

GNU T_EX_{MACS} 用户手册

目录

1. 开始使用	7
1.1. 本文档惯例	7
菜单项	7
组合键	7
快捷键	7
特殊键	7
1.2. 配置 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$	8
1.3. 新建、打开和保存文档	8
1.4. 打印文档	8
2. 撰写简单的文档	11
2.1. 概况	11
2.2. 输入结构化文本	11
2.3. 属性标记	12
2.4. 列表	12
2.5. 环境	13
2.6. 布局问题	13
2.7. 字体选择系统	14
2.8. 精通键盘	14
2.8.1. 基本前缀规则	14
2.8.2. 文本模式下的快捷键	15
2.8.3. 混合命令和 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 模拟	16
2.8.4. 动态对象	16
2.8.5. 一些有用的键盘快捷键	17
3. 数学公式	19
3.1. 在文档中输入数学公式	19
3.2. 输入数学符号	20
3.3. 主要的数学结构	20
3.4. 输入大型分隔符	21
3.5. 输入大型算符	22
3.6. 加宽的顶标	22
3.7. 语义编辑工具	23
3.8. Common errors and syntax correction	23
3.9. Semantics of mathematical symbols	25
3.10. 定制数学语义	26
4. 表格处理	29
4.1. 创建表格	29
4.2. 格式化模式	30
4.3. 指定单元格和表格对齐	30
4.4. 指定单元格和表格的大小	30
4.5. 边界、补白和背景色	31

4.6. 表格的高级特性	31
5. 链接和内容自动生成	33
5.1. 创建标签, 链接和参考	33
5.2. 插入图片	33
5.3. 生成目录	34
5.4. Compiling a bibliography	34
Editing files with bibliographic entries	34
Inserting citations and compiling bibliographies	35
5.5. Generating an index	35
5.6. 编制术语表	35
5.7. Multiple extractions	36
5.8. Books and multifile documents	36
6. 内置作图工具	37
6.1. 开始作图	37
6.2. 插入图形对象	37
6.3. 编辑图形对象	38
6.4. 样式属性详述	39
颜色	39
填充色	39
不透明度	39
点形状	39
线宽度	40
线组成	40
线箭头	40
文本对齐	40
6.5. 编辑对象的组合	41
7. 进阶布局特性	43
7.1. 文档流	43
7.2. 浮动对象	43
7.3. 分页	43
8. 编辑工具	45
8.1. 剪切与粘贴	45
8.2. 搜索和替换	45
8.3. 拼写检查	46
8.4. 撤销和恢复	46
8.5. 结构化编辑	46
8.6. 结构化光标移动	47
结构化遍历文档	47
结构化遍历相似标签	48
最深处标签的移动	48
8.7. 结构化变元	48
8.8. 移动和缩放对象	48
8.9. 版本控制工具	49
比较两个版本	49

差异的可视化	49
保留特定版本	49
粒度控制和差异重现	50
使用外部版本控制工具	50
9. LAPTOP PRESENTATIONS	51
9.1. Beamer styles	51
9.2. Traversal of a presentation	51
9.3. Overlays	52
9.4. Decorations	53
9.5. Animations	54
9.6. Exporting beamer presentations	54
10. USING GNU T_EX_{MACS} AS AN INTERFACE	57
10.1. Creating sessions	57
10.2. Editing sessions	57
10.3. Selecting the input method	58
10.4. Plug-ins as scripting languages	59
10.5. Spreadsheets	60
10.6. Remote plug-ins	61
11. WRITING T_EX_{MACS} STYLE FILES	63
11.1. Writing a simple style package	63
11.2. Rendering of style files and packages	65
11.2.1. ASCII-based or tree-based editing: an intricate choice	65
11.2.2. Global presentation	66
11.2.3. Local customization	69
11.3. The style-sheet language	70
11.3.1. Assignments	70
11.3.2. Macro expansion	70
11.3.3. Formatting primitives	71
11.3.4. Evaluation control	73
11.3.5. Flow control	74
11.3.6. Computational markup	75
11.4. Customizing the standard T _E X _{MACS} styles	76
11.4.1. Organization of style files and packages	76
11.4.2. General principles for customization	77
11.4.3. Customizing the general layout	77
11.4.4. Customizing list environments	78
11.4.5. Customizing numbered textual environments	80
Defining new environments	80
Customization of the rendering	80
Customization of the numbering	81
11.4.6. Customizing sectional tags	81
11.4.7. Customizing the treatment of title information	83
11.5. Further notes and tips	84
11.5.1. Customizing arbitrary tags	84
11.5.2. Standard utilities	85
12. 定制T_EX_{MACS}	87

12.1. GUILE扩展语言入门	87
12.2. 撰写初始化配置文件	87
12.3. 定制动态菜单	87
12.4. 定制快捷键	88
12.5. 其它值得一看的文件	89
13. THE \TeX_{MACS} PLUG-IN SYSTEM	91
13.1. Installing and using a plug-in	91
13.2. Writing your own plug-ins	92
13.3. Example of a plug-in with SCHEME code	93
The world plug-in	93
How it works	93
13.4. Example of a plug-in with C++ code	93
The minimal plug-in	93
How it works	94
13.5. Summary of the configuration options for plug-ins	94
索引	97

章 1

开始使用


1.1. 本文档惯例

菜单项.


在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 文档中，菜单项以加粗的字体和箭头表示：文档，文件→打开或者格式→字型→斜体。


组合键.

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 使用下列组合键：

. Shift 组合键








. Control 组合键






. Alt 组合键

. Meta 组合键

例如，X 表示同时按下三个键 ， 和 X。

快捷键.

更加复杂一些的快捷键是通过连续按多个键或者组合键得到的。比如，快捷键 -> 是先按下 ，再按下  得到的。在数学模式中，这个快捷键会插入 \rightarrow 。类似地，先按下 +和X的组合，再按下 +和F的组合，你就能得到快捷键 +X+F。如果你使用的是 Emacs 快捷键风格，那么这个快捷键是用来打开文件的。

为了避免 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的快捷键和操作系统的快捷键产生冲突，我们使用通用的键前缀，详细内容见[通用键盘规则](#)一节。简单的说，使用  键可以得到[组合键的等价键](#)。例如， 等价于  n+， 等价于 。

值得注意的是， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的菜单和键盘的行为是上下文相关的，具体来说，是依赖于当前的模式（文本模式或者数学模式等等），当前文档的语言和当前光标的位置。比如说，在数学模式中，会有很多方便你输入数学公式的快捷键，但是在文本模式中，这些快捷键就会失效。

特殊键.

在某些平台上，一些特殊键诸如换行是用特殊字符来描述的。下面这个表格列出了所有这些特殊键及其含义。

键	含义	键	含义
	Shift 组合键		光标左移
	大写锁定		光标右移
	Ctrl 组合键		光标上移
	Alt 组合键		光标下移
	Meta 组合键		Home
	换行		End
	删除		Page up
	退格		Page down
	Escape		空格
	Tab		

表格 1.1. 特殊键。

1.2. 配置 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$

第一次启动 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 时，程序将自动启用默认配置，一般来说，默认配置就已经够用了。 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 会根据你的系统设置决定文档的语言和页面的类型。当然，有些时候，自动配置也会失灵，这个时候你可以自己配置。最简单的方式是，直接在编辑→首选项中指定你的偏好。

特别地，我们建议你配置好自己喜欢的快捷键风格和菜单布局。默认地，我们使用和系统一致的快捷键风格和菜单布局。当然，我们也提供类Emacs的快捷键风格和菜单布局。

1.3. 新建、打开和保存文档

不加任何命令行参数就启动 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ ，编辑器会自动为你新建一个空白文档。你也可以用文件→新建自己新建一个文档。新建文档默认未命名。点击文件→另存为可以给文档命名。我们建议你新建文档后马上另存为该文档，这样可以避免你丢失该文档。

必要时，建议你设定文档的全局设置，这些设置一般都在文档菜单中。首先，你可能需要在文档→样式中指定文档的样式比如article, book, seminar。如果你使用多种语言编辑文档，你可以在文档→语言中指定这个文档的默认语言。类似地，在文档→页面→大小中可指定页面的类型。

对于新建的文档，样式和页面大小也可以通过焦点菜单或者焦点工具栏上的按钮改变。一般来说，焦点工具栏就是菜单下面的第三行工具栏。焦点菜单和焦点工具栏在编辑结构化文档时尤为方便，其菜单项会跟随光标所在的上下文改变。

更改文档之后，点击文件→保存保存文档。文档可以通过文件→打开打开。注意，在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中你可以在同一个窗口下编辑多个文档，我们把这样的每一个打开的文档叫做缓冲区，使用菜单转到可以在不同的缓冲区间切换。

1.4. 打印文档

点击文件→打印→全部打印打印当前文档。默认地， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 假定你的打印机是600dpi，用的是A4纸。当然，在编辑→首选项→打印机中你可以更改成你需要的设置。你也可以用文件→打印→全部打印为文件或者文件→导出→Postscript将文档导出成Postscript文档。两者的差别在于，前者使用前面提到的默认打印设置导出文档，后者则是直接忽略了打印设置。

当你配置好 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 之后，编辑器一定是所见即所得的——打印出来之后的结果和你在屏幕上看到的完全一样。特别地，为了完整的所见即所得支持，你应该选择文档→页面→类型→Paper和文档→页面→Screen layout→在纸上显示边白。你还应该确认屏幕上的字符显示和你的打印机的每英寸所打印的点数是否相同。这种字符渲染精度可以在文档→字体→Dpi中调整。当前，改变dpi时排版上细微的改变也许会影响到文档中的行与页的分割。在未来的版本中，这个缺陷将不复存在。

章 2

撰写简单的文档

2.1. 概况

一般的字符和标点符号直接键盘输入即可。大多数现代系统也会为重音字符和其他特殊符号提供快捷键。在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 上，**Wi n+**键前缀可用于输入重音字符。比如，由快捷键**Wi n+ ' E**可得“é”，快捷键**Wi n+ ` A**则是“à”。

较长的单词在行边界处会自动使用连字符连接。对于中文，您需要在菜单文档→语言中指定语言，以正确换行。

在底部状态栏的左边，您可看到文档的样式、当前光标处的文本属性。比如，底部状态栏显示“generic text roman 10”意味着您是在文本模式工作，所用的roman字体大小为10，所用文档的样式为generic。您可在格式菜单中改变当前光标处的文本属性(字体、字体大小、颜色、语言)。也可以选中一块文本后再对该文本操作。默认的全局文本属性在文档菜单中更改。

底部状态栏的右边显示的是光标前字符或者对象的属性。我们会显示当前光标所在位置的环境。所有这些信息都有助于您在文档中光标的精确定位。

2.2. 输入结构化文本

长文档往往具有良好的结构：即章、节和子节。其中往往包含不同类型的文本，比如纯文本，引用，脚注，定理等。选择好文档→样式后， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 会依照样式为文本排版，比如节、页、定理的编号，又如引用和脚注的排版等等。

目前可用的标准文档样式有：**generic**，**article**，**book**，**letter**，**exam**，**beamer**，**seminar**，**source**。比如，article样式可用于写文章。另外的样式可用于期刊论文或其它特殊用途，比如 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 官方文档的样式。

选择好文档的样式后，您可以用节来组织文本，还可以使用特定的环境。环境的例子有定理，命题，注意等(见插入→说明)。有序列表和无序列表也可用来组织文本。常用的属性标记有**strong**，**name**等等，可用于改变特定文本的属性。

用熟 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 后，您可在自己的样式文件中添加环境。假设您经常做引用，且您希望引用文本是斜体的，且左右边距皆为1cm。最好的方法是制作一个引用环境，而不是一次次手动的改变文本和段落的属性。这不仅仅能够加快引用的输入，还可以直接改变引用环境的定义，同步改变文档中所出现的所有引用环境。比如您需要把引用的字体全部改小，此时，直接改定义明显比一次次手动修改方便。

