

Customizing LyX: Features for the Advanced User

by the LyX Team*

Version 2.4.x

May 13, 2024

*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 Kimberly Heck <rikiheck@lyx.org>.

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	5
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	6
3. The Preferences dialog	7
3.1. Formats	7
3.2. Copiers	8
3.3. Converters	9
4. Internationalizing LyX	13
4.1. Translating LyX	13
4.1.1. Translating the graphical user interface (text messages).	13
4.1.1.1. Ambiguous messages	14
4.1.2. Translating the documentation.	14
4.2. International Keymap Stuff	15
4.2.1. The .kmap File	16
4.2.2. The .cdef File	17
4.2.3. Dead Keys	18
4.2.4. Saving your Language Configuration	18
5. Installing New Document Classes	19
5.1. Installing new L ^A T _E X files	20
5.2. Types of layout files	22
5.2.1. Layout modules	22
5.2.1.1. Local Layout	23
5.2.2. Layout for .STY files	23
5.2.3. Layout for .CLS files	25
5.2.4. Creating templates	25
5.2.5. Upgrading old layout files	26
5.2.6. Cite engine files	26

5.3.	The layout file format	26
5.3.1.	The document class declaration and classification	27
5.3.2.	The Module declaration	28
5.3.3.	The CiteEngine file declaration	29
5.3.4.	Format number	30
5.3.5.	General text class parameters	30
5.3.6.	<code>ClassOptions</code> section	35
5.3.7.	Paragraph styles	35
5.3.8.	Internationalization of Paragraph Styles	43
5.3.9.	Floats	44
5.3.10.	Flex insets and <code>InsetLayout</code>	46
5.3.11.	Arguments	52
5.3.12.	Counters	54
5.3.13.	Font description	55
5.3.14.	Cite engine description	56
5.3.15.	Cite format description	58
5.4.	Tags for XHTML output	62
5.4.1.	Paragraph styles	62
5.4.2.	<code>InsetLayout XHTML</code>	64
5.4.3.	Float XHTML	65
5.4.4.	Bibliography formatting	66
5.4.5.	LyX-generated CSS	66
5.5.	Tags for DocBook output	66
5.5.1.	Paragraph styles	67
5.5.2.	New-line policy	68
5.5.3.	<code>InsetLayout DocBook</code>	68
5.5.4.	Float DocBook	72
5.5.5.	Bibliography formatting	72
6.	Including External Material	73
6.1.	How does it work?	73
6.2.	The external template configuration files	74
6.2.1.	The template header	75
6.2.2.	The Format section	76
6.2.3.	Preamble definitions	77
6.3.	The substitution mechanism	77
6.4.	Security discussion	80
A.	List of supported LyX functions to be used in layouts	81
B.	Names of available colors to be used in layouts	83
B.1.	Color functions	83
B.2.	Static colors	83
B.3.	Dynamic colors	84

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 L^AT_EX 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 system and user directories are by using `HELP ▷ ABOUT LYX`. The system 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 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 found in `USERDIR` are generated when you configure LyX. They contain various default values that are automatically detected during reconfiguration. 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 L^AT_EX document class and their description.

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

`*files.lst` Lists of various sorts of L^AT_EX-related files found on your system

2. LyX configuration files

`doc/TeXConfig.lyx` is automatically generated during configuration from the file `TeXConfig.lyx.in`. It contains information on your L^AT_EX configuration.

2.1.2. Directories

The following directories in `LyXDir` can be duplicated in `UserDir`. If a particular file exists in both places, the one in `UserDir` will be used.

- `bind/` This directory contains files with the extension `.bind` that define the keybindings used in LyX. If there exists an internationalized version of the bind file in a subdirectory `bind/xx` where “xx” is the ISO language code, that will be used first.
- `citeengines/` Contains files with the extension `.citeengine` which define the diverse citation possibilities (natbib, biblatex etc.). See sec. 5.2.6 for details.
- `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 sec. 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 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 sec. 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.
- `xtemplates/` Contains files with the extension `.xtemplate` which define the templates for the insertion of external material to a LyX document; see chapter 6.

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 L^AT_EX script used during the configuration process. Do not run directly.

configure.py This is a Python 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

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

latexfonts Contains information about the supported fonts.

layouttranslations This file contains translations for internationalized paragraph styles (see sec. 5.3.8).

unicodesymbols This file contains information about Unicode-encoded glyphs and the way they are supported by LyX via L^AT_EX.

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.

2. LyX configuration files

- If you get some updated documentation from a LyX ftp site and cannot install it because you do not have sysadmin rights on your system, you can just copy the files to `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 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 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_24x` 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 `CTRL+D` will `DOCUMENT▷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 `DOCUMENT▷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 `MIME` type of a format is optional, but if it is specified, it must be unique across all formats. It is used to detect files of this format from the file contents. For some important file formats there is no `MIME` type officially registered with the [IANA](http://iana.org). Therefore LyX uses the extended list of `MIME` types as specified by freedesktop.org.

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 `DOCUMENT▷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

3. The Preferences dialog

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 eight variables:

<code>\$\$s</code>	The LyX system directory (e. g. <code>/USR/SHARE/LYX</code>).
<code>\$\$i</code>	The input file
<code>\$\$o</code>	The output file
<code>\$\$b</code>	The base name (without filename extension) in the LyX temporary directory
<code>\$\$p</code>	The full directory path of the LyX temporary directory
<code>\$\$r</code>	The full pathname to the original LyX file being processed
<code>\$\$f</code>	The filename (without any directory path) of the LyX file.
<code>\$\$l</code>	The 'L ^A T _E X' name'

The latter should be the filename as it would be used in a L^AT_EX'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

¹For 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.

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.

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=flavor This converter runs some form of L^AT_EX. This will make LyX’s L^AT_EX error logs available. The optional **flavor** value specifies the form of L^AT_EX that is run (**latex**, **pdflatex**, **platex**, **xetex**, **luatex**). If no value is specified, **latex** is used.

needauth This converter is considered non-safe and needs authorization by the user. Depending on the settings in **Tools**▷**Preferences**▷**File Handling**▷**Converters**, users will (a.) be asked whether they trust the current document temporarily, permanently, or not, (b.) be informed that conversion is not possible due to security concerns, or (c.) not be informed as they gave permanent consent. Set this flag for converters that might execute arbitrary programs.

²This 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`.

3. The Preferences dialog

- needaux=flavor** Needs the \LaTeX .AUX file for the conversion. The optional **flavor** value specifies the form of \LaTeX that is run in order to generate the .AUX file (**latex**, **pdflatex**, **platex**, **xetex**, **luatex**). If no value is specified, **latex** is used.
- nice** Needs a “nice” file from the backend, which in practice means a \LaTeX file like the one we would export, without **input@path**.
- xml** Output is XML.

The following three flags are not really flags at all because they take an argument in the **KEY = VALUE** format:

- hyperref-driver** The name of the driver that needs to be loaded with the **hyperref** package for this converter. The loading of the correct driver is necessary to get some PDF-specific features. See the **hyperref** manual for details.
- 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’.

A suitable **hyperref-driver** is set for some converters that are installed with LyX. The last three flags, however, are presently not used in any of the pre-installed converters.

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 five ways to convert \LaTeX to PDF:

1. Directly, using **PDFLATEX**
2. via (DVI and) PostScript, using **PS2PDF**

3. via DVI, using DVIPDFM
4. directly using XETEX
5. directly using LUALATEX

To define such alternate chains, you must define multiple target ‘file formats’, as described in sec. 3.1. For example, in the standard configuration, the formats named PDF (for PS2PDF), PDF2 (for PDFLATEX), PDF3 (for DVIPDFM), PDF4 (for XETEX), and PDF5 (for LUALATEX) are defined, all of which share the extension .PDF, and which correspond to the conversion methods just mentioned.

4. Internationalizing L_YX

L_YX supports using a translated interface. Last time we checked, L_YX 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 L_YX.

4.1. Translating L_YX

4.1.1. Translating the graphical user interface (text messages).

L_YX uses the GNU `gettext` library to handle the internationalization of the interface. To have L_YX 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 L_YX community. Send a message to the L_YX 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 L_YX 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).

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

4. Internationalizing LyX

- 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 available³ and the locale is set accordingly, these will be used automatically by LyX. LyX looks for translated

²This 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, see https://www.gnu.org/software/gettext/manual/html_node/PO-Mode.html#PO-Mode.

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

versions as `LyXDir/doc/xx/DocName.lyx`, where `xx` is the code for the language currently in 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 <https://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. **Note:** For a complex document with external material (images, etc.), if you make a copy e.g. in a temp dir, beware that the links to external material may be broken when the document is moved to a different place. The best way is to retrieve the LyX tree from git (see <https://www.lyx.org/HowToUseGIT>) and to edit the doc file in place.
- Sometimes the original document (from the LyX team) will be updated. Use the source viewer at <https://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

4. Internationalizing LyX

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 key⁴ 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):

⁴The 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.

⁵Note 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. Creating and 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$ document classes.

First, let us say a few words about how one ought to think about the relation between LYX and $\text{L}\text{A}\text{T}\text{E}\text{X}$. The thing to understand is that, in a certain sense, LYX doesn't know anything about $\text{L}\text{A}\text{T}\text{E}\text{X}$. Indeed, from LYX 's point of view, $\text{L}\text{A}\text{T}\text{E}\text{X}$ is just one of several “backend formats” in which it is capable of producing output. Other such formats are DocBook, plaintext, and XHTML. $\text{L}\text{A}\text{T}\text{E}\text{X}$ is, of course, a particularly important format, but very little of the information LYX has about $\text{L}\text{A}\text{T}\text{E}\text{X}$ 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$, 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$, DocBook, and XHTML commands and tags. The `article.layout` file basically just includes several of these `std*.inc` files.

