters for all formats between which you want to convert. For example, you will note that there is no 'LyX to PostScript' converter, but LyX will export PostScript. It does so by first creating a LaTeX file (no converter needs to be defined for this) which is then converted to DVI using the 'LaTeX to DVI' converter, and finally converting the resulting DVI file to PostScript. LyX finds such 'chains' of converters automatically, and it will always choose the shortest possible chain. You can, though, still define multiple conversion methods between file formats. For example, the standard LyX configuration provides three ways to convert LaTeX to PDF: Directly, using pdflatex; via (DVI and) PostScript, using ps2pdf; or via DVI, using dvipdfm. To define such alternate chains, you must define multiple target 'file formats', as described in section 3.1. For example, in the standard configuration, the formats named pdf, pdf2, and pdf3 are defined, all of which share the extension .pdf, and which correspond to the conversion methods just mentioned.

4 Internationalizing LyX

LyX supports using a translated interface. Last time we checked, LyX provided text in thirty languages. The language of choice is called your *locale*. (For further reading on locale settings, see also the documentation for locale that comes with your operating system. For Linux, the manual page for `locale(5)` could be a good place to start).

Notice that these translations will work, but do contain a few flaws. In particular, all dialogs have been designed with the English text in mind, which means that some of the translated text will be too large to fit within the space allocated. This is only a display problem and will not cause any harm. Also, you will find that some of the translations do not define shortcut keys for everything. Sometimes, there are simply not enough free letters to do it. Other times, the translator just hasn't got around to doing it yet. Our localization team, which you may wish to join,[1] will of course try to fix these shortcomings in future versions of LyX.

4.1 Translating LyX

4.1.1 Translating the graphical user interface (text messages).

LyX uses the GNU `gettext` library to handle the internationalization of the interface. To have LyX speak your favorite language in all menus and dialogs, you need a `po`-file for that language. When this is available, you'll have to generate a `mo`-file from it and install the `mo`-file. The process of doing all of this is explained in the documentation for GNU `gettext`. It is possible to do this just for yourself, but if you're going to do it, you might as well share the results of your labors with the rest of the LyX community. Send a message to the LyX developers' list for more information about how to proceed.

In short, this is what you should do (xx denotes the language code):

- Check out the LyX source code. (See the information on the web.)
- Copy the file `lyx.pot` to the folder of the `**.po` files. Then rename it to `xx.po`. (If `lyx.pot` doesn't exist anywhere, it can be remade with the console command `make lyx.pot` in that directory, or you can use an existing po-file for some other language as a template).

[1] If you are a fluent speaker of a language other than English, joining these teams is a great way to give back to the LyX community!

- Edit `xx.po`.[2] For some menu- and widget-labels, there are also shortcut keys that should be translated. Those keys are marked after a '|', and should be translated according to the words and phrases of the language. You should also fill also out the information at the beginning of the new `po`-file with your email-address, etc., so people know where to reach you with suggestions and entertaining flames.

If you are just doing this on your own, then:

- Generate `xx.mo`. This can be done with `msgfmt -o xx.mo < xx.po`.
- Copy the `mo`-file to your locale-tree, at the correct directory for application messages for the language xx, and under the name `lyx.mo` (e. g. `/usr/local/share/locale/xx/L`

As said, however, it would be best if the new `po`-file could be added to the LyX distribution, so others can use it. Adding it involves making additional changes to LyX. So send an email to the developers' mailing list if you're interested in doing that.

4.1.1.1 Ambiguous messages

Sometimes it turns out that one English message needs to be translated into different messages in the target language. One example is the message `To` which has the German translation Nach or Bis, depending upon exactly what the English "to" means. GNU `gettext` does not handle such ambiguous translations. Therefore you have to add some context information to the message: Instead of `To` it becomes `To[[as in 'From format x to format y']]` and `To[[as in 'From page x to page y']]`. Now the two occurrences of `To` are different for `gettext` and can be translated correctly to Nach and Bis, respectively.

Of course the context information needs to be stripped off the original message when no translation is used. Therefore you have to put it in double square brackets at the end of the message (see the example above). The translation mechanism of LyX ensures that everything in double square brackets at the end of messages is removed before displaying the message.

4.1.2 Translating the documentation.

The online documentation (in the Help-menu) can (and should!) be translated. If there are translated versions of the documentation available[3] and the locale is set accordingly, these will be used automagically by LyX. LyX looks for translated versions as `LyXDir/doc/xx/DocName.lyx`, where `xx` is the code for the language currently in

[2] This is just a text file, so it can be edited in any text editor. But there are also specialized programs that support such editing, such as `Poedit` (for all platforms) or `KBabel` (for KDE). `Emacs` contains a 'mode' for editing `po` files, as well.

[3] As of March 2008, at least some of the documents have been translated into fourteen languages, with the Tutorial available in a few more.

use. If there are no translated documents, the default English versions will be displayed. Note that the translated versions must have the same filenames (`DocName` above) as the original. If you feel up to translating the documentation (an excellent way to proof-read the original documentation by the way!), there are a few things you should do right away:

- Check out the documentation translation web page at http://www.lyx.org/Translation. That way, you can find out which (if any) documents have already been translated into your language. You can also find out who (if anyone) is organizing the effort to translate the documentation into your language. If no one is organizing the effort, please let us know that you're interested.

Once you get to actually translating, here's a few hints for you that may save you trouble:

- Join the documentation team! There is information on how to do that in `Intro.lyx` (Help ▷ Introduction), which by the way is the first document you should translate.

- Learn the typographic conventions for the language you are translating to. Typography is an ancient art and over the centuries, a great variety of conventions have developed throughout different parts of the world. Also study the professional terminology amongst typographers in your country. Inventing your own terminology will only confuse the users. *(Warning! Typography is addictive!)*

- Make a copy of the document. This will be your working copy. You can use this as your personal translated help-file by placing it in your `UserDir/doc/xx/` directory.

- Sometimes the original document (from the LyX-team) will be updated. Use the source viewer at http://www.lyx.org/trac/timeline to see what has been changed. That way you can easily see which parts of the translated document need to be updated.

If you ever find an error in the original document, fix it and notify the rest of the documentation team of the changes! (You didn't forget to join the documentation team, did you?)

4.2 International Keymap Stuff

The next two sections describe the `.kmap` and `.cdef` file syntax in detail. These sections should help you design your own key map if the ones provided do not meet your needs.

4.2.1 The .kmap File

A .kmap file maps keystrokes to characters or strings. As the name suggests, it sets a keyboard mapping. The .kmap file keywords kmap, kmod, kxmod, and kcomb are described in this section.

kmap Map a character to a string

\kmap *char string*

This will map *char* to *string*. Note that in *string*, the double-quote (") and the backslash (\) must be escaped with a preceding backslash (\).

An example of a kmap statement to cause the symbol / to be output for the keystroke & is:

```
\kmap & /
```

kmod Specify an accent character

\kmod *char accent allowed*

This will make the character *char be an accent* on the *allowed* character(s). This is the dead key[4] mechanism.

If you hit *char* and then another key not in *allowed*, you will get a *char* followed by the other, not allowed key, as output. Note that a Backspace cancels a dead key, so if you hit *char Backspace*, the cursor will not go one position backwards but will instead cancel the effect that *char* might have had on the next keystroke.

The following example specifies that the character ' is to be an acute accent, allowed on the characters a, e, i, o, u, A, E, I, O, and U:

```
\kmod ' acute aeiouAEIOU
```

kxmod Specify an exception to the accent character

\kxmod *accent char result*

This defines an exception for *accent* on *char*. The *accent* must have been assigned a keystroke with a previous \kmod declaration and *char* must not belong in the *allowed* set of *accent*. When you enter the *accent char* sequence, *result* is produced. If such a declaration does not exist in the .kmap file and you enter *accent char*, you get *accent_key char* where *accent_key* is the first argument of the \kmod declaration.

The following command produces causes äi to be produced when you enter acute-i ('i):

[4] The term *dead key* refers to a key that does not produce a character by itself, but when followed with another key, produces the desired accent character. For example, a German character with an umlaut like *ä* can be produced in this manner.

```
\kxmod acute i "\\'{\\i}"
```

`kcomb` Combine two accent characters

`\kcomb` *accent1 accent2 allowed*

This one is getting pretty esoteric. It allows you to combine the effect of *accent1* and *accent2* (in that order!) on *allowed* chars. The keystrokes for *accent1* and *accent2* must have been set with a `\kmod` command at a *previous* point in the file.

Consider this example from the `greek.kmap` file:

```
\kmod ; acute aeioyvhAEIOYVH \kmod : umlaut iyIY \kcomb acute umlaut iyIY
```

This allows you to press ;:i and get the effect of `\'{\"{i}}`. A backspace in this case cancels the last dead key, so if you press ;: Backspace i you get `\'{i}`.

4.2.2 The .cdef File

After the `.kmap` mapping is performed, a `.cdef` file maps the strings that the symbols generate to characters in the current font. The LyX distribution currently includes at least the `iso8859-1.cdef` and `iso8859-2.cdef` files.

In general the `.cdef` file is a sequence of declarations of the form

char_index_in_set string

For example, in order to map \'{e} to the corresponding character in the iso-8859-1 set (233), the following declaration is used

```
233 "\\'{e}"
```

with `\` and `"` being escaped in *string*. Note that the same character can apply to more than one string. In the `iso-8859-7.cdef` file you have

```
192 "\\'{\\\"{i}}"
192 "\\\"{\\'{i}}"
```

If LyX cannot find a mapping for the string produced by the keystroke or a deadkey sequence, it will check if it looks like an accented char and try to draw an accent over the character on screen.

4.2.3 Dead Keys

There is a second way to add support for international characters through so-called dead-keys. A dead-key works in combination with a letter to produce an accented character. Here, we'll explain how to create a really simple dead-key to illustrate how they work.

Suppose you happen to need the circumflex character, "^". You could bind the ^-key [a.k.a. Shift-6] to the LyX command `accent-circumflex` in your `lyxrc` file. Now, whenever you type the ^-key followed by a letter, that letter will have a circumflex accent on it. For example, the sequence "^e" produces the letter: "ê". If you tried to type "^t", however, LyX will complain with a beep, since a "t" never takes a circumflex accent. Hitting Space after a dead-key produces the bare-accent. Please note this last point! If you bind a key to a dead-key, you'll need to rebind the character on that key to yet another key. Binding the ,-key to a cedilla is a bad idea, since you'll only get cedillas instead of commas.

One common way to bind dead-keys is to use Meta-, Ctrl-, and Shift- in combination with an accent, like "~" or "," or "^". Another way involves using `xmodmap` and `xkeycaps` to set up the special `Mode_Switch` key. The `Mode_Switch` acts in some ways just like Shift and permits you to bind keys to accented characters. You can also turn keys into dead-keys by binding them to something like `usldead_cedilla` and then binding this symbolic key to the corresponding LyX command.[5] You can make just about anything into the `Mode_Switch` key: One of the Ctrl- keys, a spare function key, etc. As for the LyX commands that produce accents, check the entry for `accent-acute` in the *Reference Manual*. You'll find the complete list there.

4.2.4 Saving your Language Configuration

You can edit your preferences so that your desired language environment is automatically configured when LyX starts up, via the Edit ▷ Preferences dialog.