了解一些通用的编辑规则可以极大的方便编辑 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的结构化文本。一个重要的概念是焦点，请看下面这个例子。假设我们正在输入如下经典定理：

下面这个定理是欧拉的：

定理 2.1. $e^{\pi i} = -1$ 。

在光标处，灰色和蓝绿色的盒子显示的是激活的标记：在本例中，光标是在定理和公式中。最内部的激活被蓝绿色框包围的标记称为当前焦点。

焦点菜单和焦点工具栏（最后一行工具栏）是高度上下文相关的，它们都依赖于当前焦点。在本例中，焦点工具栏中有一个弹出菜单公式，当您选择单行方程后，文本会变为：

下面这个定理是欧拉的：

定理 2.2.

$$e^{\pi i} = -1.$$

类似地，焦点工具栏左部的箭头按钮可允许您跳转到相似标记。在本例中，它们会让你迅速在文档中所有的公式或者方程间浏览。关于[结构化编辑](#)和[结构化编辑工具](#)，还请移步细看。

第二个重要的概念是编辑模式。有五种主要的模式：文本模式，数学模式，程序模式，图形模式和源代码模式。模式切换比焦点的切换要少得多，其工具栏就在焦点工具栏的上方。插入和格式菜单都是模式相关的。

2.3. 属性标记

文本中的结构性最简单的一种体现即属性标记。属性标记能够表明一段文字特殊性质或者为特殊目的服务。比如重要的文本应当用**strong**来标记。该标记默认将字体渲染成粗体。需要注意的是，文档样式不同，加强文本后渲染效果也会有所不同。比如在幻灯片中，加强文本的渲染效果可能是透明的不同的颜色。下面是最常用的一些属性标记及其目的。

标记	例子	目的
strong	this is important	加粗文本以显示其重要性
em	the real thing	强调一段文本
<code>dfn</code>	A <i>gnu</i> is a horny beast	一些概念的定义
<code>samp</code>	the ae ligature æ	字母序列
<code>name</code>	the LINUX system	某物的名称
<code>person</code>	I am JORIS	某人的名字
<code>cite*</code>	Melville's <i>Moby Dick</i>	参考书目
<code>abbr</code>	I work at the C.N.R.S.	缩写
<code>acronym</code>	the HTML format	首字母缩写
<code>verbatim</code>	the program said hello	纯文本， 比如计算机程序的输出
<code>kbd</code>	Please type <code>return</code>	应当用键盘输入的文本
<code>code*</code>	<code>cout << 1+1; yields 2</code>	代码
<code>var</code>	<code>cp src-file dest-file</code>	程序中的变量

表格 2.1. 常用属性标记一览。

2.4. 列表

插入→无序列表即插入无序列表。可选择 • (圆点)，- (短线) 或者 → (箭头) 之一作为条目的标记。如下所示，列表中还可嵌入子列表：

- 第一项。
- 下面是子列表：
 - 一个子项。

- 另一个子项。
- 最后一项。

默认的标记根据其层次自动以不同样式显示。在最外面一层，我们使用•，第二层则用◦。光标在列表中时，按下`Return`键将自动新建一个新项目。如果您的项目内容由多个段落构成，则可以用`Shift+Return`新起一段。

有序列表可用菜单项插入→有序列表得到，与无序列表类似，只是各个项目是顺序显示的。下面是一个按罗马数字顺序显示的例子(插入→有序列表→Roman)：

- I. 第一项
- II. 第二项
- III. 最后一项

最后一种列表是描述列表。相应的菜单项是 插入→描述列表，功能是演示一串概念：

Gnu. A hairy but gentle beast.

Gnat. Only lives in a zoo.

2.5. 环境

与属性标记类似，环境的作用是赋予一段文字以特殊含义。属性标记通常作用于一小段文字，而环境的作用范围往往横跨多行。数学中常用的环境是`theorem`和`proof`，如下所示：

定理 2.3. 不存在这样的正整数 a, b, c 和 $n(n \geq 3)$ 使得 $a^n + b^n = c^n$ 。

证明. 空白太小，容不下我的证明。(费马)

□

插入→环境即插入环境。与定理类似的环境有`proposition`, `lemma`, `corollary`, `axiom`, `definition`。使用`dueto`宏(按下`\DUETO`输入该宏)可以指定与定理相关的学者，像这样：

定理 2.4. (毕达哥拉斯) *Under nice circumstances, we have $a^2 + b^2 = c^2$.*

其它常用的环境还有`remark`, `note`, `example`, `warning`, `exercise`和`problem`，其渲染效果与定理类似，只是没有强调显示其中内容。其余环境如`verbatim`, `code`, `quote`, `quotation`和`verse`可用于输入多行文本，代码，引用或者诗歌。

2.6. 布局问题

一般情况下，`TeXMACS`会自己处理文档的布局。所以，尽管我们不想禁止，但是我们也不鼓励你手动排版文档。比如，你不应该用插入空格或者空行的方法作为水平方向上字与字之间的空白或者竖直方向上行与行之间的空白。额外的空白应该用插入→空白明确指定。这可以让你的文档更加鲁棒，因为你从此无需顾虑由于某些细小的更改而影响到断行、分页或更大改变等布局问题。改变文档的样式就会使得`TeXMACS`重新排版。

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 实现了多种精确插入空白的命令。首先，你可以精确地插入指定高度和宽度的空白。水平方向上的空白无需高度这一属性，而且可以是可伸缩的。可伸缩的空白的长度由段落的断行方式决定。更进一步，你还可以插入制表符类型的空白。竖直方向上的空白可以在段落中任意位置插入，根据类型的不同会插入到段落的上面或者下面，我们建议将插入点放在段落起始处或结束处。两个段落间额外的竖直空白高度是其间的竖直空白高度的最大值（和 $\text{T}_{\text{E}}\text{X}$ 相反，这可以防止两个连续定理之间多余的空白）。

至于段落的布局，用户可以指定段落的对齐方式（自调整、左对齐、中央对齐、右对齐）、边白和段落首行（末行）的向左（右）缩进。用户还可以控制行间以及段落间的间距。

页面布局的菜单在文档→页面可以找到。首先，你可以指定页面在屏幕上的展示方式：在文档→页面的格式标签下页面渲染选项中选中“paper”作为页面类型，你就可以看到明确的分页。默认页面类型“papyrus”在你撰写文档时不会分页。页面类型“automatic”假定你的页面大小正好是你的窗口大小。页面的边白和文本宽度在文档→页面→边白中指定。

2.7. 字体选择系统

在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中，字体有以下五种主要属性：

- 名称 (roman, pandora, concrete等等)
- 字族 (roman, typewriter或者sans serif)
- 大小 (基准大小和相对大小)
- 字型 (粗黑, 适中 或者 细黑)
- 字形 (倾斜, 斜体, 小体大写等等)








注意，在 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2\epsilon$ 的字体选择系统中，没有细分字体名称和字族。注意，基准字体大小是整个文档的属性，在文档→字体→大小中指定。










2.8. 精通键盘



本节主讲快捷键，相关内容除本节外还可以参考[配置键盘](#)。

2.8.1. 基本前缀规则

由于快捷键多而杂，将它们分类将便于记忆。这里按照键前缀来分类。当前激活的键前缀依赖于编辑→首选项中的快捷键风格。主要的键前缀如以下所示：























- . 标准快捷键。比如，在Emacs快捷键风格下，可以用来粘贴文本。
- . $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 快捷键，往往依赖于当前的编辑模式。比如， 在文本模式下产生strong文本，在数学模式下则是 $\sqrt{\quad}$ （平方根）。
- . 组合式 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 快捷键。通常，这些快捷键是用于先指示命令将要应用的标记的类别再指定具体的命令。比如E键前缀用于插入可执行的标记以用于撰写样式文件。一个例子是快捷键E+用于插入加法。

- . 此键前缀用于和方向键及其它特殊键组合，用于移动和缩放对象。
- . 此键前缀用于和方向键及其它特殊键组合，用于结构化光标移动。
- . 此键前缀偶用于和字母和标点符号组合以创建一些额外的简单易记的快捷键。
- F5. 此键前缀可和常用字符组合使用以插入特殊符号。比如，使用F5 S得到ß以及F5 A得到И。F5键前缀还用于插入字面意义上的字符。比如，F5 \总是产生\，而用于进入混合命令。

不巧的是，基于快捷键在一些系统上面被系统的快捷键所覆盖。比如，在Mac OS上重音字符和常见的特殊字符是使用该快捷键输入的。此时，你可以使用作为替代。更多信息，参见配置键盘。















2.8.2. 文本模式下的快捷键

使用键盘输入欧洲语言的特殊键时，您可以使用下面的快捷键输入重音字符。注意，这并不依赖于当前的语言设置。


Shortcut		Example		Shortcut		Example	
	Acute ´	 E	é		Grave `	 E	è
	Hat ^	 E	ê		Umlaut ¨	 E	ë
	Tilde ~	 A	ã	 C	Cedilla ¸	 C C	ç
	Breve ˘	 U G	ğ	 V	Check ˇ	 V S	š
	Above ring ˆ	 O A	â		Above dot ˙	 . Z	ž
	Hungarian ˝	 H O	ó				

表格 2.2. 输入重音字符

特殊字符也可以在任意语言环境下输入：

Shortcuts							
 F5 A	æ	 F5 A	Æ	 F5 A E	æ	 F5 A E	Æ
 F5 O	ø	 F5 O	Ø	 F5 O E	œ	 F5 O E	Œ
 F5 S	ß	 F5 S	SS				
 F5 !	¡	 F5 ?	¿	 F5 P	§	 F5 P	£

表格 2.3. 输入特殊字符

当你按下键时，恰当的引号将被插入。这里的引号由当前的语言和周围的文本决定。如果默认的引号不正确，你可以在 编辑→首选项→键盘→自动补全引号 改变设置。你也可以用快捷键这样输入引号：

快捷键			
 F5 "	"	 F5 "	"
 F5 <	<	 F5 >	>
 F5 <<	<<	 F5 >>	>>

表格 2.4. 输入引号和书名号

“英文”的引号是相继两个重音符或者单引号的连接。它们可以使用`` ``创建`' '`，但是实际上这是两个字符的特殊展示，而不是单个特殊的字符。

一些快捷键在特定语言环境下生效。你可以使用 文档→语言 设置文档的语言，或者 格式→语言 设置选中区域的语言(见[输入文本概况](#))。

匈牙利语		西班牙语		波兰语			
	ő		í		ą		ó
	Ö		¿		Ą		Ó
	ű		ï		ć		ś
	Ű		ı		Ć		Ś
					ę		ź
					Ę		Ź
					ł		ż
					Ł		Ż
					ń		ź
					Ń		Ź

表格 2.5. 语言相关的文本快捷键

语言相关的快捷键会覆盖更加一般的快捷键：比如，在匈牙利语环境下，你无法轻易地输入字符“ø”。

2.8.3. 混合命令和L^AT_EX模拟

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 允许你直接从键盘输入L^AT_EX命令。 具体步骤如下： 你先按下`\`以进入L^AT_EX/ $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 混合命令模式，再输入你需要执行的命令。当你输完命令时，底部的状态栏左边会显示如下信息：

```
 > action to be undertaken
```

此时当你按下回车键， 你的命令就会被执行。 比如在数学模式中， 你可以通过输入`\frac`创建分数。

如果你输入的命令不是一个（可识别的）L^AT_EX命令，我们首先看该命令是否是一个存在的 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 宏，函数或者环境（由样式文件提供）。如果是这样，相应的宏展开，函数或者环境将被应用。否则，我们认为你的命令对应于一个环境变量且我们提取其值。`\`键总是等价于这些命令`%I L`，`%I E`，`%I A`，`%I #`或者`%I V`之一。

