LyX の高度設定
熟練ユーザのための各機能

LyX 開発チーム[1]

第 2.0.x 版

2011 年 5 月 15 日

[1]もしコメントや誤りの修正をお持ちでしたら、LyX 文書化メーリングリストlyx-docs@lists.lyx.org宛お送りください。件名ヘッダに「[Customization]」という文字を入れ、このファイルの現在のメンテナ Richard Heck <rgheck@comcast.net>を cc にして送ってください。

目次

第1章　はじめに

この取扱説明書は、LyX に備わっている高度設定機能を取り扱います。ここでは、短絡キーや画面プレビューオプション、プリンタオプション、LyX サーバ経由での LyX へのコマンド送信、国際化、新しい LaTeX クラスや LyX レイアウトの導入などの題材について論じます。おそらくは変更可能なことすべてについて触れることは無理でしょうが—私たちの開発者たちは私たちが文書化できる速さよりも速く新しい機能を付け加えてしまうので—、もっとも一般的な高度設定については説明を行い、わかりにくいものについては正しい方向を指し示すことができるようにしていくつもりです。

第2章　LyX設定ファイル

本章の目的は、LyX 設定ファイル群を理解するための一助となることです。本章を読み進める前に、ヘルプ▷LyX についてを使って、LyX ライブラリとユーザディレクトリがどこにあるかを確認しておいてください。ライブラリディレクトリは、LyX がシステム全体の設定ファイルを置いておくところです。一方、ユーザディレクトリは、自身がそれを修正した版を置いておくところです。私たちは、本書の以下の部分で、前者を LyXDir と呼び、後者を UserDir と呼ぶことにします。

2.1　LyXDir にはなにがあるの？

LyXDir とそのサブディレクトリには、多くのファイルがあり、LyX の挙動を高度設定するのに使用されます。これらのファイルの多くは、LyX 内のツール▷設定ダイアログから変更することができます。LyX 中で行いたいと思うような高度設定は、ほとんどこのダイアログから行うことができるようになっています。しかしながら、LyX の他の多くの内部動作は、LyXDir のファイルを修正することで高度設定されます。これらのファイルは様々なカテゴリに分類しうるので、以下の各小節で説明します。

2.1.1　自動的に生成されるファイル

UserDir にある各ファイルは、LyX が自動設定を行ったときに生成されます。これらのファイルは、内部調査中に推測された様々な既定値が置かれています。これらは、随時上書きされてしまうので、一般的には修正しないことが望まれます。

`lyxrc.defaults` このファイルには、様々な既定コマンドが置かれています。

`packages.lst` このファイルには、LyX が認識したパッケージの一覧が収められています。現在のところ、これは LyX プログラム自体には使用されていませんが、抽出された情報その他は、ヘルプ▷LaTeX の設定で見ることができます。

`textclass.lst` ユーザの `layout/` ディレクトリで検出されたテキストクラスと、関連した LaTeX 文書クラスおよびその説明の一覧です。

`lyxmodules.lst` ユーザの `layout/` ディレクトリで検出されたレイアウトモジュールの一覧です。

`*files.lst` ご使用のシステムで検出された様々な種類の LaTeX 関連ファイルの一覧です。

`doc/LaTeXConfig.lyx` このファイルは、自動設定中に `LaTeXConfig.lyx.in` から自動的に生成されます。ご使用中の LaTeX の設定に関する情報が納められています。

2.1.2　ディレクトリ

以下の各ディレクトリは、`LyXDir` と `UserDir` に重複して存在します。特定のファイルが両方の場所にある場合には、`UserDir` の方にあるものが使用されます。

`bind/` このディレクトリには、LyX で使用されるキー割当を定義している、拡張子が `.bind` のファイルが置かれています。`$LANG_xxx.bind` という名称の割当ファイルの各国語版がある場合には、そちらが用いられます。

`clipart/` このディレクトリには、文書に取り込むことのできる画像ファイルが納められています。

`doc/` このディレクトリには、LyX の取扱説明書ファイル（今お読みのものも含めて）が納められています。上述のように、`LaTeXConfig.lyx` ファイルは特に注目に値します。各国語版のヘルプ文書は、`doc/xx`（「xx」は ISO 言語コード）サブディレクトリにあります。詳しくは、第 4 章をご覧ください。

examples/ このディレクトリには、何らかの機能の使い方を説明する例示ファイルが納められています。ファイルブラウザ中で用例ボタンを押すと、このディレクトリが表示されます。

images/ このディレクトリには、文書ダイアログで使用される画像ファイルが納められています。さらに、ツールバーの各アイコンや、LyX を起動したときに現れるバナーも納められています。

kbd/ このディレクトリには、キーボードのキー割当ファイルが納められています。詳細については、第 4.2章をご覧ください。

layouts/ このディレクトリには、第 5章に述べられているテキストクラスおよびモジュールのファイルが納められています。

lyx2lyx このディレクトリには、LyX の各バージョン間の変換に使用される lyx2lyx Python スクリプトが納められています。たとえば、複数のファイルの変換をバッチ処理したい場合には、これらをコマンドラインから実行することもできます。

scripts/ このディレクトリには、外用ひな型機能の有用性を示すためのファイルがいくつか納められています。LyX 自身が使用するスクリプトもいくつか収められています。

templates/ このディレクトリには、第 5.2.4章で述べられている標準の LyX ひな型ファイルが納められています。

ui/ このディレクトリには、LyX の操作画面を定義する拡張子 .ui のファイルが納められています。つまり、これらのファイルは、どのメニュー項目がどのメニューに現れるかを定義し、どの項目がツールバーに現れるかを定義しています。

2.1.3 変更を加えない方がよいファイル

これらのファイルは LyX が内部的に使用するもので、あなたが開発者でない限りは、凡そこれらに変更を加える必要はありません。

CREDITS このファイルは、LyX 開発陣の名簿です。この内容は、メニュー項目ヘルプ▷LyX についてで表示されます。

`chkconfig.ltx` これは、自動設定プロセスによって使用される LaTeX スクリプトです。直接実行しないでください。

`configure.py` これは、LyX の再設定によって使用されるスクリプトです。これは、このスクリプトを実行したディレクトリに設定ファイルを生成します。

2.1.4　ひとこと必要なファイル群...

`encodings` このファイルには、各文字エンコーディングがどのように Unicode にマップされるかを示した表が載っています。

`external_templates` このファイルには、新しく導入された外用ひな型機能で使用できるひな型が載っています。

`languages` このファイルには、現在 LyX がサポートしている言語の全一覧が載っています。

2.2　ユーザのローカル設定ディレクトリ

LyX を非特権ユーザとして利用している場合でも、自分自身で使うために、LyX の設定を変更したいと思うかもしれません。`UserDir` ディレクトリには、すべての個人設定ファイルが収められています。これは、ヘルプ▷LyX についてで「ユーザーディレクトリ」として言及されているディレクトリです。このディレクトリは、`LyXDir` のミラーとして使用されており、これは `UserDir` 内のすべてのファイルが、`LyXDir` の対応するファイルを置き換えるものとして機能していることを意味します。前節で述べられた設定はどれも、全ユーザに影響する全システム用ディレクトリに置くこともできますし、自分自身で使うために個人のローカルディレクトリに置くこともできます。

わかりやすくするために、いくつか例を挙げましょう。

- ツール▷設定ダイアログで設定されるユーザ設定は、`UserDir` 中の `preferences` ファイルに保存されます。
- ツール▷再設定を使用して再設定を行うと、LyX は `configure.py` スクリプトを実行し、その結果のファイルは、ご自身のローカル設定

ディレクトリに書き込まれます。これはすなわち、UserDir/layouts にご自身で追加したテキストファイルは、文書▷設定ダイアログのクラス一覧に表示されるようになることを意味します。

- たとえば、LyX の FTP サイトから最新の取扱説明書をとってきたものの、使用中のシステム上で管理者権限がないために、それをインストールすることができなかったとしても、それらのファイルを UserDir/doc/ディレクトリにコピーすれば、ヘルプメニュー項目はこれらを開くようになります！

2.3 LyX を複数の設定を使って実行するには

ローカル設定ディレクトリにおいて設定の自由度があるだけでは、2 つ以上の設定を自由に使いこなしたい場合には充分ではないかもしれません。たとえば、使用する度に異なるキー割当を使用したり、異なるプリンタ設定を使用したいことがあるかもしれません。これは、複数の設定ディレクトリを作ることで実現することができます。そして、実行時にどのディレクトリを使用するか指定するのです。

LyX をコマンドラインスイッチ-userdir <**ディレクトリ名**>と共に起動すると、設定を既定のディレクトリではなく、指定したディレクトリから読み込むように、指示することになります（LyX を-userdir スイッチなしで実行すれば、既定ディレクトリを指定することになります）。指定したディレクトリが存在しない場合には、LyX は、初めて LyX を実行したときに既定ディレクトリを訊いてくるのと同様に、そのディレクトリを作るかどうか訊いてきます。この追加したユーザディレクトリでは、既定ディレクトリで行うのと全く同じように設定オプションを修正することができます。これらのディレクトリは完全に独立しています（が、読み進めてください）。また、環境変数 LYX_USERDIR_20x を特定の値に設定しても、全く同じ効果があります。

複数の設定を持つことはまた、維持の手間も増えるということです。もし新しいレイアウトを NewUserDir/layouts に加えて、これをすべての設定で利用できるようにしたいならば、これをすべての設定ディレクトリで個々に付け加えなくてはなりません。これを避けるには、次のようなトリックを使用してください。LyX が新しい設定ディレクトリを生成すると、そのサブディレクトリ（上記参照）はほとんど空です。新しい設定が既存のものをミラーするようにするには、空のサブディレクトリ

を、既存の設定の対応するサブディレクトリへのシンボリックリンクに置き換えてください。ただし doc/サブディレクトリには、設定スクリプト（ツール▷再設定で使用可能）が書き出した、設定毎に異なるファイルが含まれていますので、注意を払ってください。

第3章　設定ダイアログ

設定ダイアログのオプションのすべては、**ユーザーの手引き**の付録**設定ダイアログ**に述べられています。オプションのうちいくつかについて、ここでさらに詳細に説明します。

3.1　書式

ファイル書式が定義されていない場合、はじめの一歩は、使いたいと思うファイル書式を定義することです。それには、ツール▷設定ダイアログを開いてください。ファイル処理▷ファイル書式の中で新規...ボタンを押して、登録する新しい書式を定義してください。書式フィールドは、GUI 中で書式を認識するために用いられる名称です。短縮名は、書式を内部的に識別するために用いられます。さらにファイル拡張子も入力する必要があります。これらはすべて必須事項です。オプションの短絡キーフィールドは、メニュー中で短絡キーを提供するのに使用されます（たとえば、Alt-V D を押すと表示▷DVI となります）。

書式には、閲覧プログラムと編集プログラムを関連づけることができます。たとえば、PostScript ファイルを閲覧するのに Ghostview を使用したいとしましょう。このプログラムを起動するのに必要なコマンドを対応するフィールドに入力します。ここで、コマンドを定義するのに、次節に掲げる 4 つの変数を用いることができます。この閲覧プログラムは、LyX 中で画像を閲覧したり表示メニューを使用したときに起動されます。一方、編集プログラムは、たとえば、画像を右クリックして現れるコンテクストメニューで外部で編集を選択したときに起動します。

文書書式オプションは、LyX に、この書式が文書として書き出すのに適していることを指示するものです。このオプションが有効となっていて、適切な変換経路が存在する場合には（第 3.3節を参照）、この書式がファイル▷書き出しメニューに表示されます。また、この書式に閲覧プログラムが指定されている場合には、この書式は表示メニューにも表示さ

れます。png のような純粋な画像書式は、このオプションを有効にしてはいけません。pdf のようにベクター画像であると同時に文書でもあるような書式は、これを有効にします。

ベクター画像書式オプションは、LyX にこの書式がベクター画像を含みうることを教示するものです。この情報は、pdflatex を書き出す際に、内包されている画像をどの書式に変換するかを決定するのに使用されます。pdflatex は、pdf・png・jpg 以外の画像書式を取り扱うことができないので、内包されている画像は、これらの書式に変換される必要があるかもしれない為です。内包されている画像が既に pdf・png・jpg のいずれかになっていない場合には、ベクター画像書式オプションが有効になっている場合には pdf に変換され、そうでない場合には png に変換されます。

3.2　複写子

書式の変換はすべて、LyX の一時ディレクトリで行われるため、変換用にファイルを一時ディレクトリにコピーする前段階で、ファイルに変更を加える必要のあることがあります[1]。これは複写子によって取り扱われ、複写子は、ファイルを一時ディレクトリに（あるいは一時ディレクトリから）コピーすると同時に、その過程でファイルに変更を加えます。

複写子の定義においては、以下の 4 つの変数を用いることができます。

`$$s`　　LyX のシステムディレクトリ（例：/usr/share/lyx）

`$$i`　　入力ファイル

`$$o`　　出力ファイル

`$$l`　　「LaTeX 名」

最後の変数は、LaTeX の **\include** コマンドで使用されるのと同形式のファイル名です。これは、書き出すファイルがそのようなインクルードに適している場合のみ、使用さるべきものです。

複写子は、出力ファイルに関する操作であれば、ほとんどすべてに対応することができます。たとえば、生成した PDF ファイルを、/home/you/pdf/と

[1] たとえば、ファイルが他のファイル—たとえば画像—を、相対ファイル名を用いて参照している場合、このファイルが一時ディレクトリにコピーされると参照が無効になる場合があります。

いう特別なディレクトリにコピーしたいものとしましょう。その場合には、以下のようなシェルスクリプトを書きます。

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

これを、自身のローカルLyXディレクトリ—たとえば/home/you/.lyx/scripts/pdfcopier.sh—に保存し、お使いのプラットフォームが必要とするならば、実行可能属性を付与します。それから、ツール▷設定ダイアログのファイル処理▷ファイル書式の中で、PDF(pdflatex) 書式—あるいは他の PDF 書式のうちどれか—を選択し、複写子フィールドに `pdfcopier.sh $$i $$o` と入力します。

複写子は、LyX 自身が様々な変換に使用します。たとえば、適切なプログラムが検出された場合、LyX は自動的に HTML 書式と HTML (MS Word) 書式の複写子を導入します。これらの書式を書き出す際、複写子は、本体の HTML ファイルだけでなく、関連した様々なファイル（スタイルファイルや画像など）もコピーされるように手配します。これらのファイルはすべて、元の LyX ファイルのあるディレクトリのサブディレクトリに書き込まれます[2]。

3.3 変換子

各書式間でファイルを変換するために、ご自身の変換子を定義することができます。これは、ツール▷設定▷ファイル処理▷変換子ダイアログで行います。

新規に変換子を定義するには、ドロップダウンリストから変換元の書式と変換先の書式を選択し、変換に必要なコマンドを入力してから追加ボタンを押してください。変換子の定義には、以下のような変数を使用することができます。

[2] この複写子の挙動は調整することができます。非必須の「-e」オプションは、コピーする拡張子をコンマ区切りで羅列したものを引数にとります。これを省略した場合には、すべてのファイルがコピーされます。「-t」引数は、生成したディレクトリに書き加える拡張子を指定するものです。既定値では、これは「LyXconv」となっているので、/path/to/filename.lyx から生成された HTML ファイルは、/path/to/filename.html.LyXconv となります。.