[5] Note from John Weiss: This is exactly what I do in my `~/.lyx/lyxrc` and my `~/.xmodmap` files. I have my Scroll Lock key set up as `Mode_Shift` and a bunch of these "`usldead_*`" symbolic keys bound such things as Scroll Lock-^ and Scroll Lock-~. This is how I produce my accented characters.

5 Installing New Document Classes, Layouts, and Templates

In this chapter, we describe the procedures for creating and installing new LyX layout and template files, as well as offer a refresher on correctly installing new LaTeX document classes.

First, let us a say a few words about how one ought to think about the relation between LyX and LaTeX. The thing to understand is that, in a certain sense, LyX doesn't know anything about LaTeX. Indeed, from LyX's point of view, LaTeX is just one of several "backend formats" in which it is capable of producing output. Other such formats are DocBook, plaintext, and XHTML. LaTeX is, of course, a particularly important format, but very little of the information LyX has about LaTeX is actually contained in the program itself.[1] Rather, that information, even for the standard classes like `article.cls`, is contained in 'layout files'. Similarly, LyX itself does not know much about DocBook or XHTML. What it knows is contained in layout files.

You can think of the layout file for a given document class as a translation manual between LyX constructs—paragraphs with their corresponding styles, certain sorts of insets, etc—and the corresponding LaTeX, DocBook, or XHTML constructs. Almost everything LyX knows about `article.cls`, for example, is contained in the file `article.layout` and in various other files it includes. For this reason, anyone intending to write layout files should plan to study the existing files. A good place to start is with `stdsections.inc`, which is included in `article.layout`, `book.layout`, and many of the other layout files for document classes. This file is where sections and the like are defined: `stdsections.inc` tells LyX how paragraphs that are marked with the Section, Subsection, etc, styles can be translated into corresponding LaTeX, DocBook, and XHTML commands and tags. The `article.layout` file basically just includes several of these `std*.inc` files.

Defining the LyX–LaTeX correspondence is not the only thing layout files do, though. Their other job is to define how the LyX constructs themselves will appear on-screen. The fact that layout files have these two jobs is often a source of confusion, because they are completely separate. Telling LyX how to translate a certain paragraph style into LaTeX does not tell LyX how to display it; conversely, telling LyX how to display a certain paragraph style does not tell LyX how to translate it into LaTeX (let alone tell LaTeX how to display it). So, in general, when you define a new LyX construct, you must always do two quite separate things: (i) tell LyX how

[1] Some commands are sufficiently complex that they are "hardcoded" into LyX. But the developers generally regard this as a Bad Thing.

to translate it into LaTeX and (ii) tell LyX how to display it.

Much the same is true, of course, as regards LyX's other backend formats, though XHTML is in some ways different, because in that case LyX *is* able, to some extent, to use information about how it should display a paragraph on the screen to output information (in the form of CSS) about how the paragraph should be displayed in a browser. Even in this case, however, the distinction between what LyX does internally and how things are rendered externally remains in force, and the two can be controlled separately. See 5.4 for the details.

5.1 Installing new LaTeX files

Some installations may not include a LaTeX package or class file that you would like to use within LyX. For example, you might need FoilTeX, a package for preparing slides for overhead projectors. Modern LaTeX distributions like TeXLive (2008 or newer) or MiKTeX provide a user interface for installing such packages. For example, with MiKTeX, you start the program "Package Manager" to get a list of available packages. To install one of them, right click on it or use the corresponding toolbar button.

If your LaTeX distribution does not provide such a 'package manager', or if the package is not available from your distribution, then follow these steps to install it manually:

1. Get the package from CTAN or wherever.

2. If the package contains a file with the ending "`.ins`" (is the case for FoilTeX) then open a console, change to the folder of this file and execute the command `latex foiltex.ins`. You have now unpacked the package and have all files to install it. Most LaTeX-packages are not packed and you can skip this step.

3. Now you need to decide if the package should be available for all users or only for you.

   a) On *nix systems (Linux, OSX, etc.), if you want the new package to be available for all users on your system, then install it in your 'local' TeX tree, otherwise install it in your own 'user' TeX tree. Where these trees should be created, if they do not already exist, depends on your system. To find this out, look in the file `texmf.cnf`.[2] The location of the 'local' TeX tree is defined by the `TEXMFLOCAL` variable; this is usually somewhere like `/usr/local/share/texmf`. The location of the 'user' TeX tree is defined by `TEXMFHOME` and is commonly `$HOME/texmf`. (If these variables are not predefined, you have to define them.) You'll probably need root permissions to create or modify the 'local' tree, but not for your 'user'

[2] This is usually in the directory `$TEXMF/web2c`, though you can execute the command `kpsewhich texmf.cnf` to locate it.

tree.
In general, it is recommended to install in the user tree because your user will not be modified or even overwritten when you upgrade your system. It will typically also be backed up together with everything else when you backup your home directory (which, of course, you do on a regular basis).

b) On Windows, if you want the new package to be available for all users on your system, change to the folder where LaTeX is installed and then change to the subfolder `~\tex\latex`. (For MiKTeX, this would be by default the folder `~:\Programs\MiKTeX\tex\latex`.)[3] Create there a new folder `foiltex` and copy all files of the package into it.
If the package should only available for you or you don't have admin permissions, do the same, but in the local LaTeX folder. E. g., for MiKTeX 2.8 under Windows XP, this would be the folder:
`~:\Documents and Settings\<username>\Application Data\MiKTeX\2.8\tex\latex`
On Vista, it would be:
`~:\Users\<username>\AppData\Roaming\2.8\MiKTeX\tex\latex`

4. Now one only need to tell LaTeX that there are new files. This depends on the used LaTeX-Distribution:

   a) For TeXLive execute the command `texhash` from a console. If you installed the package for all users, then you will probably need to have root permissions for that.

   b) For MiKTeX, if you have installed the package for all users, start the program "Settings (Admin)" and press the button marked "Refresh FNDB". Otherwise start the program "Settings" and do the same.

5. Finally, you need to tell LyX that there are new packages available. So, in LyX, use the menu Tools ▷ Reconfigure and then restart LyX.

Now the package is installed. In our example, the document class `Slides (FoilTex)` will now be available under Document ▷ Settings ▷ Document Class.

If you would like to use a LaTeX document class that is not even listed in the menu Document ▷ Settings ▷ Document Class, then you need to create a 'layout' file for it. That is the topic of the next section.

5.2 Types of layout files

This section describes the various sorts of LyX files that contain layout information. These files describe various paragraph and character styles, determining how

[3] Note that this will be the correct path only on English installations. On a German one, it would be `~:\Programme\MiKTeX\tex\latex`, and similarly for other languages.

LyX should display them and how they should be translated into LaTeX, DocBook, XHTML, or whatever output format is being used.

We shall try to provide a thorough description of the process of writing layout files here. However, there are so many different types of documents supported even by just LaTeX that we can't hope to cover every different possibility or problem you might encounter. The LyX users' list is frequented by people with lots of experience with layout design who are willing to share what they've learned, so please feel free to ask questions there.

As you prepare to write a new layout, it is extremely helpful to look at the layouts distributed with LyX. If you write a LyX layout for a LaTeX document class that might also be used by others, or write a module that might be useful to others, then you should consider posting your layout to the layout section on the LyX wiki or even to the LyX developers' list, so that it might be included in LyX itself.[4]

5.2.1 Layout modules

We have spoken to this point about 'layout files'. But there are different sorts of files that contain layout information. Layout files, strictly so called, have the `.layout` extension and provide LyX with information about document classes. As of LyX 1.6, however, layout information can also be contained in layout *modules*, which have the `.module` extension. Modules are to LaTeX packages much as layouts are to LaTeX classes, and some modules—such as the `endnotes` module—specifically provide support for one package. In a sense, layout modules are similar to included[5] files—files like `stdsections.inc`—in that modules are not specific to a given document class but may be used with many different classes. The difference is that using an included file with `article.cls` requires editing that file. Modules, by contrast, are selected in the Document ▷ Settings dialog.

Building modules is the easiest way to get started with layout editing, since it can be as simple as adding a single new paragraph style or flex inset. But modules may, in principle, contain anything a layout file can contain.

After creating a new module and copying it to the `layouts/` folder, you will need to reconfigure and then restart LyX for the module to appear in the menu. However, changes you make to the module will be seen immediately, if you open Document ▷ Settings, highlight something, and then hit "OK". *It is strongly recommended that you save your work before doing this.* In fact, *it is strongly recommended that you not attempt to edit modules while simultaneously working on actual documents.* Though of course the developers strive to keep LyX stable in such situations, syntax errors and the like in your module file could cause strange behavior.

[4] Note that LyX is licensed under the General Public License, so any material that is contributed to LyX must be similarly licensed.

[5] These can have any extension, but by convention have the `.inc` extension.

5.2.1.1 Local Layout

Modules are to LyX as packages are to LaTeX. Sometimes, however, you find yourself wanting a specific inset or character style just for one document and writing a module that will also be available to other documents makes little sense. What you need is LyX's "Local Layout".

You will find it under Document ▷ Settings ▷ Local Layout. The large text box allows you to enter anything that you might enter in a layout file or module. You can think of a document's local layout, in fact, as a module that belongs just to it. So, in particular, you must enter a `Format` tag. Any format is acceptable, but one would normally use the format current at the time. (In LyX 2.0, the current layout format is 35.) You should be aware that local layout is not supported by versions of LyX prior to 1.6, so you should not use it if you want to be able to export your document to LyX 1.5 or earlier (without, that is, losing the local layout information). If you wish to be able to export to 1.6—local layout is supported in 1.6, though there is no UI for it—then you should use format 11 and, of course, use only layout constructs that were available in LyX 1.6.

When you have entered something in the `Local Layout` pane, LyX will enable the "Validate" button at the bottom. Clicking this button will cause LyX to determine whether what you have entered is valid layout information for the chosen format. LyX will report the result but, unfortunately, will not tell you what errors there might have been. These will be written to the terminal, however, if LyX is started from a terminal. You will not be permitted to save your local layout until you have entered something valid.

The warnings at the end of the previous section apply here, too. Do not play with local layout while you are actually working, especially if you have not saved your document. That said, using local layout with a test document can be a very convenient way to try out layout ideas, or even to start developing a module.

5.2.2 Layout for .sty files

There are two situations you are likely to encounter when wanting to support a new LaTeX document class, involving style (`.sty`) files and LaTeX 2_ε class (`.cls`) . Supporting a style file is usually fairly easy. Supporting a new class file is a bit harder. We'll discuss the former in this section and the latter in the next. Similar remarks apply, of course, if you want to support a new DocBook DTD.

The easier case is the one in which your new document class is provided as a style file that is to be used in conjunction with an already supported document class. For the sake of the example, we'll assume that the style file is called myclass.sty and that it is meant to be used with report.cls, which is a standard class.

Start by copying the existing class's layout file into your local directory:[6]

[6] Of course, which directory is your local directory will vary by platform, and LyX allows you to specify your local directory on startup, too, using the `-userdir` option.

```
cp report.layout ~/.lyx/layouts/myclass.layout
```

Then edit `myclass.layout` and change the line:

```
\DeclareLaTeXClass{report}
```

to read

```
\DeclareLaTeXClass[report, myclass.sty]{report (myclass)}
```

Then add:

```
Preamble
    \usepackage{myclass}
EndPreamble
```

near the top of the file.