使用`F5 \`可以输入字面意义上的`\`(backslash)。



2.8.4. 动态对象

某些更加复杂的对象在编辑的过程中可以有不同状态。这种动态对象的一个例子便是标签和引用，因为引用的标号必须动态确定。另外一些动态标记的例子可以在[撰写样式文件](#)这章中找到。




编辑进入动态对象时，比如使用`^!`插入标签，默认的状态是不激活的。在这种不激活的状态中，你可以输入和这个动态对象相关的信息，比如在标签这个例子中就是标签的名字。有一些动态对象的参数数量不是固定的，可以用`->`插入新的参数。

```
<label|pythagoras>
```

图 2.1. 不激活的标签

当你将动态对象的相关信息输入完毕后，你可以输入激活这个对象。在光标移动到对象后面并敲击就可以反激活已经激活的动态对象。

2.8.5. 一些有用的键盘快捷键

一些常用的组合快捷键如下表 2.6所示。特别注意的是由Space, ^Space和 ^↑Space插入的空白可用和改变大小。这种缩放的方式实际上适用于更一般的从菜单格式→空白插入的或水平或竖直的空白，也可以是一些其它的对象如图片。

快捷键	动作
	除去所包含的对象或环境。
Space 	插入一个不间断的空白。
Space 	插入一个quad空白。
^Space	插入一个小空白。
^↑Space	插入一个负空白。
	插入一个“tab”。
	移动到文档头部。
	移动到文档尾部。
	插入一个换行符。
	插入一段不能断行（rigid）的文字。

表格 2.6. 一些有用的键盘快捷键

章 3

数学公式

$\text{\TeX}_{\text{MACS}}$ 要解决的主要问题之一便是数学公式的编辑。当光标处于公式中时，模式敏感的菜单和工具栏就会提供相应的数学符号输入方法。类似地，键盘的行为也会跟着改变，以便您快速输入数学公式。比如，输入 \rightarrow 可得 \rightarrow 。上述行为改变将在本节中详细说明。

自从1.0.7.10版， $\text{\TeX}_{\text{MACS}}$ 为数学公式增添了一些语义编辑功能。只要使用得当，这能使您的公式至少在句法上保证正确。您可用语法纠错来达到这一目标。带有语义的文档显然更加实用，比如使用公式作为计算机代数系统的输入。这样的文档会比一般的文档typos更少。其它一些有意思的特性，比如语义搜索和替换，会在不久的将来开发。

3.1. 在文档中输入数学公式

$\text{\TeX}_{\text{MACS}}$ 对数学公式有着非常强大的支持。 $\text{\TeX}_{\text{MACS}}$ 下有三种输入数学公式的方式：

插入 \rightarrow 数学 \rightarrow 公式或者 $\$$.

这个菜单条目输入的是像 $a^2 + b^2 = c^2$ 那样嵌在文本段落中的简短公式。注意公式是一种特殊的排版方式，不会占用过多的垂直方向上的空间。比如求和符号的上下界限往往处于右边（ $\sum_{i=1}^n$ ）。此时选中整个数学公式，再在格式 \rightarrow 数学 \rightarrow 展示风格中选择打开，求和符号的界限就会显示在其上下两侧（ $\sum_{i=1}^n$ ）。缺省情况下，展示风格是关闭的。

插入 \rightarrow 数学 \rightarrow 单行公式或者 $\text{Alt}+\$$.

这个条目适合输入较长的公式，比如：

$$x^n + y^n = z^n,$$

公式的排版会占据整个段落。使用快捷键 $\text{Alt}+\#$ 可以给公式加上编号（或者删去编号）。快捷键 $\text{Ctrl}+\text{Tab}$ 能够让公式在行内模式和单行模式间切换。

插入 \rightarrow 数学 \rightarrow 多行公式或者 $\text{Alt}+\&$.

这个条目创建了一个`eqnarray*`，一个三列宽的类似表格的环境（详见[创建表格](#)）。在这个环境中可以方便地输入多行公式，比如：

$$\begin{array}{lcl} x + 0 & = & x \\ x + (-x) & = & 0 \\ x + y & = & y + x \\ (x + y) + z & = & x + (y + z) \end{array}$$

第一列右对齐，第二列居中，第三列左对齐。`eqnarray*`环境的典型用途就是显示多步计算：

$$\begin{aligned}(e^{\sin x} + \sin e^x)' &= (e^{\sin x})' + (\sin e^x)' \\ &= (\sin x)' e^{\sin x} + (e^x)' \cos e^x \\ &= e^{\sin x} \cos x + e^x \cos e^x,\end{aligned}$$

在这个例子中，大多数行的第一列是空着的。

3.2. 输入数学符号

使用F5输入希腊字母，比如，F5 A输入 α ，F5 ↑G输入 Γ 。类似地，F6，F7，F8 和 ↑F6可用来输入粗体，手写体，德文黑体和黑板粗体。例如，F8 M 得到m，↑F6 ↑R得到 \mathbb{R} 以及F6 F7 ↑Z得到 \mathbb{Z} 。

使用→还可以从拉丁字符得到希腊字母。比如，P →得到 π 。→键还可以在希腊字母间变换，比如，F5 P →和P →→都可以得到 ϖ 。输入黑板粗体的另外一种方法是输入该大写字母两次。比如，↑Z ↑Z得到 \mathbb{Z} 。

一些符号中包含了诸多变元。比如<得到<，< →得到 \in ，< →→得到 \subset ，< →→→得到 \subsetneq 等等。反向轮换可用↑→。即，< →→↑→等价于< →。

其它数学符号可通过自然的键组合得到。比如->得到 \rightarrow ，-->得到 \longrightarrow 和>=得到 \geq 。类似地，| →-得到 \vdash ，| ->得到 \mapsto 以及-><-得到 \rightrightarrows 。一般地，在数学符号的输入过程中有如下规则：

- . 在变元间轮换的键。比如，>=得到 \geq ，而>=→得到 \geq 。类似地，< →→得到 \subset ，< →→=得到 \subseteq 以及< →→=→得到 \subseteq 。↑P →得到 φ 以及E →得到常量 $e = \exp(1)$ 。
- @. 用来将符号嵌入圈或者框中。比如@+得到 \oplus 以及@X得到 \otimes 。类似地，@→+得到 \boxplus 。
- / . 用于否定。例如，=/得到 \neq 以及<=/得到 \nless 。注意<=→→/得到 \nless ，而<=→→/→得到 \nless 。
- !. 用于强制使得箭头的上标或下标置于箭头的竖直方向上。比如，-->^X得到 \longrightarrow^x ，而-->!^X得到 \xrightarrow{x} 。

逻辑关系 \wedge 和 \vee 通过&和%得到。操作符 \cap 和 \cup 是其自然而然的变元，可用&→和%→得到。许多杂七杂八的符号可通过↑F5得到。

注意到某些符号的数学意义不一样，然而长得很是相似；这类符号被称为homoglyphs。比如竖直条|可以是集合定义中的分隔符，如 $R^> = \{x \in R | x > 0\}$ ，也可以是二元关系整除，如 $11 | 1001$ 。往往，homoglyphs之间的空白是不同。最恼人的多意性可以以空白为例，如不可见的乘法空白 xy 和函数空白 $\sin x$ ，两者分别是使用* 和Space输入的。

为了方便T_EX_{MACS}自动处理你的文档以方便数学公式的语法检查等，我们鼓励用户在输入公式时注意到homoglyph的问题。更多信息，可参见[数学符号的语义](#)。

3.3. 主要的数学结构

最主要的几个数学结构可使用^前缀以如下方式创建：

快捷键	目标	例子
	文本	$L = \{x x \text{ 足够大} \}$
	分式	$\frac{a}{b+c}$
	平方根	$\sqrt{x+y}$
	n 次方根	$\sqrt[n]{x^3+y^3}$
	否定	$\frac{a}{b/\text{ } c}$

表格 3.1. 主要数学标记的创建

下面是撇和上下标：

快捷键	目标	例子
	撇号	f' or $(g+h)'''$
	反撇号	$\backslash f$
	下标	x_n or x_{i_3}
	上标	x^2, x_n^2 or e^{e^x}
	左下标	$2x$
	左上标	πx or ${}^*\text{He}^*$

表格 3.2. 撇、上下标的创建

另外一些重要的数学结构实际上是广义的表格，如矩阵、行列式等，详细请参见[表格结构](#)。

3.4. 输入大型分隔符

数学公式中的括号必须匹配：在你输入“(”时， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 会自动插入“)”。你可以编辑→首选项→键盘→自动补全括号→禁用禁用此特性。注意：文档中旧有的括号可能会自动与你输入的括号匹配。

有时，你不需要封闭的括号，或者你的括号匹配的种类不同。没有关系：如果光标正好在右括号的前面(a, b)，此时按下可以得到表达式(a, b)|。类似地，删除一个括号会导致其变为不可见括号，然后你就可以使用任意类型左或者右括号去替换。

默认地，括号的大小会随着其中公式的大小自动调整。使用键前缀产生的小分隔符大小不会随之改变。你还可以使用让分隔符在大小之间切换。

对某些分隔符，如，其左右符号恰好一致。比如输入垂直方向上的竖条产生的是绝对的值。小竖条分隔符通过F5得到，也可以是。大竖条分隔符通过。在 $\text{T}_{\text{E}}\text{X}$ 和 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 中，这类大型分隔符是不存在的；他们可以用来输入下列公式

$$\left\langle \frac{a}{b+c} \middle| \frac{p}{q+r} \middle| \frac{a}{b+c} \right\rangle.$$

中间分隔符也是非常常见的一种分隔符。比如，二元关系整除可以通过F5或得到。

在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中，大型分隔符分为左分隔符，右分隔符和中间分隔符。默认地，(, [, {和<是左分隔符；),], }和>是右分隔符。使用，和可以改变分隔符的状态。比如，产生的)|，可以认为是大型左分隔符。

有时候，你可能需要固定大小的大型分隔符，而不是自动调整。这可以通过改变分隔符之间的表达式大小的方法实现，具体是使用菜单项格式→变换→Resize object。

还可以使用`?`输入一对不可见的括号。这对于计算性质的文本非常有用，特别是在公式不仅仅是显示的语义，还要照顾到其精度。还可以用`%^↑R`将公式保护在固定框内，即防止公式被连字符强行断开。

3.5. 输入大型算符

下面的快捷键用于输入较大的数学符号：

快捷键	效果	快捷键	效果
<code>↑F5 ↑I</code>	\int	<code>↑F5 ↑O</code>	\oint
<code>↑F5 ↑P</code>	\prod	<code>↑F5 ↑A</code>	\coprod
<code>↑F5 ↑S</code>	\sum	<code>↑F5 @+</code>	\oplus
<code>↑F5 @X</code>	\otimes	<code>↑F5 @.</code>	\odot
<code>↑F5 ↑U</code>	\cup	<code>↑F5 ↑N</code>	\cap
<code>↑F5 ↑V</code>	\vee	<code>↑F5 ↑W</code>	\wedge

表格 3.3. 大型算符

标签周围的浅蓝色小框显示了这些大型算符的作用域。

积分符号由两种表示，其不同之处在于上下标的位置。默认地，我们使用如下方式：

$$\int_0^{\infty} \frac{dx}{1+x^2}.$$

另外一种

$$\int \limits_0^{\infty} \frac{dx}{1+x^2}.$$

则可用快捷键`↑F5 ↑L ↑I`获得。类似地，使用快捷键`↑F5 ↑L ↑O`可得到 \oint ，其中上下标的位置是第二种形式。

3.6. 加宽的顶标

下表展示的是如何在符号乃至整个公式上打上顶标。一般来说，顶标会随着下面公式的加宽而加宽。

快捷键	例子	加宽变元	快捷键	结果
<code>^~</code>	\tilde{x}	$\widetilde{x+y}$	<code>^'</code>	\acute{x}
<code>^^</code>	\hat{x}	$\widehat{x+y}$	<code>^ˆ</code>	\hat{x}
<code>^↑B</code>	\bar{x}	$\overline{x+y}$	<code>^.</code>	\dot{x}
<code>^↑V</code>	\vec{x}	\overrightarrow{AB}	<code>^"</code>	\ddot{x}
<code>^↑C</code>	\check{x}	$\widetilde{x+y}$		
<code>^↑U</code>	\breve{x}	$\overline{x+y}$		