`$$s`　LyX システムディレクトリ

`$$i`　入力ファイル

`$$o`　出力ファイル

`$$b`　入力ファイルのベースファイル名（拡張子をとった部分）

`$$p`　入力ファイルのパス

`$$r`　元の入力ファイルのパス（変換子が連鎖して呼び出されたときの挙動が$$p とは異なります）

`$$e`　文書エンコーディングの iconv 名

追加フラグフィールドには、以下のフラグをコンマで区切って入力することができます。

`latex`　この変換子が LaTeX の一種を実行することを示します。これによって、LyX の LaTeX エラーログに記録を残せるようになります。

`needaux`　変換に LaTeX の.aux ファイルが必要であることを示します。

`xml`　出力が XML であることを示します。

以下の 3 つのフラグは key = value 形式の引数をとります（したがって厳密にはフラグとは呼べません）。

`parselog`　これを指定すると、変換子の標準エラーが `infile.out` ファイルにリダイレクトされ、引数に指定されたスクリプトが `script < infile.out > infile.log` の形で実行されるようになります。引数には`$$s` を指定することができます。

`resultdir`　これには、変換子が生成したファイルをダンプするディレクトリ名を指定します。LyX はこのディレクトリを作成せず、ここに何もコピーしませんが、このディレクトリを宛先にコピーします。引数には、`$$b` を使用することができ、これはディレクトリがコピーされる際に、入力ファイルおよび出力ファイルのベース名で置換されます。
resultdir と usetempdir は、同時に用いることはできませんのでご注意ください。前者が指定されているときには、後者は無視されます。

`resultfile` これは出力ファイル名を指定するもので、`$$b` を使用することができます。resultdir が指定されているときのみ有効で、必ず用いる必要はありません。指定されていなければ、既定値は「index」です。

最後の 3 つは、LyX とともに導入される変換子には、現在いずれも使用されておりません。

変換しようとするすべての書式のあいだに変換子を定義する必要はありません。たとえば、「LyX から PostScript」変換子が定義されていないのに、LyX は PostScript を書き出していることに気づかれることでしょう。これは、まず LaTeX ファイルを生成した後に（これには変換子を定義する必要はありません）、「LaTeX から DVI」変換子を使用して DVI に変換し、最後に、得られた DVI を PostScript に変換することによって実現しています。LyX はこのような変換子の「連鎖」を自動的に見つけ、つねに最も短い連鎖を選択します。しかしながら、なお書式間に複数の変換方法を定義することも可能です。たとえば、標準的な LyX 設定は、LaTeX から PDF へ変換するのに、以下の 3 つの方法を用意しています。(1) 直接 pdflatex を使用するもの。(2) （DVI と）PostScript を経由して ps2pdf を使用するもの。(3) DVI 経由で dvipdfm を使用するもの。このように代替連鎖を定義するには、第 3.1節に述べられているように、ターゲットとなる「ファイル書式」を複数定義しなくてはなりません。たとえば、標準設定では、pdf・pdf2・pdf3 と命名された書式が定義されていて、すべて共通の拡張子.pdf を持ち、上記で言及した各変換方法に対応しています。

第4章　LyXの各国語対応

LyXは、翻訳された操作画面の利用をサポートしています。私たちが最後に確かめたところでは、LyXは30言語の翻訳を提供しています。選択した言語は、使用する**ロケール**と呼ばれます（ロケール設定についての詳しい資料は、お使いの基本ソフトに添付のロケール関連説明書をご覧ください。Linuxの場合は、マニュアルページの `locale(5)` から見ると良いかもしれません）。

これらの翻訳は適切に機能しますが、欠点もいくつかあることに注意してください。たとえば、ダイアログはすべて英文を念頭にデザインされているため、翻訳文の一部は、割り当てられたスペースに収めるには大きすぎるかもしれません。これは表示上の問題に過ぎず、他の障害は引き起こしません。また、翻訳によっては、すべての短絡キーが定義されていないことに気づかれるでしょう。短絡キーのために空いている文字が十分ないことが時々あるのです。単に翻訳者がまだ短絡キーを定義していないこともあるでしょう。もちろん、私たちの各国語対応チーム—あなたも参加したいと思われるかもしれません[1]—は、LyXの将来のバージョンでこれらの欠点を修正しようとするでしょう。

4.1　LyXを翻訳する

4.1.1　グラフィカル・ユーザ・インタフェース（テキスト・メッセージ）を翻訳する

LyXは、操作画面の国際化対応にGNU `gettext` ライブラリを使用します。LyXのすべてのメニューやダイアログでお好みの言語を話させたいときには、その言語の `po` ファイルが必要です。このファイルが利用可能であれば、そこから `mo` ファイルを生成して、この `mo` ファイルをイン

[1] もしあなたが英語以外の言語を流暢に操れるならば、これらのチームに参加することは、LyXコミュニティに報いるたいへん素晴らしい方法です！

ストールしなくてはなりません。この全過程は、GNU gettext の取扱説明書に説明があります。この作業をあなたのためだけに行うこともできますが、もしせっかくするのであれば、あなたの骨折りの結果を LyX コミュニティの他の人々と分かち合いませんか。どのように段取りを進めればよいか、詳しくは LyX 開発者メーリングリストにメールを送ってください。

要約すれば、以下のように行います（xx は言語コードを表します）。

- LyX ソースコードをチェックアウトしてください（ウェブ上の情報参照）。
- lyx.pot ファイルを**.po ファイルのあるフォルダにコピーして、xx.po に名前を付け替えてください（lyx.pot がどこにもない場合には、コンソールからそのディレクトリで make lyx.pot コマンドを実行し、作成し直すか、他言語の既存の po ファイルをひな型として使用することができます）。
- xx.po を編集します[2]。メニューラベルやウィジェットラベルのうちには、翻訳しなくてはならない短絡キーがある場合があります。これらのキーは「|」の後に記されており、当該言語の単語やフレーズに対応して翻訳しなくてはなりません。さらに、新しい po ファイルの冒頭に、あなたの電子メールアドレスなどの情報も書き加えて、人々があなたに提案や、滑稽な怒りのメッセージを届けることができるようにしてください。

もし、あなたがこれを自身のためだけに行っているのであれば、

- xx.mo を生成してください。これは msgfmt -o xx.mo < xx.po でできます。
- この mo ファイルを、お使いのロケールツリー中、言語 xx のアプリケーションメッセージ用の正式なディレクトリにコピーして、lyx.mo という名称にしてください（例:/usr/local/share/locale/xx/LC_MESSAGES/lyx.mo

[2] これは単なるテキストファイルなので、どのテキストエディタでも編集できます。しかし、Poedit（全プラットフォーム用）や KBabel（KDE 用）のように、この目的の編集をサポートする特別なプログラムがあります。Emacs にも po ファイルを編集するための「モード」があります。

しかしながら前述のように、この新しい po ファイルを他の人たちが使用できるよう、LyX 頒布版に追加できることが最善です。これを追加するには、LyX に変更を加える必要がありますので、もしその気があれば、開発者メーリングリストに電子メールを送ってください。

4.1.1.1 多義訳語メッセージ

時には、一つの英語のメッセージが、翻訳先の言語では複数のメッセージに翻訳されなくてはならないことが判明することがあります。一つの例は、To というメッセージで、これは英語で「to」がどういう意味を持っているかによって、独語では Nach と訳されたり Bis と訳されたりします。GNU gettext は、このような多義訳語を To の代わりに、`To[[as in 'From format x to format y']]` や `To[[as in 'From page x to page y']]` としなくてはなりません。これによって、これら 2 つの To は、gettext には別物と解釈され、それぞれ正しく Nach と Bis に訳すことができるようになります。

もちろん、この文脈情報は、翻訳が存在しないときには取り去られる必要がありますので、メッセージの終わりに二重大括弧で囲わなくてはなりません（上例参照）。LyX の翻訳機構では、メッセージの終わりに二重大括弧で囲われているものはすべて、メッセージを表示する前に取り去るようにされています。

4.1.2 説明書を翻訳する

（Help メニュー中の）オンライン説明書は翻訳することができます（そして翻訳されるべきです！）。説明書の翻訳版が利用可能であり[3]、ロケールがその言語に設定されている場合、LyX はこれを自動的に使用します。LyX は、翻訳版を `LyXDir/doc/xx/DocName.lyx`（xx は現在使用している言語コード）で探します。翻訳文書がない場合には、既定の英語版が表示されます。翻訳版は、原典と同じファイル名（上述の DocName）を持っていなくてはならないことに注意してください。説明書を翻訳する気がおありであれば（これは原典の校正としてもたいへん役立ちます！）、以下のような点をすぐに行うべきです。

[3] 2008 年 3 月現在、説明書の少なくとも一部が翻訳されている言語は 14 言語に上り、入門編が訳されているものはさらにいくつかあります。

- 説明書翻訳ウェブページhttp://www.lyx.org/Translationを確認してください。ここで、どの文書が（もしあれば）お使いの言語に既に翻訳されているかを見つけることができます。また、説明書をお使いの言語に翻訳する作業の面倒を見ている人を（もしいれば）見つけることができます。この作業の面倒を見ている人がいない場合には、私たちにあなたが興味をお持ちであることを知らせてください。

いったん実際の翻訳の仕事に取りかかったならば、トラブルを回避するいくつかのヒントがあります。

- 文書化チームに所属してください！そのための情報が `Intro.lyx`（ヘルプ▷はじめの一歩）にあります。また、この `Intro.lyx` が最初に訳すべき文書です。

- 翻訳しようとする言語での印刷慣行を学んでください。活版印刷は古来の技術であり、何世紀にもわたって世界の至る所で、様々な慣行を発達させてきました。また、あなたの国で活版工が用いる専門用語も学んでください。自分で勝手な専門用語を捻出するとユーザを混乱させるだけです。（**警告！活版技術は病みつきになる可能性があるので注意してくさい！**）

- 文書のコピーをとってください。これを作業用コピーとします。これをお使いの `UserDir/doc/xx/` ディレクトリにコピーすれば、個人用の翻訳ヘルプファイルとして使用することができます。

- （LyX チームが維持している）原典の説明書は、時折更新されます。変更点については、http://www.lyx.org/trac/timelineのソースビューアでご覧ください。この方法で、翻訳文書のどの部分を更新しなくてはならないか、たやすく見つけることができます。

もし原典に誤りを見つけたならば、修正して文書化チームの他のメンバーに変更したことを知らせてください（文書化チームに参加することをお忘れになっていませんよね）。

4.2　国際キー配列

以下の 2 節では、`.kmap` および `.cdef` ファイルの文法を詳細に解説します。これらの節は、提供されているキー配列があなたのニーズに合わない場合に、自身用のキー配列をデザインする手助けとなるでしょう。

4.2.1 .kmap ファイル

.kmap ファイルは、打鍵したものを文字や文字列に割り当てます。名前が示唆するように、これはキーボード配列表を定義します。.kmap ファイルは、以下の各項で説明するように、kmap・kmod・ksmod・kcomb のキーワードを定義します。

kmap　　文字を文字列に割り当てる

　　\kmap **文字 文字列**

これは、**文字**を**文字列**に割り当てます。**文字列**中では、二重引用符（"）とバックスラッシュ（\）は、前にバックスラッシュ（\）を付けてエスケープしなくてはならないことに注意してください。

&を打鍵すると/記号が出力される kmap ステートメントを、一例としてあげると、

```
\kmap & /
```

のようになります。

kmod　　アクセント文字を指定する

　　\kmod **文字 アクセント 許可文字**

これは**文字**を**許可文字**のアクセントとするものです。これはデッドキー[4]機構です。

文字を打鍵してから**許可文字**にないキーを打鍵すると、**文字**の後に許可文字ではないその文字が出力として表示されます。Backspace はデッドキーを取り消しますので、**文字**→ Backspace と打鍵すると、カーソルは一文字戻ることなく、文字が次の打鍵したものに及ぼしたはずの効力を取り消します。

以下の例は、' 文字を acute アクセントとして、a・e・i・o・u・A・E・I・O・U の文字に許可するものです。

```
\kmod ' acute aeiouAEIOU
```

[4]**デッドキー**という用語は、それ自身で文字を出力しないけれども、別のキーを続けて打つと、望んだアクセント文字を出力するキーのことを指し示します。たとえば、独語で$\ddot{a}$のようなウムラウトのついた文字は、このようにして出すことができます。

ksmod　　アクセント文字に例外を指定する

\kxmod　**アクセント 文字 結果**

これは**文字**上の**アクセント**について例外を指定するものです。ここで**アクセント**には、前出の\kmod 宣言で打鍵キーを既に割り当てられてなくてはならず、**文字**は**アクセント**の**許可文字**の集合に属していてはなりません。こうして**アクセント→文字**の順に入力すると、**結果**が出力されるようになります。.kmap ファイルにこの宣言がない場合には、**アクセント→文字**と入力すると、**アクセントキー→文字**（**アクセントキー**は\kmod 宣言の最初の変数）と出力されます。

以下のコマンドを用いると、acute-i（'i）と入力した場合、äi と出力されるようになります。

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

kcomb　　2 つのアクセント文字を結合する

\kcomb **アクセント** *1* **アクセント** *2* **許可文字**

これはなかなか難解になってきます。これは**アクセント** *1* と**アクセント** *2* を（この順番で）結びつけて、**許可文字**に効果を及ぼすようにします。**アクセント** *1* と**アクセント** *2* の打鍵キーは、ファイル内のこのコマンド**よりも前に**、\kmod コマンドで設定されていなくてはなりません。

greek.kmap ファイル上にある例をとってみましょう。

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

これは;:i を押すと\'{\"{i}}という効果を得るようにするものです。この場合のバックスペースは、最後のデッドキーを取り消すので、;: Backspace i と押した場合には、\'{i}となります。

4.2.2　.cdef ファイル

.kmap による割り当てが行われた後、.cdef ファイルは、記号の作り出す文字列を現在のフォントの文字に割り当てます。LyX 頒布版には、現在のところ、少なくとも iso8859-1.cdef ファイルと iso8859-2.cdef ファイルが含まれています。

一般的に.cdef ファイルは、

セット中の文字番号　文字列

という形の宣言の羅列です。たとえば、\'{e}を iso-8859-1 セットの対応する文字 (233) に割り当てるには、以下の宣言を用います。

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

ここで、**文字列**中の\と"はエスケープされています。同一の文字を二つ以上の文字列に充てることができることに注意してください。`iso-8859-7.cdef` ファイルには、

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

という例があります。LyX は、キー打鍵やデッドキーの組み合わせで生成される文字列の割り当てを見つけることができないとき、それがアクセント付き文字のように解釈ができないかどうかチェックして、画面上の文字にアクセントを引くことを試みます。

4.2.3 デッドキー

国際文字のサポートを追加する第 2 の方法として、いわゆるデッドキーによる方法があります。デッドキーは文字と一緒に用いて、アクセント付き文字を生成します。ここではその機能を説明するために、きわめて単純なデッドキーの作り方を説明します。

仮に、曲折アクセント記号「ˆ」が必要になったものとしましょう。この場合、自身の `lyxrc` ファイル中で、^キー（すなわち Shift-6 キー）を、LyX コマンドの `accent-circumflex` に結びつけることができます。すると^キーの後に文字を打ったときはいつでも、この文字上に曲折アクセントが付けられるようになります。たとえば「^e」という打鍵順は「ê」という文字を生成します。しかしながら、もし「^t」と打鍵したならば、「t」は曲折アクセントをとることは決してないために、LyX はビープを鳴らして文句を付けます。デッドキーの後にスペースを打つと、アクセントだけが生成されます。この最後の点に注意してください。あるキーをデッドキーに割り当てる場合には、このキー上の文字を別のキーに割り当て直す必要があります。たとえば、, キーをセディーユに割り当てるのはよ

くありません。コンマを入力しようとすると必ずセディーユが出てくるようになるためです。

デッドキーを割り当てるのによく用いられる方法は、Meta-・Ctrl-・Shift-キーを、「~」・「¸」・「^」のようなアクセントと一緒に用いる方法です。また、xmodmap や xkeycaps を使って、特別な Mode_Switch キーを設定する方法もあります。Mode_Switch キーは、ちょうど Shift キーのように機能するので、アクセント文字を割り当てるのに使用できます。また、特定のキーを usldead_cedilla などに割り当てることで、これらのキーをデッドキーに仕立て、このシンボリックキーを対応する LyX コマンドに割り当てることもできます[5]。この Mode_Switch キーには、Ctrl-キーの片方や使われていないファンクションキーなど、ほぼ何でも指定することができます。アクセントを生み出す LyX コマンドについては、*LyX* **関数説明書**の LFUN_ACCENT_*の項をご覧ください。ここには完備した一覧があります。