Start LyX and select Tools ▷ Reconfigure. Then restart LyX and try creating a new document. You should see "report (myclass)" as a document class option in the Document ▷ Settings dialog. It is likely that some of the sectioning commands and such in your new class will work differently from how they worked in the base class—`report` in this example—so you can fiddle around with the settings for the different sections if you wish. The layout information for sections is contained in `stdsections.inc`, but you do not need to copy and change this file. Instead, you can simply add your changes to your layout file, after the line `Input stdclass.inc`, which itself includes `stdsections.inc`. For example, you might add these lines:

```
Style Chapter
    Font
        Family Sans
    EndFont
End
```

to change the font for chapter headings to sans-serif. This will override (or, in this case, add to) the existing declaration for the Chapter style.

Your new package may also provide commands or environments not present in the base class. In this case, you will want to add these to the layout file. See 5.3 for information on how to do so.

If myclass.sty can be used with several different document classes, and even if it cannot, you might find it easiest just to write a module that you can load with the base class. The simplest possible such module would be:

```
#\DeclareLyXModule{My Package}
#DescriptionBegin
#Support for mypkg.sty.
```

```
#DescriptionEnd
Format 21
Preamble
    \usepackage{mypkg}
EndPreamble
```

A more complex module might modify the behavior of some existing constructs or define some new ones. Again, see 5.3 for discussion.

5.2.3 Layout for .cls files

There are two possibilities here. One is that the class file is itself based upon an existing document class. For example, many thesis classes are based upon book.cls. To see whether yours is, look for a line like

```
\LoadClass{book}
```

in the file. If so, then you may proceed largely as in the previous section, though the `DeclareLaTeXClass` line will be different. If your new class is `thesis` and it is based upon `book`, then the line should read:[7]

```
\DeclareLaTeXClass[thesis,book]{thesis}
```

If, on the other hand, the new class is not based upon an existing class, you will probably have to "roll your own" layout. We strongly suggest copying an existing layout file which uses a similar LaTeX class and then modifying it, if you can do so. At least use an existing file as a starting point so you can find out what items you need to worry about. Again, the specifics are covered below.

5.2.4 Creating templates

Once you have written a layout file for a new document class, you might want to consider writing a *template* for it, too. A template acts as a kind of tutorial for your layout, showing how it might be used, though containing dummy content. You can of course look at the various templates included with LyX for ideas.

Templates are created just like usual documents: using LyX. The only difference is that usual documents contain all possible settings, including the font scheme and the paper size. Usually a user doesn't want a template to overwrite his preferred settings for such parameters. For that reason, the designer of a template should remove the corresponding commands like `\font_roman` or `\papersize` from the template LyX file. This can be done with any simple text-editor, for example `vi` or `notepad`.

[7] And it will be easiest if you save the file to `thesis.layout`: LyX assumes that the document class has the same name as the layout file.

Put the edited template files you create in `UserDir/templates/`, copy the ones you use from the global template directory in `LyXDir/templates/` to the same place, and redefine the template path in the Tools ▷ Preferences ▷ Paths dialog.

Note, by the way, that there is a template which has a particular meaning: `defaults.lyx`. This template is loaded every time you create a new document with File ▷ New in order to provide useful defaults. To create this template from inside LyX, all you have to do is to open a document with the correct settings, and use the Save as Document Defaults button.

5.2.5 Upgrading old layout files

The format of layout files changes with each LyX release, so old layout files need to be converted to the new format. This process has been automated since LyX 1.4: If LyX reads a layout file in an older format, it automatically calls the script `layout2layout.py` to convert it to a temporary file in current format. The original file is left untouched. If you use the layout file often, then, you may want to convert it permanently, so that LyX does not have to do so itself every time. To do this, you can call the converter manually:

```
mv myclass.layout myclass.old
python LyXDir/scripts/layout2layout.py myclass.old myclass.layout
```

You need to replace `LyXDir` with the name of your LyX system directory, of course.

Note that manual conversion does not affect included files, so these will have to be converted separately.

5.3 The layout file format

When it's finally time to get your hands dirty and create or edit your own layout file, the following sections describe what you're up against. Our advice is to go slowly, save and test often, listen to soothing music, and enjoy one or two of your favorite adult beverages; more if you are getting particularly stuck. It's really not that hard, except that the multitude of options can become overwhelming if you try to do too much in one sitting. Go have another adult beverage, just for good measure.

Note that all the tags used in layout files are case-insensitive. This means that `Style`, `style` and `StYlE` are really the same tag. The possible values are printed in brackets after the feature's name. The default value if a feature isn't specified inside a text class-description is typeset *`emphasized`*. If the argument has a data type like "string" or "float", the default is shown like this: `float=`*`default`*.

5.3.1 The document class declaration

Lines in a layout file which begin with `#` are comments. There is one exception to this rule. All `*.layout` files should begin with a line like:

```
#% Do not delete the line below; configure depends on this
#  \DeclareLaTeXClass{article}
```

The second line is used when you (re)configure LyX. The layout file is read by the LaTeX script `chkconfig.ltx`, in a special mode where `#` is ignored. The first line is just a LaTeX comment, and the second one contains the declaration of the text class. If these lines appear in a file named `article.layout`, then they define a text class of name `article` (the name of the layout file) which uses the LaTeX document class `article.cls` (the default is to use the same name as the layout). The string "article" that appears above is used as a description of the text class in the Document ▷ Settings dialog.

Let's assume that you wrote your own text class that uses the `article.cls` document class, but where you changed the appearance of the section headings. If you put it in a file `myarticle.layout`, the header of this file should be:

```
#% Do not delete the line below; configure depends on this
#  \DeclareLaTeXClass[article]{article (with my own headings)}
```

This declares a text class `myarticle`, associated with the LaTeX document class `article.cls` and described as "article (with my own headings)". If your text class depends on several packages, you can declare it as:

```
#% Do not delete the line below; configure depends on this
#  \DeclareLaTeXClass[article,foo.sty]{article (with my own headings)}
```

This indicates that your text class uses the `foo.sty` package. Finally, it is also possible to declare classes for DocBook code. Typical declarations will look like:

```
#% Do not delete the line below; configure depends on this
# \DeclareDocBookClass[article]{SGML (DocBook article)}
```

Note that these declarations can also be given an optional parameter declaring the name of the document class (but not a list).

So, to be as explicit as possible, the form of the layout declaration is:

```
#  \DeclareLaTeXClass[class,package.sty]{layout description}
```

The class need only be specified if the name of the LaTeX class file and the name of the layout file are different or if there are packages to load. If the name of the class file is not specified, then LyX will simply assume that it is the same as the name of the layout file.

When the text class has been modified to your taste, all you have to do is to copy it either to `LyXDir/layouts/` or to `UserDir/layouts`, run Tools ▷ Reconfigure, exit LyX and restart. Then your new text class should be available along with the others.

Once the layout file is installed, you can edit it and see your changes without having to reconfigure or to restart LyX. [8] You can force a reload of the current layout by using the LyX function layout-reload. There is no default binding for this function—though, of course, you can bind it to a key yourself. But you will normally use this function simply by entering it in the mini-buffer.

Warning: layout-reload is very much an 'advanced feature'. It is *strongly* recommended that you save your work before using this function. In fact, it is *strongly* recommended that you not attempt to edit layout information while simultaneously working on a document that you care about. Use a test document. Syntax errors and the like in your layout file could cause peculiar behavior. In particular, such errors could cause LyX to regard the current layout as invalid and to attempt to switch to some other layout.[9] The LyX team strives to keep LyX stable in such situations, but safe is better than sorry.[10]

5.3.2 The Module declaration

A module must begin with a line like the following:

```
#\DeclareLyXModule[endnotes.sty]{Endnotes}
```

The mandatory argument, in curly brackets, is the name of the module, as it should appear in Document ▷ Settings ▷ Modules. The argument in square brackets is optional: It declares any LaTeX packages on which the module depends. Please note that only packages about which LyX knows should be listed in the square brackets.[11] LyX will not check for arbitrary packages. It is also possible to use the form from->to as an optional argument, which declares that the module can only be used when there exists a conversion chain between the formats '`from`' and '`to`'.

The module declaration should then be followed by lines like the following[12]:

```
#DescriptionBegin
#Adds an endnote command, in addition to footnotes.
#You will need to add  \theendnotes in TeX code where you
#want the endnotes to appear.
#DescriptionEnd
#Requires: somemodule | othermodule
#Excludes: badmodule
```

[8] In versions of LyX prior to 1.6, this was not true. As a result, editing layout files was very time consuming, since you had constantly to restart LyX to see changes.

[9] Really bad syntax errors may even caused LyX to exit. This is because certain sorts of errors may make LyX unable to read *any* layout information. Please be careful.

[10] While we're giving advice: make regular backups. And be nice to your mother.

[11] The list of such packages is documented only in the source code.

[12] Preferably in English if the module should be published with LyX. This description will appear in the list of messages to be translated and will be thus translated with the next interface update.

The description is used in Document ▷ Settings ▷ Modules to provide the user with information about what the module does. The `Requires` line is used to identify other modules with which this one must be used; the `Excludes` line is used to identify modules with which this one may not be used. Both are optional, and, as shown, multiple modules should be separated with the pipe symbol: |. Note that the required modules are treated disjunctively: *at least one* of the required modules must be used. Similarly, *no* excluded module may be used. Note that modules are identified here by their *filenames* without the `.module` extension. So `somemodule` is really `somemodule.module`.

5.3.3 Format number

The first non-comment line of any layout file, included file, or module *must* contain the file format number:

`Format` [`int`] The format of the layout file.

This tag was introduced with LyX 1.4.0. Layout files from LyX 1.3.x and earlier don't have an explicit file format and are considered to be of format 1. The format for the present version of LyX is format 21. But each version of LyX is capable of reading earlier versions' layout files, just as they are capable of reading files produced by earlier versions of LyX. There is, however, no provision for converting to earlier formats. So LyX 1.6.x will not read layout files in format 21 but only files in format 11 or earlier.

5.3.4 General text class parameters

These are general parameters that govern the behavior of an entire document class. (This does *not* mean that they must appear in `.layout` files rather than in modules. A module can contain any layout tag.)

`AddToHTMLPreamble` Adds information that will be output in the `<head>` block when this document class is output to XHTML. Typically, this would be used to output CSS style information, but it can be used for anything that can appear in `<head>`. Must end with "`EndPreamble`".

`AddToPreamble` Adds information to the document preamble. Must end with "`EndPreamble`".

`CiteFormat` Defines formats for use in the display of bibliographic information. See Section 5.3.12 for details. Must end with "`End`".

`ClassOptions` Describes various global options supported by the document class. See Section 5.3.5 for a description. Must end with "`End`".

`Columns` [*1*, 2] Whether the class should *default* to having one or two columns. Can be changed in the Document ▷ Settings dialog.

Counter [string] This sequence defines the properties for a counter. If the counter does not yet exist, it is created; if it does exist, it is modified. Must end with "End".
See Section 5.3.10 for details on counters.

DefaultFont Sets the default font used to display the document. See Section 5.3.11 for how to declare fonts. Must end with "EndFont".

DefaultModule [string] Specifies a module to be included by default with this document class. The module should be specified by filename without the .module extension. The user can still remove the module, but it will be active at the outset. (This applies only when new files are created, or when this class is chosen for an existing document.)