表格 3.4. 顶标的快捷键

使用键前缀`^U`可将相应的顶标插入到表达式下面。比如，快捷键`^U BX+Y`可用于输入 $x+y$ 。

3.7. 语义编辑工具

自从1.0.7.10, $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 新增了一些数学公式的语义编辑特性。合理使用这些特性，就能使您的公式至少在句法层面上保证正确。比如，在公式 $a+b$ 中，计算机能识别 $+$ 是作用在变量 a 和 b 上的运算符。注意这里所谓“语义”仅限于此，即 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 不会去关注加法的数学本质。

语义编辑需要用户注意一些额外的事物，至少是适应它们。比如，用`*`输入乘号和用`Space`输入函数是用户的需要了解且实践的知识。诚然，展现在图像上，这些操作的细节无法辨识，因为它们都是以空格的形式为用户所见。然而这些操作的语义显然截然不同。

尽管通常情况下，就排版而言，语义正确的文档和以打印为目的的正式文档没有太多的区别。我们认为用户为此付出的努力是有价值的，具体理由如下

- 比如，当您输入的公式用于计算机的代数系统时，语义丰富的文档更显优势。
- 句法上正确的文档出现“typos”或者更复杂的数学错误的可能性更小。
- 对某些编辑操作，诸如剪切和粘贴，您可以更方便的选中句法上意义完整的子公式。
- 这降低了您使用非标准标记的可能性，非标准标记会增加您的作品潜在用户的阅读难度。

更进一步，其它语义相关的工具其实已经集成到了某些特性中，如语义搜索和替换或者互联网上的语义搜索。

点击编辑→首选项→数学→语义编辑激活语义编辑工具。在语义编辑模式下，您将使用到针对公式句法结构的结构化编辑特性。比如，语义焦点通常是当前焦点的子公式。又如，您只能选中句法上有意义的子公式。

语义焦点的意义如下所述：首先，句法上正确的公式显示为绿色，错误的则为红色。这样您就可以在输入公式时迅速发现“typos”。其二，如果您对某个数学操作符或者关系的优先级存在任何疑问，从语义焦点便可以看出缺省优先级。具体地，将您的光标置于操作符的右边，则该操作符所作用的子表达式会高亮显示。在加法（或者更一般的结合算符）这个例子中，所有的被加数都会高亮显示。

3.8. COMMON ERRORS AND SYNTAX CORRECTION

By default, the semantic editing mode “understands” most classical mathematical notations. This is achieved through the use of a carefully designed grammar for mainstream mathematics. Obviously, the use of a fixed grammar may cause the following problems:

- Mathematical formulas frequently contain *ad hoc* notations. For instance, the formulas might contain some text or meaningful whitespace. Another example of an *ad hoc* notation is the sign sequence $++--++$. In such cases, the user should [explicitly annotate](#) the appropriate parts of the formula in order to make them semantically meaningful.

- The $\text{\TeX}_{\text{MACS}}$ grammar used for the interpretation of mathematical formulas may be incomplete or inadequate for certain situations. It is possible to customize or extend the grammar using the standard $\text{\TeX}_{\text{MACS}}$ macro mechanism. Notations for specific areas may be grouped together in dedicated style packages.

Besides these intrinsically hard to avoid problems, the following common and “easy-to-make” mistakes are a further source of trouble for associating semantics to mathematical formulas:

- Since $\text{\TeX}_{\text{MACS}}$ is a wysiwyg editor, some of the structure of the document is invisible for the user. For instance, the presence of a mathematical formula $x + y$ is indicated through the use of an italic slant and special spacing. However, in the formula $f(x)$ it is easy to type the closing bracket outside the formula, with no visual difference.
- Various mathematical notations are visually ambiguous. For instance, $a(b + c)$ would usually be understood as $a \cdot (b + c)$, whereas $f(x + y)$ rather corresponds to a function application. In the semantic editing mode, the user is expected to resolve this ambiguity by hand by entering multiplications using `*` and spaces using `Space`. The multiply/apply ambiguity is one of the main sources of syntax errors, since many users do not pay attention to invisible differences. Similarly, the \wedge glyph could be the “logical and” or the “wedge product”. This “homoglyph” issue will be addressed in more detail in the section on the [semantics of mathematical symbols](#).
- It could be that a text was originally written in \LaTeX or an old version of $\text{\TeX}_{\text{MACS}}$. In that case, the document contains no special indication on matching brackets or the scopes of big operators. For instance, in the formula $[x, y[$, should we interpret the second bracket as a closing bracket? This is indeed the standard french notation for an interval with an open right end. More generally, all problems that we have mentioned so far tend to be present simultaneously when trying to associate semantics to existing documents.

After activation of the semantic editing mode, you may check whether a formula is correct by positioning your cursor inside it and looking at the color of the bounding box of the [semantic focus](#): a green color corresponds to a correct formula and a red color indicates an error in the formula. Alternatively, assuming that the focus is on a mathematical formula, you may select 焦点→首选项→Highlight incorrect formulas, in which all incorrect formulas are highlighted inside red boxes.

For the second kind of “easy-to-make” errors, $\text{\TeX}_{\text{MACS}}$ includes an automatic syntax corrector. Assuming that your cursor is inside a formula, you may use 编辑→Correct→Correct all for the correction of all formulas in your document, or the correction of the current selection. If the versioning tool is activated, then you may use 编辑→Correct→Correct manually to show the differences between the original and the corrected versions. You may then use the versioning tool to go through these differences and select the preferred versions.

The precise algorithms which are used for the correction may be enabled or disabled from 编辑→首选项→数学→手动纠正:

Remove superfluous invisible operators. This algorithm is used in order to remove any superfluous function applications or multiplications. For instance, users who are accustomed to editing ASCII files often type spaces around binary infixes such as addition. Such “function applications” will be removed by this algorithm.

Insert missing invisible operators. In L^AT_EX, multiplications and function applications are never entered explicitly. When importing a L^AT_EX document, it is therefore important to detect and insert missing multiplications and function applications.

Homoglyph substitutions. This algorithm may perform some other useful substitutions of symbols by visually similar, but semantically distinct symbols. For instance, the backslash symbol `\` is replaced by the binary set differences infix (as in $X \setminus Y$), whenever appropriate.

From the 编辑→首选项→数学→自动纠正, you may also select those corrections algorithms which should be applied automatically whenever you open a file. The various corrections are always carried out when importing a L^AT_EX file.

After syntax correction, the remaining errors indicate genuine typos at worst or non standard or non supported notations at best. We also notice that “correct” formulas do not necessarily have the intended meaning. In order to check whether the operators indeed apply to the intended arguments, you should keep an eye on the current focus while typing your formulas.

3.9. SEMANTICS OF MATHEMATICAL SYMBOLS

The mathematical symbols in T_EX_{MACS} all come with a certain number of properties which correspond to their intended meaning. For instance, T_EX_{MACS} is aware that $+$ is an infix operator, whereas $!$ is rather a postfix, and $,$ a separator.

T_EX_{MACS} has special symbols $e = 2.71828\dots$, $\pi = 3.14159\dots$ and i for important mathematical constants, which display differently from the mere characters e , π and i , and which can be entered using the shortcuts `E→→→`, `P→→→` and `I→→→`. We recommend to systematically use these shortcuts.

Inversely, semantically distinct symbols may display in a similar way. For instance, the comma separator, as in $f(x, y)$, is different from the decimal comma, as in $3,14159\dots$. Notice that the two symbols admit different spacing rules. Semantically distinct symbols which are rendered by the same glyph are called *homoglyphs*. Notice that our semantics is purely syntactic: for instance, the $+$ infix is commonly used for addition, but sometimes also for the concatenation of strings. Nevertheless, these two uses do not differ from a syntactical point of view, since the $+$ symbol remains a binary infix with the same precedence with respect to other symbols.

The most confusing homoglyphs are the various invisible symbols supported by T_EX_{MACS}:

- The multiplication, entered by `*`. Example: $a\,b$.
- Function application, entered by `Space`. Example: $\sin x$.
- An invisible separator, entered by `,`. Example: the matrix $A = (a_{ij})$.
- An invisible addition, entered by `+`. Example: $17\frac{3}{8}$.
- An invisible symbol, entered by `.`. Example: the increment $+1$.
- An invisible bracket (mainly for internal use). A matching pair of invisible brackets is entered using `(` and `)`.

Again it is recommended that authors carefully enter these various invisible symbols when appropriate. It is particularly important to distinguish between multiplication and function application, since there is no 100% safe automatic way to make this distinction (we already mentioned the formulas $a(b+c)$ and $f(x+y)$ before).

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ supports two quite general schemes for entering homoglyphs. On the one hand, we often rely on the standard variant system. For instance, \times and $*$ are obtained using $\text{*}\rightarrow$ and $\text{*}\rightarrow\rightarrow$. In table 3.5 we have given the complete list of homoglyphs supported by $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$.

Shortcut	Glyph	Example	Semantics
*		$a\,b$	Multiplication
Space		$\sin x$	Function application
$\text{,}\rightarrow\rightarrow$		$a_{ij}=a_{ji}$	Invisible separator
$\text{+}\rightarrow\rightarrow\rightarrow\rightarrow$		$17\frac{3}{8}$	Invisible addition
$\text{.}\rightarrow\rightarrow\rightarrow\rightarrow$		$+1$	Invisible symbol
$\text{(}\rightarrow$		$\Phi\equiv\forall x, P(x)$	Invisible bracket(s)
 	$ $	$ -x = x $	Absolute value
$\text{ }\rightarrow$	$ $	$\{x\in\mathbb{R} x>0\}$	Separating bar
$\text{ }\rightarrow\rightarrow$	$ $	$\langle a_i^2 a_j^2\rangle$	Extensible middle bar
$\text{ }\rightarrow\rightarrow\rightarrow\rightarrow$	$ $	$11 1001$	Divides relation
,	$,$	$f(x,y)$	Comma separator
$\text{,}\rightarrow$	$,$	$123,456$	Decimal comma
.	$.$	123.456	Decimal point
$\text{.}\rightarrow$	$.$	$\lambda x. x^2$	Dot connector
$\text{*}\rightarrow\rightarrow\rightarrow\rightarrow$	\cdot	$\boldsymbol{v}\cdot\boldsymbol{w}$	Dot multiplication
$\text{.}\rightarrow\rightarrow$	\cdot	$\cdot+1$	Dummy wildcard
:	$:$	$\{x\in E: P(x)\}$	Separator
$\text{:}\rightarrow$	$:$	$x:\text{Int}$	Type satisfaction
$\text{/}\rightarrow$	$:$	$121:11=11$	Division
$\text{\backslash}\rightarrow$	\backslash	$\backslash x$	Backslash
$\text{\backslash}\rightarrow\rightarrow$	\backslash	$\mathbb{N}^>=\mathbb{N}\backslash\{0\}$	Set minus
\&	\wedge	$1=1\wedge 2=2$	Logical and
$\text{*}\&$	\wedge	$dx\wedge dy$	Wedge product

表格 3.5. Homoglyphs supported by $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$.