Defining the LYX – $\text{L}\text{A}\text{T}\text{E}\text{X}$ 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$ 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

¹Some commands are sufficiently complex that they are “hardcoded” into LYX . But the developers generally regard this as a Bad Thing.

5. Installing New Document Classes

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 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 sec. 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` or `/usr/local/texlive/XXXX` where `XXXX` is the year of the installed TeXLive distribution. The location of the

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

‘user’ T_EX tree is defined by `TEXMFHOME` and is commonly `$HOME/texmf` or `$HOME/.texliveXXXX`. (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’ 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 L^AT_EX is installed and then change to the subfolder `~\tex\latex`. (For MiK_TE_X, 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 L^AT_EX folder. E. g., for MiK_TE_X 2.8 under Windows XP, this would be the folder:

```
~:\Documents and Settings\\Application Data\  
    MiKTeX\2.8\tex\latex
```

On Vista, it would be:

```
~:\Users\\AppData\Roaming\2.8\MiKTeX\tex\latex
```

4. Now one only need to tell L^AT_EX that there are new files. This depends on the used L^AT_EX-Distribution:
- a) For T_EXLive 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 MiK_TE_X, 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 L^AX that there are new packages available. So, in L^AX, use the menu `TOOLS▷RECONFIGURE` and then restart L^AX.

Now the package is installed. In our example, the document class `FoilTeX` will now be available under `DOCUMENT▷SETTINGS▷DOCUMENT CLASS` (in the “Presentations” category).

If you would like to use a L^AT_EX 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.

³Note 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.

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 LyX should display them and how they should be translated into L^AT_EX, 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 L^AT_EX 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 L^AT_EX 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. Since LyX 1.6 layout information can also be contained in layout *modules*, which have the `.module` extension. Modules are to L^AT_EX packages much as layouts are to L^AT_EX classes, and some modules—such as the `endnotes` module—specifically provide support for one package. In a sense, layout modules are similar to included⁵ 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

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

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

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.

5.2.1.1. Local Layout

Modules are to LyX as packages are to L^AT_EX. 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, the current layout format is .)

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 L^AT_EX document class, involving style (.sty) files and L^AT_EX 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.

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

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

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

⁶Of 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.

5. Installing New Document Classes

```
\DeclareLaTeXClass{Report (Standard Class)}
```

to read

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

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 (MY CLASS)" 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 sec. 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 Class}
#DescriptionBegin
#Support for myclass.sty.
#DescriptionEnd
Format
Preamble
  \usepackage{myclass}
EndPreamble
```

A more complex module might modify the behavior of some existing constructs or define some new ones. Again, see sec. 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 L^AT_EX 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 L^YX for ideas.

Templates are created just like usual documents: using L^YX. 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 L^YX file. This can be done with any simple text-editor, for example `vi` or `notepad`.

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` \triangleright `PREFERENCES` \triangleright `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` \triangleright `NEW` in order to provide useful defaults. To create this template from inside L^YX, all you have to do is to open a document with the correct settings, and use the `SAVE AS DOCUMENT DEFAULTS` button.

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

5.2.5. Upgrading old layout files

The format of layout files changes with each L^AT_EX release, so old layout files need to be converted to the new format. If L^AT_EX 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 L^AT_EX does not have to do so itself every time. To do this, you can call the converter manually:

1. rename the file `myclass.layout` to `myclass.old`
2. Call the command

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

where `LyXDir` is the name of your L^AT_EX system directory.

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

5.2.6. Cite engine files

A specific form of layout files are the so-called `*.citeengine` files that are located in the `citeengines/` sub-directory. Their purpose is to define the specifics of L^AT_EX packages aimed at bibliography generation, such as `natbib`, `jurabib` or `biblatex`, but also the way how normal Bib_TE_X citations (without additional packages) are handled in L^AT_EX is defined in such a file.

More specifically, it is defined which packages L^AT_EX needs to load, which citation commands are available, how these are to be displayed in L^AT_EX (in the workarea, the dialogs, the context menus) as well as in the XHTML and plain text output. Furthermore, the files specify available style variants (author-year, numerical, etc.) and their specifics. The cite engine files are also used to generate the options that are available in `Document > Settings > Bibliography > Style engine`.

Even though a cite engine file is essentially a normal layout file that could theoretically include any layout information, it usually primarily includes some specific parameters such as `MaxCiteNames`, `CiteFramework`, `CiteEngine` and `CiteFormat` blocks. The syntax of the latter two is described in sec. 5.3.14 and sec. 5.3.15, as well as in the files themselves.

5.3. The layout file format

The following sections describe how layout files are structured and written. Our advice is to go slowly, save and test often. It is really not that hard, except that the multitude of options can become overwhelming, especially if you try to check out too many at once. It becomes easier if you use existing layouts of L^AT_EX as examples/reference or if you modify an existing layout to your needs.

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 arguments are printed in brackets after the tag's name. The default argument 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 and classification

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

```
#% Do not delete the line below; configure depends on this
# \DeclareLaTeXClass{Article (Standard Class)}
# \DeclareCategory{Articles}
```

The second and third lines are used when you (re)configure LyX. The layout file is read by the L^AT_EX script `chkconfig.ltx`, in a special mode where `#` is ignored. The first line is just a L^AT_EX comment, the second one contains the mandatory declaration of the text class and the third line contains the optional classification of the 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 L^AT_EX document class `article.cls` (the default is to use the same name as the layout). The string “Article (Standard Class)” that appears above is used as a description of the text class in the DOCUMENT▷SETTINGS dialog. The category (“Articles” in the example) is also used in the DOCUMENT▷SETTINGS dialog: the text classes are grouped by these categories (which are usually genres, so typical categories are “Articles”, “Books”, “Reports”, “Letters”, “Presentations”, “Curricula Vitae” etc.). If no category has been declared, the class will be put in the “Uncategorized” group.

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)}
# \DeclareCategory{Articles}
```

This declares a text class `myarticle`, associated with the L^AT_EX 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)}
# \DeclareCategory{Articles}
```

5. Installing New Document Classes

This indicates that your text class uses the `foo.sty` package.

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}
# \DeclareCategory{category}
```

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}
#\DeclareCategory{Foot- and Endnotes}
```

The mandatory argument of `\DeclareLyXModule`, 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

⁸In 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.

⁹Really 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.

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

module depends. 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 `\DeclareCategory` declaration is not strictly mandatory, but you should add it, since it is helpful to find the module. Please have a look at the existing module categories and if appropriate, use one of those.

The module and category declaration should then be followed by lines like the following:¹¹

```
#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
```

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. The CiteEngine file declaration

A cite engine file must begin with a line like the following:

```
#\DeclareLyXCiteEngineModule[biblatex.sty]{Biblatex}
```

The mandatory argument, in curly brackets, is the name of the cite style, as it should appear in `DOCUMENT▷SETTINGS▷BIBLIOGRAPHY`. The argument in square brackets is optional: It declares any \LaTeX packages on which the cite engine depends.

The cite engine declaration should then be followed by lines like the following:¹²

```
# DescriptionBegin
# Biblatex supports many author-year and numerical styles.
# It is mainly aimed at the Humanities. It is highly
# customizable, fully localized and provides many features
```

¹¹Preferably in English if the module should be published with \LaTeX . This description will appear in the list of messages to be translated and will be thus translated with the next interface update.

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

5. Installing New Document Classes