4.2.4　自分の言語設定を保存する

ツール▷設定ダイアログを使えば、LyX を起動したときに、ご希望の言語環境に自動的に設定されるように、設定を編集することができます。

[5] John Weiss からの註：これはまさに私が、自分の~/.lyx/lyxrc と~/.xmodmap で行っていることです。私は、Scroll Lock キーを Mode_Shift に仕立てて、多数の usldead_*シンボリックキーを Scroll Lock-^や Scroll Lock-~などに割り当てています。私はこの方法でアクセント文字を入力しています。

第5章　文書クラスやレイアウトやひな型を新規に導入する

この章では、新しくLyXのレイアウトファイルやひな型ファイルを作成して、導入する手順を説明すると共に、新規にLaTeX文書クラス（ドキュメントクラス）を正しく導入する方法を復習します。

まず、LyXとLaTeXの間の関係をどのように考えるべきか、若干の注釈を加えておくことにしましょう。理解していただきたいことは、ある意味において、LyXは、LaTeXについて何も知らないと云うことです。実際のところ、LyXの観点からは、LaTeXは、LyXが出力を生成することができる、複数の「バックエンド書式」のうちの一つに過ぎないということです。同種のバックエンド書式には、DocBook・平文・XHTMLがあります。もちろんLaTeXは、とくに重要な書式ですが、LyXがLaTeXについて持っている情報のほとんどは、実はプログラム本体には含まれていないのです[1]。このような情報は、`article.cls`のような標準クラスでも、「レイアウトファイル」に保管されています。同様に、LyXは、DocBookやXHTMLについてもほとんど知りません。LyXが知っていることは、レイアウトファイルの中にあります。

文書クラス用のレイアウトファイルは、LyX構成体—対応する様式や何らかの差込枠などを有する段落群—と、それに対応するLaTeX構成体・DocBook構成体・XHTML構成体との間の翻訳指南書のようなものです。たとえば、LyXが`article.cls`について知っていることのほとんど総ては、`article.layout`と、それが呼び出す他の様々なファイルに書き込まれています。このことから、レイアウトファイルを書こうとする人は、既存のファイルを研究することを勧めます。とっかかりとしては、`article.layout`

[1]過度に複雑なため、LyXに「ハードコード化」されているコマンドもありますが、一般的に開発者は、これを「わるいこと」とみなしています。

や book.layout や、文書クラス用の他のレイアウトファイルに取り込まれている stdsections.inc から見始めるのがよいでしょう。このファイルは、節などの定義が為されている場所です。stdsections.inc は、節様式や小節様式などとしてマークされている段落を、対応する LaTeX・DocBook・XHTML のコマンドやタグにどのように翻訳すべきかを LyX に知らせるものです。基本的に article.layout ファイルは、これらの std*.inc ファイルを取り込んでいるだけのものです。

しかしながら、LyX–LaTeX 間の対応を定義するだけが、レイアウトファイルが行うことではありません。レイアウトファイルが行うもう一つの仕事は、LyX 構成体自身が画面上にどのように表示されるべきかを定義することです。この 2 つの仕事は全く独立したものであるので、レイアウトファイルが 2 つの仕事を行うという事実は、しばしば混乱を引き起こす元となります。ある段落様式を LaTeX に翻訳する仕方を LyX に指示することは、その表示の仕方を LyX に指示するものではありません。逆に、ある段落様式の表示の仕方を LyX に指示することは、その段落様式をどのように LaTeX に翻訳するかを LyX に指示するものではありません（ましてや LaTeX に表示の仕方を指示するものではありません）。つまり、一般的に、新しい LyX 構成体を定義する際には、(i) LaTeX にどのように翻訳するかを LyX に指示する、(ii) それをどのように表示するかを LyX に指示する、という、二つのかなり異なることを行わなくてはならないのです。

もちろん、LyX の他のバックエンド書式に関しても、ほぼ同じことが言えますが、XHTML の場合には若干事情が異なり、LyX が、ブラウザ中での段落の表示方法を（CSS の形で）出力するにあたって、当該段落を LyX が画面上に出力する仕方の情報を、ある程度利用することができます。しかし、この場合でも、LyX が内部的に行うことと、外部的に行う物事の間の区別は、依然として有効であり、この 2 つは独立して制御することができます。詳細に関しては、第 5.4節をご覧ください。

5.1　新しい LaTeX ファイルの導入

頒布版によっては、LyX で使いたい LaTeX パッケージやクラスファイルが含まれていないことがあるかもしれません。たとえば、オーバーヘッドプロジェクタ用のスライドを準備するためのパッケージである、FoilTeX がないかもしれません。TeXLive（2008 年以降）や MiKTeX のような最

近の LaTeX 頒布版には、これらのパッケージを導入するためのユーザーインタフェースが用意されています。たとえば、MiKTeX では、付属の「Package Manager」プログラムを起動すると、利用できるパッケージの一覧を得ることができます。どれかを導入するには、その上で右クリックするかツールバーボタンを押してください。

お使いの LaTeX 頒布版がこのような「パッケージマネジャー」を提供していなかったり、使用中の頒布版にそのパッケージが入っていない場合には、以下のステップに従って手動で導入してください。

1. CTANなどから欲しいパッケージを入手してください。

2. パッケージに「.ins」で終わるファイル名が入っている場合（FoilTeX がその一例です）は、コンソールを開いて、このファイルのフォルダに移動し、コマンド `latex foiltex.ins` を実行してください。すると、パッケージが解凍されて、導入すべきすべてのファイルが展開されます。たいていの LaTeX パッケージは圧縮されていないので、このステップは飛ばすことができます。

3. ここで、パッケージを全ユーザーに使用可能にするか自分自身だけで使うかを決定する必要があります。

   (a) （Linux・OSX などの）*nix 系システムでは、システム上の全ユーザーに新パッケージを利用可能にしたければ、「ローカル」TeX ツリーに導入し、そうでなければ「ユーザー」TeX ツリーに導入してください。これらのツリーが存在しない場合にどこに作成すればよいかは、お使いのシステムに依存します。これを見いだすには、`texmf.cnf` ファイルを参照してください[2]。「ローカル」TeX ツリーの場所は、`TEXMFLOCAL` 変数で定義されており、通常は`/usr/local/share/texmf/`のような場所になっています。「ユーザー」TeX ツリーの場所は、`TEXMFHOME` で定義されており、通常は`$HOME/texmf/`です（もしこれらの変数が事前定義されていなければ、定義しなくてはなりません）。「ローカル」ツリーを作成したり変更したりするには、おそらく root 権限が必要ですが、「ユーザー」ツリーにはこのような制限はありません。

[2] このファイルは、通常`$TEXMF/web2c` ディレクトリにありますが、コマンド `kpsewhich texmf.cnf` を実行してその場所を見つけることもできます。

一般的に、システムをアップグレードした際に、ユーザーが修正されたり上書きされたりということが起こらないので、ユーザーツリーに導入することが推奨されます。こうすると、自分のホームディレクトリをバックアップする際に、パッケージも他のものと一緒にバックアップされます（もちろん通常行われるようにすればの話です）。

(b) Windowでは、システム上の全ユーザーに新パッケージを利用可能にしたければ、LaTeXの導入されているフォルダに移動し、それからサブフォルダ`~\tex\latex`に移動します（MiKTeXでは、これは既定では`~:Programs\MiKTeX\tex\latex`です）[3]。ここに新規フォルダ`foiltex`を作成し、パッケージの全ファイルをそこにコピーしてください。パッケージを自分だけで使用したい場合や、admin権限を持っていない場合には、ローカルLaTeXフォルダで同じことを行います。たとえばMiKTeX 2.8では、これはWinXP上では

```
~:\Documents and Settings\<ユーザー名>\Application Data\
   MiKTeX\2.8\tex\latex
```

フォルダ、WinVista上では

```
~:\Users\<ユーザー名>\AppData\Roaming\2.8\MiKTeX\tex\latex
```

フォルダになります。

4. ここまで来れば、あとはLaTeXに新しいファイルがあることを告げるだけです。これは使用しているLaTeX頒布版に依存します。

   (a) TeXLiveの場合には、コンソールから`texhash`コマンドを実行してください。パッケージを全ユーザー用に導入した場合には、おそらくroot権限で行う必要があります。

   (b) MiKTeXでは、パッケージを全ユーザー用に導入した場合には、「Settings (Admin)」を起動し、「Refresh FNDB」と記してあるボタンを押してください。そうでない場合には、「Settings」を起動して同様に行ってください。

5. 最後に、LyXに新しいパッケージがあることを告げなくてはなりません。そこで、LyXからツール▷再初期設定メニューを実行して、

[3] これは、英語版でのみ正しいパスになっています。独語版では`~:Programme\MiKTeX\tex\latex`となり、他の言語でも同様です。

LyX を再起動します。

これでパッケージが導入されました。この例では、文書クラス Slides (FoilTeX) が文書▷設定▷文書クラスで利用可能になっているはずです。

文書▷設定▷文書クラスメニューに列挙されてもいない LaTeX 文書クラスを使用したい場合には、その「レイアウト」ファイルを作り出さなくてはなりません。これが次節のトピックです。

5.2 レイアウトファイルの型

この節は、レイアウト情報を含む各種 LyX ファイルについて述べます。これらのファイルは、各種段落様式や文字様式についての記述がされているものであり、LyX がそれらをどのように表示すべきなのか、また、それらをどのように LaTeX や DocBook、XHTML その他の出力書式に翻訳すればよいのかが記されています。

ここでは、レイアウトファイル作成過程の包括的な解説を試みたいと思いますが、LaTeX クラスだけでもサポートする文書の種類があまりにたくさんあるので、読者が出会いうるケースや問題をすべてカバーすることはとても望めません。LyX ユーザーメーリングリストには、自身の経験を人々と分かち合いたいと望む、レイアウトデザインの経験豊かな人々がよく顔を出していますので、気軽に質問を投げかけてみてください。

新しいレイアウトを準備するに当たっては、LyX と共に頒布されているレイアウトの例を見るのがたいへん役立ちます。他の人々も使用できる LaTeX 文書クラス用の LyX レイアウトを作ったり、他の人々にも有用なモジュールをお書きになった場合には、LyX Wiki のレイアウトに関する節か、LyX 開発者メーリングリストに投稿して、LyX 頒布版に同梱することができるようにしてください[4]。

5.2.1 レイアウトモジュール

ここまで、「レイアウトファイル」についてお話してきました。しかし、レイアウト情報を含むものには、他の種類のファイルもあります。厳密にレイアウトファイルと呼ぶとき、それは .layout 拡張子を持ち、文書クラ

[4] LyX は General Public License の下でライセンスされていますので、LyX に寄贈されたものは総て同じライセンス下に置かれることに注意してください。

スに関する情報を LyX に提供するものを指します。しかしながら、LyX 1.6 以降、レイアウト情報は、拡張子が .module のレイアウト**モジュール**にも含めることができます。レイアウトが LaTeX クラスに対応しているように、モジュールは LaTeX パッケージに対応するものであり、endnotes モジュールのように、特定のパッケージにサポートを提供するためのモジュールもあります。レイアウトモジュールは、特定の文書レイアウトに特化したものではなく、多くのレイアウトで使用できるという意味において、一面、stdsections.inc 等のインクルードファイル[5]のようなものです。相異なる点といえば、article.cls でインクルードファイルを使用するには、そのファイルを編集しなくてはなりませんが、モジュールの場合は、文書▷設定ダイアログで選択するだけですみます。

モジュール作成は、新しく段落様式を一つ加えたり、任意設定差込枠を加えたりするだけで済むことも多いので、レイアウト編集を学ぶ上でもっとも易しい方法です。しかし原理的には、レイアウトファイルに入れることのできるものはすべて入れることができます。

新しいモジュールを作成し、それを layout/ フォルダにコピーした後、モジュールがメニューに現れるようにするためには、LyX の再設定を行って再起動しなくてはなりません。しかしながら、モジュールの修正の場合には、文書▷設定を開いてどれかを選択し「OK」を押せば、直ちに反映されます。**これを実行する前に、作業中の文書を保存しておくことを強く勧めます**。もっと言えば、**実際の文書で作業しているときに、同時にモジュールの編集をしようとしないことを強く勧めます**。もちろん開発陣は、そのような場合でも LyX が安定性を維持するように努力していますが、あなたが作成したモジュール中の文法エラー等によって、奇妙な挙動が引き起こされることがあるからです。

5.2.2　.sty ファイル用のレイアウト

新しく LaTeX 文書クラスをサポートしようとするとき、LaTeX 2ε クラス（.cls）ファイルが絡む場合と、スタイル（.sty）ファイルが絡む場合の 2 つの状況があり得ます。スタイルファイルのサポートは、通常は、かなり容易ですが、新しくクラスファイルをサポートすることは、もう少し難しくなります。この節では、前者について述べることとし、後者につ

[5] これらは任意の拡張子をつけることができますが、慣習的に .inc 拡張子が用いられます。

いては次節に譲ります。当然のことながら、新しく DocBook DTD をサポートしたい場合にも、同様の所見が当てはまります。

この易しい方の場合では、新しい文書クラスは、既にサポートされている文書クラスと共に使うスタイルファイルとして提供されています。例示のために、スタイルファイルは myclass.sty という名称で、標準的なクラスである report.cls と共に用いられるものと仮定しましょう。

既存の文書クラスのレイアウトファイルを、以下のように、お使いのローカルディレクトリにコピーすることから始めてください[6]。

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

それから、myclass.layout を編集して、

```
\DeclareLaTeXClass{report}
```

という行を

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

のように変更してください。それから、ファイル冒頭辺りに

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

と書き加えてください。

LyX を起動してツール▷再設定を選択してください。それから LyX を再起動し、新規文書を作成してみてください。すると、文書▷設定ダイアログの文書クラスオプションに「report (myclass)」が現れるはずです。新しいクラスにおいて、節区切り用コマンドなどの一部が、基礎となったクラス—この例では report—とは違う挙動をすることはよくありますので、希望に応じて、各節の設定をいじると良いでしょう。各節のレイアウト情報は、stdsections.inc に含まれていますが、このファイルをコピーしたり変更したりする必要はありません。代わりに、自身のレイアウトファイル中、stdsections.inc も取り込む Input stdclass.inc の後に変更を加えるだけです。たとえば、章見だしのフォントをサンセリフ体に変更するには、以下のような行を加えます。

[6] もちろん、どのディレクトリがローカルディレクトリとなるかは、プラットフォームに依存します。LyX では、起動時に-userdir オプションを指定することによって、ローカルディレクトリを指定することも可能です。

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

これは、既存の章様式宣言を上書き（あるいはこの場合には追加）します。

新しいパッケージでは、基礎となったクラスには存在しないコマンドや環境を提供することもできます。この場合には、これらをレイアウトファイルに加えます。そのやり方については、第5.3節の情報を参照してください。

もしmyclass.styが他の文書クラスで使用することができたり、あるいはできない場合でも、基礎となるクラスから読み込むことのできるモジュールを書くのが最も簡単であることがわかるでしょう。最も簡単なモジュールの例としては、以下のようなものになります。

```
#\DeclareLyXModule{My Package}
#DescriptionBegin
#Support for mypkg.sty.
#DescriptionEnd
Format 21
Preamble
    \usepackage{mypkg}
EndPreamble
```

もう少し複雑なモジュールでは、既存の構成物の挙動を修正したり、新しい構成物を定義したりすることになるでしょう。この辺りの議論については、第5.3節を参照してください。

5.2.3 .cls ファイル用のレイアウト

これには2つのケースがありえます。ひとつは、クラスファイル自体が既存の文書クラスに立脚している場合です。たとえば、多くの学位論文用クラスはbook.clsに基づいています。お使いのものがどうであるかを見るには、クラスファイル中に

```
\LoadClass{book}
```

という行がないかどうか探してください。もしこれがあれば、DeclareLaTeXClass 行は異なりますが、おおよそ前節のように進めることができます。あなたが新しく作るクラスが thesis であり、book クラスに基づいていれば、DeclareLaTeXClass 行は以下のようにします[7]。