3.10. 定制数学语义

我们已经尽其所能，支持了大多数经典的数学符号。然而，用户有时候需要自定义非标准语义的数学符号。而且，某些区域也可能需要一些未被支持的特殊符号。

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 提供了一个非常简单的`syntax`原语，以便用户手动覆盖公式的默认句法语义。假定语义编辑功能已经开启，快捷键`^XX`或菜单项插入 \rightarrow 语义 \rightarrow 其它可插入`syntax`原语。它的第一个参数是公式应该显示的模样，参数二则是公式内在的语义。

比如，如果我们输入 \mathcal{R} 作为参数一， $<$ 作为参数二，那么 \mathcal{R} 将被解释为二元关系。更进一步， \mathcal{R} 周围空格分布已经变了，可以认为是在模拟 $<$ 周围的空格分布。在这个特例中，我们也可以使用`math-relation`原语得到等价的结果。大部分标准操作符都可从菜单插入 \rightarrow 语义找到，或者使用`^X`键前缀。特别地，您可以使用`^X Space`简单地忽略一个公式(FIXME)，使用`^X 0`将公式变成普通符号。

`syntax`原语与 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 宏的结合异常灵活强大。比如，考虑这个公式 $C = 1/2\pi i \oint f(z) dz$ 。 $1/2\pi i$ 很有可能应当解释为 $1/(2\pi i)$ 而不是 $(1/2)\pi i$ 。因此我们往往使用常量 $2\pi i$ 。首先，需要定义宏`twopii`

```
<assign|twopii|<macro|<syntax|2\pi i|(2\pi i)>>>
```

用户可以根据自己的偏好将这些宏聚合在一个宏包里面。未来的 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 将提供特定区域所需符号的样式包。

我们注意到定义公式语义的方法往往有多种。比如我们可以这样重定义`twopii`这个宏：

```
<assign|twopii|<macro|2\pi i>
```

我们在 $2\pi i$ 外插入了一对不可见的括号。类似地，在公式

$$e^{\sqrt{x} + e^{\sqrt{\log x} + e^{\sqrt{\log \log x} + \cdots + \log \log \log x} + \log \log x + \log x}},$$

我们可以选中整个公式，然后使用`^X0`赋上一个普通符号的值。然而，更优美的解决方案是，只选中子公式 \cdots ，再赋上普通符号的值。另外的例子是早先提到的符号序列 $++-+-+$ 。可在不同的符号间使用`Space`快捷键插入不可见的分隔符使得这个序列可被正确的解释。

章 4

表格处理

表格实际上为我们提供了一种复杂而通用的对齐标记的方法。它既可以用来呈现数据,也可以用于

计算机程序和网页的排版。 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 提供了丰富的参数以精确控制表格及其单元格的样式。

4.1. 创建表格

你可以使用插入→表格或者以下的快捷键来创建表格：

Meta+T Shift+N T. 创建普通无框表格。

Meta+T ↑N ↑T. 创建居中无框表格。

Meta+T ↑N B. 创建普通有框表格 (block) 。

Meta+T ↑N ↑B. 创建居中有框表格。

在数学模式下，提供如下类表格结构：

Meta+T ↑N M. 创建一个矩阵。

Meta+T ↑N D. 创建一个行列式。

Meta+T ↑N. 创建一个选择。

下面演示的是普通表格，居中有框表格和矩阵。下面的带标题的小表格是通过插入→表格→小表格标题得到的。小表格允许你将多个表格并列排在一行上，而大表格一行只能有一个。通过插入→表格→大表格标题可插入大表格。

boom	tree
hallo	hello
wiskunde	mathematics

表格 4.1. 普通无框表格

boom	tree
hallo	hello
wiskunde	mathematics

表格 4.2. 居中有框表格

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

表格 4.3. 矩阵

还有其他类似的类表格环境，除此之外新的环境也可以由用户自己创建。比如，通过插入→数学→多行公式或者 **Alt+&**，你可以进入 eqnarray^* 环境。在此环境下，关注数学排版的用户可以轻易地对齐一组等式。例如：

$$\begin{aligned} \sin(f(x)g(x))' &= (f(x)g(x))'\cos(f(x)g(x)) \\ &= (f'(x)g(x) + f(x)g'(x))\cos(f(x)g(x)) \end{aligned}$$

刚创建一个新表时，默认大小为最小（一般是 1×1 ），且所有单元格都是空的。通过快捷键 $\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ 和 $\text{^}\downarrow$ 可以插入新列和新行。比如说， $\text{^}\rightarrow$ 在当前光标所在位置的右边插入新列，详情参见下图。你也可以用`Return`在当前光标所在位置的下面插入新行。

$$\begin{pmatrix} d & b \\ c & d \end{pmatrix} \longrightarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

图 4.1. 一个在矩阵中插入新列的例子。如果光标在左边的矩阵所示位置处，此时按下 $\text{^}\rightarrow$ 所得到的即右边的矩阵。

4.2. 格式化模式

在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中，可以为任意单元格设定格式。比如，你可以为某个单元格设定背景颜色，也可以使整列水平居中。默认地，格式化命令在当前所在单元格上生效，你可以在单元格 \rightarrow 操作模式中更改操作模式。下列模式可供选择：

^TMC . 只对所在单元格生效

^TMH . 只对所在行生效

^TMV . 只对所在列生效

^TMT . 对整个表格生效

也可以使用鼠标选中一个矩形区域，再在这个区域上执行操作。

4.3. 指定单元格和表格对齐

在水平或垂直方向上对齐单元格是最常用的格式化操作。可以使用 $\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ 和 $\text{^}\downarrow$ 这些快捷键迅速对单元格执行左，右，上，下对齐。

您也可以在菜单单目单元格 \rightarrow 水平对齐和单元格 \rightarrow 竖直对齐中完成上述操作。相应的快捷键是 ^THx 和 ^TVx 。

类似地，您或许还需要调整表格及其周遭文字相对位置。此时请选择菜单项表格 \rightarrow 水平对齐和 表格 \rightarrow 竖直对齐。相应的快捷键是 $\text{^T}\uparrow\text{Hx}$ 和 $\text{^T}\uparrow\text{Vx}$ 。

请您注意表格和单元格作用对象的区别，以及上面快捷键中的 x 的含义。 x 本身并不是一个键，分别用`L` `R` `T` `B` `C`替换时，各自表示左、右、上、下和居中对齐。

4.4. 指定单元格和表格的大小

使用表格 \rightarrow 单元格宽度 \rightarrow Set width和表格 \rightarrow 单元格高度 \rightarrow Set height指定单元格的宽度和高度。实际上，宽度（或者高度）由以下三种模式指定：

最小模式. 实际的宽度=指定宽度和单元格中内容宽度的较小值

固定模式. 直接指定实际宽度

最大模式. 实际宽度=指定宽度和单元格中内容宽度的较大值

边界和补白决定了单元格和其中内容的相对位置。

使用表格→Special table properties指定整个表格的宽度和高度。特别地，你还可以使表格正好延伸至与段落等宽。当你对整个表格指定宽度或者高度时，还可以使用表格→Special cell properties→分配未使用空隙。默认地，未使用空隙将被平均分配。

4.5. 边界、补白和背景色

您可以自由指定单元格上下左右四个方向上的边界宽度和补白大小（参见单元格→边界）。相应的快捷键分别是`⌘^TB x`和`⌘^TP x`。

单元格默认的边界宽度是`1ln`，即当前字体的标准宽度（比如分式中间那一行的宽度）。水平方向上默认的单元格补白大小为`1spc`，即当前字体下空格的宽度。垂直方向上默认的单元格补白大小为`1sep`，即当前两个闭合区域之间标准的最小距离。

单元格的背景色可在菜单项单元格→背景色中指定。

整个表格的边界和补白可在表格→边界和表格→补白中指定。其快捷键和相应的单元格的快捷键相同。特别注意的是，在这种情况下，补白是在边界之外。

4.6. 表格的高级特性

在菜单中，您还会发现许多表格的特性。下面是这些特性的简介：

- 改变单元格的延伸属性，以使其边界侵占其右或者其下单元格，输入`n+1`则侵占其右或其下`n`个单元格
- 在单元格中创建子表格
- 文本高度修正，以使基准对齐
- 单元格内容的水平字符连接以及整个表格的垂直字符连接
- 将多行多列单元格合并，这样合并后大单元格的边界就会是剩下单元格的边界了
- 反激活表格，以显示其“源码”
- 指定表格的模板单元格。这样，此后新建的单元格格式属性与模板的相同
- 指定表格的最小和最大尺寸，这对接下来的编辑非常有帮助。（在撰写表格宏时也非常实用）

目前，所有的表格其实大多是在`tabular`，`block`，`matrix`等环境下。在撰写您自己的表格宏时，可以使用表格→特定→Extract format从选中的表格导出格式。

章 5

链接和内容自动生成

5.1. 创建标签， 链接和参考

通过 **Meta+!** 或者插入→链接→标签创建新的未激活标签，通过 **Meta+?** 或者插入→链接→参考创建对应的参考。在输入标签或者参考的名字后，莫忘按下回车键**激活**。在编辑参考的名字时，按下 **Tab** 可以自动补全。对小节加标签时，建议加在小节标题后。给单行公式加标签时，建议加在公式前。给多行公式加标签时，建议加在公式编号后。回顾一下，按下 **Ctrl+#** 可以退出或进入编号环境。

通过 **%^I >** 或者插入→链接→超链接插入超链接。第一个域填写的是相关的文本，激活后显示为蓝色。第二个域包含了超链接。超链接可以是本地文档的路径，也可以是网络上文档的地址。使用 **#label** 作为超链接则会指向当前文档中的标签，使用 **url#label** 则指向url所定位到的文档中标签的位置。

类似地，通过 **%^I *** 或者插入→链接→动作可以把一个动作关联到一段文本或者图片上。第二个域填写的是一个Guile/Scheme脚本命令。回车激活后，双击这个动作链接，命令就被执行了。为了系统安全，有些脚本会被拒绝执行。**TEX_{MACS}**默认会弹出对话框询问是否执行，但可以在首选项→安全中更改这一默认行为。注意，这条Guile/Scheme命令

```
(system "shell-command")
```

会将shell-command作为shell命令执行。

通过 **%^I I** 或者插入→链接→包含文件可以直接将其它文档的内容包含到当前文档中。这意味着，如果你在那些被包含的文档中做了更改，那么这些更改会自动地更新到当前的文档中。

5.2. 插入图片

通过插入→图片在文档中插入图片，**TEX_{MACS}**支持这些图片格式：**ps**，**eps**，**tif**，**pdf**，**pdm**，**gif**，**ppm**，**xpm**和**fig**。**TEX_{MACS}**使用**gs**即ghostscript来渲染postscript图像。如果你还没有安装ghostscript，可从这个页面下载：

www.cs.wisc.edu/~ghost/index.html

目前其他格式的文件是通过**tiff2ps**，**pdf2ps**，**pnmtops**，**giftopnm**，**ppmtogif**，**xpmtoppm**这些脚本转换的。如果你的系统上没有安装这些脚本，请联系你的系统管理员。

图片默认是以实际大小显示，并和底部对齐。图片的宽度、高度以及对齐时偏移量可在插入图片时弹出的对话框中指定。

- 如果指定了新的宽度，而没有指定高度，则图片会保持宽度和高度的比例（反之亦成立）。比如指定宽度为**1par**就可以让图片的宽度正好等于段落的宽度，而高度会依比例自动调整。
w 和 **h** 可以作为指定图片大小的单位，比如 **2w** 表示是默认宽度的两倍，而 **2h** 则表示默认高度的两倍。

- 在指定图片的对齐的时候，可以用 w 和 h 表示宽度和高度显示的大小（注意此时这两个单位已经不再表示图片的实际宽度和高度了）。例如，给 y 方向上的偏移赋值为 $0.5h$ ，将使图片在竖直方向上向上平移一半的高度。

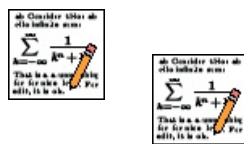


图 5.1. 左图的偏移为 $(0w, 0.5h)$ ，右图的偏移为 $(0.5w, 0h)$

关于如何编辑以及插入带有 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 公式的图片，请参考本文档英文原文的最后一段。

5.3. 生成目录


只要你以正确的方式撰写好文档，生成目录非常容易。只要将你的光标置于目录要生成的位置，然后点击插入→自动→目录。

接着是生成目录，首先要让文档分页显示，在文档→页面→类型中选择Pager即可，这样每一页就有了一个可供引用的页码。再点击文档→更新→全部或者文档→更新→目录就生成了目录。注意：可能需要您要重复更新多次，直到目录没有变化为止。因为每一次更新后，页码都可能会由于目录大小的改变而变化！


5.4. COMPILING A BIBLIOGRAPHY

Editing files with bibliographic entries

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ uses the $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ model for its bibliographies. Manuals about $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ can easily be found at various places on the web. $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ files can either be entered and edited using $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ itself or using an external tool. Some external tools offer possibilities to search and retrieve bibliographic entries on the web, which can be a reason to prefer such tools from time to time. $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ implements good converters for $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ files, so several editors can easily be used in conjunction.