```
# that are not possible with BibTeX. The use of 'biber' as
# bibliography processor is advised.
# DescriptionEnd
```

The description is used in DOCUMENT▷SETTINGS▷BIBLIOGRAPHY to provide the user with information about the cite engine.

5.3.4. 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 number of the layout file.

This tag was introduced with LyX 1.4.0. Layout files from older LyX versions do not have an explicit file format and are considered to have **Format 1**. The format for the present version of LyX is format 60. 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.

5.3.5. 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.)

AddToCiteEngine <engine> Extends the possibilities for displaying citation references. See sec. 5.3.14 for details. Must end with **End**.

AddToHTMLPreamble Adds information that will be output in the <head> block when the document 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**”.

BibInToc [0, 1] If the document class adds the bibliography to the table of contents, add this option with value 1 (or **true**). This prevents the bibliography from being added twice.

CiteEngine <engine> Defines the possibilities for displaying citation references. See sec. 5.3.14 for details. Must end with “**End**”. Primarily used in cite engine files (see sec. 5.2.6). Note that if you specify this in a layout file or module, any cite engine definition will be overridden. Also see **AddToCiteEngine**.

- CiteFormat** Defines formats for use in the display of bibliographic information. See sec. 5.3.15 for details. Must end with “End”. Primarily used in cite engine files (see sec. 5.2.6). A cite format defined in a layout or module will override the cite engine definition.
- CiteFramework** [*bibtex*,*biblatex*] Determines whether Bib_lat_ex or Bib_TE_X is used to generate a Bibliography. Primarily used in cite engine files (see sec. 5.2.6).
- ClassOptions** Describes various global options supported by the document class. See sec. 5.3.6 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 sec. 5.3.12 for details on counters.
- DefaultFont** Sets the default font used to display the document. See sec. 5.3.13 for how to declare fonts. Must end with “EndFont”.
- DefaultModule** [*<module>*] Specifies a module to be included by default with this document class. The module is specified as 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** [*<style>*] 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.
- DocBookRoot** [*string*] The root element (at the top of the document) to use when outputting documents with this class in DocBook. The default value is “*article*”.
- DocBookForceAbstract** [*boolean*] If “*true*”, the root element will always have an *<abstract>* tag. The default value is “*false*”.
- ExcludesModule** [*<module>*] This tag indicates that the module (which is 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 sec. 5.2.1).
- Float** Defines a new float. See sec. 5.3.9 for details. Must end with “End”.

5. Installing New Document Classes

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 [`<style>`] The style 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 [`<counter>`] Modifies the properties of the given counter. If the counter does not exist, the statement is ignored. Must end with “`End`”.
See sec. 5.3.12 for details on counters.

Input [`<filename>`] This 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.

InputGlobal [`<filename>`] is a variant of the `Input` command which does not look for files in the user directory. This allows to create a file `name.layout` or `name.inc` in the user directory which includes a global file with the same name via `InputGlobal name` or `InputGlobal name.inc`, respectively (with `Input`, the file would recursively include itself). This way, you can modify global files without having to completely copy them.

InsetLayout [`<type>`] 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 sec. 5.3.10 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`”.)

MaxCiteNames [`<integer>`] An integer that determines the maximal number of names displayed in an author-year citation before the citation switches to “`FirstAuthor et al.`”. Primarily used in cite engine files (see sec. 5.2.6).

ModifyInsetLayout [`<type>`] Modifies the layout of an inset. If the layout does not exist, this section is ignored. Must end with “`End`”.

ModifyStyle [`<style>`] Modifies the properties of the given paragraph style. If the style does not exist, this section is ignored. Must end with “`End`”.

NoCounter [`<counter>`] Deletes an existing counter, usually one defined in an included file.

- NoFloat** [`<float>`] Deletes an existing float. This is particularly useful when you want to suppress a float that has been defined in an input file.
- NoStyle** [`<style>`] Deletes an existing style.
- OutlinerName** [`<type>`] [`<string>`] Define a new table of contents with type `<type>` and name `<string>`. See also the `AddToToc` commands.
- OutputFormat** [`<format>`] The file format (as defined in the LyX preferences) produced by this document class. It is mainly useful when `OutputType` is `literate` and one wants to define a new type of literate document. The format is reset to “`latex`” when the corresponding `OutputType` parameter is encountered.
- OutputType** [`latex`, `literate`] Specifies what sort of output documents using this class will produce.
- PackageOptions** [`string string`] Specifies options, given in the second string, for the package named by the first string. For example, “`PackageOptions natbib square`” will cause `natbib` to be loaded with the `square` option. (For T_EXperts, this causes LyX to output: `\PassOptionsToPackage{natbib}{square}` prior to loading `natbib`.)
- PageSize** [`custom`, `letter`, `legal`, `executive`, `a0`, `a1`, `a2`, `a3`, `a4`, `a5`, `a6`, `b0`, `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `c0`, `c1`, `c2`, `c3`, `c4`, `c5`, `c6`, `b0j`, `b1j`, `b2j`, `b3j`, `b4j`, `b5j`, `b6j`] The default page size. This is used by some converters.
- PageStyle** [`plain`, `empty`, `headings`] The default pagestyle. Can be changed in the DOCUMENT▷SETTINGS dialog.
- Preamble** Sets the preamble for the L^AT_EX 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`”.
- ProvideInsetLayout** [`<type>`] Defines the layout of an inset if it does not already exist. If the layout does exist, this section is ignored. Must end with “`End`”.
- Provides** [`string`] [`0, 1`] Whether the class already provides the feature `string`. A feature is in general the name of a package (e.g. `amsmath` or `makeidx`) or a macro (e.g. `url` or `boldsymbol`). See Appendix A for the list of features.
- 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.

5. Installing New Document Classes

ProvideStyle [`<style>`] Creates a new paragraph style if it does not already exist. If the style does exist, this section is ignored. Must end with “End”.

Requires [`string`] Whether the class requires the feature `string`. Multiple features must be separated by commas. Note that you can only request supported features. (See Appendix A for the list of features.). If you require a package with specific options, you can additionally use `PackageOptions`.

RightMargin [`string`] A string that indicates the width of the right margin on the screen, for example, “MMMMM”.

SecNumDepth [`int=3`] Sets which divisions get numbered. Corresponds to the `secnumdepth` counter in L^AT_EX.

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 [`<name>`] 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 sec. 5.3.7 for details on paragraph styles.

TableStyle [`<name>`] defines the default table style that is used when inserting a table. The following styles are available:

- `Formal_with_Footline`: formal (“booktabs”) style with horizontal lines only, using a bold top and bottom line, the first and last row are additionally separated from the table body with a thin middle line.
- `Formal_without_Footline`: same as the above, but the last row is not separated with a middle line from the body.
- `Simple_Grid`: Simple table lines.
- `Grid_with_Head`: Like `Simple_Grid`, but with the header row offset with a second horizontal line. This is also the default style of L_YX.
- `No_Borders`: Table without lines.

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 all layouts which have “InTitle 1” should be enclosed into the `TitleLatexName` environment.

TocDepth [`int=3`] Sets which divisions are included in the table of contents. Corresponds to the `tocdepth` counter in L^AT_EX.

5.3.6. 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 “|”. Any number is possible.

FontSizeFormat [`string`] The format for the font size option. Default: `$$spt. $$s` is a placeholder for the font size.

PageSize [`string="letter|legal|executive|a0|a1|a2|a3|a4|a5|a6|b0|b1|b2|b3|b4|b5|b6|c0|c1|c2|c3|c4|c5|c6|b0j|b1j|b2j|b3j|b4j|b5j|b6j"`] The list of available page sizes, separated by “|”. Currently, only the listed sizes are supported. Other sizes might be entered as custom class option.

PageSizeFormat [`string`] The format for the page size option. Default: `$$paper. $$s` is a placeholder for the paper size.

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”.

5.3.7. Paragraph styles

A paragraph style description looks like this:¹³

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

where the following commands are allowed:

AddToToc [`string=""`] This paragraph will appear in the table of contents of the given type. An empty string disables. See also the `OutlinerName` and the `IsTocCaption` commands. Default: disabled.

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.)

¹³Note that this will either define a new style or modify an existing one.

5. Installing New Document Classes

Argument [`int`] Defines argument number `<int>` of a command/environment associated with the current style. The definition must end with `EndArgument`. See sec. 5.3.11 for more information.

AutoNests Includes a comma-separated list of layouts that should be nested in and after the current layout. Only makes sense for nestable layouts (such as environments). Must be ended by “`EndAutoNests`”. See also `IsAutoNestedBy`.

BabelPreamble Note that this will completely override any prior `BabelPreamble` declaration for this style. Must end with “`EndBabelPreamble`”. See sec. 5.3.8 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.

CopyStyle [`string`] Copies all the features of an existing style into the current one. Note that this copies the style as it is defined at that point. Later changes to it will not affect styles into which it has been copied.

DependsOn [`<name>`] 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

DocBookGenerateTitle [`bool=false`] Generates a `title` tag after the wrapper tag. This parameter should only be used with `DocBookWrapperTag`, otherwise the title will be output *before* the contents of the environment. The generated title is the same as the LyXHTML label: a combination of the environment type and its number. A major use is when DocBook has no close mapping for LaTeX environments and users must fallback to using a generic container such as `asfigure`, which requires a title although there is none in LaTeX. This feature is heavily used for theorem-like environments.

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.

¹⁴Note that a ‘float’ here is a real number, such as: 1.5.

¹⁵Note 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 sec. 5.3.13. 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`.
- ForceLocal** [int=0] Used for backporting new styles to stable LyX versions. The first stable version that supports this tag is LyX 2.1.0. The argument is a number which may either be 0, -1 or any value greater than zero. If the `ForceLocal` flag of a style is greater than zero, it will always be written to the document header. If a .lyx file is read, the style definitions from the document header are added to the document class. Therefore even older LyX versions can handle the style. The argument of `ForceLocal` is a version number: if the style is read, and the version number is less than the version number of the already existing style in the document class, the new style is ignored. If the version number is greater, the new style replaces the existing style. A value of -1 means an infinite version number, i. e. the style is always used.
- FreeSpacing** [0, 1] Usually LyX does not 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 L^AT_EX code. For this reason, `FreeSpacing` can be enabled. LyX will create non-breaking spaces for the additional blanks when `Passthru 1` is not specified. Note that `FreeSpacing` implies `KeepEmpty`.
- HTML*** These tags are used with XHTML output. See sec. 5.4.1.
- InPreamble** [0, 1] If 1, marks the style as to be included in the document preamble and not 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** [0, 1] If 1, marks the style as being part of a title block (see also the `TitleLatexType` and `TitleLatexName` global entries).
- IsAutoNestedBy** Includes a comma-separated list of layouts after which this one should be nested. Only makes sense with regard to nestable layouts (such as environments). Must be ended by “`EndIsAutoNestedBy`”. See also `AutoNests`.
- IsTocCaption** [0, 1] If this is set to 1 and `AddToToc` is enabled, the paragraph adds a summary of its contents in its item in the table of contents. Otherwise, only the label, if it exists, appears.

5. Installing New Document Classes

ItemCommand [string="item"] The \LaTeX command sequence declaring an item in a list. The command is to be defined without the preceding backslash (the default is “item”, resulting in \backslash item in the \LaTeX output).

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*.

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** and **Centered_Top_Environment**).

LabelCounter [string=""] The name of the counter for automatic numbering. In order to have the counter appear with your label, you will need to reference it in the **LabelString**. This will work with **LabelTypes**, **Static**, **Above** and **Centered**, at least.

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 sec. 5.3.12 for details on counters.

LabelFont The font used for the label. See sec. 5.3.13.

LabelIndent [string=""] Text that indicates how far a label should be indented.

LabelSep [string=""] Text that indicates the amount of 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 the label. When **LabelCounter** is set, this string can be contain the special formatting commands described in sec. 5.3.12.

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

LabelType [*No_Label*, *Manual*, *Static*, *Above*, *Centered*, *Sensitive*, *Enumerate*, *Itemize*, *Bibliography*]

Manual means the label is the very first word (up to the first real blank). Use non-breaking spaces if you want more than one word as the label.

Static means the label is simply what is declared as `LabelString`. This will be displayed “inline”, at the beginning of the paragraph. If the `LatexType` is `Environment`, then it will be displayed only in the first paragraph of any sequence of paragraphs with the same `Style`.

Above and Centered are special cases of `Static`. The label will be printed above the paragraph either at the beginning of the line or centered.

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. For the case that a caption is inserted outside of a float the `LabelString` will appear as “Senseless!”.

Enumerate produces the usual sort of enumeration labels. The number type needs to be set in the `Counter`, see sec. 5.3.12.

Itemize produces various bullets at the different levels. The bullet types displayed can be set via `DOCUMENT`▷`SETTINGS`▷`BULLETS`.

Bibliography 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 sec. 5.3.8 for details on its use.

LatexName [`<name>`] The name of the corresponding `LATEX` stuff. Either the environment or command name.

LatexParam [`<parameter>`] An optional parameter for the corresponding `LatexName` stuff. This parameter cannot be changed from within `LyX` (use `Argument` for customizable parameters). This will be output as is after all `LATEX Arguments`.

LatexType [`Paragraph`, `Command`, `Environment`, `Item_Environment`, `List_Environment`, `Bib_Environment`] How the style should be translated into `LATEX`.¹⁶

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.

¹⁶`LatexType` is perhaps a bit misleading, since these rules apply to SGML classes, too. Visit the SGML class files for specific examples.

5. Installing New Document Classes

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.

Bib_Environment is like **Environment** but adds the necessary mandatory argument (the longest label) to the begin statement of the bibliography environment:

`\begin{thebibliography}{99}`. It is therefore only useful for bibliography environments. The default longest label “99” can be changed by the user in the paragraph settings of a bibliography item.

Putting the last few things together, the L^AT_EX output will be either:

```
\LatexName[LatexParam]{...}
```

or:

```
\begin{LatexName}[LatexParam] ... \end{LatexName}.
```

depending upon the L^AT_EX type.

LeftDelim [**string**] A string that is put at the beginning of the style content. A line break in the output can be indicated by `
`.

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

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.)

NeedCProtect $[-1, 0, 1]$ Value 1 causes macros that contain this layout to be protected with `\cprotect` (cf. package `cprotect`) if necessary and thus allows (some) verbatim stuff in macros. With the default setting (value 0), `\cprotect` is used if a nested element requires it. Value -1 prevents any use of `\cprotect` within this layout, even if required by a nested element.

NeedMBoxProtect $[0, 1]$ Whether specific commands in this style (such as `\cite` and `\ref`) should be protected in an `\mbox`. This is particularly needed for styles that draw on `ulem` or `soul` commands which parse their content in complex ways.

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 $[0, 1]$ 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 $[\langle \text{name} \rangle]$ Name of a style that has replaced this style. This is used to rename a style, while keeping backward compatibility.

ParagraphGroup $[0, 1]$ Determines whether consecutive paragraphs of the same type are treated as belonging together. This has the effect that the `GuiLabel` is only printed once before such a group. By default, this is true for `LaTeXTypeEnvironment` and `Bib_Environment` and false for all other types.

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 $[\text{string}=""]$ 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.

5. Installing New Document Classes

- ParSep** [`float=0`] The vertical space between two paragraphs of this style.
- Parskip** [`float=0`] L^AT_EX allows the user to choose either “indent” or “skip” to separate paragraphs. When “indent” is chosen, **Parskip** is ignored. When “skip” is chosen, **ParIndent** is ignored and all paragraphs are separated by the **Parskip** argument. The vertical space is calculated with **Parskip** * **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 L^AT_EX would require.
- PassThruChars** [`string`] Defines individual characters that should be output in raw form, meaning without special translations that L^AT_EX would require.
- Preamble** Information to be included in the L^AT_EX 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.
- Requires** [`string`] Whether the style requires the feature **string** (see Appendix A for the list of features). If you require a package with specific options, you can additionally use **PackageOptions** as a general text class parameter (see sec. 5.3.5).
- ResetArgs** [`0,1`] Resets the L^AT_EX arguments of this style (as defined via the **Argument** tag). This is useful if you have copied a style via **CopyStyle**, but you do not want to inherit its (required and optional) arguments.
- ResumeCounter** [`0,1`] Resumes a counter that is usually reset at each new sequence of layouts. This is currently only useful when **LabelType** is **Enumerate**.
- RightDelim** [`string`] A string that is put at the end of the layout content. A line break in the output can be indicated by `
`.
- 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 must also provide a value argument which will be the actual multiplier value. Note that, contrary to other parameters, **Spacing** implies the generation of specific L^AT_EX code, using the L^AT_EX package **setspace**.
- Spellcheck** [`0,1`] Allow spell-checking paragraphs of this style. Default is true.

StepParentCounter [*0,1*] Steps the parent counter of a given counter at the beginning of a new sequence of layouts. This is currently only useful when **LabelType** is **Enumerate**.

TextFont The font used for the text body . See sec. 5.3.13.

TocLevel [*int=3*] The level of the style in the table of contents. This is used for automatic numbering of section headings.

ToggleIndent [*default, always, never*] This tag determines whether the first line indentation of this paragraph can be toggled via the Paragraph settings dialog. If *default* is set, indentation can be toggled if the document settings use “indent” paragraph style, with *always*, indentation can always be toggled, notwithstanding the document settings, with *never*, indentation can never be toggled.

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.

5.3.8. 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 L^AT_EX may appear in the **LangPreamble** and **BabelPreamble** tags, but in practice they will typically look as they do here. The key to correct

5. Installing New Document Classes

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\captionsgerman{\renewcommand{\claimname}{Behauptung}}
\providecommand{\claimname}{Behauptung}
```

\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 file `layouttranslations`. 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 unless the `layouttranslations` file is modified accordingly. That said, however, any layout created with the intention that it will be included with LyX should use these tags where appropriate. Please note that the paragraph style translations provided by LyX will never change with a minor update (e. g. from version 2.1.x to 2.1.y). It is however quite likely that a major update (e. g. from 2.0.x to 2.1.0) will introduce new translations or corrections.

5.3.9. Floats

It is 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:

AllowedPlacement [string=lhtbpH] Allowed placement options for this float type. The value is a string of placement characters. Possible characters include: *h* (“here if possible”), *t* (“top of page”), *b* (“bottom of page”), *p* (“page of floats”), *H* (“here definitely”) and *!* (“ignore LaTeX rules”). The order of the characters in the string does not matter. If no placement options are allowed, use the string *none*.

AllowsSideways [0, 1] Defines whether the float allows to be rotated via the L^AT_EX package `rotfloat` (`sidewaysfloat`). Set to 0 if the float does not support this feature.

AllowsWide [0, 1] Defines whether the float has a starred variant that spans columns in a two column paragraph. Set to 0 if the float does not support this feature.

Extension [string=“”] The file name extension of an auxiliary file for the list of figures (or whatever). L^AT_EX 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 tags control the XHTML output. See sec. 5.4.

IsPredefined [0, 1] Indicates whether the float is already defined in the document class or if instead the L^AT_EX package `float` needs to be loaded to define it on-the-fly. The default is 0, which means: `float` is used. It should be set to 1 if the float is already defined by the L^AT_EX document class.

ListCommand [string=“”] The command used to generate a list of floats of this type; the leading ‘\’ must 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 L^AX, it is used by L^AT_EX for the title and it is used as the title in the 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 `NumberWithin` is set 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 L^AT_EX: `t`, `b`, `p` and `h` for top, bottom, page, and here, respectively.¹⁷ On top of that there is a new type, `H`, which does not

¹⁷Note that the order of these letters in the string is irrelevant, like in L^AT_EX.

5. Installing New Document Classes

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`”.

PrettyFormat [`string=`”] A format for use with formatted references to this counter. For example, one might want to have references to tables appear as “Table 2”. The string may contain “`##`” or a counter specification. (See the documentation for `LabelString` in sec. 5.3.12.) The former will be replaced by the counter number itself. So, for sections, it would be: Section `##`, or perhaps: section `\arabic{section}` (which might render as: section 2.7).

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.

Requires [`string`] As with paragraph styles, see sec. 5.3.7.

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`] Specifies whether this float is defined using the \LaTeX package `float`, either by the class file, another package or on-the-fly by LyX itself.

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

5.3.10. Flex insets and InsetLayout

Flex insets come in two 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 $\text{T}_{\text{E}}\text{X}$ `code`, `footnote`, and the like. An obvious example is an endnote inset, which is defined in the `endnote` module.

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 captions, 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 four 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`, `Caption:Standard`, `ERT`, `Figure`, `Foot`, `Index`, `Info`, `Info:menu`, `Info:shortcut`, `Info:shortcuts`, `Listings`, `Marginal`, `Note:Comment`, `Note>Note`, `Note:Greyedout`, `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 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.
3. The layout for user specific branch is being defined. In this case, `<Type>` must be of the form “`Branch:<name>`”, where `name` may be any valid identifier of branch defined in user’s document. The identifier may include spaces, but in that case the whole thing must be wrapped in quotes. The main purpose of this feature is to allow L^AT_EX wrapping around specific branches as user needs.
4. The layout of a user (or class) specific caption is being defined. In this case, `<Type>` must be of the form “`Caption:<name>`”, where `name` specifies the name of the caption as it appears in the menu. Have a look at the standard caption (`Caption:Standard`), the specific captions of the KOMA-Script classes (`Caption:Above`, `Caption:Below`) or the module `Multilingual Captions` (`Caption:Bicaption`) for applications.

The `InsetLayout` definition can contain the following entries:

AddToToc [`string=""`] This inset will appear in the table of contents of the given type. An empty string disables. See also the `OutlinerName` and the `IsTocCaption` commands. This is only implemented for Flex insets. Default: disabled.