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

他方、新しいクラスが既存のクラスに基づいていない場合には、おそらくあなた自身のレイアウトをしたためる必要があります。もし可能であれば、類似した LaTeX クラスを使用している既存のレイアウトファイルをコピーして、それに修正を加えるようにすることを強くお勧めします。少なくとも、どの項目を考慮すべきかがわかるように、既存のファイルを作業の開始点としてください。

5.2.4 ひな型を作成する

新しい文書クラス用のレイアウトファイルを書いたならば、そのレイアウト用のひな型も書くことを検討されるかもしれません。ひな型は、内容はダミーですが、レイアウトの使い方を示す一種のチュートリアルとして動作します。もちろん、イメージを得るために、LyX 添付のひな型をあれこれ見てみるのもよいでしょう。

ひな型は、通常の文書と同様、LyX を使って作成することができます。唯一違う点は、通常の文書では、フォント構成や用紙寸法を含め、すべてのあり得る設定が為されている点です。これらの場合、通常ユーザーはひな型が彼の設定値を上書きすることを望みません。この理由から、ひな型の設計者は、\fontscheme や\papersize などの対応するコマンドをひな型 LyX ファイルから取り除く必要があります。これは、たとえば vi や notepad のような、どの軽いテキストエディタでも行うことができます。

編集したひな型を UserDir/templates/に置き、グローバルなひな型ディレクトリ LyXDir/templates/から使用したいものを同じ場所にコピーし、ツール▷設定▷パスダイアログのひな型パスを再定義してください。

ところで、特別な意味を持つひな型 defaults.lyx があることに注意してください。このひな型は、ファイル▷新規を使って新規文書を作成する際、便利な既定値を提供する目的で必ず読み込まれます。このひな型

[7] さらに LyX は、文書クラス名がレイアウトファイル名と同じだと仮定するので、クラスファイルを thesis.layout という名前で保存するのが最も簡単です。

をLyX内部から作成するのにしなくてはならないことは、対応する設定を持つ文書を開き、文書既定値として保存ボタンを押すことです。

5.2.5 旧レイアウトファイルの更新

レイアウトファイルの書式は、LyXのリリース毎に変更されますので、古いレイアウトファイルは変換されなくてはなりません。この過程は、LyX第1.4版から自動化されました。LyXが古い書式のレイアウトファイルを読み込むと、LyXは、自動的に変換ツールLyXDir/scripts/layout2layout.pyを呼び出し、それを現在の書式の一時ファイルに変換します。元のファイルは変更を加えられずに措かれます。もしこのレイアウトファイルをよく使うならば、LyXがこれを毎回行なわくて済むように、レイアウトファイルを恒久的に変換しておきたいと思うかもしれません。これを行うには、以下のように変換子を手動で呼び出してください。

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

もちろん、LyXDirは、お使いのLyXシステムディレクトリ名に置き換えてください。

手動変換は、インクルードされているファイル内部の変更までは取り扱いませんので、それらのファイルは別に変換されなくてはなりません。

5.3 レイアウトファイルの書式

以下の各節では、いよいよ自分の手を汚してレイアウトファイルを作成したり編集したりする段階になった際、直面することの説明を行います。私たちからのアドバイスとしては、ゆっくりと進めるようにして、ちょっと進むごとに保存やテストを行い、心休まる音楽を聴き、好きな大人の飲み物を一二杯口にしながら行うのがいいでしょう。特に行き詰まってしまったときにはそうです。実際にはそんなに難しいことではないのですが、特に一度に多くのことをやろうとすると、選択肢が多くありすぎて圧倒されてしまうのです。さて、もう一杯大人向け飲料をどうぞ。適量ね。

この章で述べられているタグは、すべて大文字小文字を区別しません。つまり、Style・style・StYlEは同じコマンドとなります。機能名の後

にある角括弧は、その機能が取り得る値を示します。テキストクラス設定内で機能が特定されていない場合には、既定値は**強調**で表記されます。引数が「文字列」や「浮動小数点型」などのデータ型をとる場合には、既定値は浮動小数点型=**既定値**のように表示されます。

5.3.1 文書クラス宣言

レイアウトファイル中の#で始まる行はコメントです。この規則には一つだけ例外があります。すべての*.layout ファイルは、以下のような行で始めなくてはならないのです。

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

この 2 行目は、LyX を初期設定や (再) 設定するときに用いられるのです。このレイアウトファイルは、LaTeX スクリプト chkconfig.ltx が、#を無視する特別なモードで読み込みます。1 行目は単なる LaTeX コメントですが、2 行目にはテキストクラスの宣言が書かれています。これらの行が article.layout と名付けられたファイルにあると、article（レイアウトファイル名）という名称のテキストクラスを定義し、LaTeX 文書クラス article.cls を使用するようになります（既定ではレイアウトと同じ名称のものを使用します)。上記に現れる「article」という文字列は、文書▷設定ダイアログのテキストクラスの説明に使用されます。

節見出し表示に変更を加えた、article.cls 文書クラスを使用するテキストクラスを自分で書いたものとしましょう。これを myarticle.layout というファイルに置いたとすると、このファイルのヘッダは以下のようになります。

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

これは、LaTeX 文書クラス article.cls に関連づけられ、「article (with my own headings)」と表示される、myarticle テキストクラスを宣言するものです。もしこのテキストクラスが複数のパッケージに依存するならば、以下のように宣言すると良いでしょう。

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

これは、このテキストクラスが foo.sty パッケージを使用することを示しています。最後に、DocBook コード向けのクラスを宣言することもできることを見ます。典型的な宣言は以下のようになります。

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

これらの宣言には、文書クラス名を宣言する非必須パラメータ（ただしリストではない）を与えることができることに注意してください。

できる限り明示的に要約すると、レイアウト宣言は以下の形をとります。

```
#  \DeclareLaTeXClass[クラス, パッケージ名.sty]{レイアウト
の説明}
```

ここで「クラス」は、LaTeX クラスファイル名とレイアウトファイル名が異なるときのみ、指定する必要があります。クラスファイル名が指定されなければ、LyX は単純に、クラスファイル名がレイアウトファイル名と同じであると仮定します。

テキストクラスがあなたの嗜好に合うように修正できたならば、他にしなくてはならないことは、それを LyXDir/layouts/か UserDir/layouts にコピーし、ツール▷再初期設定を実行し、LyX を終了して再起動するだけです。そうすれば、この新しいテキストクラスが、他のテキストクラスと同様に使用できるようになります。

レイアウトファイルが導入されたならば、これを編集して、再初期設定したり LyX を再起動したりすることなく、その変更を確認することができます[8]。LyX 関数 layout-reload を使用すれば、現在使っているレイアウトの再読み込みを強制することができるのです。この関数への既定のキー割り当てはありません—もちろん自分でどれかのキーに割り当てることもできますが—。しかし、通常は、この関数を使用する場合は、これをミニバッファに入力します。

[8]第 1.6 版よりも前の LyX では、これを行うことはできませんでした。その結果、レイアウトファイルに加えた変更を反映させるには、その度に LyX を再起動しなくてはならなかったので、レイアウトファイルを編集する作業は、たいへん時間を浪費する作業だったのです。

注意：layout-reload はかなり「高度な機能」です。この機能を利用する前に、作業中の文書を保存しておくことを**強く**勧めます。もっと言えば、大事な文書の作業をしているときに、同時にレイアウト情報の編集をしようとしないことを**強く**勧めます。テスト用文書を使用してください。レイアウトファイル中の文法エラー等が奇妙な挙動を引き起こす可能性があります。特に、そのようなエラーが起こると、LyX は現在のレイアウトが無効であるものと判断して、別のレイアウトに切り替えようとする可能性があります[9]。LyX 開発陣は、このような状況下でも安定性を保つよう努力していますが、後悔よりも安心の方が良いでしょう[10]。

5.3.2 モジュール宣言

モジュールは、以下のような行で始まらなくてはなりません。

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

波括弧内に入っている必須引数はモジュール名で、これは文書▷設定内に表示されます。角括弧内の引数は非必須です。これは、モジュールが依存する LaTeX パッケージをすべて宣言します。角括弧の中には、LyX にとって既知のパッケージだけしか、列挙することはできないことに注意してください。[11]LyX は、任意のパッケージをチェックすることはしません。また、非必須引数として、変換元->変換先の形を使用することができます。これは、変換元書式から変換先書式への変換鎖が存在するときのみ、このモジュールを使用できることを宣言するものです。

それから、以下のようなモジュール宣言を行います。

```
#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
```

[9]非常に悪質な文法エラーの場合には、LyX が終了してしまうことさえあります。これは、ある種のエラーでは、LyX がレイアウト情報を全く読めなくなる可能性があるからです。ご注意ください。

[10]重ねての助言ですが、つねにバックアップを取ってください。それから、お母さんのお片づけに注意。

[11]既知のパッケージのリストは、ソースコード中でしか、ドキュメント化されていません。

```
#Requires: somemodule | othermodule
#Excludes: badmodule
```

ここで説明（Description）は、文書▷設定でこのモジュールが何をするものか、ユーザに情報を与えるために使用されます。Requires 行は、このモジュールが共に使用する必要がある、他のモジュールを特定するのに用いられます。一方、Excludes 行は、このモジュールが共に使用してはならない、他のモジュールを特定するのに用いられます。この 2 つの行は必須ではなく、上記のようにモジュールが複数ある場合には、パイプ記号「|」で区切らなくてはなりません。Requires に指定されたモジュールは、選言的に取り扱われることに注意してください。つまり、Requires に指定されたモジュールのうち、**少なくとも一つ**が使用されていればよいということです。同様に、Excludes に指定されたモジュールは、一つも使用されてはなりません。ここでモジュールは、.module 拡張子を除いた**ファイル名**で認識されることに注意してください。つまり somemodule とは、実のところ somemodule.module のことです。

5.3.3 ファイル書式

レイアウトファイルやインクルードされたファイル、またはモジュールの最初の非コメント行には、以下のように、かならずファイル書式番号が記されていなくてはなりません。

Format [整数型] このレイアウトファイルの書式

このタグは LyX 1.4.0 で導入されました。LyX 1.3.x 以前のレイアウトファイルには、明示されたファイル書式がないため、書式 1 と解されます。LyX 現行版のファイル書式は、書式 21 です。しかし、LyX の各版は、旧版の LyX で作成されたファイルを読むことができるように、旧版のレイアウトファイルも読むことができます。しかしながら、以前の書式に変換する方法はありません。したがって、LyX 1.6.x は、書式 11 以前のファイルのみを読むことができますが、書式 21 は読むことができません。

5.3.4 汎用テキストクラスパラメータ

以下は、文書クラス全体の挙動を決定する汎用パラメータです（これは、.layout ファイルのみに使用されるべきで、モジュールでは使っては

ならない、ということを意味するものでは**ありません**。モジュールには、すべてのレイアウトタグを使用することができます)。

AddToHTMLPreamble この文書クラスが XHTML に出力されるときに、<head>ブロックに追加出力される情報です。典型的には、これは CSS スタイル情報を出力するのに用いられますが、<head>に出力するものであれば、何でも使用することができます。「EndPreamble」で閉じる必要があります。

AddToPreamble 文書プリアンブルに書き加えられる情報です。「EndPreamble」で閉じる必要があります。

CiteFormat 書誌情報の表示に使う書式を定義します。詳細については、第 5.3.12節をご覧ください。「End」で閉じる必要があります。

ClassOptions 文書クラスがサポートする様々な大域オプションを記します。説明は、第 5.3.5節を参照してください。「End」で閉じる必要があります。

Columns [*1*, 2] 文書クラスが既定で 1 段組か 2 段組かを指定します。文書▷設定ダイアログで変更することができます。

Counter [文字列] この部分はカウンタの特性を定義します。カウンタがまだ存在していなければ、生成されます。もし存在しなければ修正されます。「End」で閉じる必要があります。
カウンタについての詳細は、第 5.3.10節を参照してください。

DefaultFont 文書を表示するのに用いられる既定フォントを設定します。フォントの宣言の仕方については、第 5.3.11節を参照してください。「EndFont」で閉じる必要があります。

DefaultModule [文字列] この文書クラスに、既定で取り込むモジュールを指定します。モジュールは、.module 拡張子を除いたファイル名で指定します。ユーザはこのモジュールを除外することができますが、当初は有効の状態になっています（これは新しいファイルが作成されたときや、既存の文書にこの文書クラスが選択したときのみ該当します)。

DefaultStyle [文字列] これは新規段落に割り当てられる様式であり、通常は標準です。もしこれを指定しなければ、最初に定義される様式

がこれに割り当てられるようにはなっていますが、このディレクティブを使用することが推奨されます。

`ExcludesModule` [文字列] このタグは、指定されたモジュール—.module 拡張子を除いたファイル名で指定します—が、この文書クラスでは使用できないように設定します。これはたとえば、特定の学術誌用レイアウトファイルの中で、定理番号を節毎に振る theorems-sec モジュールが使用されるのを防ぐために用いたりすることができます。このタグは、モジュール内で**使用してはいけません**。モジュールは、他のモジュールを排除する独自の枠組みがあります（第5.2.1節参照）。

`Float` フロートを新規に定義します。詳細は、第5.3.8節を参照してください。「End」で閉じる必要があります。

`HTMLPreamble` この文書クラスがXHTMLに出力されるときに、<head>ブロックに出力される情報です。これより前に出現したHTMLPreambleやAddToHTMLPreamble宣言は、すべて完全に上書きされることに注意してください。（プリアンブルに何かを追加したい時には、AddToHTMLPreambleを使用してください。）「EndPreamble」で閉じる必要があります。

`HTMLTOCSection` [文字列] 文書がHTMLに出力されるときに、目次や書誌情報などに使用されるレイアウトです。articleの場合には、これは通常「節」であり、bookの場合は「章」です。これを指定しない場合には、LyXはどのレイアウトを使用するべきか、解析しようと試みます。

`IfCounter` [文字列] 与えられたカウンタの特性を修正します。カウンタが存在しない場合には、この節は無視されます。「End」で閉じる必要があります。
カウンタについての詳細は、第5.3.10節をご覧ください。

`IfStyle` [文字列] 与えられた段落様式の特性を修正します。様式が存在しない場合には、この節は無視されます。「End」で閉じる必要があります。

`Input` 名称の指し示すように、このコマンドは、同じコマンドを何度も指定せずに済むように、別のレイアウト定義ファイルを取り込ませ

ます。よく使われる例は、基本的なレイアウトのほとんどを収録している stdclass.inc のような標準レイアウトファイルです。

InsetLayout このセクションは、差込枠のレイアウトを定義（再定義）します。これは、既存の差込枠にも、新しい文字様式のような新規のユーザ定義差込枠にも使用することができます。「End」で閉じる必要があります。
詳しい情報は、第 5.3.9節をご覧ください。

LeftMargin 画面上の左余白の幅を指示する文字列。例：「MMMMM」。（これは、「2ex」のような「長さ」ではないことに注意してください。）

NoFloat このコマンドは既存のフロートを削除します。これは特に、Input で取り込んだファイルに定義されていたフロートを抑制するのに便利です。

NoStyle このコマンドは既存の様式を削除します。これは特に、Input で取り込んだファイルに定義されていた様式を抑制するのに便利です。

OutputFormat このクラスによって生成されるファイル書式を示す文字列（設定ダイアログで定義される形のもの）。おもに、OutputType が「literate」になっていて、新しい型の literate 文書を定義したい時に便利です。対応する OutputType パラメーターに遭遇したときには、この文字列は、「docbook」・「latex」・「literate」のいずれかにリセットされます。

OutputType このクラスを使用する文書がどのような種類の出力をするかを示す文字列。現在のところ、取り得る値は「docbook」・「latex」・「literate」です。

PageStyle [*plain*, empty, headings] 既定ページ様式。文書▷設定ダイアログで変更することができます。

Preamble LaTeX 文書のプリアンブルを設定します。前に行った Preamble 宣言や AddToPreamble 宣言は、すべて上書きされてしまうので注意してください。（プリアンブルに何かを追加したい時には、AddToPreamble を使用してください。）「EndPreamble」で閉じる必要があります。

Provides [文字列] [*0*, 1] このクラスが文字列で示される機能を既に提供しているかどうかを示します。機能は、一般的にパッケージ名

（amsmath・makeidx・...）やマクロ名（url・boldsymbol・...）です。サポートされている機能の全覧は、LyX ソースコード以外には残念ながら文書化されていませんが、興味があれば LaTeXFeatures.cpp をご覧ください。ヘルプ▷LaTeX の設定もサポートされているパッケージの概要を提供します。

ProvidesModule [文字列] このレイアウトが文字列で表されているモジュールの機能を提供することを示し、.module 拡張子を除いたファイル名で指定します。DefaultModule タグを使用すると、モジュールを使用しなくてはならないことを示しますが、このタグは主に、このレイアウトがモジュールを直接取り込んでしまっていることを示すのに用いられます。同じ機能を別に実装しているモジュール中で使用するなどすることもできます。

Requires [文字列] このクラスが文字列で表されている機能を要求することを示します。機能が複数ある場合には、コンマで区切らなくてはなりません。サポートされている機能以外は要求できないことに注意してください。（機能一覧については、これも LaTeXFeatures.cpp をご覧ください。）

RightMargin 画面上の右余白の幅を指示する文字列。例：「MMMMM」。

SecNumDepth どの節区切りまで連番を振るかを指定します。LaTeX における secnumdepth カウンタに対応します。

Sides [*1*, 2] クラスの既定値として、用紙の片面に印字するか両面に印字するかを指定します。文書▷設定ダイアログで変更することができます。

Style この部分は段落様式を定義します。様式がまだ存在していなければ、生成されます。既に存在していれば、そのパラメータが修正されます。「End」で閉じる必要があります。
段落様式に関する詳細は、第 5.4.1節をご覧ください。

TitleLatexName [文字列="maketitle"] TitleLatexType で使用すべきコマンド名あるいは環境名。The name of the command or environment to be used with TitleLatexType.