The built-in editor for $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ files is automatically used for files with the `.bib` extension. New items can easily be added using 插入→Database entry. When creating a new entry, required fields appear in dark blue, alternative fields in dark green and optional fields in light blue. The special field inside the header of your entry is the name of your entry, which will be used later for references to this entry. When editing a field, you may use  to confirm it and jump to the next one (blank optional fields will automatically be removed when doing this). When the cursor is inside a bibliographic entry, additional fields may also be added using 焦点→插入行(在上方) and 焦点→插入行(在下方).

$\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ contains a few unnatural conventions for entering names of authors and managing capitalization inside titles. When editing $\text{BIB}_{\text{T}_{\text{E}}\text{X}}$ files using $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$, these conventions are replaced by the following more user friendly conventions:

- When entering authors (inside “Author” or “Editor” fields), use the `name` tag for specifying last names (using 插入→Last name or ) For instance, “Albert Einstein” should be entered as “Albert EINSTEIN” or as “A. EINSTEIN”. Special particles such as “von” can be entered using 插入→Particle. Title suffices such as “Jr.” can be entered similarly using 插入→Title suffix.

- When entering titles, do not capitalize, except for the first character and names or concepts that always must be. For instance, use “Riemannian geometry” instead of “Riemannian Geometry” and “Differential Galois theory” instead of “Differential Galois Theory”.

Inserting citations and compiling bibliographies

Assuming that you have created a `.bib` file with your bibliographic references, the mechanism to automatically compile a bibliography is the following:

- Use 插入→链接→引用 and 插入→链接→Invisible citation to insert citations, which correspond to entries in your `.bib` file.
- At the place where your bibliography should be compiled, click on 插入→自动→参考文献. At the prompt, you should enter a `bibtex` style (such as `plain`, `alpha`, `abbrv`, etc.) and your `.bib` file.
- Use 文档→更新→参考文献 in order to compile your bibliography.

Notice that additional BiBTeX styles should be put in the directory `~/TeXmacs/system/bib`.

5.5. GENERATING AN INDEX

For the generation of an index, you first have to put index entries in your document using 插入→链接→索引项. At a second stage, you must put your cursor at the place where you want your index to be generated and click on 插入→自动→索引. The index is then generated in a similar way as the table of contents.

In the 插入→链接→索引项 menu, you find several types of index entries. The simplest are “main”, “sub”, “subsub”, which are macros with one, two and three arguments respectively. Entries of the form “sub” and “subsub” may be used to subordinate index entries with respect to other ones.

A complex index entry takes four arguments. The first one is a key how the entry has to be sorted and it must be a “tuple” (created using `%^I <`) whose first component is the main category, the second a subcategory, etc. The second argument of a complex index entry is either blank or “strong”, in which case the page number of your entry will appear in a bold typeface. The third argument is usually blank, but if you create two index entries with the same non-blank third argument, then this will create a “range” of page numbers. The fourth argument, which is again a tuple, is the entry itself.

It is also possible to create an index line without a page number using “interject” in 插入→链接→索引项. The first argument of this macro is a key for how to sort the index line. The second argument contains the actual text. This construct may be useful for creating different sections “A”, “B”, etc. in your index.

5.6. 编制术语表

术语表的编制与索引的编制类似，但无需对条目排序。“常规”的术语条目只包含若干文字和一个自动生成的页码。“详细”的术语条目还包含额外的参数，以解释该术语。“重复”的术语条目则用于显示该术语第二次出现的页码。

5.7. MULTIPLE EXTRACTIONS

$\text{\TeX}_{\text{MACS}}$ allows users to create multiple bibliographies, tables of contents, indexes, etc. inside the same document. Let us explain how to do this for bibliographies; the procedure is similar for other types of automatically generated content.

First of all, every separate bibliography is identified by a “name”. The default name of the bibliography is `bib`. Using 插入→链接→Alternate→参考文献, it is possible to specify a different bibliography (than the default one) for a certain region of text.

For instance, to specify that a given citation should appear in a second bibliography with name `bib2`, you should proceed as follows:

- Click on 插入→链接→Alternate→参考文献 and enter `bib2` on the prompt. This will insert an empty `with-bib` tag into your document, with the cursor inside.
- Inside this `with-bib` tag, enter your citation, using 插入→链接→引用.

If needed, the `with-bib` tag can be made to span over a large portion of text. All citations inside this span will be put into the bibliography with name `bib2`.

The bibliography `bib2` itself should be created in a similar way: first click on 插入→链接→Alternate→参考文献 and enter `bib2` on the prompt. Next insert the bibliography as usual, *via* 插入→自动→参考文献. Now do 文档→更新→全部 as many times as need in order to generate the bibliography and get all links right.

5.8. BOOKS AND MULTIFILE DOCUMENTS

When a document gets really large, you may want to subdivide it into smaller pieces. This both makes the individual pieces more easily reusable in other works and it improves the editor’s responsiveness. An entire file can be inserted into another one using 插入→链接→包含文件. In order to speed up the treatment of included documents, they are being buffered. In order to update all included documents, you should use 工具→更新→包含文件.

When writing a book, one usually puts the individual chapters in files `c1.tm`, `c2.tm` until `cn.tm`. One next creates one file `book.tm` for the whole book, in which the files `c1.tm`, `c2.tm` until `cn.tm` are included using the above mechanism. The table of contents, bibliography, etc. are usually put into `book.tm`.

In order to see cross references to other chapters when editing a particular chapter `ci.tm`, one may specify `book.tm` as a “master file” for the files `c1.tm` to `cn.tm` using 工具→项目→关联主文件.... Currently, the chapter numbers themselves are not dealt with by this mechanism, so you may want to manually assign the environment variable `chapter-nr` at the start of each chapter file in order to get the numbering right when editing.

章 6

内置作图工具

除了可以插入其他程序生成的图片之外, $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 还内置了基本的作图工具以供使用。尽管这套工具较大多数专业的作图工具而言, 功能稀少, 然而其优点在于, 它是与 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 完美整合在一起的。比如, 很容易就能在图片中插入文本、数学内容和超链接。又如, 内置工具所作之图往往更美观, 因为它们和周围的文本使用的是一样的字体和线宽。

6.1. 开始作图










点击插入→图片→绘制图形开始作图。在某些情况下, 你也许想在已有的图片(或者其他内容)上绘图。此时, 你可以先选中该区域, 再点击插入→图片→在选择区域内绘制。


插入图片的默认宽度是整个段落宽度。使用快捷键 $\text{Ctrl}+\leftarrow$, $\text{Ctrl}+\rightarrow$, $\text{Ctrl}+\uparrow$, $\text{Ctrl}+\downarrow$ 可以调整图片的尺寸。使用 $\text{Ctrl}+\text{Shift}+\leftarrow$, $\text{Ctrl}+\text{Shift}+\rightarrow$, $\text{Ctrl}+\text{Shift}+\uparrow$, $\text{Ctrl}+\text{Shift}+\downarrow$ 调整得更快。你也可以通过插入→几何→大小精确指定图片的长和宽。绘图完毕, 你可以使用插入→几何→剪裁自动剪裁掉绘图内容上下左右的空白(当然, 会留下一部分作为补白)。



对于技术类的图, 用坐标纸来绘制是再好不过的了。点击插入→网格→类型→直角坐标即可。在菜单插入→网格中, 还可以调整坐标和网格线的颜色以及网格的单位长度。打印时, 网格默认会被打印出来。如果不希望打印网格, 那么打印前需要先移除网格。



$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 默认将网格的原点置于屏幕的中央并且使用 1cm 作为单位长度。使用 \leftarrow , \rightarrow , \uparrow , \downarrow 可以控制图片在网格上的移动, 使用 $\text{Shift}+\leftarrow$, $\text{Shift}+\rightarrow$, $\text{Shift}+\uparrow$, $\text{Shift}+\downarrow$ 则可以加大移动的步伐。通过插入→几何→单位可以改变网格的单位长度。使用 $+$ 和 $-$ 可以放缩图片, 对应的菜单项是插入→几何→放缩。



6.2. 插入图形对象



插入新图片或点击进入已有图片后, 依赖于模式的第二行工具栏会显示一系列图形模式的图标。特别地, 其中的第二组图标 , , , , , , , ,  是一系列可以点击并插入图片的图形对象。 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 当前支持下列这些原生的图形对象:

点. 点击  或者插入→点, 你就能在图片上通过鼠标左键插入点。

线和多边形. 点击  或者插入→线后, 每一次左击都将插入一个虚拟的点, 直线将顺次连接这些点, 双击可以插入最后一个点, 并结束本次绘制。点击  或者插入→多边形后, 行为与插入直线类似, 只是最后一个点插入后, 将有一条额外的直线使之与第一个点相连。

样条曲线和闭合样条曲线.  即插入→样条曲线,  即插入→闭合样条曲线。插入曲线的操作方式与插入直线的方式一模一样, 不同之处在于, 连接这些点的是样条曲线。

圆弧和圆圈.  即插入→圆弧,  即插入→圆圈。平面上不在同一直线上的三点可以确定一个圆。所以指定该对象后, 只需要左击插入三个点即可。

文本和数学.  即插入→文本,  即插入→数学。指定该对象后, 在图片上左击选中插入的位置, 再编辑即可。值得注意的是, 这里的文本其实不限于于纯粹的文本, 其本质是一个容器, 大部分 \TeX -MACS内容都可以栖身其中。

这些基本对象的典型例子如下图所示:

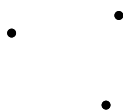


图 6.1. 点

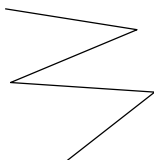


图 6.2. 线

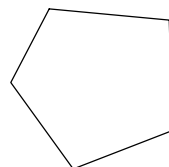


图 6.3. 线段

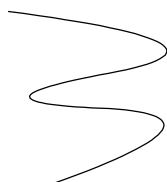


图 6.4. 样条曲线

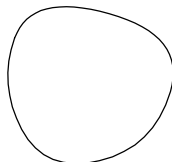


图 6.5. 闭合样条曲线

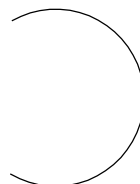


图 6.6. 圆弧

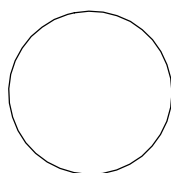


图 6.7. 圆圈

Hello

图 6.8. 文本

$$e^{\pi i} = -1$$

图 6.9. 数学

6.3. 编辑图形对象

所有支持插入新对象(点, 线, 多边形等)的模式也同样支持直接编辑已存在的对象。一旦鼠标移动到一个对象上方, 该对象的控制点就会被自动高亮, 并一些编辑操作, 如下所示:

移动控制点. 当鼠标足够接近一个控制点时, 你可以用鼠标左键拖拽控制点到某处。

插入新的控制点. 对于那些拥有一个以上控制点的对象, 比如线、多边形、样条曲线和闭合样条曲线, 可以在边上插入新的控制点。首先将鼠标移动到目标边上, 此时目标边两端的控制点将高亮显示。然后按下鼠标左键不放并拖拽, 在合适的地方松开。

移除控制点. 使用鼠标中键可以移除当前控制点。也可以用 **Del** 或者 **Backspace**。

移除整个对象。使用鼠标中键并同时按下 \downarrow 将移除当前高亮的整个对象。

在编辑过程中，你可能注意到了， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 会试着自动将鼠标附着到控制点、对象的边缘、曲线的交点和网格上的点。这就使得我们能够快速绘制复杂的精确图形，不仅仅精确到一两个像素（指那种发大或者打印后很丑的精度）。在文本或者数学公式四周有八个隐藏的控制点， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 会尽可能地把鼠标附着到这些点上。这使得图6.10的绘制变得容易得多。

图形对象在图片上的绘制依照一种栈的顺序，这种顺序会影响到重叠对象的覆盖和被覆盖。使用 \uparrow 和 \downarrow ，你可以使当前高亮的对象上浮或者下沉。同样地，控制点也可能由于被其它控制点覆盖而无法访问。在这种情况下，你可以使用 \rightarrow 遍历在当前光标所在位置下的所有控制点。

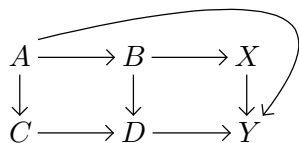


图 6.10. 使用数学公式方框四周八个控制点的附着特性作图的一个例子。注意该图已经被剪裁过了。

6.4. 样式属性详述

每一种基础图形对象都有对应的一些影响渲染的样式属性。这些属性如下所示：

颜色。 该属性用于指定颜色，应用于任何图形对象。

填充色。 该属性用于指定对象的填充色，应用于所有图形对象，文本和数学除外。

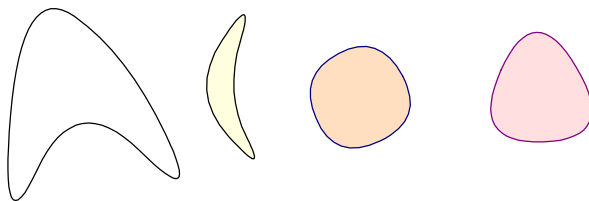


图 6.11. 一些不同颜色和填充色的闭合样条曲线示例。

不透明度。 该属性用于指定对象从0%到100%的不透明度，也是应用于所有图形对象。默认的不透明度为100%，不透明度越低，对象就越透明。

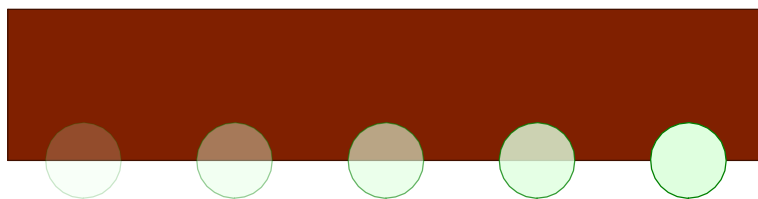


图 6.12. 覆盖在另一个对象上的同一个对象不透明度逐渐增加的一个过程的示例。

点形状。 支持这些不同的形状：圆盘、圆圈和方框。



图 6.13. 黑色和红色填充色的不同的点形状。

线宽度. 线宽度属性应用于所有曲线（包括线段，多边形，样条曲线，闭合样条曲线，圆弧和圆）。其默认大小为 $11n$ ，即数学公式分式中线段的宽度。你可以指定其为任何 $T_{E}X_{MACS}$ 长度。

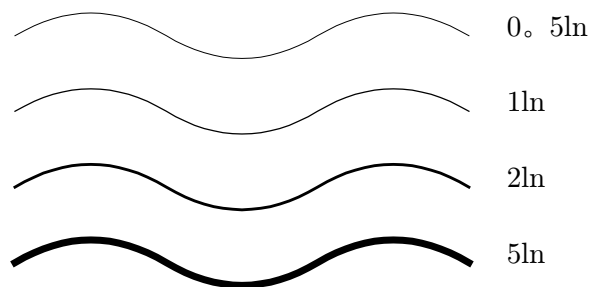


图 6.14. 使用不同线宽度的同一条曲线。

线组成. 可以在焦点→Line dashes中指定曲线的组成样式。

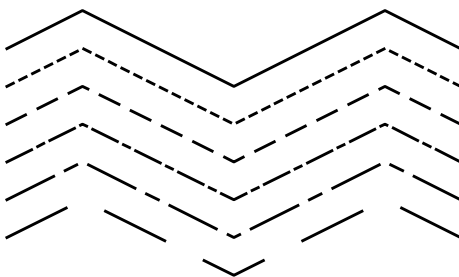


图 6.15. 使用不同组成样式的同一条曲线。

线箭头. 可以使用焦点→线箭头指定曲线两端的箭头样式。

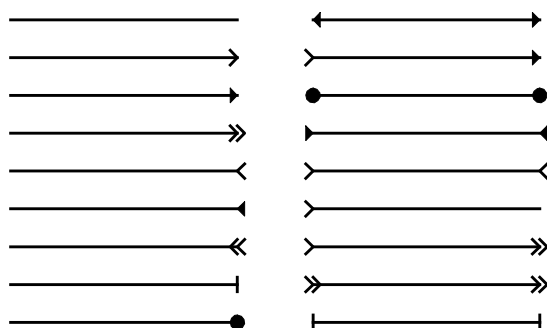


图 6.16. 两端使用不同类型箭头的同一条线段。

文本对齐. 对于文本和数学方格，可以指定其水平和竖直对齐的属性，如下所示：

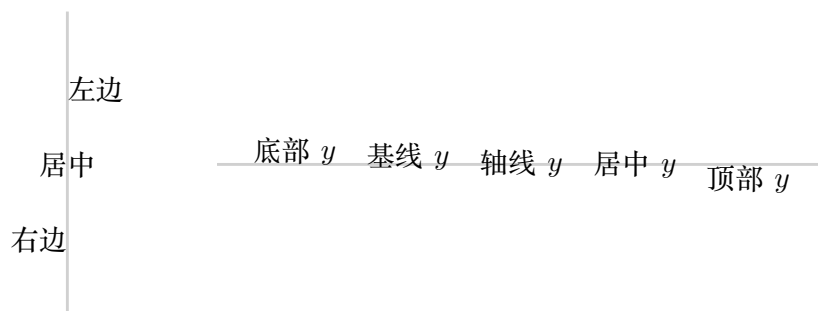




图 6.17. 文本方格的水平和垂直方向上对齐的图解。


6.5. 编辑对象的组合


第二栏工具栏最右端的图标可用于编辑图形对象的组合。在组合编辑模式中，你可以右击选中或者取消选中对象。你也可以拖拽鼠标右键通过矩形选框选中其中的所有对象。此时左击，则当前的操作将作用在所有选中的对象上。


所支持的对组合的操作如下所示：

改变属性.  即插入→Set properties。左击对象后，焦点栏中的当前属性就应用到了选中对象上。

移动对象.  即插入→Move objects。选中对象会跟着鼠标移动直到你左击。

缩放对象.  即插入→Resize objects。选中对象会跟着鼠标缩放直到你左击。

旋转对象.  即插入→Rotate objects。选中对象会跟着鼠标旋转直到你左击。

组合或者打散.  即插入→分组/打散。左击之后，选中对象将被组合在一起。若是组合对象，则左击之后将被打散。

在组合编辑模式中，复制和粘贴操作仍然适用。

章 7

进阶布局特性

7.1. 文档流

复杂的文档通常包含脚注及浮动对象（指区别于正文文本的对象）。实际上，那些复杂文档是分流处理的，其一为正文文本，其二为脚注，其三为浮动对象，其四是两栏文本。不同的文档流，其分页特征各不相同。

使用格式→页面插入→脚注插入脚注。 使用段落→栏数更改文本的栏数。

7.2. 浮动对象

浮动对象可以在页面上相对正文独立移动。图片或者表格太大，而无法完美插入正文，此时便可使用浮动对象解决问题。使用格式→页面插入→浮动对象可插入浮动对象。

您也可以分别使用格式→页面插入→浮动图形或者格式→页面插入→浮动表格直接插入图片或者表格。有时候，需要将多个小图片或者多个小表格合并成一个浮动对象，此时分别插入→图片→小图片标题或者插入→表格→小表格标题。

创建浮动对象后，您可用插入→Position float来移动对象。您可以指定它是在页面顶部、底部、文本中还是在下一页。默认地，浮动对象可在任何位置出现。然而，浮动对象不可能在离顶部或者底部小于三行文本距离处中出现。

7.3. 分页

用户可在文档→页面→断行中精确控制分页。在其子菜单算法中,您可以指定分页使用的算法。专业选项适用于打印，但是在分页模式下会使编辑器变慢。粗略选项速度最快，适中选项对普通文档的分页还算专业。多栏文档应使用专业选项，并且不可避免地显得缓慢。

您还可以在子菜单极限中使得分页算法可以在特定情况下增大或者减小页面长度。段落间隔的可伸展性可在弹性子菜单中指定。默认系数是1；偏小的系数会使段落间隔控制更加严格，而使分页质量降低。

章 8

编辑工具

本章我们讨论 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中的通用编辑方法。除了经典的特性，如剪切和粘贴，搜索和替换等， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 还充分挖掘文档的结构性以提供额外的优秀功能。这些功能的代表是结构化光标移动和结构化变元。我们也在传统的功能如搜索和替换中寻求结构化的美感。例如，当您搜索数学模式中的 x ，结果只会匹配同样是数学模式中的 x 。

8.1. 剪切与粘贴

使用鼠标左键，您可以选中文本。选中后，点击编辑→剪切，将会删除选中文本；而点击编辑→复制则会留下选中文本。此时，再点击编辑→粘贴，您可以将此所选中的文本粘贴至光标所在位置。注意，您也可以使用鼠标中键快速复制选中区域，只需选中文本，在将光标置于某处，按下鼠标中键，此时此前选中文本被粘贴至此处。

选中文本后，您还可以改变选中区域的属性。比如，使用鼠标左键选中一些黑色文字，点击格式→颜色→红色，将选中文字变成红色。又如，选中公式后点击插入→分式，则此公式将成为新插入分式的分子。

与其它应用程序之间通过复制和粘贴机制通信时，文本是以 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的数据格式传输的。这意味直接从外部粘贴有格式的文本至 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ ，容易导致格式混乱。您可以通过文件→导入和文件→导出规避这种风险。默认地，复制和粘贴使用文本缓冲区。使用编辑→粘贴自和编辑→复制到，您可以指定选中文本的来源和去向。

使用键盘选中文本有以下两种方法：其一是按中 ␣ 不放，使用 ← ， → 在文本上移动。其二是使用 ␣Space 固定一个起始点，再移动光标，移动过程中，起始点和光标所在点之间的区域即为选中区域。按下 ^G ，您可以退出这种选中的状态。中文用户在GNU/Linux环境下使用的输入法切换和 ␣Space 产生冲突，建议自定义其快捷键。

注意使用 ␣Space 还可以用于结构化选中。按下 ␣Space 两次，当前光标所在处的单词将被选中。再次按下 ␣Space ，选中的范围将依照结构逐层扩大，比如一开始选中的是单词，下一步可能是段落。最终，将选中全文，此时，再次按下 ␣Space ，选中区域被清空。这里的 ␣Space ，也可以用鼠标左键替代。