DefaultStyle [string] This is the style that will be assigned to new paragraphs, usually Standard. This will default to the first defined style if not given, but you are encouraged to use this directive.

ExcludesModule [string] This tag indicates that the module in question—which should be specified by filename without the .module extension—cannot be used with this document class. This might be used in a journal-specific layout file to prevent, say, the use of the theorems-sec module that numbers theorems by section. This tag may *not* be used in a module. Modules have their own way of excluding other modules (see 5.2.1).

Float Defines a new float. See Section 5.3.8 for details. Must end with "End".

HTMLPreamble Sets the information that will be output in the <head> block when this document class is output to XHTML. Note that this will completely override any prior HTMLPreamble or AddToHTMLPreamble declarations. (Use AddToHTMLPreamble if you just want to add material to the preamble.) Must end with "EndPreamble".

HTMLTOCSection [string] The layout to use for the table of contents, bibliography, and so forth, when the document is output to HTML. For articles, this should normally be Section; for books, Chapter. If it is not given, then LyX will attempt to figure out which layout to use.

IfCounter [string] Modifies the properties of the given counter. If the counter does not exist, the section is ignored. Must end with "End".
See Section 5.3.10 for details on counters.

IfStyle [string] Modifies the properties of the given paragraph style. If the style does not exist, the section is ignored. Must end with "End".

Input As its name implies, this command allows you to include another layout definition file within yours to avoid duplicating commands. Common examples are the standard layout files, for example, stdclass.inc, which contains most of the basic layouts.

InsetLayout This section (re-)defines the layout of an inset. It can be applied to an existing inset or to a new, user-defined inset, e.g., a new character style. Must end with "End".
See Section 5.3.9 for more information.

LeftMargin [string] A string that indicates the width of the left margin on the screen, for example, "MMMMM". (Note that this is not a 'length', like "2ex".)

NoCounter [string] This command deletes an existing counter, usually one defined in an included file.

NoFloat This command deletes an existing float. This is particularly useful when you want to suppress a float that has been defined in an input file.

NoStyle This command deletes an existing style. This is particularly useful when you want to suppress a style that has be defined in an input file.

OutputFormat A string indicating the file format (as defined in the Preferences dialog) produced by this class. It is mainly useful when OutputType is 'literate' and one wants to define a new type of literate document. This string is reset to 'docbook', 'latex', or 'literate' when the corresponding OutputType parameter is encountered.

OutputType A string indicating what sort of output documents using this class will produce. At present, the options are: 'docbook', 'latex', and 'literate'.

PageStyle [*plain*, empty, headings] The default pagestyle. Can be changed in the Document ▷ Settings dialog.

Preamble Sets the preamble for the LaTeX document. Note that this will completely override any prior Preamble or AddToPreamble declarations. (Use AddToPreamble if you just want to add material to the preamble.) Must end with "EndPreamble".

Provides [string] [*0*, 1] Whether the class already provides the feature string. A feature is in general the name of a package (amsmath, makeidx, ...) or a macro (url, boldsymbol,...); the complete list of supported features is unfortunately not documented outside the LyX source code—but see LaTeXFeatures.cpp if you're interested. Help ▷ LaTeX Configuration also gives an overview of the supported packages.

ProvidesModule [string] Indicates that this layout provides the functionality of the module mentioned, which should be specified by the filename without the .module extension. This will typically be used if the layout includes the module directly, rather than using the DefaultModule tag to indicate that it ought to be used. It could also be used in a module that provided an alternate implementation of the same functionality.

Requires [string] Whether the class requires the feature `string`. Multiple features must be separated by commas. Note that you can only request supported features. (Again, see `LaTeXFeatures.cpp` for a list of these.)

RightMargin A string that indicates the width of the right margin on the screen, for example, "MMMMM".

SecNumDepth Sets which divisions get numbered. Corresponds to the `secnumdepth` counter in LaTeX.

Sides [*1*, 2] Whether the class-default should be printing on one or both sides of the paper. Can be changed in the Document ▷ Settings dialog.

Style This sequence defines a paragraph style. If the style does not yet exist, it is created; if it does exist, its parameters are modified. Must end with "`End`". See Section 5.3.6 for details on paragraph styles.

TitleLatexName [`string="maketitle"`] The name of the command or environment to be used with `TitleLatexType`.

TitleLatexType [*`CommandAfter`*, `Environment`] Indicates what kind of markup is used to define the title of a document. `CommandAfter` means that the macro with name `TitleLatexName` will be inserted after the last layout which has "`InTitle 1`". `Environment` corresponds to the case where the block of paragraphs which have "`InTitle 1`" should be enclosed into the `TitleLatexName` environment.

TocDepth Sets which divisions are included in the table of contents. Corresponds to the `tocdepth` counter in LaTeX.

5.3.5 `ClassOptions` section

The `ClassOptions` section can contain the following entries:

FontSize [`string="10|11|12"`] The list of available font sizes for the document's main font, separated by "`|`".

Header Used to set the DTD line with XML-based output classes. E. g.: PUBLIC "-//OASIS//DTD DocBook V4.2//EN".

PageStyle [`string="empty|plain|headings|fancy"`] The list of available page styles, separated by "`|`".

Other [`string=""`] Some document class options, separated by a comma, that will be added to the optional part of the `\documentclass` command.

The `ClassOptions` section must end with "`End`".

5.3.6 Paragraph styles

A paragraph style description looks like this:[13]

```
Style name
  ...
End
```

where the following commands are allowed:

Align [*block*, left, right, center] Paragraph alignment.

AlignPossible [*block*, left, right, center] A comma separated list of permitted alignments. (Some LaTeX styles prohibit certain alignments, since those wouldn't make sense. For example a right-aligned or centered enumeration isn't possible.)

BabelPreamble Note that this will completely override any prior BabelPreamble declaration for this style. Must end with "EndBabelPreamble". See section 5.3.7 for details on its use.

BottomSep [float=0][14] The vertical space with which the last of a chain of paragraphs with this style is separated from the following paragraph. If the next paragraph has another style, the separations are not simply added, but the maximum is taken.

Category [string] The category for this style. This is used to group related styles in the style combobox on the toolbar. Any string can be used, but you may want to use existing categories with your own styles.

CommandDepth Depth of XML command. Used only with XML-type formats.

CopyStyle [string] Copies all the features of an existing style into the current one.

DependsOn The name of a style whose preamble should be output *before* this one. This allows to ensure some ordering of the preamble snippets when macros definitions depend on one another.[15]

EndLabeltype [*No_Label*, Box, Filled_Box, Static] The type of label that stands at the end of the paragraph (or sequence of paragraphs if LatexType is Environment, Item_Environment or List_Environment). No_Label means "nothing", Box (resp. Filled_Box) is a white (resp. black) square suitable for end of proof markers, Static is an explicit text string.

[13] Note that this will either define a new style or modify an existing one.
[14] Note that a 'float' here is a real number, such as: 1.5.
[15] Note that, besides that functionality, there is no way to ensure any ordering of preambles. The ordering that you see in a given version of LyX may change without warning in later versions.

EndLabelString [string=""] The string used for a label with a Static EndLabelType.

Font The font used for both the text body *and* the label. See section 5.3.11. Note that defining this font automatically defines the LabelFont to the same value. So you should define this one first if you also want to define LabelFont.

FreeSpacing [*0*, 1] Usually LyX doesn't allow you to insert more than one space between words, since a space is considered as the separation between two words, not a character or symbol of its own. This is a very fine thing but sometimes annoying, for example, when typing program code or plain LaTeX code. For this reason, FreeSpacing can be enabled. Note that LyX will create protected blanks for the additional blanks when in another mode than LaTeX-mode.

HTML* These tags are used with XHTML output. See 5.4.1.

InnerTag [[FIXME]] (Used only with XML-type formats.)

InPreamble [1, 0] If 1, marks the style as to be included in the document preamble rather than in the document body. This is useful for document classes that want such information as the title and author to appear in the preamble. Note that this works only for styles for which the LatexType is Command or Paragraph.

InTitle [1, 0] If 1, marks the style as being part of a title block (see also the TitleLatexType and TitleLatexName global entries).

ItemSep [float=0] This provides extra space between paragraphs that have the same style. If you put other styles into an environment, each is separated with the environment's Parsep. But the whole items of the environment are additionally separated with this Itemsep. Note that this is a *multiplier.*

ItemTag [[FIXME]] (Used only with XML-type formats.)

KeepEmpty [*0*, 1] Usually LyX does not allow you to leave a paragraph empty, since it would lead to empty LaTeX output. There are some cases where this could be desirable however: in a letter template, the required fields can be provided as empty fields, so that people do not forget them; in some special classes, a style can be used as some kind of break, which does not contain actual text.

LabelBottomsep [float=0] The vertical space between the label and the text body. Only used for labels that are above the text body (Top_Environment, Centered_Top_Environ

LabelCounter [string=""]
The name of the counter for automatic numbering.
This *must* be given if LabelType is Counter. In that case, the counter will be stepped each time the style appears.
This *may* also be given if LabelType is Enumerate, though this case is a bit complicated. Suppose you declare "LabelCounter myenum". Then the actual

counters used are `myenumi`, `myenumii`, `myenumiii`, and `myenumiv`, much as in LaTeX. These counters must all be declared separately.
See Section 5.3.10 for details on counters.

`LabelFont` The font used for the label. See section 5.3.11.

`LabelIndent` Text that indicates how far a label should be indented.

`Labelsep` [`string=""`] The horizontal space between the label and the text body. Only used for labels that are not above the text body.

`LabelString` [`string=""`] The string used for a label with a `Static` labeltype. When `LabelCounter` is set, this string can be contain the special formatting commands described in Section 5.3.10.[16]

`LabelStringAppendix` [`string=""`] This is used inside the appendix instead of `LabelString`. Note that every `LabelString` statement resets `LabelStringAppendix` too.

`LabelTag` [FIXME] (Used only with XML-type formats.)

`LabelType` [*`No_Label`*, `Manual, Static, Top_Environment, Centered_Top_Environment, Counter, Sensitive, Enumerate, Itemize, Bibliography`]

- `Manual` means the label is the very first word (up to the first real blank). Use protected spaces (like that one) if you want more than one word as the label.
- `Static` means the label is simply whatever `LabelString` declares it to be. Note that this really is 'static'.
- `Top_Environment` and `Centered_Top_Environment` are special cases of `Static`. The label will be printed above the paragraph, but only at the top of an environment or the top of a chain of paragraphs with this style. This might be used with the Abstract style, for example.
- `Sensitive` is a special case for the caption-labels "Figure" and "Table". `Sensitive` means the (hardcoded) label string depends on the kind of float: It is hardcoded to be 'FloatType N', where N is the value of the counter associated with the float.
- The `Counter` label type defines automatically numbered labels. The `LabelString` will be expanded to resolve any counter references it contains: For example, it might be "`Section \thechapter.\thesection`". See Section 5.3.10 for more information on counters.

[16] For the sake of backwards compatibility, the string `@`*`style-name`*`@` will be replaced by the expanded `LabelString` of style *`style-name`*. This feature is now obsolete and should be replaced by the mechanisms of Section 5.3.10.