TitleLatexType [*CommandAfter*, Environment] 文書のタイトルを定義するのに、どのようなマークアップを使用するのかを示します。CommandAfter は、「InTitle 1」が指定されている最後のレイアウトの後に、TitleLatexName で指定したマクロ名を挿入することを意味します。Environment は、「InTitle 1」を持つ段落群を TitleLatexName で指定した環境でくるむ場合に対応します。

TocDepth どの節区切りまで目次に取り込むかを指定します。LaTeX の tocdepth カウンタに対応します。

5.3.5 ClassOptions 部

ClassOptions 部は、以下の項目を取り得ます。

FontSize [文字列="10|11|12"] 文書のメインフォントが使用できるフォント寸法の一覧です。「|」で区切ります。

Header XML ベースの出力クラスで、DTD 行を設定するのに使用されます。例：PUBLIC "-//OASIS//DTD DocBook V4.2//EN"

PageStyle [文字列="empty|plain|headings|fancy"] 使用できるページ様式の一覧です。「|」で区切ります。

Other [文字列=""] \documentclass コマンドの非必須パラメータとして付け加える文書クラスオプションです。コンマで区切ります。

ClassOptions 部は「End」で閉じる必要があります。

5.3.6 段落様式

段落様式の記述は、以下のようになります[12]。

```
Style 名称
  ...
End
```

[12] これは新しいレイアウトを定義するか、既存のレイアウトを修正することになることに注意してください。

ここでは、以下のコマンドを使用することができます。

Align [*block*, left, right, center] 段落の揃え。

AlignPossible [*block*, left, right, center] 使用できる揃えのコンマ区切りリスト（LaTeX スタイルには、意味を成さない一部の揃えが禁じられているものがあります。たとえば、連番箇条書きを右揃えや中央揃えにすることはできません）。

BabelPreamble これは、前に現れたこの様式の BabelPreamble 宣言をすべて、完全に上書きしますので注意してください。「EndBabelPreamble」で閉じる必要があります。これの利用法についての詳細は、第5.3.7節をご覧ください。

BottomSep [浮動小数点型=0][13] このレイアウト型の段落塊の最後の段落と、次の段落とを分離する垂直空白。次の段落が別のレイアウト型である場合、分離幅は足し上げられるのではなく、最大値がとられます。

Category [文字列] この様式のカテゴリです。これは、ツールバーのレイアウト・コンボボックスで関連した様式をグループ化するのに用いられます。任意の文字列を使用することができますが、作成した様式に既存のカテゴリを使用したいと思うことが多いでしょう。

CommandDepth XML コマンドの深度。XML 型の書式でのみ使用されます。

CopyStyle [文字列] 既存の様式から、すべての機能を現在の様式にコピーします。

DependsOn この**前に**プリアンブルを出力させる様式名。マクロ定義がお互いに依存関係にある場合に、プリアンブルの断片の順序を確実にするためのものです[14]。

EndLabeltype [*No_Label*, Box, Filled_Box, Static] 段落の最後（あるいはLatexTypeが、Environment・Item_Environment・List_Environment

[13] ここで「浮動小数点型」とは 1.5 のような実数を指します。

[14] この機能以外には、プリアンブルの順序を確定する方法はないことに注意してください。LyX の特定のバージョンで観察された順序は、将来のバージョンで警告なしに変わる可能性があります。

のいずれかの場合は、段落群の最後）に置くラベル。No_Label の場合は「何もない」ことを指し、Box（あるいは Filled_Box）の場合は、証明終了マーカ用の白い箱型（あるいは黒い箱型）を指し、Static は明示したテキスト文字列を指します。

EndLabelString [文字列=""] Static 型 EndLabelType のラベルで用いる文字列。

Fill_Bottom [*0*,1] Fill_Top と同様。

Fill_Top [*0*,1] このパラメータは、この様式で段落を初期化する際に設定される、編集▷段落設定ダイアログの「上部垂直余白」リストの Fill 値を設定します[15] 。

Font 本文テキストとラベルの**両方で**用いられるフォント。第 5.3.11節を参照。このフォントを定義すると、自動的に LabelFont も同じ値で定義されることに注意してください。したがって、LabelFont も同時に定義したい場合には、これを先に定義してください。

FreeSpacing [*0*, 1] LyX は、空白をそれ自体文字や記号ではなく、2 つの単語の間の分割子として捉えているため、単語間に 2 つ以上の空白を入れることは、通常許可していません。これ自体はとても素晴らしいことですが、たとえばプログラムコードや生の LaTeX コードを入力しようとするときなどに、煩わしくなることがあります。このことから、FreeSpacing を有効にすることが認められています。LyX は、LaTeX モード以外では 2 つめ以降の空白には、保護された空白を使用することに注意してください。

HTML* これらのタグは、XHTML 出力で使用されます。第 5.4.1節をご覧ください。

InnerTag [[FIXME]] （XML 型書式でのみ使用されます。）

InTitle [1 , 0] 1 の場合、このレイアウトをタイトルブロックの一部としてマークします（大域項目の TitleLatexType と TitleLatexName も参照）。

[15] *Jean-Marc* **による註**：この設定にどれほど使途があるかは定かでなく、おそらく将来のバージョンで取り除かれます。

ItemSep [浮動小数点型=0] これは、同じレイアウトを持つ段落群の間に追加する空白を与えるものです。複数のレイアウトを一つの環境に入れると、それぞれのレイアウトは、その環境の Parsep だけ分離されます。しかし、その環境の項目全体は、さらにこの Itemsep 分だけ離されます。これは**乗数**であることに注意してください。

ItemTag [[FIXME]] （XML 型書式でのみ使用されます。）

KeepEmpty [*0*, 1] 段落を空のままにすると、LaTeX 出力が空になってしまうので、通常、LyX は段落を空にすることを許可しません。しかしながら、これを無効にすることが望ましい場合が存在します。たとえば、書簡のひな型では、必須フィールドを人々が忘れないように、空のフィールドのまま提供する手もあります。特別なクラスにおいては、レイアウトを実際には文章を含まないある種の改行として使用することもあります。

LabelBottomsep [浮動小数点型=0] ラベルと本文テキストとの間の垂直余白。本文テキストの上に来るラベルにのみ使用されます（Top_Environment および Centered_Top_Environment）。

LabelCounter [文字列=""]
自動連番に使われるカウンタ名（詳しくは第 5.3.10節参照）。
LabelType が Counter 型のときには、本項目は**必須**です。この場合には、この様式が出現するたびにカウンタが増えます。
また、LabelType が Enumerate 型のときにも、若干複雑にはなりますが、本項目を使うことができます。たとえば、「LabelCounter myenum」と宣言したものとしましょう。すると、LaTeX におけるのと同様、実際に使われるカウンタは、myenumi・myenumii・myenumii・myenumiv のようになります。これらのカウンタは、全て別々に宣言されなくてはなりません。
カウンタの詳細については、第 5.3.10節をご覧ください。

LabelFont ラベルに使用されるフォント。第 5.3.11節を参照。

LabelIndent ラベルをどれくらい行頭下げすべきかを示す文字列。

Labelsep [文字列=""] ラベルと本文テキストの間の水平余白。本文テキストの上に来ないラベルにのみ使用されます。

LabelString [文字列=""] Static ラベル型でラベルに使用する文字列。LabelCounter を設定している場合、第 5.3.10節に述べられている特別な整形コマンドを含めることができます[16]。

LabelStringAppendix [文字列=""] これは付録の中で LabelString の代わりに用いられます。各LabelString ステートメントは、LabelStringAppendix をもリセットすることに注意してください。

LabelTag [FIXME] （XML 型書式でのみ使用されます。）

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

- Manual は、ラベルが最初の単語（最初の本当の空白まで）であることを示します。ラベルに 2 単語以上使用したいときは、保護された空白を使用してください。
- Static は、ラベルが LabelString で宣言したものであることを示します。これは「静的」であることに注意してください。
- Top_Environment と Centered_Top_Environment は、Static の特別な場合です。ラベルは段落の上に印字されるのですが、それは単一の環境の上か、このレイアウトを持つ連続した段落群の上だけに印字されます。たとえば、Abstract レイアウトが使用したりします。
- Sensitive はキャプションラベルの「図」や「表」の特別な場合です。Sensitive は、（ハードコードされた）ラベル文字列がフロートの種類に依存することを示します。これは、フロートに関連付けられたカウンタの値が N であるものとすると、「FloatType N」にハードコードされています。
- Counter ラベル型は、自動的に連番が付されるラベルを定義します。LabelString は、それが含むカウンタ参照をすべて解決するために展開されます。これは、たとえばSection \thechapter.\thesection のようになります。第 5.3.10節を参照してください。

[16]後方互換性のために、@*style-name*@という文字列は、*style-name* 様式の LabelString を展開したものに置換されます。この機能は既に廃されたものとなっているので、第 5.3.10節の機構に置き換えられなくてはなりません。

- Enumerate は、通常の連番ラベルを生成します。現在のところ、これは、第 4 階層まで順に、アラビア数字・小文字・小文字ローマ数字・大文字を使用するようにハードコードされています。
- Itemize は、各階層でさまざまなブリットを生成します。これもハードコードされています。
- Bibliography は、LyX で内部的に使用され、LatexType BibEnvironment とともにのみ使用されます。

LangPreamble これは、この様式で既に出現した LangPreamble 宣言をすべて、完全に上書きしますので、注意してください。使用法についての詳細は、第 5.3.7節をご覧ください。

LatexName 対応する LaTeX の名称です。環境名かコマンド名を指します。

LatexParam 対応する LatexName の非必須パラメータです。このパラメータは、LyX 内部から変更することはできません。

LatexType [*Paragraph*, Command, Environment, Item_Environment, List_Environmen, Bib_Environment] レイアウトがどのように LaTeX に変換されるべきかを示します[17]。

- Paragraph は、何も特別なことは意味しません。
- Command は、*LatexName*{...}を意味します。
- Environment は、\begin{*LatexName*}...\end{*LatexName*}を意味します。
- Item_Environment は Environment と同じですが、\item がこの環境のすべての段落に付けられるところだけが異なっています。
- List_Environment は Item_Environment と同じですが、LabelWidthString が環境の引数として渡されるところだけが異なっています。LabelWidthString は、編集▷段落設定ダイアログで定義することができます。

[17] これらのルールは SGML クラスにも適用されるので、LatexType の名称は、少しミスリーディングかもしれません。特定の例については、SGML クラスファイルを見てください。

上記最後のいくつかをまとめると、LaTeX 出力は、LaTeX 型に依存して

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

のようになるか、

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

となります。

LeftMargin [文字列=""] レイアウトを環境の中に入れた場合、左余白は単純に加えられるのではなく、因子 $\frac{4}{depth+4}$ をかけて加えられます。このパラメータは、Margin が Manual あるいは Dynamic に設定されているときにも用いられることに注意してください。その場合には、これは手動設定余白または動的設定余白に加えられることに注意してください。
引数は文字列として渡されます。たとえば「MM」と指定すると、段落を通常フォントの「MM」の幅だけ行頭下げを行います。文字列の前に「-」を付けると、負の幅を与えることができます。この方法が採用されたのは、どの画面フォントでも見かけが同じになるようにするためです。

Margin [*Static*, Manual, Dynamic, First_Dynamic, Right_Address_Box]
このレイアウトの左余白の種類です。Static は固定余白を示します。Manual は、左余白が編集▷段落設定ダイアログで入力した文字列によって決められることを示します。これは、タブを用いずに整った一覧表を組むのに使用されます。Dynamic は、余白がラベルの大きさに依存することを示します。これは、自動連番の見出しに使用されます。「5.4.3.2.1 非常に長い見出し」という見出し行が、「3.2 非常に長い見出し」よりも広い左余白（5.4.3.2.1 足す空白と同じ幅）を必要とすることは明らかでしょう（標準的「ワープロ」はこんなことはやってくれませんが）。First_Dynamic は似ていますが、段落の最初の行だけが Dynamic でその他の行は Static です。これは、たとえば、箇条書き（記述）に使用されます。Right_Address_Box は、段落中、最も長い行が右余白に合うように余白を選択します。これは、ページの右端に住所を組版するのに用いられます。

NeedProtect [*0*,1] このレイアウト中の脆弱なコマンドが\protect されるべきか否か（註：これはこのコマンド自体が\protect されるべきかでは**ありません**）。

Newline [0, *1*] 新規行を LaTeX の新規行（\\）に変換するか否か。LyX 中で LaTeX 編集をやりやすくするために、変換は無効にすることができます。

NextNoIndent [1, *0*] 次の段落に最初の行の行頭下げを許すか否か。1 は許可しないことを意味し、0 は望むならば行頭下げできることを意味します。

ObsoletedBy このレイアウトが置き換えられたレイアウト名。これは、後方互換性を維持しながら、レイアウトの名称を変更するのに使用されます。

OptionalArgs [整数型=0] このレイアウトに使用することのできる非必須引数の数。節見出しのようなものに有用であり、LaTeX でのみ意味を持ちます。出力時には、非必須引数は、すべての必須引数の前に来ることに注意してください（以下参照）。つまり、

```
\mycmd[非必須引数 1]{必須引数 1}{段落の内容}
```

のようなコマンドは生成できますが、

```
\mycmd[非必須引数 1]{必須引数 1}[非必須引数 2]{段落の内容}
```

のようなコマンドは、TeX コード（これを使えば何でもできます）を使用することなしには、生成することができません。

ParbreakIsNewline [*0*, 1] LaTeX 出力中で、段落を空行ではなく、改行で区切るよう指定します。PassThru 1 と併用すれば、（TeX コードを使用したときのように）テキストエディタをエミュレートすることができます。

ParIndent [文字列=""] 段落の最初の行の行頭下げ。レイアウトによっては Parindent は固定されています。例外には標準レイアウトがあり、標準レイアウトの段落の行頭下げは、NextNoIndent で禁止す

ることができるようになっています。また、環境中の標準レイアウト段落は、当該段落の Parindent ではなく、この環境の Parindent を使用します。たとえば、箇条書き（連番）内の標準段落は、行頭下げされません。