AllowedInInsets Includes a comma-separated list of insets to which this inset can be inserted. Must be ended by “`EndAllowedInInsets`”. If you you also want the insertion to be allowed in specific arguments of the target insets, append the argument name after `@` (e.g., `My_Inset@post:1`). Note that this currently only supports immediately containing insets (no deeper nesting). See also `AllowedInLayouts`.

5. Installing New Document Classes

AllowedInLayouts Includes a comma-separated list of layouts within which this inset can be inserted. Must be ended by “`EndAllowedInLayouts`”. Note that this currently only supports immediately containing layouts (no deeper nesting). See also `AllowedInInsets`.

AllowedOccurrences [`int`] If `AllowedInInsets` or `AllowedInLayouts` has been defined, this can be used to determine how many times the inset can be inserted to a given inset or the paragraph (group).

AllowedOccurrencesPerItem [`0, 1`] If this is set to true, `AllowedOccurrences` applies to single paragraphs if we are in a list-type environment (with `\items`).

Argument [`int`] Defines argument number of a command/environment associated with the current layout. The definition must end with `EndArgument`. See sec. 5.3.11 for more information.

BabelPreamble Preamble for changing language commands; see sec. 5.3.8.

BgColor [`<name>`] The color for the inset’s background. See Appendix B for a list of the available color names.

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

CopyStyle [`<type>`] As with paragraph styles, see sec. 5.3.7. Note that you need to specify the complete type, e. g. `CopyStyle Flex:<name>`.

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 the 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.

EditExternal [`0, 1`] Allow the contents of the inset to be edited externally (using whatever editor is defined for the document’s output format).

End Required at the end of the `InsetLayout` declarations.

Font The font used for both the text body *and* the label. See sec. 5.3.13. 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.

- FixedWidthPreambleEncoding** $[0, 1]$ Force a fixed width encoding for the translated contents of `BabelPreamble` and `LangPreamble` code generated by this layout. This is needed for special \LaTeX -packages like `listings` that do not work with variable width encodings such as `utf8`. This setting is ignored if fully Unicode aware \LaTeX backends such as \XeTeX or \LuaTeX are used.
- ForceLocalFontSwitch** $[0, 1]$ When using `babel`, always use a local font switch (`\foreignlanguage`), never a global one (such as `\selectlanguage`).
- ForceLTR** $[0, 1]$ Force the “`latex`” language, leading to Left-to-Right (Latin) output, e.g. in \TeX code or URL. A kludge.
- ForceOwnlines** $[0, 1]$ Force a line break in the \LaTeX output before the inset starts and after the inset ends. This assures the inset itself is output on its own lines, for parsing purposes.
- 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** $[0, 1]$ As with paragraph styles, see sec. 5.3.7.
- HTML*** These tags control the XHTML output. See sec. 5.4.
- InheritFont** $[0, 1]$ The font inside the inset is inherited from the parent for \LaTeX export if this parameter is 1, as well as on screen. Otherwise the document default font is used.
- InToc** $[0, 1]$ Whether to include the contents of this inset in the strings generated for the ‘Outline’ pane for all table of contents, regardless of the `AddToToc` setting. 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.
- IsTocCaption** $[0, 1]$ If this is set to 1 and `AddToToc` is enabled, the inset adds a summary of its contents in its item in the table of contents. Otherwise, only the label appears.
- KeepEmpty** $[0, 1]$ As with paragraph styles, see sec. 5.3.7.
- LabelFont** The font used for the label. See sec. 5.3.13. Note that this definition can never appear before `Font`, lest it be ineffective.
- LabelString** $[\text{string=}“”]$ 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.
- LangPreamble** Language dependent preamble; see sec. 5.3.8.

5. Installing New Document Classes

LatexName [`<name>`] The name of the corresponding L^AT_EX stuff. Either the environment or command name.

LatexParam [`<parameter>`] The optional parameter for the corresponding **LatexName** stuff, including possible bracket pairs like `[]`. This parameter cannot be changed from within LyX (use **Argument** for customizable parameters). It will be output as is after all L^AT_EX **Arguments**.

LatexType [`Command`, `Environment`, `None`] How the style should be translated into L^AT_EX.¹⁸

None means nothing special

Command means `\LatexName{...}`

Environment means `\begin{LatexName}... \end{LatexName}`

Putting the last few things together, the L^AT_EX output will be either:

```
\LatexName[LatexParam]{...}
```

or:

```
\begin{LatexName}[LatexParam] ... \end{LatexName}
```

depending upon the L^AT_EX type.

LeftDelim [`string`] A string that is put at the beginning of the layout content. A line break in the output can be indicated by `
`.

LyxType Can be `charstyle`, `custom`, 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 automatically set **MultiPar** to false and **ForcePlain** to true. **MultiPar** can be set to true, or **ForcePlain** to false, for `charstyle` insets by setting it *after* you set the **LyxType**.

MenuString [`string`] A dedicated string for the menu. You can define an accelerator by appending the respective character to the string, divided by “|” (e.g. “My Inset|M”). This specification is optional. If it is not given the inset name as specified in the type declaration will be used instead for the menu.

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.

¹⁸**LatexType** is perhaps a bit misleading, since these rules apply to SGML classes too. Visit the SGML class files for specific examples.

- 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.
- NeedCProtect** [*0, 1*] This causes macros that contain this inset to be protected with `\cprotect` (cf. package `cprotect`) if necessary and thus allows (some) verbatim stuff in macros. Default is false.
- NeedMBoxProtect** [*0, 1*] Whether specific commands in this inset (such as `\cite` and `\ref`) should be protected in an `\mbox`. This is particularly needed for insets that draw on `ulem` or `soul` commands which parse their content in complex ways. Default is false.
- NewlineCmd** [*string*] Option to define a different command (from the default `\`) to be used for line breaks. The initial backslash must not be specified.
- NoInsetLayout** [*<layout>*] Deletes an existing `InsetLayout`.
- ObsoletedBy** [*<layout>*] Name of an `InsetLayout` that has replaced this `InsetLayout`. This is used to rename an `InsetLayout`, while keeping backward compatibility.
- ParbreakIgnored** [*0, 1*] If this is set to 1, paragraph breaks will be ignored in the output. This might be useful for insets where the content should be alignable on different lines only in the L^AT_EX workarea, without any effect in the output.
- ParbreakIsNewline** [*0, 1*] As with paragraph styles, see sec. 5.3.7. Default is false.
- PassThru** [*0, 1*] As with paragraph styles, see sec. 5.3.7. Default is false.
- Preamble** As with paragraph styles, see sec. 5.3.7.
- 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 sec. 5.3.7.
- ResetArgs** [*0, 1*] Resets the L^AT_EX arguments of this layout (as defined via the `Argument` tag). This is useful if you have copied a style via `CopyStyle`, but you do not want to inherit its (required and optional) arguments.
- ResetsFont** [*0, 1*] If 1, font changes are redone inside the respective inset (in the output) even if the inset itself is in the context of this font changes (e.g., `\textbf{Sourrounding text \myinset{\textbf{content}}}\dots`) rather than `\textbf{Sourrounding text \myinset{content}}\dots`. Setting this makes sense for commands that internally reset font settings (e.g., footnotes). Note that wrongly setting this might lead to unwanted result (e.g., with `\emph{Sourrounding text \myinset{\emph{content}}}\dots`), content is upright, as `\emph` toggles. Default is 0: font changes are not redone inside the inset.

5. Installing New Document Classes

RightDelim [`string`] A string that is put at the end of the layout content. A line break in the output can be indicated by `
`.

Spellcheck [`0, 1`] Allow spell-checking the contents of this inset. Default is true.

5.3.11. Arguments

Both paragraph styles and inset layouts allow for *arguments* as well as the main content. This is especially useful for things like section headings and only makes sense with \LaTeX . Each (optional or required) argument of a command or environment—except for the required argument that is associated with the content—has a separate definition, where the number specifies the order of the arguments. The definition must end with `EndArgument`. So a command with two optional arguments looks like:

```
Argument 1
...
EndArgument
Argument 2
...
EndArgument
```

Inside the `Argument` definition, the following specifications are possible:

- **LabelString** [`string`] The string that will appear both in the menu (to insert this argument) and on the argument inset button (unless you also specify a separate `MenuString`). For the menu, you can define an accelerator by appending the respective character to the string, divided by “|” (e.g. “`Short Title|S`”).
- **MenuString** [`string`] A separate string for the menu. You can define an accelerator by appending the respective character to the string, divided by “|” (e.g. “`Short Title|S`”). This specification is optional. If it is not given the `LabelString` will be used instead for the menu.
- **Tooltip** [`string`] A longer explanatory text that appears in the tooltip when hovering over the argument inset.
- **Mandatory** [`0, 1`] Declare if this is a mandatory (1) or an optional (0) argument. Mandatory arguments will be output empty if not given, while optional arguments will not be output at all. By default, mandatory arguments are delimited by `{...}`, while optional arguments are delimited by `[...]`
- **NewlineCmd** [`string`] Option to define a different command (from the default `\`) to be used for line breaks. The initial backslash must not be specified.