- `Enumerate` produces the usual sort of enumeration labels. At present, it is hardcoded to use Arabic numerals, lowercase letters, small Roman numerals, and uppercase letters for the four possible depths.
- `Itemize` produces various bullets at the different levels. It is also hardcoded.
- `Bibliography` is used internally by LyX and should be used only with `LatexType BibEnvironment`.

`LangPreamble` Note that this will completely override any prior `LangPreamble` declaration for this style. Must end with "`EndLangPreamble`". See section 5.3.7 for details on its use.

`LatexName` The name of the corresponding LaTeX stuff. Either the environment or command name.

`LatexParam` An optional parameter for the corresponding `LatexName` stuff. This parameter cannot be changed from within LyX.

`LatexType` [*`Paragraph`*, `Command, Environment, Item_Environment, List_Environment, Bib_Environment`] How the style should be translated into LaTeX.[17]

- `Paragraph` means nothing special.
- `Command` means `\`*`LatexName`*`{...}`.
- `Environment` means `\begin{`*`LatexName`*`}...\end{`*`LatexName`*`}`.
- `Item_Environment` is the same as `Environment`, except that an `\item` is generated for each paragraph of this environment.
- `List_Environment` is the same as `Item_Environment`, except that `LabelWidthString` is passed as an argument to the environment. `LabelWidthString` can be defined in the Edit ▷ Paragraph settings dialog.

Putting the last few things together, the LaTeX output will be either:

```
\latexname[latexparam]{...}
```

or:

```
\begin{latexname}[latexparam] ... \end{latexname}.
```

depending upon the LaTeX type.

`LeftMargin` [`string=""`] If you put styles into environments, the leftmargins are not simply added, but added with a factor $\frac{4}{depth+4}$. Note that this parameter is also used when the margin is defined as `Manual` or `Dynamic`. Then it is added to the manual or dynamic margin.

[17] `LatexType` is perhaps a bit misleading, since these rules apply to SGML classes, too. Visit the SGML class files for specific examples.

The argument is passed as a string. For example "MM" means that the paragraph is indented with the width of "MM" in the normal font. You can get a negative width by prefixing the string with "-". This way was chosen so that the look is the same with each used screen font.

Margin [*Static*, Manual, Dynamic, First_Dynamic, Right_Address_Box] The kind of margin that the style has on the left side. Static just means a fixed margin. Manual means that the left margin depends on the string entered in the Edit ▷ Paragraph Settings dialog. This is used to typeset nice lists without tabulators. Dynamic means that the margin depends on the size of the label. This is used for automatic enumerated headlines. It is obvious that the headline "5.4.3.2.1 Very long headline" must have a wider left margin (as wide as "5.4.3.2.1" plus the space) than "3.2 Very long headline", even if standard "word processors" are not able to do this. First_Dynamic is similar, but only the very first row of the paragraph is dynamic, while the others are static; this is used, for example, for descriptions. Right_Address_Box means the margin is chosen in a way that the longest row of this paragraph fits to the right margin. This is used to typeset an address on the right edge of the page.

NeedProtect [*0*,1] Whether fragile commands in this style should be \protect'ed. (Note: This is *not* whether this command should itself be protected.)

Newline [0, *1*] Whether newlines are translated into LaTeX newlines (\\) or not. The translation can be switched off to allow more comfortable LaTeX editing inside LyX.

NextNoIndent [1, *0*] If set to true, and if DefaultStyle (usually Standard) paragraphs are being indented, then the indentation of such a paragraph following one of this type will be suppressed. (So this will not affect the display of non-default paragraphs.)

ObsoletedBy Name of a style that has replaced this style. This is used to rename a style, while keeping backward compatibility.

OptionalArgs [int=0] The number of optional arguments that can be used with this style. This is useful for things like section headings, and only makes sense with LaTeX. Note that, on output, the optional arguments will all precede any required arguments (see below). So one can have constructs like:

```
\mycmd[opt1]{req1}{contents of paragraph}
```

but one cannot have things like:

```
\mycmd[opt1]{req1}[opt2]{contents of paragraph}
```

at least, not without ERT (with which you can have anything).

ParbreakIsNewline [*0*, 1] Indicates that paragraphs will not be separated by an empty line in LaTeX output, but only by a line break; together with **PassThru** 1, this allows to emulate a plain text editor (like the ERT inset).

ParIndent [string=""] The indent of the very first line of a paragraph. The **Parindent** will be fixed for a certain style. The exception is the default style, since the indentation for these paragraphs can be prohibited with **NextNoIndent**. Also, **Standard** style paragraphs inside environments use the **Parindent** of the environment, not their native one. For example, **Standard** paragraphs inside an enumeration are not indented.

Parsep [float=0] The vertical space between two paragraphs of this style.

Parskip [float=0] LyX allows the user to choose either "indent" or "skip" to typeset a document. When "indent" is chosen, this value is completely ignored. When "skip" is chosen, the parindent of a LaTeXtype "Paragraph" style is ignored and all paragraphs are separated by this parskip argument. The vertical space is calculated with **value * DefaultHeight** where **DefaultHeight** is the height of a row with the normal font. This way, the look stays the same with different screen fonts.

PassThru [*0*, 1] Whether the contents of this paragraph should be output in raw form, meaning without special translations that LaTeX would require.

Preamble Information to be included in the LaTeX preamble when this style is used. Used to define macros, load packages, etc., required by this particular style. Must end with "**EndPreamble**".

RefPrefix [string] The prefix to use when creating labels referring to paragraphs of this type. This allows the use of formatted references.

RequiredArgs [int=0] The number of required arguments that the LaTeX command or environment corresponding to this style expects. In the case of a command, these are required arguments *other than* that associated with the content of the paragraph itself. These do not actually have to be provided: LyX will output empty arguments if necessary. Note that optional arguments will be output before required arguments. See the discussion of the **OptionalArgs** tag above for more information.

Requires [string] Whether the style requires the feature **string**. See the description of **Provides** above (page 32) for information on 'features'.

RightMargin [string=""] Similar to **LeftMargin**.

Spacing [*single*, **onehalf**, **double**, **other** *value*] This defines what the default spacing should be in the style. The arguments **single**, **onehalf** and **double**

correspond respectively to a multiplier value of 1, 1.25 and 1.667. If you specify the argument `other`, then you should also provide a numerical argument which will be the actual multiplier value. Note that, contrary to other parameters, `Spacing` implies the generation of specific LaTeX code, using the package `setspace.sty`.

`Spellcheck` [`0`,*`1`*] Spellcheck paragraphs of this style. Default is true.

`TextFont` The font used for the text body . See section 5.3.11.

`TocLevel` `[int]` The level of the style in the table of contents. This is used for automatic numbering of section headings.

`TopSep` [`float=0`] The vertical space with which the very first of a chain of paragraphs with this style is separated from the previous paragraph. If the previous paragraph has another style, the separations are not simply added, but the maximum is taken.

5.3.7 Internationalization of Paragraph Styles

LyX has long supported internationalization of layout information, but, until version 2.0, this applied only to the user interface and not to, say, PDF output. Thus, French authors were forced to resort to ugly hacks if they wanted 'Théorème 1' instead of 'Theorem 1'. Thanks to Georg Baum, that is no longer the case.

If a `Style` defines text that is to appear in the typeset document, it may use `LangPreamble` and `BabelPreamble` to support non-English and even multi-language documents correctly. The following excerpt (from the `theorems-ams.inc` file) shows how this works:

```
Preamble
  \theoremstyle{remark}
  \newtheorem{claim}[thm]{\protect\claimname}
EndPreamble
LangPreamble
  \providecommand{\claimname}{_(Claim)}
EndLangPreamble
BabelPreamble
    \addto\captions$$lang{\renewcommand{\claimname}{_(Claim)}}
EndBabelPreamble
```

In principle, any legal LaTeX may appear in the `LangPreamble` and `BabelPreamble` tags, but in practice they will typically look as they do here. The key to correct translation of the typeset text is the definition of the LaTeX command `\claimname` and its use in `\newtheorem`.

The `LangPreamble` tag provides for internationalization based upon the overall language of the document. The contents of the tag will be included in the preamble, just as with the `Preamble` tag. What makes it special is the use of the "function" `_()`, which will be replaced, when LyX produces LaTeX output, with the translation of its argument into the document language.

The `BabelPreamble` tag is more complex, since it is meant to provide support for multi-language documents and so offers an interface to the `babel` package. Its contents will be added to the preamble once for each language that appears in the document. In this case, the argument to `_()` will be replaced with its translation into the language in question; the expression `$$lang` is replaced by the language name (as used by the babel package).

A German document that also included a French section would thus have the following in the preamble:

```
\addto\captionsfrench{\renewcommand{\claimname}{Affirmation}}   \addto\captio
```

LaTeX and `babel` will then conspire to produce the correct text in the output.

One important point to note here is that the translations are provided by LyX itself, through the same mechanism it uses for internationalization of the user interface. This means, in effect, that `LangPreamble` and `BabelPreamble` are really only of use in layout files that are provided with LyX, since text entered in user-created layout files will not be seen by LyX's internationalization routines. That said, however, any layout created with the intention that it will be included with LyX should use these tags where appropriate.

5.3.8 Floats

Since version 1.3.0 of LyX, it is has been both possible and necessary to define the floats (figure, table, ...) in the text class itself. Standard floats are included in the file `stdfloats.inc`, so you may have to do no more than add

```
Input stdfloats.inc
```

to your layout file. If you want to implement a text class that proposes some other float types (like the AGU class bundled with LyX), the information below will hopefully help you:

`Extension` [`string=""`] The file name extension of an auxiliary file for the list of figures (or whatever). LaTeX writes the captions to this file.

`GuiName` [`string=""`] The string that will be used in the menus and also for the caption. This is translated to the current language if babel is used.

`HTML*` These are used for XHTML output. See 5.4.

IsPredefined [*0*, 1] Indicates whether the float is already defined in the document class or if we instead need to load `float.sty` and use what it provides to define it on-the-fly. The default is `0`, which means: use `float.sty`. It should be set to `1` if the float is already defined by the LaTeX document class.

ListCommand [string=""] The command used to generate a list of floats of this type; the leading '\' should be omitted. This *must* be given if `UsesFloatPkg` is false, since there is no standard way to generate this command. It is ignored if `UsesFloatPkg` is true, since in that case there is a standard way to define the command.

ListName [string=""] A title for a list of floats of this kind (list of figures, tables, or whatever). It is used for the screen label within LyX; it is passed to LaTeX for use as the title there; and it is used as the title in XHTML output. It will be translated to the document language.

NumberWithin [string=""] This (optional) argument determines whether floats of this class will be numbered within some sectional unit of the document. For example, if within is equal to `chapter`, the floats will be numbered within chapters.

Placement [string=""] The default placement for the given class of floats. The string should be as in standard LaTeX: `t`, `b`, `p` and `h` for top, bottom, page, and here, respectively.[18] On top of that there is a new type, `H`, which does not really correspond to a float, since it means: put it "here" and nowhere else. Note however that the `H` specifier is special and, because of implementation details, cannot be used in non-built in float types. If you do not understand what this means, just use "`tbp`".

RefPrefix [string] The prefix to use when creating labels referring to floats of this type. This allows the use of formatted references. Note that you can remove any `RefPrefix` set by a copied style by using the special value "OFF", which must be all caps.

Style [string=""] The style used when defining the float using `\newfloat`.