Parsep [浮動小数点型=0] このレイアウトの 2 段落間の垂直余白。

Parskip [浮動小数点型=0] LyX では、文書を組版するのに、ユーザが「行頭下げ」か「スキップ」を選ぶことができます。「行頭下げ」を選択した際には、この値は完全に無視されます。「スキップ」を選択した際には、LaTeX 型「段落」レイアウトの ParIndent は無視され、すべての段落はこの Parskip 引数分だけ引き離されます。垂直余白は、DefaultHeight を標準フォントでの 1 行の高さとすると、Parskip の値 ×DefaultHeight によって計算されます。このようにして、画面フォントを変更しても同じように表示されるのです。

PassThru [*0*, 1] この段落の内容が、LaTeX が必要とするような特別な変換を行わずに、生の形で出力されるべきかどうか。

Preamble この様式が使用されたときに、LaTeX プリアンブルに付け加えるべき情報。この特定の様式が要求するマクロを定義したり、パッケージを読み込んだりと言ったことに使用します。「EndPreamble」で閉じる必要があります。

RefPrefix [文字列] この型の段落を参照する際、生成されるラベルに使用する前置句。これによって、整形参照を使用することができるようになります。

RequiredArgs [整数型=0] このレイアウトに対応する LaTeX コマンドや LaTeX 環境が予期している、必須引数の数。コマンドの場合には、段落の内容自体に関連付けられている引数**以外に**、必要な引数の数のことです。これらの引数は、実際には渡されなくても構いません。必要な場合は、LyX が空の引数を生成します。非必須引数は、必須引数の前に出力されることに注意してください。詳細については、上記 OptionalArgs タグにおける議論をご覧ください。

Requires [文字列] このレイアウトが文字列で表される機能を必要するか否か。「機能」に関する情報は、上記 Provides の説明（43ページ）をご参照ください。

RightMargin [文字列=""] LeftMargin に同様。

Spacing [*single*, onehalf, double, other **値**] これはレイアウト中の既定の行間をどうすべきか定義するものです。引数の single・onehalf・double は、それぞれ乗数1・1.25・1.667に対応します。引数 other を指定した場合には、実際の乗数値も引数として指定しなくてはなりません。他のパラメータと違って Spacing は、setspace.sty パッケージを使用した、限定的な LaTeX コードを生成することを意味することにご注意ください。

Spellcheck [0,1] この様式の段落をスペルチェックするか否か。既定値は真です。

TextFont 本文に使うフォント。第5.3.11節参照。

TocLevel [整数型] 目次中でのこの様式の階層。これは、節見出しの自動連番に使用されます。

TopSep [浮動小数点型=0] このレイアウトを持つ一連の段落群の最初の段落と、その前の段落の間の垂直余白。前の段落が別のレイアウトを持っていれば、余白は単純に追加されるのではなく、それらの最大値がとられます。

5.3.7 段落様式の国際化

LyX は、長きにわたってレイアウト情報の国際化をサポートしてきましたが、第2.0版までは、これは操作画面にのみ適用されるものであって、たとえばPDF出力には適用されませんでした。たとえば、フランスの著者が、「Theorem 1」の代わりに「 Th 駮 r 鑪 e 1」としたければ、醜いハックに頼るしかありませんでした。Georg Baum のおかげで、これは解消されました。

もし Style が、組版文書に出力される文字列を定義するのであれば、非英語文書や複数言語文書をサポートするために、LangPreamble や Babel-Preamble を使用することができます。以下の抜粋（theorems-ams.inc より）は、これがどう動作するかを示すものです。

```
Preamble
```

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

原則として、LangPreamble と BabelPreamble タグ内には、有効な LaTeX コードはすべて用いることができますが、実際においては、ここで典型的に示したような形になるでしょう。組版文字列が正しく翻訳されるための鍵となるのは、LaTeX コマンド\claimname とその\newtheorem 中での用い方です。

LangPreamble タグは、文書全体の言語に基づいた国際化を提供します。タグの内容は、Preamble タグと同様、プリアンブルに置かれるのですが、これを特別なものにしているのは、「関数」_() が使用されていることです。これは、LyX が LaTeX 出力を生成する際、その引数を文書言語に翻訳したもので置き換えられます。

BabelPreamble タグは、複数言語文書をサポートし、babel パッケージへのインタフェースを提供することを意図しているので、もう少し複雑です。その内容は、文書に現れる言語それぞれについて一度、プリアンブルに追加されます。この場合には、_() の引数は、その当該言語への翻訳で置き換えられ、$$lang は言語名（babel パッケージで使用されるもの）で置き換えられます。

したがって、フランス語のセクションを持つドイツ語文書では、以下のような内容がプリアンブルに追加されます

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

それから、LaTeX と babel は協力して、出力に正しい文字列を生成します。

ここで注意しなくてはならない重要な点は、翻訳は、操作画面の国際化に使われるのと同じ機構を通じて、LyX 自身によって提供されるということです。つまり、ユーザー作成のレイアウトファイルに入力された文字列は、LyX の国際化ルーチンでは取り扱われないので、LangPreamble

と BabelPreamble は、事実上、LyX とともに提供されるレイアウトファイルでのみ、使うことができるということを意味します。とはいえ、こういうことでありますので、将来的に LyX に同梱させようという意図を以て作成されたレイアウトは、適切なところではすべて、これらのタグを使用するべきです。

5.3.8　フロート

LyX 第 1.3.0 版以来、テキストクラス自体の中でフロート（figure・table・...）を定義することが可能となり、かつ必要となりました。標準的なフロートは `stdfloats.inc` ファイルに含まれているので、作業中のレイアウトファイルに

```
Input stdfloats.inc
```

と加えるだけで済むことも多いでしょう。LyX に同梱されている AGU クラスのように、それ以外のフロート型を提供するテキストクラスを実装するには、以下の情報が役立つでありましょう。

`Extension` [文字列=""] 図などのリストを含む外部ファイルのファイル拡張子名。LaTeX がキャプションを書き込むファイルです。

`GuiName` [文字列=""] メニューとキャプションに使用される文字列。babel が使用される場合には、これは現在の言語に翻訳されます。

`HTML*` これらは、XHTML 出力で使用されます。第 5.4節をご覧ください。

`ListCommand` [文字列=""] この型のフロートの一覧を生成するのに使用するコマンド。頭部の「\」は書きません。`NeedsFloatPkg` が偽の時には、このコマンドを生成する標準的な方法はないので、これは**必ず**指定しなくてはなりません。`NeedsFloatPkg` が真の時は、標準的な方法が存在するので、これは無視されます。

`ListName` [文字列=""] この種類のフロート一覧（図一覧・表一覧など）に使用される見出し。LyX 中では、これは画面上のラベルとして使用されます。また、見出しとして使用するために、LaTeX に渡され、XHTML 出力でも見出しとして使用されます。これは、文書言語に翻訳されます。

NeedsFloatPkg [0,*1*] フロートが文書クラス中に既に定義されているか、あるいは代わりにfloat.styを読み込んで、それが提供しているものを使用する必要があるかを示します。既定値は1であり、float.styを使用します。LaTeX 文書クラスでフロートが既に定義されているときには、0に設定しなくてはなりません。

NumberWithin [文字列=""] この（非必須の）引数は、このクラスのフロートが文書中のある節単位ごとに番号を振り直されるべきかどうかを規定します。たとえば、引数にchapterと指定されていれば、フロートは章ごとに番号が振り直されます。

Placement [文字列=""] このクラスのフロートの既定の配置法。文字列は、標準的なLaTeX 表記に従い、tならば上部（top）、bならば下部（bottom）、pならばページ（page）、hならばここ（here）を表します[18]。これらの他に新しい型Hがあり、これはフロートを「ここ」に置いていいけれども他の場所はだめ、というものなので、本当はフロートにあるものではありません。しかし、H指定子は特別なものであり、その細かい実装上の理由で、組み込み以外のフロート型では使用することができません。これが何を意味するかおわかりにならない場合には、代わりに「tbp」を指定してください。

RefPrefix [文字列] この型のフロートを参照する際、生成されるラベルに使用する前置句。これによって、整形参照を使用することができるようになります。コピーした様式が設定したRefPrefixは、特別な文字列「OFF」（すべて大文字）を使えば、いつでも削除することができます。

Style [文字列=""] \newfloatを使用してフロートを定義する際に使用される様式。

Type [文字列=""] プログラムやアルゴリズムのような、フロートの新しいクラス「型」。適切な\newfloatの後で、\begin{program}や\end{algorithm*}といったコマンドが利用できます。

type 型のフロートを定義すると、自動的に**対応する** *type* 名カウンタが定義されます。

[18] LaTeX 同様、文字列中でのこれらの文字の順序は関係ありません。

5.3.9　任意設定差込枠と差込枠レイアウト

LyX は第 1.4.0 版から文字様式をサポートしています。第 1.6.0 版以降、これは任意設定差込枠と呼ばれています。

任意設定差込枠には次の 3 種類があります。

- 文字様式（CharStyle）。これは、\noun や\code などの LaTeX コマンドに対応した意味論的マークアップを定義するものです。
- ユーザ設定（Custom）。これは、TeX コードや脚注などに似たユーザ設定の折りたたみ式差込枠を定義するのに使用することができます。わかりやすい例は endnote 差込枠で、これは endnote モジュール中で定義されています。
- XML 要素（Element）。DocBook クラスで使用するものです。

任意設定差込枠は、以下で説明する InsetLayout タグを使用して定義されます。

InsetLayout タグは、もう一つ別の機能も提供します。これを使えば、いろいろな種類の差込枠全体のレイアウトを設定するのに使用することができるのです。現在のところ、InsetLayout は任意設定差込枠を定義することの他に、脚注・傍注・注釈差込枠・TeX コード（ERT）差込枠・派生枝・リスト・索引・ボックス・表・アルゴリズム・URL・オプション引数のレイアウトパラメータを設定するのに使用することができます。

InsetLayout 定義は以下の形の行では始まらなくてはなりません。

```
InsetLayout <型>
```

ここで<型>は、レイアウトを定義しようとしている差込枠を指し、以下の 2 つの場合がありえます。

1. 既存の差込枠のレイアウトを変更する場合。この場合、<型>は以下のいずれかになります：Algorithm・Branch・Box・Box:shaded・ERT・Figure・Foot・Index・Info・Info:menu・Info:shortcut・Info:shortcuts・Listings・Marginal・Note:Comment・Note:Note・Note:GreyedOut・OptArg・Table・URL。
2. 任意設定差込枠のレイアウトを定義する場合。この場合には、<型>は既存の差込枠で使用されていないもので有効な識別子であれば、

何でも指定できます。任意設定差込枠の定義には、LyXType 項目も**同時に**含まれていて、これがどの型の差込枠なのかが宣言されている必要があります。

InsetLayout 定義には以下の項目を入れることができます。

BgColor 差込枠の背景色。有効な色彩は src/ColorCode.h で定義されています。

ContentAsLabel [*0*,1] 差込枠を閉じた際、差込枠の内容をラベルとして使用するか否か。既定値は偽です。

CopyStyle 段落様式と同様です（5.3.6）ページ参照)。

CustomPars [*0*,1] 段落を設定するのに、段落設定ダイアログをユーザが使えるかどうかを指定します。

Decoration 差込枠の枠とボタンをレンダリングするのに使用する様式を指定するもので、Classic・Minimalistic・Conglomerate のいずれかを指定することができます。脚注は通常 Classic を使用し、TEX コード差込枠は通常 Minimalistic、文字様式は Conglomerate を使用します。

End InsetLayout 宣言を閉じるのに必要です。

Font 本文本体とラベル**両方に**使用されるフォントです。第 5.3.11節を参照。このフォントを定義すると自動的に LabelFont も同じ値に定義されるので、これらを別々の値にしたいときは、これを先に定義してから後に LabelFont を定義しなくてはならないことに注意してください。

ForceLTR たとえば TEX コードや URL で「latex」言語が「左から右」（ラテン式）出力になるように強制します。うまく機能しません。

ForcePlain [*0*,1] PlainLayout を使用するべきなのか、それともユーザが差込枠で使用されている段落様式を変更できるのかを指定します。既定値は偽です。

FreeSpacing 段落様式と同様です（43ページ参照)。既定値は偽です。

HTML* これらは、XHTML 出力で使用されます。第 5.4節をご覧ください。

InToc [*0*,1] 「文書構造」ペイン用に出力される文字列に、この差込枠の内容を含めるかどうか。たとえば、節見出しの脚注の内容が、文書構造の目次に表示されることは望まないでしょうが、通常、文字様式の内容は表示されることを望むでしょう。既定値は偽、すなわち含めません。

KeepEmpty 段落様式と同様です（44ページ参照）。既定値は偽です。

LabelFont ラベルに使用されるフォント。第 5.3.11節を参照。非効率を回避するため、この定義は Font の前には決して現れてはなりません。

LabelString ボタンなどに差込枠のラベルとして表示されるもの。差込枠型によっては（TEX コードや派生枝）、ラベルが動的に変更されます。

LatexName 対応する LATEX 関連物の名称。環境名ないしはコマンド名。

LatexParam 対応する LatexName 関連物の非必須パラメータ。[] のような括弧対を含む。このパラメータは LYX 内部から変更することはできません。

LatexType 段落様式と同様です（46ページ参照）。

LyxType charstyle・custom・element・end（charstyle の定義の終わりなどを示すダミー定義）の各値を取ることができます。この項目は、任意設定差込枠に必須であり、かつ任意設定差込枠でしか意味を持ちません。この項目は、就中、差込枠がどのメニューに表示されるかを決定します。LyxType を charstyle に設定すると、MultiPar が偽に設定されます。文字様式差込枠で MultiPar を真にしたい時には、LyxType を設定した**後に**設定すれば、真に設定することができます。

MultiPar [*0*,1] この差込枠中に複数の段落を入れることができるかどうか。これは同時に、CustomPars を同じ値に設定し、ForcePlain を逆の値に設定します。これらは、MultiPar の**後に**指定されれば、他の値に指定し直すことができます。既定値は真です。

NeedProtect [*0*,1] 本レイアウト中で脆弱なコマンドを\protect するか否か（註：当該コマンド自身を protect するかどうかでは**ありません**）。既定値は偽です。

ParbreakIsNewline [*0*,1] 段落様式と同様です（5.3.6ページ参照）。既定値は偽です。

PassThru [*0*,1] 段落様式と同様です（5.3.6ページ参照）。既定値は偽です。

Preamble 段落様式と同様です（49ページ参照）。

RefPrefix [文字列] この型の差込枠を参照する際、生成されるラベルに使用する前置句。これによって、整形参照を使用することができるようになります。

Requires [文字列] 段落様式と同様です（49ページ参照）。

ResetFont [0, *1*] この差込枠が周囲の環境と同じフォントを用いるか、独自のフォントを用いるか。既定値は真、すなわち独自のフォントを用います。

Spellcheck [0,*1*] この差込枠の内容をスペルチェックするか否か。既定値は真です。

5.3.10 カウンタ

LyX 第 1.3.0 版以来、テキストクラス自体の中でカウンタ（chapter・figure・...）を定義することが可能となり、かつ必要となりました。標準的なフロートは stdcounters.inc ファイルに含まれているので、作業中のレイアウトファイルに

```
Input stdcounters.inc
```

と加えるだけで済むことも多いでしょう。しかし自製カウンタを定義したければ、そうすることもできます。カウンタ宣言は、

```
Counter 名称
```

で始まらなくてはなりません。ここで「名称」はカウンタ名で置き換えます。また、宣言は「End」で終わらなくてはなりません。以下のパラメータを使用することができます。

LabelString [**文字列**=""] 定義されていると、ここで指定した文字列がカウンタの表示の仕方を定義します。この値を指定すると、LabelStringAppendix も同じ値に設定されます。文字列中では、以下の構成要素を使用することができます。

- \thecounter は、カウンタ counter の LabelString（または LabelStringAppendix）を展開したもので置き換えられます。
- カウンタ値は、LaTeX 型マクロ*numbertype*{**カウンタ**}を用いて表現することができます。ここで *numbertype* は以下のいずれかです[19]。arabic：1, 2, 3,...；alph：a, b, c, ...（小文字）；Alph：A, B, C, ...（大文字）；roman：i, ii, iii, ...（小文字ローマ数字）；Roman：I, II, III...（大文字ローマ数字）；hebrew（ヘブライ語数字）。