- **Requires** [*int=0*] defines another argument or arguments (by their number) which this argument requires to be output if it is itself output. E. g., in L^AT_EX commands, optional arguments often require previous optional arguments to be output (at least empty), as in `\command[] [argument]{text}`. This can be achieved by the statement **Requires 1** within **Argument 2**. If multiple arguments are required, separate them by comma (e.g., **Requires 1,2**).
- **LeftDelim** [*string*] defines a custom left delimiter (instead of { or [). A line break in the output can be indicated by `
`.
- **RightDelim** [*string*] defines a custom right delimiter (instead of } or]). A line break in the output can be indicated by `
`.
- **DefaultArg** [*string*] defines an argument that is inserted if and only if no user-specified arguments were given, i. e. if no argument inset has been inserted (note that also an empty argument inset omits the **DefaultArg**). Multiple arguments need to be separated by comma.
- **PresetArg** [*string*] defines an argument that is inserted in any case (alone or in addition to user-specified arguments). Multiple arguments need to be separated by comma.
- **Font** The font used for the argument content, see sec. 5.3.13.
- **FreeSpacing** [*0, 1*] As with paragraph styles, see sec. 5.3.7.
- **LabelFont** The font used for the label; see sec. 5.3.13.
- **Decoration** [*Classic, Minimalistic, Conglomerate*] describes the rendering style used for the inset's frame and buttons.
- **AutoInsert** [*int=0*] If this is set to 1, this argument is automatically inserted when the respective style is selected.
- **InsertOnNewline** [*int=0*] If this is set to 1, this argument will be inserted on a new line with **AutoInsert** (only available within Flex insets).
- **InsertCotext** [*int=0*] If this is set to 1, this argument will be inserted with a copy of the co-text (either selected text or the whole paragraph) as content.
- **PassThru** [*inherited, true, false*] Whether the contents of this argument should be output in raw form, meaning without special translations that L^AT_EX would require. By default, the **PassThru** status is inherited by the inset or paragraph layout the argument belongs to, true and false change the status for the given argument only.

5. Installing New Document Classes

- **PassThruChars** [`string of characters`] Defines individual characters that should be output in raw form, meaning without special translations that \LaTeX would require. Note that, contrary to `PassThru`, this needs to be explicitly defined for arguments. That is, arguments do not inherit `PassThruChars` from their parent inset or layout.
- **IsTocCaption** [`0, 1`] If this is set to 1, the argument will output its content in the corresponding item in the table of contents. See `AddToToc`.

By default, the text entered in the \LaTeX workarea in the respective layout is the last (mandatory) argument of a command if the `LatexType` is `Command`. However, arguments with the prefix `post:` are output after this workarea argument. Note that post-argument numbering restarts at 1, so the first argument following the workarea argument is `post:1`. Post-arguments are ignored in any other `LatexType` than `Command`.

Arguments for list `\items` (as in `\item[foo]`) have the prefix `item:` followed by the number (e.g. `Argument item:1`).

Finally, there is a special argument type with the prefix `listpreamble:`. It is not really an argument, but uses the argument interface (thus, the prefix is also followed by a number, e.g. `Argument listpreamble:1`). As the name implies, it is targeted at lists such as `Itemize`, `Enumerate`, `Description`, or `Bibliography`. Its content will be output at the list start, before the first `\item`, on an own line (a place that is otherwise not accessible in \LaTeX). This way, users can insert redefinitions (of lengths etc.) to individual lists. By default, these arguments do not have a delimiter.

5.3.12. Counters

It is 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:

InitialValue [`int=1`] Sets the initial value for the counter, to which it will be reset whenever that happens. Normally, one will want the default, 1.

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 L^AT_EX-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 parent counter `parent` (defined via `Within`), the string `\theparent.\arabic{counter}` is used; otherwise the string `\arabic{counter}` is used.

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

LaTeXName [`string =` “”] The counter name as used in L^AT_EX. (e.g., in LyX, there is a counter named ‘theorem’, but it is output to L^AT_EX as ‘thm’.)

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 may contain “##” or a counter specification as in `LabelString`. The former will be replaced by the counter number itself. So, for sections, it would be: Section ##, or perhaps: `\S\arabic{section}` (which might render as §2.7).

RefFormat [`string, string`] For use with ‘formatted references’, specifically when a single counter is used with multiple sorts of styles. For example, the `theorem` counter is often used for all sorts of theorem-like environments: Theorem, Lemma, etc. The first argument gives a prefix used in the labels (e.g., “thm” or “lem”), and the second a formatting string, as for `LabelString` or `PrettyFormat`. If this is not given, then `PrettyFormat` is used.

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`.

5.3.13. Font description

A font description looks like this:

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

¹⁹Actually, 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.

5. Installing New Document Classes

The following commands are available:

Color [string] See appendix Appendix B for valid arguments.

Family [*Roman*, Sans, Typewriter]

Misc [string] Valid arguments are: `emph`, `noun`, `strikeout`, `underbar`, `uuline`, `uwave`, `no_emph`, `no_noun`, `no_strikeout`, `no_bar`, `no_uuline` and `no_uwave`. 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]

5.3.14. Cite engine description

The `CiteEngine` blocks, as used mainly in cite engine files (see sec. 5.2.6), define the citation commands provided by a specific “cite engine”. A cite engine, in LYX terms, is way specific way to format citations, using numbers, author names and/or years. Currently, LYX supports three such engine types, namely:

1. **default**: the default $\text{BibT}\text{E}\text{X}$ way to format citations, a simple numeric style (e. g., “[1]”)
2. **authoryear**: Harvard-styled citations using author names and publication year (e. g., “Smith and Miller (2017b)”)
3. **numerical**: extended numerical citations that also allow for author or title next to the number (e. g., “Smith and Miller [1]”)

`CiteEngine` blocks look like this:

```
CiteEngine default
  cite
  Citep* [] []
  citeyearpar [] []=parencite*
  ...
End
```

The tag following `CiteEngine` denotes the engine. The individual lines respectively define a cite command or cite command paradigm supported by this engine. The line can be as simple as a cite command that is used both to name the respective `LYX` command and the `LATEX` output or more complex in order to differentiate things. The full syntax is:

```
LyXName|alias$*<!_stardesc!_stardescstooltip>[] []=latexcmd
```

- `LyXName`: The name as used in the `*.lyx` file.
For portability reasons, we try to use the same name for same-formatted commands in different cite packages (thus many names stem from `natbib`, and thus we need to differentiate a `latexcmd` sometimes, if the `LATEX` command names differ).
- `alias`: a (comma-separated) list of commands that fall back to the given `LyXName` in the current engine. This eases the switch of citation packages and engines. The `alias` can be compared to `ObsoletedBy` in layout definitions.
- `latexcmd`: The actual `LATEX` command that is output.

`Alias` and `latexcmd` are optional. If no `latexcmd` is given, the `LyXName` will be output to `LATEX`.

Note further:

- Capitalization indicates that the command also has a capitalized form (`\Latexcmd` vs. `\latexcmd`). These usually enforce up-casing of name prefixes (*von Goethe* ⇒ *Von Goethe*).
- Brackets `[]` indicate the number of optional arguments (there can be 0–2).
- A star `*` indicates there is a starred version of the command (`\latexcmd*` vs. `\latexcmd`).

By default, the starred version means: Output all authors even if it should be shortened with “et al.” due to the `MaxCiteNames` threshold.

If the star has a different meaning for a given command, it can be specified in angle brackets: `<!_stardesc!_stardescstooltip>`. Maximal two translatable macro keywords, marked by the prefix `!_`, can be given. The first points to the string that replaces the “Full aut&hor list” checkbox label in the citation dialog, the second one to an optional tool tip for this checkbox.

Note that these two macros have to be defined in a `CiteFormat` (see next section), dropping the `!` from the prefix, like this:

```
_stardesc Sta&rred command label
_stardescstooltip Tooltip for the starred command checkbox.
```

5. Installing New Document Classes

- A dollar sign \$ indicates that this command features “qualified citation lists”. This is a Bib_lat_ex-specific feature for multi-reference citations where an individual pre- and postnote can be given to each reference in the list. Please refer to the Bib_lat_ex manual for details.

If you want to add a cite command to a cite engine (e. g., add a specific command provided by a class), you can use `AddToCiteEngine <engine type> ... End`. Note that only cite commands that do not exist yet are added.

5.3.15. Cite format description

The `CiteFormat` blocks are used to describe how bibliographic information should be displayed, both within L_YX 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
```

or