Type [string=""] The "type" of the new class of floats, like program or algorithm. After the appropriate `\newfloat`, commands such as `\begin{program}` or `\end{algorithm*}` will be available.

UsesFloatPkg [0, *1*] Tells us whether this float is defined using the facilities provided by `float.sty`, either by the class file or a package, or on-the-fly by LyX itself.

Note that defining a float with type ***type*** automatically defines the corresponding counter with name ***type***.

[18] Note that the order of these letters in the string is irrelevant, like in LaTeX.

5.3.9 Flex insets and InsetLayout

LyX has supported character styles since version 1.4.0; as of version 1.6.0, these are called Flex insets.

Flex insets come in three different kinds:

- character style (`CharStyle`): These define semantic markup corresponding to such LaTeX commands as `\noun` and `\code`.
- user custom (`Custom`): These can be used to define custom collapsible insets, similar to TeX code, footnote, and the like. An obvious example is an endnote inset, which is defined in the `endnote` module.
- XML elements (`Element`): For use with DocBook classes.

Flex insets are defined using the `InsetLayout` tag, which shall be explained in a moment.

The `InsetLayout` tag also serves another function: It can be used to customize the general layout of many different types of insets. Currently, `InsetLayout` can be used to customize the layout parameters for footnotes, marginal notes, note insets, TeX code (ERT) insets, branches, listings, indexes, boxes, tables, algorithms, URLs, and optional arguments, as well as to define Flex insets.

The `InsetLayout` definition must begin with a line of the form:

```
InsetLayout <Type>
```

Here `<Type>` indicates the inset whose layout is being defined, and here there are two cases.

1. The layout for a pre-existing inset is being modified. In this case, can be `<Type>` any one of the following: `Algorithm`, `Branch`, `Box`, `Box:shaded`, `ERT`, `Figure`, `Foot`, `Index`, `Info`, `Info:menu`, `Info:shortcut`, `Info:shortcuts`, `Listings`, `Marginal`, `Note:Comment`, `Note:Note`, `Note:Greyedout`, `OptArg`, `Table`, or `URL`.
2. The layout for a Flex inset is being defined. In this case, `<Type>` must be of the form "`Flex:<name>`", where `name` may be be any valid identifier not used by a pre-existing Flex inset. The identifier may include spaces, but in that case the whole thing must be wrapped in quotes. Note that the definition of a flex inset *must* also include a `LyXType` entry, declaring which type of inset it defines.

The `InsetLayout` definition can contain the following entries:

`BgColor` The color for the inset's background. The valid colors are defined in `src/ColorCode.h`.

`ContentAsLabel` [*0*,1] Whether to use the content of the inset as the label, when the inset is closed. Default is false.

CopyStyle As with paragraph styles (see page 5.3.6).

CustomPars [*0*,1] Indicates whether the user may employ the Paragraph Settings dialog to customize the paragraph.

Decoration can be `Classic`, `Minimalistic`, or `Conglomerate`, describing the rendering style used for the inset's frame and buttons. Footnotes generally use `Classic`, ERT insets generally `Minimalistic`, and character styles `Conglomerate`.

Display [0,*1*] Only useful if `LatexType` is `Environment`. Indicates whether the environment will stand on its own in LaTeX output or will appear inline with the surrounding text. If set to false, it is supposed that the LaTeX environment ignores white space (including one newline character) after the `\begin{`*LatexName*`}` and `\end{`*LatexName*`}` tags. Default is true.

End Required at the end of the InsetLayout declarations.

Font The font used for both the text body *and* the label. See section 5.3.11. Note that defining this font automatically defines the `LabelFont` to the same value, so define this first and define `LabelFont` later if you want them to be different.

ForceLTR Force the "latex" language, leading to Left-to-Right (latin) output, e. g. in TeX code or URL. A kludge.

ForcePlain [*0*,1] Indicates whether the `PlainLayout` should be used or, instead, the user can change the paragraph style used in the inset. Default is false.

FreeSpacing As with paragraph styles (see page 32). Default is false.

HTML* These tags control XHTML output. See section 5.4.

InToc [*0*,1] Whether to include the contents of this inset in the strings generated for the 'Outline' pane. One would not, for example, want the content of a footnote in a section header to be included in the TOC displayed in the outline, but one would normally want the content of a character style displayed. Default is false: not to include.

KeepEmpty As with paragraph styles (see page 32). Default is false.

LabelFont The font used for the label. See section 5.3.11. Note that this definition can never appear before `Font`, lest it be ineffective.

LabelString What will be displayed on the button or elsewhere as the inset label. Some inset types (`TeX code` and `Branch`) modify this label on the fly.

LatexName The name of the corresponding LaTeX stuff. Either the environment or command name.

LatexParam The optional parameter for the corresponding `LatexName` stuff, including possible bracket pairs like `[]`. This parameter cannot be changed from within LyX.

LatexType As with paragraph styles (see page 34).

LyxType Can be `charstyle`, `custom`, `element`, or `end` (indicating a dummy definition ending definitions of charstyles, etc). This entry is required in and is only meaningful for Flex insets. Among other things, it determines on which menu this inset will appear. Setting `LyXType` to `charstyle` will set `MultiPar` to false. `MultiPar` can be set to true for charstyle insets, if you wish, by setting it *after* you set the `LyXType`.

MultiPar [*0*,1] Whether multiple paragraphs are permitted in this inset. This will also set `CustomPars` to the same value and `ForcePlain` to the opposite value. These can be reset to other values, if they are used *after* `MultiPar`. Default is true.

NeedProtect [*0*,1] Whether fragile commands in this inset should be `\protect`'ed. (Note: This is *not* whether the command should itself be protected.) Default is false.

ParbreakIsNewline [*0*, 1] As with paragraph styles (see page 36). Default is false.

PassThru [*0*,1] As with paragraph styles (see page 36). Default is false.

Preamble As with paragraph styles (see page 36).

RefPrefix [`string`] The prefix to use when creating labels referring to insets of this type. This allows the use of formatted references.

Requires [`string`] As with paragraph styles (see page 36).

ResetsFont [0,*1*] Whether this inset should use the font of its surrounding environment or uses its own. Default is true: uses its own.

Spellcheck [0,*1*] Spellcheck the contents of this inset. Default is true.

5.3.10 Counters

Since version 1.3.0 of LyX, it is both possible and necessary to define the counters (chapter, figure, ...) in the text class itself. The standard counters are defined in the file `stdcounters.inc`, so you may have to do no more than add

```
Input stdcounters.inc
```

to your layout file to get them to work. But if you want to define custom counters, then you can do so. The counter declaration must begin with:

```
Counter CounterName
```

where of course '`CounterName`' is replaced by the name of the counter. And it must end with "`End`". The following parameters can also be used:

`LabelString` [`string=""`] When defined, this string defines how the counter is displayed. Setting this value sets `LabelStringAppendix` to the same value. The following special constructs can be used in the string:

- `\thecounter` will be replaced by the expansion of the `LabelString` (or `LabelStringAppendix`) of the counter `counter`.
- counter values can be expressed using LaTeX-like macros `\`*`numbertype`*`{`*`counter`*`}`, where *`numbertype`* can be:[19] `arabic`: 1, 2, 3,...; `alph` for lower-case letters: a, b, c, ...; `Alph` for upper-case letters: A, B, C, ...; `roman` for lower-case roman numerals: i, ii, iii, ...; `Roman` for upper-case roman numerals: I, II, III...; `hebrew` for hebrew numerals.

If LabelString is not defined, a default value is constructed as follows: if the counter has a master counter `master` (defined via `Within`), the string `\themaster.\arabic{counter}` is used; otherwise the string `\arabic{counter}` is used.

`LabelStringAppendix` [`string=""`] Same as `LabelString`, but for use in the Appendix.

`PrettyFormat` [`string=""`] A format for use with formatted references to this counter. For example, one might want to have references to section numbers appear as "Section 2.4". The string should contain "##". This will be replaced by the counter number itself. So, for sections, it would be: Section ##.

`Within` [`string=""`] If this is set to the name of another counter, the present counter will be reset every time the other one is increased. For example, `subsection` is numbered inside `section`.

5.3.11 Font description

A font description looks like this:

```
Font or LabelFont
  ...
EndFont
```

The following commands are available:

`Color` [*`none`*, `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow`]

[19] Actually, the situation is a bit more complicated: any *numbertype* other than those described below will produce arabic numerals. It would not be surprising to see this change in the future.

Family [*Roman*, Sans, Typewriter]

Misc [string] Valid arguments are: emph, noun, underbar, no_emph, no_noun and no_bar. Each of these turns on or off the corresponding attribute. For example, emph turns on emphasis, and no_emph turns it off.
If the latter seems puzzling, remember that the font settings for the present context are generally inherited from the surrounding context. So no_emph would turn off the emphasis that was anyway in effect, say, in a theorem environment.

Series [*Medium*, Bold]

Shape [*Up*, Italic, SmallCaps, Slanted]

Size [tiny, small, *normal*, large, larger, largest, huge, giant]

5.3.12 Citation format description

The CiteFormat blocks are used to describe how bibliographic information should be displayed, both within LyX itself (in the citation dialog and in tooltips, for example) and in XHTML output. Such a block might look like this:

```
CiteFormat
 article ...
 book ...
End
```

The individual lines define how the bibliographic information associated with an article or book, respectively, is to be displayed, and such a definition can be given for any 'entry type' that might be present in a BibTeX file. LyX defines a default format in the source code that will be used if no specific definition has been given. LyX predefines several formats in the file stdciteformats.inc, which is included in most of LyX's document classes.

The definitions use a simple language that allows BibTeX keys to be replaced with their values. Keys should be enclosed in % signs, e.g.: %author%. So a simple definition might look like this:

```
misc %author%, "%title".
```

This would print the author, followed by a comma, followed by the title, in quotes, followed by a period.

Of course, sometimes you may want to print a key only if it exists. This can be done by using a conditional construction, such as: {%volume%[[vol. %volume%]]}. This says: If the volume key exists, then print "vol. " followed by the volume key. It is also possible to have an else clause in the conditional, such as: {%author%[[%author%]][[%editor%, ed.]]}. Here, the author key is printed if it exists; otherwise, the editor key is printed, followed by ", ed." Note that the key is again enclosed in % signs; the entire

conditional is enclosed in braces; and the if and else clauses are enclosed in double brackets, "`[[`" and "`]]`". There must be no space between any of these.

There is one other piece of syntax available in definitions, which looks like this: `{!<i>!}`. This defines a piece of formatting information that is to be used when creating "rich text". Obviously, we do not want to output HTML tags when writing plain text, so they should be wrapped in "{!" and "!}".

Two special sorts of definitions are also possible in a `CiteFormat` block. An example of the first would be:

```
!quotetitle "%title%"
```

This is an abbreviation, or macro, and it can be used by treating it as if it were a key: `%!quotetitle%`. LyX will treat `%!quotetitle%` exactly as it would treat its definition. So, let us issue the obvious *warning*. Do not do this:

```
!funfun %funfun%
```

or anything like it. LyX shouldn't go into an infinite loop, but it may go into a long one before it gives up.

The second sort of special definition might look like this:

```
_pptext pp.
```

This defines a translatable piece of text, which allows relevant parts of the bibliography to be translated. It can be included in a definition by treating it as a key: `%_pptext%`. Several of these are predefined in `stdciteformats.inc`. Note that these are not macros, in the sense just defined. They will not be expanded.