LabelStringが定義されていないときは、既定値は以下のように組み立てられます。このカウンタに親カウンタ master（Within で定義）があるときには、文字列\themaster.\arabic{カウンタ}が使用されます。それ以外の場合は、\arabic{カウンタ}が使用されます。

LabelStringAppendix [文字列=""] LabelString と同様ですが、付録で使用するためのものです。

PrettyFormat [文字列=””] このカウンタの整形参照で使用する書式。たとえば、節番号への参照を「Section 2.4」のように表示させたい場合には、文字列に「##」を含めます。これは、カウンタ番号で置換されます。したがって、節の場合には「Section ##」のようにします。

Within [文字列=””] これを別のカウンタ名に設定すると、現在のカウンタは、別のカウンタが増加する毎にリセットされます。たとえば、subsection は section 毎に番号がリセットされます。

[19] 実は、事態はもう少し複雑です。以下に説明されているもの以外の *numbertype* は何であれ、アラビア数字を生成します。これが将来変更されたとしても不思議ではないでしょう。

5.3.11 フォント指定

フォント指定は、以下のような形を取ります。

```
Font または LabelFont
 ...
EndFont
```

以下のコマンドを使用することができます。

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

Family [*Roman*, Sans, Typewriter]

Misc [文字列] 有効な引数は、emph・noun・underbar・no_emph・no_noun・no_bar です。それぞれ、対応する属性を有効にしたり無効にしたりします。
たとえば、emph は強調を有効にし、no_emph はそれを無効にします。もし後者がわかりにくければ、現在のコンテクストのフォント設定は、一般的に周囲のコンテクストから継承していることを思い出してください。ですから no_emph は、たとえば定理環境で、何をせずとも有効となっている強調を無効にするのです。

Series [*Medium*, Bold]

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

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

5.3.12 引用書式指定

（引用ダイアログやツールチップなどの）LyX 内部や XHTML 出力において、書誌情報をどのように表示するべきかの叙述には、CiteFormat ブロックが使用されています。このブロックは、以下のような形をしています。

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

上記の各行は、それぞれarticleやbookに関連付けられた書誌情報をどのように表示するべきかを定義するものですが、このような定義は、BibTEXファイル中に存在しうる「項目型」すべてについて与えることができます。特定の定義が与えられなければ、LYXは、ソースコード中に定義されている既定書式を使用します。LYXは、いくつかの書式を`stdciteformats.inc`ファイルで事前定義しており、これはほとんどのLYX文書クラスにインクルードされています。

この定義は、BibTEXキーをその値で置換できる機能を持った、簡単な言語を使用しています。キーは、%author%のように%記号でくくらなくてはなりません。したがって、簡単な定義は以下のようになります。

```
misc %author%, ‘‘%title%’’
```

これは、「著者名・コンマ・引用に囲まれたタイトル・終止符」を出力します。

もちろん、キーが存在するときのみ、キーを出力したい時があるはずです。このようなときには`{%volume%[[vol. %volume%]]}`のように、条件付きの構成を使用することができます。これは、volumeが存在するならば,「vol.」とvolumeキーを出力するという意味です。また、`{%author%[[%author%]][[%editor%, ed.]]}`のように、条件の中にelse節を含めることも可能です。ここでは、もしauthorキーが存在するならば出力され、そうでなければeditorキーと「, ed.」が出力されます。ここでもキーは、%記号でくくられていることに注意してください。条件全体は、波括弧で囲まれています。if節およびelse節は、「[[」と「]]」の二重角括弧で囲まれています。これらすべてのあいだには，空白は入ってはなりません。

もう一つ、定義中で使用することのできる文法として、`{!<i>!}`という形をしたものがあります。これは、「リッチテキスト」を生成するときに使われる整形情報を定義するものです。当然のことながら、平文を書き出すときには、HTMLタグを出力させたくはありませんから、HTMLタグは「{!」と「!}」でくるんでやらなくてはならないのです。

CiteFormatブロックでは、他に2つの特殊な定義が可能です。一つめの例としては、

```
!quotetitle ‘‘%title%’’
```

といった例が挙げられます。これは、短縮形ないしはマクロであり、%!quotetitle%のように、これがキーであるかのように扱って使用することができます。LYX

は、%!quotetitle%を、そこで定義されているものを扱う場合と同じように取り扱います。ですから、明白な警告を敢えてさせて頂くと、

```
!funfun %funfun%
```

のようなことはしないでください。LyX は、無限ループに陥るようなことはありませんが、諦めるまでに長いループに入るかもしれません。

特殊な定義の二つめは、

```
_pptext pp.
```

のようなものです。これは、文字列の翻訳可能な部分を定義するもので、書誌情報中の関連部分が翻訳されるようにすることができます。%_pptext%のように、これをキーとして扱って、定義の中に入れることもできます。これらのうちいくつかは、stdciteformats.inc 中に事前定義されています。これは、上記で述べたような意味でのマクロではないことに注意してください。

以下は、これらの機能を全て使った例です。

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

これは、author キーが定義されているならば、著者とコンマを出力し、author キーが定義されておらず、editor キーが定義されているならば、編集者名の後に_edtext ないしはその翻訳（既定では「ed.」）を出力します。これは実は stdciteformats.inc の中で定義されていますので、このファイルをまず読みこめば、ご自身の定義ないしは再定義の中で使用することができます。

5.4 XHTML 出力のタグ

LaTeX や DocBook と同様、LyX の XHTML 出力の書式も、レイアウト情報によって制御することができます。一般的に、LyX は適切な既定値を提供し、前述したように、他のレイアウトタグに基づいて、既定の CSS スタイルの構成まで行ないます。たとえば、章見出しを適切に整形するための CSS を書き出すために、LyX は、章様式の Font 宣言で提供されている情報を利用しようと試みます。

したがって、多くの場合、使いたい環境や任意設定差込枠などのために満足のいくXHTML出力を得るために、まったく何もしなくてよいことになるでしょう。しかしながら、これが必要になる場合もあるので、LyXは、生成されるXHTMLやCSSをカスタマイズするために使用できるレイアウトタグを、たくさん提供しています。

様式宣言や差込枠宣言の外で使用することができるタグに、HTMLPreambleとAddToHTMLPreambleの2つがあることに注意してください。これらの詳細については、第5.3.4節をご覧ください。

5.4.1 段落様式

LyXが段落のために出力するXHTMLの種類は、通常の段落を取り扱っているのか、コマンドを取り扱っているのか、あるいは環境を取り扱っているのかに依存し、これは対応するLaTeXTypeタグの内容によって決定されます。

コマンドや通常の段落の場合には、XHTML出力は以下の形になります。

```
<tag attr=''value''>
<labeltag attr=''value''>ラベル</labeltag>
段落の内容
</tag>
```

もちろん、段落にラベルがなければ、ラベルタグは省略することができます。

環境のうち、リストの変種でないものに関しては、XHTMLは以下の形を取ります。

```
<tag attr=''value''>
<itemtag attr=''value''><labeltag attr=''value''>環境ラ
ベル</labeltag>最初の段落。
</itemtag>
<itemtag>二つめの段落。</itemtag>
</tag>
```

ラベルは、たとえば定理の場合にそうであるように、最初の段落にだけ出力されることに注意してください。

リストに関しては、次のような形になります。

```
<tag attr=''value''>
<itemtag attr=''value''><labeltag attr=''value''>リスト
のラベル</labeltag>最初の項目。</itemtag>
<itemtag attr=''value''><labeltag attr=''value''>リスト
のラベル</labeltag>二つめの項目。</itemtag>
</tag>
<tag attr=''value''>
<labeltag attr=''value''>リストのラベル
</labeltag><itemtag attr=''value''>最初の項目。
</itemtag>
<labeltag attr=''value''>リストのラベル
</labeltag><itemtag attr=''value''>二つめの項目
</itemtag>
</tag>
```

ここで labeltag と itemtag の順序が違っていることに注意してください。どちらの順序になるかは、HTMLLabelFirst の設定に依存します。もし HTMLLabelFirst が偽であれば（既定値）、最初のケースのようになり、これが真であれば、二番めのケースのように、label が item の外側に来るようになります。

各段落の特定のタグ出力や属性出力は、以下に述べるようなレイアウトタグを使って制御することができます。しかしながら、前述のように、多くの場合、LyX は適切な既定値を生成するので、たいしたことをしなくても、望ましい XHTML 出力を得ることができるということになるはずです。ここで利用出来るタグは、自分の好みにあわせて微調整する目的でここにあるものと考えてください。

HTMLAttr [文字列] 主幹タグと共に出力される属性情報を指定します。たとえば、「class='mydiv'」のようなものです。既定においては、LyX は「class='レイアウト名'」と出力します。ここでレイアウト名は、レイアウトの LyX 名であり、chapter のように小文字で記述します。

HTMLForceCSS [*0*,1] HTMLStyle で追加情報が明示的に与えられているときでも、LyX がこのレイアウト用に生成する既定 CSS 情報を出力するか否か。これを 1 にすると、生成された CSS を完全に上書き

する代わりに、変更したり追加したりすることができます。既定値は0です。

HTMLItem [文字列] 環境の段落に使用されるタグ。上記各例の itemtag を置き換えます。既定値は div です。

HTMLItemAttr [文字列] item タグの属性。既定値は class='レイアウト名_item' です。

HTMLLabel [文字列] 段落と項目ラベルに使用されるタグ。上記各例の labeltag を置き換えます。既定値は span です。

HTMLLabelAttr [文字列] label タグの属性。既定値は class='レイアウト名_label' です。

HTMLLabelFirst [*0*,1] このタグは、リスト関係環境でのみ意味を持ち、label タグが、item タグの前に出力されるか、中に出力されるかを制御します。これは、たとえば、description 環境の中で、'<dt>...</dt><dd>...</dd>という形を得るために使用されます。既定値は0で、label タグは item タグの中に出力されます。

HTMLPreamble この様式が使用されたときに、<head>セクションに出力される情報。これは、たとえば、onclick ハンドラを定義するために<script>ブロックをインクルードするのに使用することができます。

HTMLStyle この様式が使用されたときに、インクルードする CSS スタイル情報。これは、レイアウトが生成する<style>ブロックで自動的に包まれますので、CSS 自体をインクルードするだけで大丈夫です。

HTMLTag [文字列] 主幹ラベルに使用されるタグ。上記各例の tag を置き換えます。既定値は div です。

HTMLTitle [*0*,1] この様式が、XHTML ファイルの<title>タグを生成するのに使用する様式であるという印をつけます。既定値は偽です。stdtitle.inc ファイルでは、title 環境のこの項目を真に設定しています。

5.4.2 差込枠レイアウト XHTML

差込枠の XHTML 出力も、レイアウトファイル内の情報によって制御することができます[20]。ここでも、LyX は適切な既定値を提供しようと試み、既定の CSS 様式を構成します。しかし、すべてカスタマイズ可能です。

LyX が差込枠用に出力する XHTML は、以下の形を取ります。

```
<tag attr=''value''>
<labeltag>ラベル</labeltag>
<innertag attr=''value''>差込枠の内容。</innertag>
</tag>
```

差込枠が多段落を許可している—つまり `MultiPar` が真—ならば、差込枠の内容は、それ自身段落として出力され、それらの段落に用いられる様式（標準、引用など）を用いて整形されます。もちろん、段落にラベルがなければ、label タグは省略され、ラベルがあれば、現在のところ、つねに `span` が用いられます。inner タグは非必須であり、既定では出力されません。各差込枠用に出力される特定のタグや属性は、以下のレイアウトタグによって制御することができます。

`HTMLAttr` [文字列] 主幹タグと共に出力される属性情報を指定します。たとえば、「`class='mydiv'`」のようなものです。既定においては、LyX は「`class='`レイアウト名`'`」と出力します。ここでレイアウト名は、レイアウトの LyX 名であり、chapter のように小文字で記述します。

`HTMLForceCSS` [*0*,1] `HTMLStyle` で追加情報が明示的に与えられているときでも、LyX がこのレイアウト用に生成する既定 CSS 情報を出力するか否か。これを `1` にすると、生成された CSS を完全に上書きする代わりに、変更したり追加したりすることができます。既定値は `0` です。

`HTMLInnerAttr` [文字列] inner タグの属性。既定値は `class='`差込枠名`_inner'` です。

[20] 現在のところ、これは「テキスト」差込枠（中に書き込みができる差込枠）にのみ有効で、「コマンド」差込枠（ダイアログボックスに関連付けられた差込枠）には適用されません。

HTMLInnerTag [文字列] inner タグです。上記各例の innertag を置き換えます。既定値はなしです。

HTMLIsBlock [0,*1*] この差込枠が（脚注のように）独立した文字列ブロックを表すのか、それとも、（派生枝のように）周囲の文字列の中に取り込まれる素材を表すのか。既定値は 1 です。

HTMLLabel [文字列] 場合によっては、カウンタへの参照を含む、この差込枠のラベル。たとえば、脚注用には\arabic{footnote}など。これは非必須であり、既定値はありません。

HTMLPreamble この様式が使用されたときに、<head>セクションに出力される情報。これは、たとえば、onclick ハンドラを定義するために<script>ブロックをインクルードするのに使用することができます。

HTMLStyle この様式が使用されたときに、インクルードする CSS スタイル情報。これは、レイアウトが生成する<style>ブロックで自動的に包まれますので、CSS 自体をインクルードするだけで大丈夫です。

HTMLTag [文字列] 主幹ラベルに使用されるタグ。上記各例の tag を置き換えます。既定値は MultiPar の設定に依存し、MultiPar が真ならば div、偽ならば span です。

5.4.3 フロート XHTML

フロートの XHTML 出力も、レイアウトファイル内の情報によって制御することができます。出力は、以下の形を取ります。

```
<tag attr=''value''>
フロートの内容。
</tag>
```

キャプションは、存在している場合には、独立した差込枠となり、そのような形で出力されます。その外観は、キャプション差込枠の InsetLayout で制御することができます。

`HTMLAttr` [文字列] 主幹タグと共に出力される属性情報を指定します。たとえば、「`class='myfloat' onclick='...'`」のようなものです。既定においては、LyX は「`class='float float-`フロート型`'`」と出力します。ここでフロート型は、フロート宣言で定義された（5.3.8参照）、この型のフロートの LyX 名です。ただし、小文字に変換され、アルファベットや数字でない文字はアンダースコアに変換されます。例：float-table。

`HTMLStyle` このフロートが使用されたときに、インクルードする CSS スタイル情報。これは、レイアウトが生成する`<style>`ブロックで自動的に包まれますので、CSS 自体をインクルードするだけで大丈夫です。

`HTMLTag` [文字列] このフロートに使用されるタグ。上記各例の `tag` を置き換えます。既定値は `div` であり、ほとんどの場合変更する必要はありません。

5.4.4 書誌情報の整形

書誌情報は、`CiteFormat` ブロックを使用して整形することができます。詳細については、第 5.3.12節を参照してください。

5.4.5 LyX が生成した CSS

LyX は、提供されている他のレイアウト情報に基づいて、差込枠と段落様式の両方の既定 CSS 様式ルールを生成ということすることを、これまでに何度か触れました。この節では、LyX がどのレイアウト情報を、どのように使うのか、ひとこと述べておきたいと思います。

LyX は、現在のところ、`Font` 宣言で指定されている `Family`・`Series`・`Shape`・`Size` を利用して、フォント情報についてのみ CSS を自動生成します（第 5.3.11節を参照）。この変換は、きわめて分かりやすく自明です。たとえば、「`Family Sans`」は「`font-family: sans-serif`」になります。LyX の寸法と CSS の寸法のあいだの対応は、少し複雑ですが、それでも直感的に分かります。詳細については、`src/FontInfo.cpp` の `getSizeCSS()` 関数をご覧ください。

第6章　外部素材を取り込む

【警告】本説明書のこの部分は、しばらく更新されていません。もちろんまだ正確であることを期待していますが、保証の限りではありません。

LyX 外部のソースから素材を使用する方法は、取扱説明書**埋込オブジェクト篇**で詳細にカバーされています。本章は、新種の素材を取り込む際に、舞台裏で何をする必要があるかをカバーします。