```
CiteFormat
  cite ...
  citet*[] [] ...
End
```

In the first case, 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 Bib_TE_X file. L_YX defines a default format in the source code that will be used if no specific definition has been given. L_YX predefines several formats in the file `stdciteformats.inc`, which is included in most of L_YX’s document classes.

In the second case, the lines define how a specific citation command (in the example `\cite`, `\citet`) is to be displayed on the citation inset label, in the citation dialog, menu or XHTML output. L_YX defines such formats for the citation style variants it supports via `Document > Setting > Bibliography...` in specific `*.citeengine` files that are shipped with L_YX (see sec. 5.2.6).

The definitions use a simple language that allows Bib_TE_X 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.

Next to the entry keys, there are some special keys that can be used for these conditionals:

- `{%dialog%[[true]] [[false]]}`: process the “true” part for dialogs and menus, the “false” part for other contexts (workarea, export)
- `{%export%[[true]] [[false]]}`: process the “true” part for export and menus, the “false” part for other contexts (workarea, dialog)
- `{%next%[[true]]}`: process the “true” part if another item follows (e. g., in a citation with multiple keys)
- `{%second%[[true]] [[false]]}`: process the “true” if this is the second of multiple items, else the “false” part
- `{%ifstar%[[true]] [[false]]}`: process the “true” part for starred citation commands (such as `\cite*`), the false part for unstarred
- `{%ifentrytype:<type>%[[true]] [[false]]}`: process the “true” if the current entry type matches `<type>`, else the false part (e.g., in a citation definition: `{%ifentrytype:book%[[this is a book]] [[this is no book]]}`)
- `{%ifmultiple:<authortype>%[[true]] [[false]]}`: process the “true” if the current author type (author, editor etc.) has multiple authors, else the false part (e.g., in a bibliography definition: `{%ifmultiple:editor%[[eds.]] [[ed.]]}`)
- `{%ifqualified%[[true]] [[false]]}`: process the “true” part if the current citation is a qualified citation list (a specific Bib_lat_ex format for multi-reference citations), the false part if this is not the case.

We said that `%author%` prints the author key as it is recorded in the bibliography file. This might not be what you want, since it will result in a string such as “Miller, Peter and Smith, Mary and White, Jane” (since “and” is used by Bib_TE_X to delimit authors). Ly_X therefore provides some methods to get properly formatted name lists (which will also get translated). The following keys are provided:

5. Installing New Document Classes

1. For name lists with pre- and surname, suitable for the main authors/editors of a bibliography item. The `<nametype>` part denotes the kind of list that is requested (e.g. `<nametype:author>`):
 - `%abbrvnames:<nametype>%`: Provides a name list which is abbreviated (with “et al.”) when `MaxCiteNames` is reached.
 - `%fullnames:<nametype>%`: Provides a full name list (never abbreviated with “et al.”).
 - `%forceabbrvnames:<nametype>%`: Provides a name list which is always abbreviated (with “et al.”) irrespective of `MaxCiteNames`.
2. Alternative name lists with pre- and surname, if the order of pre- and surname inside the bibliography item differs (as in: “Miller, John: Some text, in: Mary Smith, ed.: A volume”):
 - `%abbrvbynames:<nametype>%`: Provides a name list which is abbreviated (with “et al.”) when `MaxCiteNames` is reached.
 - `%fullbynames:<nametype>%`: Provides a full name list (never abbreviated with “et al.”).
 - `%forceabbrvbynames:<nametype>%`: Provides a name list which is always abbreviated (with “et al.”) irrespective of `MaxCiteNames`.
3. And finally name lists which consist of family names only, as used in author-year citation labels. These do not take a `<nametype>` part, but always return either an author list or, if this does not exist, an editor list (as common in author-year labels):
 - `%abbrvciteauthor%`: Provides a name list which is abbreviated (with “et al.”) when `MaxCiteNames` is reached.
 - `%fullciteauthor%`: Provides a full name list (never abbreviated with “et al.”).
 - `%forceabbrvciteauthor%`: Provides a name list which is always abbreviated (with “et al.”) irrespective of `MaxCiteNames`.

The order of pre- and surname in the former two lists can be adjusted by these macros:

- `!firstnameform %surname%, %prename%` (first author in lists of type 1)
- `!othernameform %surname%, %prename%` (other authors in lists of type 1)
- `!firstbynameform %prename% %surname%` (first author in lists of type 2)
- `!otherbynameform %prename% %surname%` (other authors in lists of type 2)

This allows you to configure namings like “Miller, Peter and Mary Smith: . . . , in: John Doe and Pat Green, eds.:. . . ”.

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:

```
B_pptext pp.
```

This defines a translatable piece of text, which allows relevant parts of the bibliography or citation to be translated. It can be included in a definition by treating it as a key: `%B_pptext%`. Note that there are two different translation paths: All definitions starting with `B_`, such as in the example above, will be translated to the currently active buffer language (so the translation will match the generated document). All definitions starting with underscore only will be translated to the GUI language. This is the proper translation for strings that only occur in the dialogs or on buttons, such as this one:

```
_addtobib Add to bibliography only.
```

Several of these translatable strings are predefined in `stdciteformats.inc` and the diverse `*.citeengine` files. Note that these are not macros, in the sense just defined. They will not be expanded.

So here then is an example that uses several of these features:

```
!authredit {%author%[[%author%, ]][[ {%editor%[[%editor%, %B_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 `B_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 sec. 5.3.5 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 TeXType 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 La-
bel</labeltag>First paragraph.</itemtag>
<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 La-
bel</labeltag>First item.</itemtag>
<itemtag attr="value"><labeltag attr="value">List La-
bel</labeltag>Second item.</itemtag>
</tag>
```

```

<tag attr="value">
<labeltag attr="value">List La-
bel</labeltag><itemtag attr="value">First item.</itemtag>
<labeltag attr="value">List La-
bel</labeltag><itemtag attr="value">Second item.</itemtag>
</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.

HTMLClass [`string`] The CSS class to use for this paragraph. Note that, if the paragraph is of `enumerate` or `itemize` type, then the default will be “`lyxenum`” or “`lyxitem`”, plus “`i`”, “`ii`”, “`iii`”, or “`iv`”, depending upon the depth. That can be over-ridden here. However, the suffix will not be added in that case. I.e., the CSS class will always be exactly what it is declared to be here.

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`.

HTMLInToc [`0,1`] Whether to include this paragraph (usually, a section or something of the sort) in the TOC. By default, true, so it should be set to false e.g. for starred sections.

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.

5. Installing New Document Classes

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. Must end with `EndHTMLStyle`.

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, L^AT_EX tries to provide sensible defaults, and it constructs default CSS style rules. But everything can be customized.

The XHTML L^AT_EX 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

²⁰At 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).

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`.

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:

5. Installing New Document Classes

```
<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 sec. 5.3.9), 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 sec. 5.3.15 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 sec. 5.3.13). 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.

5.5. Tags for DocBook output

As with \LaTeX or XHTML, the format of LyX’s DocBook output is also controlled by layout information. In general, LyX provides sensible defaults; however, much of the styling is lost during the conversion, as DocBook is strictly semantic and does

not allow formatting. When possible, information from LyX will be rendered in `role` attributes.

In many cases, then, you may not have to do anything at all to get acceptable DocBook 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 DocBook that is generated.

Labels are rarely output, as they are redundant in DocBook: this information is carried by the tags themselves, and whether labels appear in the final documents (after processing of DocBook files) is controlled by the stylesheets. However, sometimes, labels are not redundant content, such as definition lists: in this case, the term being defined will be the label.

5.5.1. Paragraph styles

The sort of DocBook 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 `TEXTType` tag.

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

```
<tag attr>
Contents of the paragraph.
</tag>
```

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

```
<tag attr>
<itemtag>First paragraph.</itemtag>
<itemtag>Second paragraph.</itemtag>
</tag>
```

For a list, the resulting DocBook takes this form:

```
<tag attr>
<itemtag attr>First item.</itemtag>
<itemtag attr>Second item.</itemtag>
</tag>
```

The specific tags and roles output for each paragraph type can be controlled by means of the layout tags we are about to describe. Please note that, due to the very nature of DocBook, no sensible defaults really exist, and the values must always be carefully chosen.

DocBookAttr [`string`] Specifies attribute information to be output with the main tag, replacing “`attr`” in the example above. This information can be used in further processing of the DocBook files.

5. Installing New Document Classes

DocBookTag [`string`] The tag to be used for this inset, replacing “`tag`” in the example above. The default is the name of the float and always needs to be changed, as DocBook provides no generic tag.

DocBookTagType [`block`, `paragraph`, `inline`] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

5.5.2. New-line policy

For all tags, there are three possible policies for outputting new lines (given in the `DocBook*TagType` attribute):

- “`block`”: the opening and closing tags are on their own lines (i.e. a line feed after and before the opening and the closing tags). Typical elements are floats. For instance:

```
Content before
<blocktag>
  Contents of the block.
</blocktag>
Content after
```

- “`paragraph`”: the opening and closing tags are on the same, new line; a line feed is output before the opening tag and after the closing tag. Typical elements are paragraphs and list items. For instance:

```
Content before
<paratag>Contents of the paragraph.</paratag>
Content after
```

- “`inline`”: the opening and closing tags are on the same line as the rest of the content. No line feeds are output. Typical elements are fonts. For instance:

```
Content before<inlinetag>Contents of the paragraph.</inlinetag>Content after
```

The default value is always “`block`”.

5.5.3. InsetLayout DocBook

The DocBook output of insets can also be controlled by information in layout files.

The DocBook L^AT_EX outputs for an inset has the following form:


```

<wrappertag wrapperattr>
  <tag attr>
    <innertag innerattr>
      Contents of the inset.
    </innertag>
  </tag>
</wrappertag>

```

For an itemising inset, it rather looks like this:

```

<wrappertag wrapperattr>
  <tag attr>
    <innertag innerattr>
      <itemwrappertag itemwrapperattr>
        <itemlabeltag itemattr>
          Label of the first item.
        </itemtag>
        <itemtag itemattr>
          <itemtag itemattr>
            Contents of the first item.
          </itemtag>
        </itemtag>
      </itemwrappertag>
      <itemwrappertag itemwrapperattr>
        <itemlabeltag itemattr>
          Label of the second item.
        </itemtag>
        <itemtag itemattr>
          <itemtag itemattr>
            Contents of the second item.
          </itemtag>
        </itemtag>
      </itemwrappertag>
      ...
    </innertag>
  </tag>
</wrappertag>

```

If the inset permits multiple paragraphs—that is, if `MultiPar` is true—then the contents of the inset will themselves be output as paragraphs formatted according to the styles used for those paragraphs (standard, quote, and the like). 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.

5. Installing New Document Classes

DocBookAttr [string] Specifies attribute information to be output with the main tag, replacing “attr” in the example above. This information can be used in further processing of the DocBook files.

DocBookInInfo [never, always, maybe] Specifies whether this tag goes into the <info> tag at the beginning of the parent layout. **never** indicates that the tag never goes into <info> (this is default value, and corresponds to usual content). **always** indicates that the tag always goes into <info> (this corresponds to usual metadata): if there is no <info> tag for the parent, one **will** be generated. **maybe** indicates that the tag may go into <info> (this is only the case for titles): if there is no <info> tag for the parent, **none** will be generated, the corresponding tag will be output directly as content.

DocBookItemAttr [string] Specifies attribute information to be output with the item tag, replacing “itemattr” in the example above. This information can be used in further processing of the DocBook files.

DocBookItemInnerAttr [string] Specifies attribute information to be output with the item inner tag, replacing “iteminnerattr” in the example above. This information can be used in further processing of the DocBook files.

DocBookItemInnerTag [string] The tag to be used for the item inner tag within the inset, replacing “iteminnertag” in the example above. The default is **NONE**, indicating that there is no item inner tag: content is directly output without it for each itemised element. This parameter only makes sense when itemising layouts are used, such as lists. The most likely value is “para”.
When a list item is split using a new line, the item inner tag will be repeated for each part of the paragraph, parts being separated by new lines.

DocBookItemInnerTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookItemLabelAttr [string] Specifies attribute information to be output with the item label tag, replacing “itemlabelattr” in the example above. This information can be used in further processing of the DocBook files.

DocBookItemLabelTag [string] The tag to be used for the item label tag within the inset, replacing “itemlabeltag” in the example above. This parameter only makes sense when itemising layouts are used with a notion of labels, such as definition lists.

DocBookItemLabelTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookItemTag [string] The tag to be used for the item tag within the inset, replacing “itemtag” in the example above. The default is **NONE**, indicating

that there is no item tag. This parameter only makes sense when itemising layouts are used, such as lists.

DocBookItemTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookItemWrapperAttr [string] Specifies attribute information to be output with the item wrapper tag, replacing “`itemwrapperattr`” in the example above. This information can be used in further processing of the DocBook files.

DocBookItemWrapperTag [string] The tag to be used for the item wrapper tag within the inset, replacing “`itemwrappertag`” in the example above. The default is `NONE`, indicating that there is no item wrapper tag: tag and content are directly output without it for each itemised element. This parameter only makes sense when itemising layouts are used, such as lists.

DocBookItemWrapperTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookInnerAttr [string] Specifies attribute information to be output with the inner tag, replacing “`innerattr`” in the example above. This information can be used in further processing of the DocBook files.

DocBookInnerTag [string] The tag to be used for the inner tag within the inset, replacing “`innertag`” in the example above. The default is `NONE`, indicating that there is no inner tag: content is directly output without it.

DocBookInnerTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookSectionTag [string] Specifies the tag that corresponds to this kind of section. This parameter only makes sense for sectioning elements (part, chapter, section, etc.). The default value is `section`, and is only overridden when DocBook uses something else for sectioning (typically, parts and chapters of a book).

DocBookTag [string] The tag to be used for this inset, replacing “`tag`” in the example above. The default is the name of the float and always needs to be changed, as DocBook provides no generic inset tag.

DocBookTagType [block, paragraph, inline] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

DocBookWrapperAttr [string] Specifies attribute information to be output with the outer wrapper tag, replacing “`wrapperattr`” in the example above. This information can be used in further processing of the DocBook files.

5. Installing New Document Classes

DocBookWrapperTag [`string`] The tag to be used for the wrapper tag around the inset, replacing “`wrappertag`” in the example above. The default is `NONE`, indicating that there is no wrapper tag: tag and content are directly output without it.

DocBookWrapperTagType [`block`, `paragraph`, `inline`] The new-line policy for this tag, see Section sec. 5.5.2 for the details.

5.5.4. Float DocBook

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

```
<tag attr>
  Contents of the float as DocBook.
</tag>
```

The caption, if there is one, is a separate inset and will be output as a title.

DocBookAttr [`string`] Specifies attribute information to be output with the main tag, replacing “`attr`” in the example above. This information can be used in further processing of the DocBook files.

DocBookTag [`string`] The tag to be used for this float, replacing “`tag`” in the example above. The default is the name of the float and always needs to be changed, as DocBook provides no generic float tag.

5.5.5. Bibliography formatting

Included bibliographies cannot be formatted: all fields are always output in the database-like DocBook format (equivalent to a BibTeX file), using the `biblioentry` tag.

When the bibliographic entries are manually inserted into the LyX document as Bibliography Items, the user deals with formatting themselves: there is no attempt of parsing what the user wrote, the string is directly used (with the `bibliomixed` tag).

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 L^AT_EX 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

6. Including External Material

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 files

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 sec. 6.4.

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

The external templates are defined in the `*.xtemplate` files that are stored in the `LyXDir/lib/xtemplates/` directory. Each template is defined in a file of its own. You can place your own templates in `UserDir/xtemplates/` or copy existing templates to that directory in order to modify them.

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 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 "[XFig: $$FName]"
FormatEnd
Format DocBook
Product "<graphic fileref=\"$$AbsOrRelPathMaster$$Basename.eps\">
        </graphic>"
UpdateFormat eps
UpdateResult "$$AbsPath$$Basename.eps"
ReferencedFile docbook "$$AbsPath$$Basename.eps"
ReferencedFile docbook-xml "$$AbsPath$$Basename.eps"
FormatEnd
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.

6. Including External Material

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 sec. 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|XHTML 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 L^AT_EX 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 L^AT_EX package. The package is included via `\usepackage{}` in the L^AT_EX preamble. This command may occur zero or more times.

TransformCommand Rotate RotationLatexCommand This command specifies that the built in L^AT_EX 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 L^AT_EX 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. If the resulting file format is PDF, you need to specify the format `pdf6`. This is the PDF format used for including graphics. The other defined PDF formats are for document export.

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

6. Including External Material

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).

\$\$pngOrjpg This will be the string “jpg” if the file is in JPEG format, otherwise it will be the string “png”. This is useful to avoid unneeded conversions for output formats that support both PNG and JPEG formats. The predefined `RasterImage` template uses this macro for the pdfTeX output format.

\$\$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.

A. List of supported \LaTeX functions to be used in layouts

accents	booktabs	feyn	listings	natbib	rsphrase	tfruppee	wasysym
amsbsy	calc	fixltx2e	longtable	nomencl	setspace	tipa	wrapfig
amscd	CJK	float	lyxskak	pdfpages	shapepar	tipx	xargs
amsmath	color	framed	makeidx	pifont	slashed	tone	xcolor
amssymb	covington	graphicx	marvosym	pmboxdraw	soul	txfonts	xy
amstext	csquotes	hhline	mathdesign	polyglossia	splitidx	ulem	yhmath
amsthm	dvipost	hyperref	mathdots	prettyref	subfig	undertilde	
array	endnotes	ifsym	mathrsfs	pxfonts	subscript	units	
ascii	enumitem	ifthen	mhchem	refstyle	tcolorbox	url	
bbding	esint	jurabib	multicol	rotating	textcomp	varioref	
bm	fancybox	latexsym	multirow	rotfloat	textgreek	verbatim	

B. Names of available colors to be used in layouts

The colors listed below are the standard colors and those that you can adjust in the LyX preferences.

B.1. Color functions

The following are no real colors, but rather act on color definitions:

ignore The color is ignored

inherit The color is inherited

none No particular color – clear or default

B.2. Static colors

These are fixed colors that cannot be customized. Please do *not* use these colors in layout definitions, since they will not work well with some color themes (such as dark themes):

black

white

blue

brown

cyan

darkgray

gray

green

lightgray

lime

B. Names of available colors to be used in layouts

magenta

olive

orange

pink

purple

red

teal

violet

yellow

B.3. Dynamic colors

These are the colors allocated to specific elements in :

added_space Added space color

addedtext Added text color

appendix Appendix marker color

background Background color

bookmark Bookmark indicator color

bottomarea Bottom area color

branchlabel Label color for branches

buttonbg Color used for button background

buttonframe Color for inset button frames

buttonhoverbg Color used for button background under focus

buttonhoverbg_broken Color used for broken inset button under focus

changebar Changebar color

changedtextauthor1 Changed text color author 1

changedtextauthor2 Changed text color author 2

changedtextauthor3 Changed text color author 3

changedtextauthor4 Changed text color author 4

changedtextauthor5 Changed text color author 5

changedtextcomparison Changed text color document comparison (workarea)

collapsible Collapsible insets text color

collapsibleframe Collapsible insets framecolor

command Text color for command insets

commandbg Background color for command insets

commandframe Frame color for command insets

command_broken Text color for broken (reference) insets

commandbg_broken Background color for broken insets

commandframe_broken Frame color for broken insets

comment Label color for comments

commentbg Background color of comments

cursor Cursor color

deletedtext Deleted text color

deletedtextmodifier Deleted text modifying color (for adjusting the brightness)

depthbar Color for the depth bars in the margin

eolmarker End of line marker color

error Color of the L^AT_EX error box

footlabel Label color for footnotes

foreground Foreground color

graphicsbg Graphics inset background color

greyedoutbg Background color of greyedout inset

greyedoutlabel Label color for greyedout insets

greyedouttext Color for greyedout inset text

indexlabel Label color for index insets

B. Names of available colors to be used in layouts

inlinecompletion Inline completion color

insetbg Inset marker background color

insetframe Inset marker frame color

language Color for marking foreign language words

latex Text color in \LaTeX mode

listingsbg Background color of listings inset

marginlabel Label color for margin notes

math Math inset text color

mathbg Math inset background color

mathcorners Math inset frame color not under focus

mathframe Math inset frame color under focus

mathline Math line color

mathmacrobg Macro math inset background color

mathmacroblend Macro math blended color

mathmacroframe Macro math frame color

mathmacrohoverbg Macro math inset background color hovered

mathmacrolabel Macro math label color

mathmacronewarg Macro template color for new parameters

mathmacrooldarg Macro template color for old parameters

newpage New page color

nonunique_inlinecompletion Inline completion color for the non-unique part

note Label color for notes

notebg Background color of notes

pagebreak Page break/line break color

paragraphmarker Color used for the pilcrow sign to mark the end of a paragraph

phantomtext Text color for phantom insets

preview The color used for previews

previewframe Preview frame color

regexpframe Color for regexp frame

scroll Color that indicates when a row can be scrolled

selection Background color of selected text

selectionmath Foreground color of selected text in math insets

selectiontext Foreground color of selected text

shadedbg Background color of shaded box

special Special chars text color

tabularline Table line color

tabularonoffline Table line color

textlabel1 Color 1 of layout and custom inset labels

textlabel2 Color 2 of layout and custom inset labels

textlabel3 Color 3 of layout and custom inset labels

urllabel Label color for URL insets

urltext Color for URL inset text