So here then is an example that use all these features:

```
!authoredit {%author%[[%author%, ]][[{%editor%[[%editor%, %_edtext%, ]]}]]}
```

This defines a macro that prints the author, followed by a comma, if the `author` key is defined, or else prints the name of the editor, followed by the `_edtext` or its translation (it is by default "ed."), if the `editor` key is defined. Note that this is in fact defined in `stdciteformats.inc`, so you can use it in your own definitions, or re-definitions, if you load that file first.

5.4 Tags for XHTML output

As with LaTeX or DocBook, the format of LyX's XHTML output is also controlled by layout information. In general, LyX provides sensible defaults and, as mentioned earlier, it will even construct default CSS style rules from the other layout tags. For example, LyX will attempt to use the information provided in the `Font` declaration for the Chapter style to write CSS that will appropriately format chapter headings.

In many cases, then, you may not have to do anything at all to get acceptable XHTML output for your own environments, custom insets, and so forth. But in some cases you will, and so LyX provides a number of layout tags that can be used to customize the XHTML and CSS that are generated.

Note that there are two tags, `HTMLPreamble` and `AddToHTMLPreamble` that may appear outside style and inset declarations. See 5.3.4 for details on these.

5.4.1 Paragraph styles

The sort of XHTML LyX outputs for a paragraph depends upon whether we are dealing with a normal paragraph, a command, or an environment, where this is itself determined by the contents of the corresponding `LaTeXType` tag.

For a command or normal paragraph, the output XHTML has the following form:

```
<tag attr="value">
<labeltag attr="value">Label</labeltag>
Contents of the paragraph.
</tag>
```

The label tags are of course omitted if the paragraph does not have a label.

For an environment that is not some sort of list, the XHTML takes this form:

```
<tag attr="value">
<itemtag attr="value"><labeltag attr="value">Environment Label</labeltag>Firs
<itemtag>Second paragraph.</itemtag>
</tag>
```

Note that the label is output only for the first paragraph, as it should be for a theorem, for example.

For a list, we have one of these forms:

```
<tag attr="value">
<itemtag attr="value"><labeltag attr="value">List Label</labeltag>First item.
<itemtag attr="value"><labeltag attr="value">List Label</labeltag>Second item
</tag>
<tag attr="value">
<labeltag attr="value">List Label</labeltag><itemtag attr="value">First item.
<labeltag attr="value">List Label</labeltag><itemtag attr="value">Second item
</tag>
```

Note the different orders of `labeltag` and `itemtag`. Which order we get depends upon the setting of `HTMLLabelFirst`: If `HTMLLabelFirst` is false (the default), you get the first of these, with the label within the item; if true, you get the second, with the label outside the item.

The specific tags and attributes output for each paragraph type can be controlled by means of the layout tags we are about to describe. As mentioned earlier, however, LyX uses sensible defaults for many of these, so you often may not need to do very much to get good XHTML output. Think of the available tags as there so you can tweak things to your liking.

HTMLAttr [string] Specifies attribute information to be output with the main tag. For example, "class='mydiv'". By default, LyX will output "class='layoutname'", where layoutname is the LyX name of the layout, made lowercase, for example: chapter. This should *not* contain any style information. Use HTMLStyle for that purpose.

HTMLForceCSS [*0*,1] Whether to output the default CSS information LyX generates for this layout, even if additional information is explicitly provided via HTMLStyle. Setting this to 1 allows you to alter or augment the generated CSS, rather than to override it completely. Default is 0.

HTMLItem [string] The tag to be used for individual paragraphs of environments, replacing itemtag in the examples above. Defaults to div.

HTMLItemAttr [string] Attributes for the item tag. Defaults to "class='layoutname_item'". This should *not* contain any style information. Use HTMLStyle for that purpose.

HTMLLabel [string] The tag to be used for paragraph and item labels, replacing labeltag in the examples above. Defaults to span, unless LabelType is either Top_Environment or Centered_Top_Environment, in which case it defaults to div.

HTMLLabelAttr [string] Attributes for the label tag. Defaults to "class='layoutname_label'". This should *not* contain any style information. Use HTMLStyle for that purpose.

HTMLLabelFirst [*0*,1] Meaningful only for list-like environments, this tag controls whether the label tag is output before or inside the item tag. This is used, for example, in the description environment, where we want '<dt>...</dt><dd>...</dd>. Default is 0: The label tag is output inside the item tag.

HTMLPreamble Information to be output in the <head> section when this style is used. This might, for example, be used to include a <script> block defining an onclick handler.

HTMLStyle CSS style information to be included when this style is used. Note that this will automatically be wrapped in a layout-generated <style> block, so only the CSS itself need be included.

HTMLTag [string] The tag to be used for the main label, replacing tag in the examples above. Defaults to div.

HTMLTitle [*0*,1] Marks this style as the one to be used to generate the `<title>` tag for the XHTML file. By default, it is false. The `stdtitle.inc` file sets it to true for the `title` environment.

5.4.2 InsetLayout XHTML

The XHTML output of insets can also be controlled by information in layout files.[20] Here, too, LyX tries to provide sensible defaults, and it constructs default CSS style rules. But everything can be customized.

The XHTML LyX outputs for an inset has the following form:

```
<tag attr="value">
<labeltag>Label</labeltag>
<innertag attr="value">Contents of the inset.</innertag>
</tag>
```

If the inset permits multiple paragraphs—that is, if `MultiPar` is true—then the contents of the inset will itself be output as paragraphs formatted according to the styles used for those paragraphs (standard, quote, and the like). The label tag is of course omitted if the paragraph does not have a label and, at present, is always `span`. The inner tag is optional and, by default, does not appear.

The specific tags and attributes output for each inset can be controlled by means of the following layout tags.

HTMLAttr [`string`] Specifies attribute information to be output with the main tag. For example, "`class='myinset' onclick='...'`". By default, LyX will output "`class='insetname'`", where `insetname` is the LyX name of the inset, made lowercase and with non-alphanumeric characters converted to underscores, for example: footnote.

HTMLForceCSS [*0*,1] Whether to output the default CSS information LyX generates for this layout, even if additional information is explicitly provided via `HTMLStyle`. Setting this to `1` allows you to alter or augment the generated CSS, rather than to override it completely. Default is 0.

HTMLInnerAttr [`string`] Attributes for the inner tag. Defaults to "`class='insetname_inner'`".

HTMLInnerTag [`string`] The inner tag, replacing `innertag` in the examples above. By default, there is none.

HTMLIsBlock [0,*1*] Whether this inset represents a standalone block of text (such as a footnote) or instead represents material that is included in the surrounding text (such as a branch). Defaults to 1.

[20] At present, this is true only for "text" insets (insets you can type into) and is not true for "command" insets (insets that are associated with dialog boxes).

HTMLLabel [string] A label for this inset, possibly including a reference to a counter. For example, for footnote, it might be: `\arabic{footnote}`. This is optional, and there is no default.

HTMLPreamble Information to be output in the `<head>` section when this style is used. This might, for example, be used to include a `<script>` block defining an `onclick` handler.

HTMLStyle CSS style information to be included when this style is used. Note that this will automatically be wrapped in a layout-generated `<style>` block, so only the CSS itself need be included.

HTMLTag [string] The tag to be used for the main label, replacing `tag` in the examples above. The default depends upon the setting of `MultiPar`: If `MultiPar` is true, the default is `div`; if it is false, the default is `span`.

5.4.3 Float XHTML

The XHTML output for floats too can be controlled by layout information. The output has the following form:

```
<tag attr="value">
Contents of the float.
</tag>
```

The caption, if there is one, is a separate inset and will be output as such. Its appearance can be controlled via the InsetLayout for caption insets.

HTMLAttr [string] Specifies attribute information to be output with the main tag. For example, "`class='myfloat' onclick='...'`". By default, LyX will output "`class='float float-floattype'`", where `floattype` is LyX's name for this type of float, as determined by the float declaration (see 5.3.8), though made lowercase and with non-alphanumeric characters converted to underscores, for example: float-table.

HTMLStyle CSS style information to be included when this float is used. Note that this will automatically be wrapped in a layout-generated `<style>` block, so only the CSS itself need be included.

HTMLTag [string] The tag to be used for this float, replacing "`tag`" in the example above. The default is `div` and will rarely need changing.

5.4.4 Bibliography formatting

The bibliography can be formatted using `CiteFormat` blocks. See Section 5.3.12 for the details.

5.4.5 LyX-generated CSS

We have several times mentioned that LyX will generate default CSS style rules for both insets and paragraph styles, based upon the other layout information that is provided. In this section, we shall say a word about which layout information LyX uses and how.

At present, LyX auto-generates CSS only for font information, making use of the `Family`, `Series`, `Shape`, and `Size` specified in the `Font` declaration. (See 5.3.11.) The translation is mostly straightforward and obvious. For example, "`Family Sans`" becomes "`font-family: sans-serif;`". The correspondence of LyX sizes and CSS sizes is a little less obvious but nonetheless intuitive. See the `getSizeCSS()` function in `src/FontInfo.cpp` for the details.

6 Including External Material

WARNING: This portion of the documentation has not been updated for some time. We certainly hope that it is still accurate, but there are no guarantees.

The use of material from sources external to LyX is covered in detail in the *Embedded Objects* manual. This part of the manual covers what needs to happen behind the scenes for new sorts of material to be included.

6.1 How does it work?

The external material feature is based on the concept of a *template*. A template is a specification of how LyX should interface with a certain kind of material. As bundled, LyX comes with predefined templates for Xfig figures, various raster format images, chess diagrams, and LilyPond music notation. You can check the actual list by using the menu Insert ▷ File ▷ External Material. Furthermore, it is possible to roll your own template to support a specific kind of material. Later we'll describe in more detail what is involved, and hopefully you will submit all the templates you create so we can include them in a later LyX version.

Another basic idea of the external material feature is to distinguish between the original file that serves as a base for final material and the produced file that is included in your exported or printed document. For example, consider the case of a figure produced with `Xfig`. The Xfig application itself works on an original file with the `.fig` extension. Within Xfig, you create and change your figure, and when you are done, you save the `fig`-file. When you want to include the figure in your document, you invoke `transfig` in order to create a PostScript file that can readily be included in your LaTeX file. In this case, the `.fig` file is the original file, and the PostScript file is the produced file.

This distinction is important in order to allow updating of the material while you are in the process of writing the document. Furthermore, it provides us with the flexibility that is needed to support multiple export formats. For instance, in the case of a plain text file, it is not exactly an award-winning idea to include the figure as raw PostScript. Instead, you would either prefer to just include a reference to the figure or try to invoke some graphics to ASCII converter to make the final result look similar to the real graphics. The external material management allows you to do this, because it is parametrized on the different export formats that LyX supports.

Besides supporting the production of different products according to the exported format, it supports tight integration with editing and viewing applications. In the

case of an Xfig figure, you are able to invoke Xfig on the original file with a single click from within the external material dialog in LyX, and also preview the produced PostScript file with Ghostview with another click. No more fiddling around with the command line and/or file browsers to locate and manipulate the original or produced files. In this way, you are finally able to take full advantage of the many different applications that are relevant to use when you write your documents, and ultimately be more productive.