6.1　どのように機能するのか

外部素材の機能は、ひな型の概念に基づいています。ひな型は、LyX がある型の素材とどのように橋渡しをするべきかを指定するものです。同梱物として、LyX は、Xfig の図や、様々なラスター形式画像、チェス棋譜、LilyPond 楽譜用のひな型を事前に定義されたものとして含んでいます。実際に何が入っているかは、挿入 ▷ ファイル ▷ 外部素材メニューで見ることができます。さらに、特定の型の素材をサポートするのに、自分自身のひな型を作成することも可能です。後でどのようなことをすればいいか詳細に説明しますが、できればあなたが作ったすべてのひな型を投稿して、我々が LyX の後の版に取り込むことができるようにしてくださることを希望します。

外部素材の機能におけるもう一つの基本的な発想は、最終素材の元となるオリジナルファイルと、書き出された文書や印刷された文書に取り込むための生成ファイルとを区別していることです。たとえば、`Xfig` で作成した図の場合を考えてみましょう。Xfig アプリケーション自体は、`.fig` 拡張子を持つオリジナルファイルを操作します。Xfig で図を作成したり変更したりして、作業が終われば `fig` ファイルに保存します。この図をお使いの文書に取り込みたいときには、LaTeX ファイルにそのままインクルードできるように、`transfig` を呼び出して PostScript ファイルを生成します。この場合には、`.fig` ファイルがオリジナルファイルであり、

PostScript ファイルが生成ファイルになります。

この区別は、文書を執筆している最中に、素材を更新することができるようにするために重要です。さらに、これによって、複数の書き出し書式をサポートするために必要な柔軟性が提供されます。たとえば、平文テキストファイルの場合には、図を生の PostScript ファイルとして取り込むのは、とても褒められた発想とはいえません。むしろ、その図への参照だけを含めるか、最終出力が実際の画像に近いものとなるように画像から ASCII への変換子を呼び出したいと考えることでしょう。LyX の外部素材マネジメントは、LyX がサポートする各書き出し書式別に仕分けしているので、ユーザがこれを行うことが可能となっています。

LyX の外部素材マネジメントは、書き出し書式によって異なる生成物をサポートすることの他に、編集・閲覧アプリケーションを緊密に統合することもサポートします。Xfig の図の場合には、LyX の外部素材ダイアログからシングルクリックでオリジナルファイルを Xfig で開くことができ、ダブルクリックすることで生成された PostScript ファイルを Ghostview で閲覧することができます。もうコマンドラインをもてあそんだり、オリジナルファイルや生成ファイルがどこにあるか探したり変更を加えるためにファイルブラウザをいじくり回す必要はないのです。このようにして、文書を執筆する際に必要となる多くのアプリケーションを最大限に利用し、最終的により生産性を上げることができるようになるのです。

6.2　外用ひな型設定ファイル

LyX に自製の外用ひな型を付け加えるのは、比較的簡単です。しかしながら、これを不用心に行ってしまうと、たいていの場合、簡単に濫用されてしまうようなセキュリティホールを作ってしまいがちであることを心に留めておいてください。したがって、これを実行に移す前に、第6.4節のセキュリティに関する議論を読んでおいてください。

このことに言及した上で、あなたが作成した面白いひな型は、ぜひ投稿してください。

外用ひな型は、`LyXDir/lib/external_templates` ファイルで定義されています。自分用の版を `UserDir/external_templates` に置くこともできます。

典型的なひな型は以下のようになります。

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

ご覧の通り、ひな型は Template ... TemplateEnd で閉じられます。ひな型には、一般的な設定を行うヘッダ部と、サポートされている主要な文書ファイル書式の設定を行う Format ... FormatEnd 部があります。

6.2.1　ひな型のヘッダ

AutomaticProduction true|false　このひな型で扱うファイルを LyX が生成しなくてはならないか否か。このコマンドは、一度だけ必ず現れなくてはなりません。

FileFilter <**パターン**>　望むファイル群を表示するために、ファイルダイアログで使用するフィルタ用 glob パターン。2 つ以上のファイル拡張子があり得る場合（たとえば、tgif には .obj と .tgo があります）、「*.{obj,tgo}」の様なパターンを使用してください。このコマンドは、一度だけ必ず現れなくてはなりません。

GuiName <GUI **名**>　この文字列はボタン上に表示されます。このコマンドは、一度だけ必ず現れなくてはなりません。

HelpText <**文章**> HelpTextEnd　外部素材ダイアログで使用されるヘルプ文。このひな型がユーザに何を提供できるのか、ユーザに説明す

るのに十分な情報を盛り込んでください。このコマンドは、一度だけ必ず現れなくてはなりません。

`InputFormat <書式>` オリジナルファイルのファイル書式。これは、LyXが知っている書式名でなくてはなりません（第3.1章参照）。このひな型が、2つ以上の書式のオリジナルファイルを取り扱える場合は、「*」を使用してください。この場合、LyXはファイル書式を推定するために、ファイル自体に詮索を試みます。このコマンドは、一度だけ必ず現れなくてはなりません。

`Template <ID>` このひな型の（他と重複しない）名称。代入マクロを含めてはなりません（下記参照）。

`Transform Rotate|Resize|Clip|Extra` このコマンドは、このひな型がどのような変換をサポートしているかを指定します。これは全く登場しなくても1回以上現れても構いません。このコマンドは、外部素材ダイアログ中の対応するタブを使用可能にします。`Transform` コマンド一つずつに応じて、`Format` 部に、対応する `TransformCommand` コマンドか `TransformOption` コマンドを置かなくてはなりません。これを行わないと、この書式での変換はサポートされません。

6.2.2 `Format` 部

`Format LaTeX|PDFLaTeX|PlainText|DocBook` この書式定義が定める主要な文書ファイル書式。すべてのひな型が、全文書ファイル書式に対して意味のある表示ができるわけではありません。それでも、全書式に対して `Format` 部を定義してください。表示する方法がないときは、ダミーテキストを使用してください。これによって、書き出した文書内で、少なくとも外部素材への参照を見ることができるようになります。

`Option <名称> <値>` このコマンドは、`Product` での代入に使うマクロ$$<名称>を新たに定義します。<値>自体にも代入マクロを使うことができます。`Product` で<値>を直接使用するよりも優れた点は、$$<名称>に代入された値が、その文書書式で有効な非必須引数となるように健全化されることです。このコマンドは全く登場しなくても1回以上現れても構いません。

`Product` <**文**> 書き出された文書に挿入される文。実のところ、これが最も重要なコマンドであり、とても複雑になることがあります。このコマンドは、一度だけ必ず現れなくてはなりません。

`Preamble` <**名称**> このコマンドは、LaTeX プリアンブルに入れるプリアンブル片を指定します。これは `PreambleDef ... PreambleDefEnd` を使用して定義しなくてはなりません。このコマンドは全く登場しなくても 1 回以上現れても構いません。

`ReferencedFile` <**書式**> <**ファイル名**> このコマンドは、変換過程で生成され、特定の書き出し書式に必要とされるファイルを示します。ファイル名が相対パスである場合には、親文書に対する相対パスとして解釈されます。このコマンドは全く登場しなくても 1 回以上現れても構いません。

`Requirement <package>` 必要とされる LaTeX パッケージ名。パッケージは、LaTeX プリアンブル中で`\usepackage{}`を使って取り込まれます。このコマンドは全く登場しなくても 1 回以上現れても構いません。

`TransformCommand Rotate RotationLatexCommand` このコマンドは、回転用に、組み込みの LaTeX コマンドを使用するように指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`TransformCommand Resize ResizeLatexCommand` このコマンドは、伸縮用に、組み込みの LaTeX コマンドを使用するように指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`TransformOption Rotate RotationLatexOption` このコマンドは、回転が非必須引数を通じて行われるように指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`TransformOption Resize ResizeLatexOption` このコマンドは、伸縮が非必須引数を通じて行われるように指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`TransformOption Clip ClipLatexOption` このコマンドは、切り抜きが非必須引数を通じて行われるように指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`TransformOption Extra ExtraLatexOption` このコマンドは、追加の非必須引数を使用することを指定します。このコマンドは、1 回現れても全く現れなくても構いません。

`UpdateFormat` <**書式**> 変換されたファイルのファイル書式。これは、LyX が知っている書式名でなくてはなりません（ツール▷設定:変換子ダイアログを参照）。このコマンドは、一度だけ必ず現れなくてはなりません。

`UpdateResult` <**ファイル名**> 変換されたファイルのファイル名。ファイル名は絶対パスでなくてはなりません。このコマンドは、一度だけ必ず現れなくてはなりません。

6.2.3 プリアンブルの定義

外用ひな型設定ファイルには、`PreambleDef ... PreambleDefEnd` で囲んだプリアンブル定義を追加することができます。これらの定義は、ひな型の `Format` 部で使用することができます。

6.3 代入機構

外部素材機構が外部プログラムを呼び出すときには、ひな型設定ファイルで定義されたコマンドにしたがって行われます。これらのコマンドには、実行前に展開されるマクロをいろいろ入れることができます。実行は、つねに元の文書があるディレクトリで行われます。

また、外部素材が表示されるときにはいつでも、その名称は代入機構によって組み立てられ、ひな型定義中の他のほとんどのコマンドも代入をサポートしています。

使用できるマクロは以下の通りです。

`$$AbsOrRelPathMaster` LyX 親文書への絶対ファイルパスないしは相対ファイルパス

`$$AbsOrRelPathParent` LyX 文書への絶対ファイルパスないしは相対ファイルパス

`$$AbsPath` 絶対ファイルパス

$$Basename パスおよび拡張子を除いたファイル名

$$Contents(“filename.ext”) このマクロは、filename.ext と云う名のファイルの中身を展開します。

$$Extension ファイル拡張子（点を含む）

$$FName 外部素材ダイアログで指定されたファイルのファイル名。これは LyX 文書への絶対パスでも良いですし、相対パスでも構いません。

$$FPath $$FName のパス部分（LyX 文書への絶対パス名か相対パス名）

$$RelPathMaster LyX 親文書への相対ファイルパス

$$RelPathParent LyX 文書への相対ファイルパス

$$Sysdir このマクロは、システムディレクトリの絶対パスを展開します。これは、典型的には、LyX に同梱されているヘルパースクリプト群を示したりするのに使用されます。

$$Tempname 元の文書が閉じられたり、挿入されていた外部素材が削除されたりすると自動的に削除される一時ファイルのフルパスとファイル名。

パスを示すマクロはすべて最後のディレクトリ区切りも含んでいますので、たとえば絶対パスのファイル名を $$AbsPath$$Basename$$Extension のようにして作ることができます。

上記マクロは、特記しない限りはすべてのコマンドで代入が行われます。Transform コマンドと TransformCommand コマンドが有効にされている場合、Product コマンドは、これらに加えて以下の代入もサポートします。

$$ResizeFront 伸縮コマンドの前置部。

$$ResizeBack 伸縮コマンドの後置部。

$$RotateFront 回転コマンドの前置部。

$$RotateBack 回転コマンドの後置部。

Option コマンドの値に入れる文字列では、Transform コマンドと TransformOption コマンドが有効にされていれば、以下の代入もサポートされます。

$$Clip 切り抜きオプション。

$$Extra 追加オプション。

$$Resize 伸縮オプション。

$$Rotate 回転オプション。

どうしてこんなに多くのパス関連マクロがあるのか不思議に思われるかもしれません。主に以下の二つの理由があります。

1. 相対ファイル名と絶対ファイル名は、それぞれ相対的あるいは絶対的なままで維持されなくてはなりません。ユーザにはどちらかの形を好む理由があるのかもしれません。たとえば相対名は、いろいろなマシンで作業をする持ち運び用の文書で役立ちます。絶対名は、プログラムによっては必要とされることがあり得ます。

2. LaTeX は、相対ファイル名に関して、LyX や入れ子にした取り込みファイル中の他のプログラムとは異なった取り扱いを行います。LyX にとって相対ファイル名とは、常にこのファイル名が書かれている文書に対して相対的なものになります。LaTeX にとっての相対ファイル名は、常に親文書に対するものになります。これら二つの定義は、一つの文書しかないときには同じですが、部分文書を含む親文書があるときには異なったものとなってきます。つまり、相対ファイル名は、LaTeX に提示されるときに変換されなくてはならないのです。幸い、正しいマクロを選びさえすれば、これは LyX が自動的に行ってくれます。

すると、新しく作ったひな型定義では、どのパス関連マクロを使うべきでしょうか。このルールは難しくありません。つまり、

- 絶対パスが必要とされるときには$$AbsPath を使う。
- 代入された文字列が、LaTeX インプットの一種である場合には、$$AbsOrRelPathMaster を使う。
- それ以外ならば、ユーザの選択を尊重するために$$AbsOrRelPathParent を使う。

このルールが機能せずに、たとえば相対名が必要となる特殊な場合もありますが、通常、上記でうまく動作します。特殊例の例としては、上述の XFig ひな型での `ReferencedFile latex "$$AbsOrRelPathMaster$$Basename.pstex_t"` というコマンドがあります。この場合、`.pstex_t` ファイルの複写子は、ファイル内容を書き換えるのに相対名を必要とするために、絶対名を使用することができないのです。

6.4　セキュリティに関する論点

外部素材機能は、多くの外部プログラムとの橋渡しをし、しかもそれを自動的に行うので、そのセキュリティ面での帰結を考慮しなくてはなりません。特に、ユーザは好きなファイル名やパラメータ文字列を含めることが許されていて、しかもそれらがコマンドに展開されるので、ユーザが文書を閲覧したり印刷したりしたときに、任意のコマンドを実行することができるような悪意ある文書を作成することが可能となりましょう。これは、我々がぜひとも避けたいことなのです。

しかしながら、外部プログラムコマンドはひな型設定ファイルでのみ指定されているので、LyX が安全なひな型でのみ適切に設定されているならば、セキュリティ上の問題は発生しません。これは、外部プログラムが `system` システムコールではなく、`execvp` システムコールで呼び出されているため、ファイル名やパラメータ部からシェル経由で任意のコマンドを実行することはできないためです。

これは、外部素材ひな型でどのようなコマンド文字列を使用することのできるかについて、制限があることを意味します。特に、パイプやリダイレクトはそのまま使用することはできません。これは、LyX の安全性を維持するためにそうしなくてはならないのです。もしシェル機能の一部を使用したいとすると、これを完全に統御の下においたまま行う安全なスクリプトを書いた上で、このスクリプトをコマンド文字列から呼び出すようにしなくてはなりません。

シェルと直接やりとりするひな型を設計することは可能ではありますが、悪意のあるユーザが狡猾なファイル名やパラメータを書くことによって、任意のコマンドを実行できるようになるため、一般的には、統御下に置いた状態で `execvp` システムコールを使用する安全なスクリプトのみを使用することをお勧めします。確かに、管理された環境下で使用する分には、通常のシェルスクリプトを使用する方に流れる誘惑はあります。

そうした場合には、お使いのシステムに簡単に濫用することのできるセキュリティホールを、**間違いなく**導入することを理解しておいてください。オープンソースの伝統に従って、私たちは人々に新しいひな型を投稿してくれるよう促していますが、そのような安全でないひな型は LyX の標準頒布版には取り入れるべきでないとすることには当然の正当性があります。公式の頒布チャンネルから出荷されている LyX には、安全でないひな型は決して入っていません。

外部素材を含めることで強力な力を手に入れることができますが、この力とともにセキュリティ上の危険を導入してしまわないように気をつける必要があります。無防備なスクリプトのたった一行に入り込んだ、ちょっとしたエラーが、巨大なセキュリティ上の問題に扉を開きうるのです。したがって、もしこの問題を完全に理解していないならば、特定のひな型が安全であるかどうか疑問がある際には、知識豊富なセキュリティの専門家か、LyX 開発チームに相談してみてください。そしてこの相談は、管理されていない環境下でこれを使用する前に、行うようにしてください。