6.2 The external template configuration file

It is relatively easy to add custom external template definitions to LyX. However, be aware that doing this in an careless manner most probably *will* introduce an easily exploitable security hole. So before you do this, please read the discussion about security in section 6.4.

Having said that, we encourage you to submit any interesting templates that you create.

The external templates are defined in the `LyXDir/lib/external_templates` file. You can place your own version in `UserDir/external_templates`.

A typical template looks like this:

```
Template XFig
GuiName "XFig: $$AbsOrRelPathParent$$Basename"
HelpText
An XFig figure.
HelpTextEnd
InputFormat fig
FileFilter "*.fig"
AutomaticProduction true
Transform Rotate
Transform Resize
Format LaTeX
TransformCommand Rotate RotationLatexCommand
TransformCommand Resize ResizeLatexCommand
Product "$$RotateFront$$ResizeFront
         \\input{$$AbsOrRelPathMaster$$Basename.pstex_t}
         $$ResizeBack$$RotateBack"
UpdateFormat pstex
UpdateResult "$$AbsPath$$Basename.pstex_t"
Requirement "graphicx"
ReferencedFile latex "$$AbsOrRelPathMaster$$Basename.pstex_t"
ReferencedFile latex "$$AbsPath$$Basename.eps"
ReferencedFile dvi "$$AbsPath$$Basename.eps"
FormatEnd
```

```
Format PDF(pdfLaTeX)
TransformCommand Rotate RotationLatexCommand
TransformCommand Resize ResizeLatexCommand
Product "$$RotateFront$$ResizeFront
         \\input{$$AbsOrRelPathMaster$$Basename.pdftex_t}
         $$ResizeBack$$RotateBack"
UpdateFormat pdftex
UpdateResult "$$AbsPath$$Basename.pdftex_t"
Requirement "graphicx"
ReferencedFile latex "$$AbsOrRelPathMaster$$Basename.pdftex_t"
ReferencedFile latex "$$AbsPath$$Basename.pdf"
FormatEnd
Format Ascii
Product "$$Contents(\"$$AbsPath$$Basename.asc\")"
UpdateFormat asciixfig
UpdateResult "$$AbsPath$$Basename.asc"
FormatEnd
Format DocBook
Product "<graphic fileref=\"$$AbsOrRelPathMaster$$Basename.eps\">
         </graphic>"
UpdateFormat eps
UpdateResult "$$AbsPath$$Basename.eps"
ReferencedFile docbook "$$AbsPath$$Basename.eps"
ReferencedFile docbook-xml "$$AbsPath$$Basename.eps"
FormatEnd
Product "[XFig: $$FName]"
FormatEnd
TemplateEnd
```

As you can see, the template is enclosed in `Template` ... `TemplateEnd`. It contains a header specifying some general settings and, for each supported primary document file format, a section `Format` ... `FormatEnd`.

6.2.1 The template header

`AutomaticProduction true|false` Whether the file represented by the template must be generated by LyX. This command must occur exactly once.

`FileFilter <pattern>` A glob pattern that is used in the file dialog to filter out the desired files. If there is more than one possible file extension (e. g. tgif has `.obj` and `.tgo`), use something like `"*.{obj,tgo}"`. This command must occur exactly once.

`GuiName <guiname>` The text that is displayed on the button. This command must occur exactly once.

HelpText <text> HelpTextEnd The help text that is used in the External dialog. Provide enough information to explain to the user just what the template can provide him with. This command must occur exactly once.

InputFormat <format> The file format of the original file. This must be the name of a format that is known to LyX (see section 3.1). Use "*" if the template can handle original files of more than one format. LyX will attempt to interrogate the file itself in order to deduce its format in this case. This command must occur exactly once.

Template <id> A unique name for the template. It must not contain substitution macros (see below).

Transform Rotate|Resize|Clip|Extra This command specifies which transformations are supported by this template. It may occur zero or more times. This command enables the corresponding tabs in the external dialog. Each `Transform` command must have either a corresponding `TransformCommand` or a `TransformOption` command in the `Format` section. Otherwise the transformation will not be supported by that format.

6.2.2 The Format section

Format LaTeX|PDFLaTeX|PlainText|DocBook The primary document file format that this format definition is for. Not every template has a sensible representation in all document file formats. Please define nevertheless a `Format` section for all templates. Use a dummy text when no representation is available. Then you can at least see a reference to the external material in the exported document.

Option <name> <value> This command defines an additional macro `$$<name>` for substitution in `Product`. `<value>` itself may contain substitution macros. The advantage over using `<value>` directly in `Product` is that the substituted value of `$$<name>` is sanitized so that it is a valid optional argument in the document format. This command may occur zero or more times.

Product <text> The text that is inserted in the exported document. This is actually the most important command and can be quite complex. This command must occur exactly once.

Preamble <name> This command specifies a preamble snippet that will be included in the LaTeX preamble. It has to be defined using `PreambleDef ... PreambleDefEnd`. This command may occur zero or more times.

ReferencedFile <format> <filename> This command denotes files that are created by the conversion process and are needed for a particular export format. If the filename is relative, it is interpreted relative to the master document. This command may be given zero or more times.

Requirement <package> The name of a required LaTeX package. The package is included via `\usepackage{}` in the LaTeX preamble. This command may occur zero or more times.

TransformCommand Rotate RotationLatexCommand This command specifies that the built in LaTeX command should be used for rotation. This command may occur once or not at all.

TransformCommand Resize ResizeLatexCommand This command specifies that the built in LaTeX command should be used for resizing. This command may occur once or not at all.

TransformOption Rotate RotationLatexOption This command specifies that rotation is done via an optional argument. This command may occur once or not at all.

TransformOption Resize ResizeLatexOption This command specifies that resizing is done via an optional argument. This command may occur once or not at all.

TransformOption Clip ClipLatexOption This command specifies that clipping is done via an optional argument. This command may occur once or not at all.

TransformOption Extra ExtraLatexOption This command specifies that an extra optional argument is used. This command may occur once or not at all.

UpdateFormat <format> The file format of the converted file. This must be the name of a format that is known to LyX (see the Tools ▷ Preferences ▷ File Handling ▷ File Format dialog). This command must occur exactly once.

UpdateResult <filename> The file name of the converted file. The file name must be absolute. This command must occur exactly once.

6.2.3 Preamble definitions

The external template configuration file may contain additional preamble definitions enclosed by `PreambleDef ... PreambleDefEnd`. They can be used by the templates in the `Format` section.

6.3 The substitution mechanism

When the external material facility invokes an external program, it is done on the basis of a command defined in the template configuration file. These commands can contain various macros that are expanded before execution. Execution always take place in the directory of the containing document.

Also, whenever external material is to be displayed, the name will be produced by the substitution mechanism, and most other commands in the template definition support substitution as well.

The available macros are the following:

$$AbsOrRelPathMaster The file path, absolute or relative to the master LyX document.

$$AbsOrRelPathParent The file path, absolute or relative to the LyX document.

$$AbsPath The absolute file path.

$$Basename The filename without path and without the extension.

$$Contents("filename.ext") This macro will expand to the contents of the file with the name `filename.ext`.

$$Extension The file extension (including the dot).

$$FName The filename of the file specified in the external material dialog. This is either an absolute name, or it is relative to the LyX document.

$$FPath The path part of `$$FName` (absolute name or relative to the LyX document).

$$RelPathMaster The file path, relative to the master LyX document.

$$RelPathParent The file path, relative to the LyX document.

$$Sysdir This macro will expand to the absolute path of the system directory. This is typically used to point to the various helper scripts that are bundled with LyX.

$$Tempname A name and full path to a temporary file which will be automatically deleted whenever the containing document is closed, or the external material insertion deleted.

All path macros contain a trailing directory separator, so you can construct e. g. the absolute filename with `$$AbsPath$$Basename$$Extension`.

The macros above are substituted in all commands unless otherwise noted. The command `Product` supports additionally the following substitutions if they are enabled by the `Transform` and `TransformCommand` commands:

$$ResizeFront The front part of the resize command.

$$ResizeBack The back part of the resize command.

$$RotateFront The front part of the rotation command.

$$RotateBack The back part of the rotation command.

The value string of the `Option` command supports additionally the following substitutions if they are enabled by the `Transform` and `TransformOption` commands:

`$$Clip` The clip option.

`$$Extra` The extra option.

`$$Resize` The resize option.

`$$Rotate` The rotation option.

You may ask why there are so many path macros. There are mainly two reasons:

1. Relative and absolute file names should remain relative or absolute, respectively. Users may have reasons to prefer either form. Relative names are useful for portable documents that should work on different machines, for example. Absolute names may be required by some programs.

2. LaTeX treats relative file names differently than LyX and other programs in nested included files. For LyX, a relative file name is always relative to the document that contains the file name. For LaTeX, it is always relative to the master document. These two definitions are identical if you have only one document, but differ if you have a master document that includes part documents. That means that relative filenames must be transformed when presented to LaTeX. Fortunately LyX does this automatically for you if you choose the right macros.

So which path macro should be used in new template definitions? The rule is not difficult:

- Use `$$AbsPath` if an absolute path is required.
- Use `$$AbsOrRelPathMaster` if the substituted string is some kind of LaTeX input.
- Else use `$$AbsOrRelPathParent` in order to preserve the user's choice.

There are special cases where this rule does not work and e. g. relative names are needed, but normally it will work just fine. One example for such a case is the command `ReferencedFile latex "$$AbsOrRelPathMaster$$Basename.pstex_t"` in the XFig template above: We can't use the absolute name because the copier for `.pstex_t` files needs the relative name in order to rewrite the file content.

6.4 Security discussion

The external material feature interfaces with a lot of external programs and does so automatically, so we have to consider the security implications of this. In particular, since you have the option of including your own filenames and/or parameter strings

and those are expanded into a command, it seems that it would be possible to create a malicious document which executes arbitrary commands when a user views or prints the document. This is something we definitely want to avoid.

However, since the external program commands are specified in the template configuration file only, there are no security issues if LyX is properly configured with safe templates only. This is so because the external programs are invoked with the `execvp`-system call rather than the `system` system-call, so it's not possible to execute arbitrary commands from the filename or parameter section via the shell.

This also implies that you are restricted in what command strings you can use in the external material templates. In particular, pipes and redirection are not readily available. This has to be so if LyX should remain safe. If you want to use some of the shell features, you should write a safe script to do this in a controlled manner, and then invoke the script from the command string.

It is possible to design a template that interacts directly with the shell, but since this would allow a malicious user to execute arbitrary commands by writing clever filenames and/or parameters, we generally recommend that you only use safe scripts that work with the `execvp` system call in a controlled manner. Of course, for use in a controlled environment, it can be tempting to just fall back to use ordinary shell scripts. If you do so, be aware that you *will* provide an easily exploitable security hole in your system. Of course it stands to reason that such unsafe templates will never be included in the standard LyX distribution, although we do encourage people to submit new templates in the open source tradition. But LyX as shipped from the official distribution channels will never have unsafe templates.

Including external material provides a lot of power, and you have to be careful not to introduce security hazards with this power. A subtle error in a single line in an innocent looking script can open the door to huge security problems. So if you do not fully understand the issues, we recommend that you consult a knowledgeable security professional or the LyX development team if you have any questions about whether a given template is safe or not. And do this before you use it in an uncontrolled environment.