8.2. 搜索和替换

使用 ^S 或者编辑→查找开始搜索文本。在搜索中，搜索字符在底端状态栏的左侧显示。你输入的每一个字符都将附加到搜索字符的末尾，文本中匹配的字符以细线红框围绕显示。在搜索过程再次按下 ^S ，搜索将继续进行。如果在下文中没有找到匹配项， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 什么也不做，底端的状态栏会显示相关信息，此时再次按下 ^S ，搜索将从正文起始处重新进行。按下 Backspace 可在搜索文本中撤销上一个字符输入。

通常情况下，搜索方向为自光标处往下。使用 ^R 可以自光标处往上搜索。在搜索中，只匹配相同模式和相同语言下的文本。模式和语言在您开始搜索时有光标所在处决定。这意味着，如果你在数学模式中搜索 x ，那么文本中普通的 x 是不会被匹配到的。当前的局限之处在于，所搜索的文本只能是普通文本而不是数学符号或者更加复杂的结构化文本。

使用`^=`或者编辑→替换可查询并替换文本。开始替换时，在底端先请您输入被替换文本，再输入替换文本。每一次匹配到文本时，底端状态栏询问您选择(y)替换文本(n)不替换文本(a)替换此处及所有匹配处三者之一。和搜索一样，查询-替换命令对模式和语言敏感。

当前的搜索和替换系统还十分基础，仅仅是对普通文本的简单处理。未来，我们计划对结构化文本实现一种更加强大的搜索和替换机制。

当前，您还可以用以下方式搜索和替换文档中的文本片段：在您的文档中某处或者是其他的窗口中选中你想搜索的文本，点击菜单项编辑→复制到→查找，接下来按`^S`便可用选中的文本搜索。类似地，选中文本，点击菜单项编辑→复制到→替换，再次选中另一个文本，接下来按`^=`就能以前后两次选中的文本为参数执行替换操作。

8.3. 拼写检查

在您的系统上已安装`ispell`的前提下，使用`⌘;`或者编辑→拼写可以检查您的文本中是否有拼写错误。使用前，还请您先检查一下您所用语言的字典是否已经安装。

对全文或者选中文本启用拼写后，您将会在底端收到错词的修正提示，有以下选项供您选择：

- a). 接受错词，这样，以后的检查中将忽略该词
- r). 使用您输入的单词替换该词
- i). 申明该词其实是正确的，并将该词添加到您的个人词典
- 1-9). 对错词的推荐修正选项

请注意，`ispell`只管拼写对错，并不检查句子的语法正确与否。

启动拼写检查后，设定语言是当前光标所在处的语言或者是选中区域的起始处的语言。拼写检查只会检查设定语言的拼写问题。这意味着，如果文档中有多种语言，您需要对每一个部分分别启用拼写检查。

8.4. 撤销和恢复

`TeXMACS`只在运行时支持撤销和恢复操作。您可以使用菜单项编辑→撤销和编辑→恢复以及相应快捷键`⌘Z`和`⌘⇧Z`来控制您对文档的改变。

为了节约内存空间，默认的撤销和恢复操作限于百步之内。在您的配置文件中添加

```
(set-maximal-undo-depth 1000)
```

这行代码，可以深度改为1000。若是将深度赋值为负值，则任何数量的操作都能被撤销。

8.5. 结构化编辑

作为普适法则，几乎所有的结构化编辑命令是上下文相关的，从文档源代码的角度上讲，结构化编辑的效果由光标所在结构最深处的标签所决定。当您选中某些文本时，焦点会定位于选中结构最深处的标签。在诸如相似标签间导航的结构化操作中，当前的焦点会临时设置为其它某处。当前的焦点由光标所在结构中最深处的蓝绿色方框显式指示。

比如，结构化插入命令 $\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ 和 $\text{^}\downarrow$ 在表格和树中含义不同。在表格中是插入行与列(见图 8.1)。在树中是插入新节点(见图 8.2)。倘若您的光标在表格中的树中，则从文档的源代码上说，最里面的tag实际上是树，所以使用结构化插入将优先插入节点，而不是行与列。

在大多数情况下，结构化操作下标签的默认行为已预定义。比如 $\text{^}\leftarrow$ 或 $\text{^}\rightarrow$ 是在标签的左边或者右边插入一个新的参数。

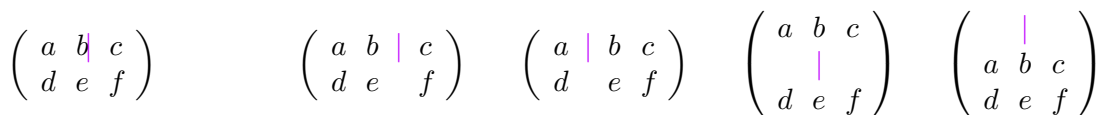


图 8.1. 假定光标在最左矩阵的处。那个右边的四个矩阵按顺序分别对应往左边($\text{^}\leftarrow$)插入一新列，往右边($\text{^}\rightarrow$)插入一新列，往下边($\text{^}\downarrow$)插入一新行，往上边($\text{^}\uparrow$)插入一新行。

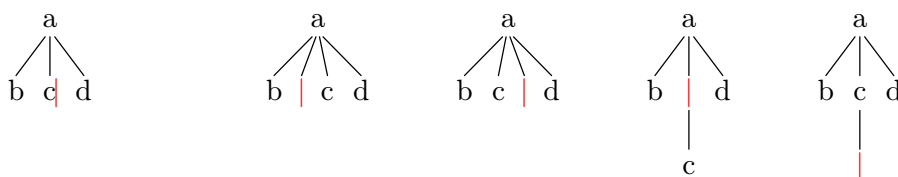


图 8.2. 假定光标位于最左边树的处。那么右边的四棵树分别对应 $\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ ， $\text{^}\downarrow$ 后的效果

类似地，在矩阵中， $\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ 和 $\text{^}\downarrow$ 可用于插入新的首列，末列，首行，末行。 $\text{^}\leftarrow$ 和 $\text{^}\rightarrow$ 可用于结构化后向或前向删除。在矩阵中，这将删除光标前或者后的一列(见图8.3)。使用 $\text{^}\leftarrow$ 或者 $\text{^}\rightarrow$ 可删除光标所在标签外的环境。

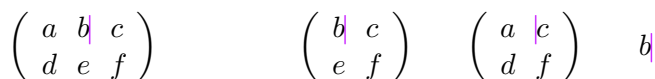


图 8.3. 假定光标在最左矩阵的处。中间两个矩阵对应按下 $\text{^}\leftarrow$ 和 $\text{^}\rightarrow$ 后的结果。最右边的矩阵是按下 $\text{^}\leftarrow$ 或者 $\text{^}\rightarrow$ 后的效果，其作用是用单元格内的内容替换掉整个矩阵结构。

8.6. 结构化光标移动

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 实现了三种主要的结构化光标移动的机制





1. 整个文档结构的遍历
2. 相似标签的遍历
3. 最深处标签的移动

大部分结构化光标移动的快捷键支持与 $\text{^}\leftarrow$ 组合适用，以在移动的同时选中文本。





To do: customizing the behaviour



结构化遍历文档.

$\text{^}\leftarrow$ ， $\text{^}\rightarrow$ ， $\text{^}\uparrow$ and $\text{^}\downarrow$ 用于结构化遍历整个文档。 在普通的文本中， $\text{^}\leftarrow$ 和 $\text{^}\rightarrow$ 用于按词移动，而 $\text{^}\uparrow$ 和 $\text{^}\downarrow$ 用于按段落移动。







在其它标记中，和用于移动到文档中光标可访问的位置，在您持续在文本间按词移动时，则没有这个效果。和行为更显上下文相关性，在矩阵中，可用于将某行向上或者向下移动。





结构化遍历相似标签.

此类光标移动是为了方便在文档中访问与当前最深处的标签相似的标签。, 对应与上一个和下一个，而和对应于第一个和最后一个。



比如，当光标处于某节的标题中，用可以跳转到前一节的标题中。用S则可以跳回先前的标签处。



最深处标签的移动.


也可以在最深处的标签内移动而不必退出。, , 和对应于跳转到光标的前一个，后一个，第一个，最后一个参数处。而或用于从左边或者右边跳出标签。

在特定场景下，这些默认行为可能无法生效。比如，在表格或树中，这些快捷键对应于按单元格或者按节点的光标移动。除此之外，垂直方向上则使用, , 和移动光标。

8.7. 结构化变元




在创建定理，公式或者列表这些环境的时候，我们经常需要在事后更改环境。您可以使用或者循环选择最深处标签的*结构化变元*。前者顺序，后者逆序。

假定您的光标在定理中，那么不断按下会在命题，引理，推论，证明，定理间轮换。而不断按下快捷键则是逆向轮换:定理 → 证明 → 推论 → 引理 → 命题 → 证明。



在数学公式这个例子中，用于将行内公式 $a^2 + b^2 = c^2$ 转化为单行公式：


$$a^2 + b^2 = c^2$$


同时顾及尾随空格(trailing space)和标点符号的处理。


$\text{\TeX}_{\text{MACS}}$ 还提供了#快捷键以供您在编号模式和普通模式间切换。此快捷键在定理，注意，表格，公式等环境适用。注意#和的区别，前者显示和隐藏编号，而后者是在一组可用环境如列表项目(圆点，短线，箭头等)间轮换。


8.8. 移动和缩放对象


键前缀用于移动和缩放对象。比如说，在表格中的单元格内部，使用可使单元格更靠右。在使用格式→空白插入的空格中，使用该键可增加空格的宽度。下面是结构化移动和缩放快捷键的通用功能描述：


. 减小对象的水平宽度，或者更向左对齐。


. 增加对象的水平宽度，或者更向右对齐。


. 减小/增加对象的竖直高度，或者更向底部对齐。


. 减小/增加对象的竖直高度，或者更向顶部对齐。



. 减小对象的水平位移或者对齐。



. 增加对象的水平位移或者对齐。

. 减小对象的垂直位移或者底部对齐。

. 增加对象的垂直位移或者顶部对齐。

. 将几何性质(大小, 位置, 对齐)重设为默认值。



, . 在可用长度单元间循环以指定其几何结构。

, . 减小或者增加移动或者缩放的步长。

下面是一些特定标签下的特定说明:

空格. 空格指的是格式→空白中的水平或者竖直空格。若要使用快捷键, 必须把光标置于空格标签之后。

盒修饰器. 格式→变换菜单中的move, shift, resize和clipped标签

动画. 动画的长度可用和更改。

图片. 其大小和对齐可供更改。







8.9. 版本控制工具

与他人合作撰写文档时, 会产生这样的需求, 即审阅他人对文档的更改, 然后或接受或舍弃或更正之。在 编辑→首选项→工具→版本菜单启用版本控制菜单后, 菜单栏中会新增版本这一项。下面我们来谈谈这个菜单是如何使版本控制的过程更加方便。

当前流行的版本控制工具有 SUBVERSION, GIT, DARCS, GNU ARCH 等。T_EX_{MACS} 目前只对Subversion提供基础支持, 不过要集成其他的版本控制工具也并不困难。





比较两个版本.

假定我们有同一个文档的两个版本old.tm和new.tm。通过以下步骤可以显示旧版本和新版本的差异: 首先打开new.tm这个文件, 然后点击版本→比较→与旧版选中old.tm打开。此时, 缓冲区的命名仍为new.tm, 更改通过特殊的标记显示在缓冲区中。假定存在差异, 那么光标初始定位在第一个差异处。类似地, 你还可以先打开旧版本, 在点击版本→比较→与新版和新版本比较。

通过子菜单版本→移动, 你可以在文档中的差异间上下移动。相应的快捷键是  和 。如果你已经了解了结构化光标移动, 不妨使用更加通用的快捷键, 即 , ,  和 。

差异的可视化.

版本间的差异有三种方式显示: 只显示旧版本, 只显示新版本, 或者两者兼顾。旧版本字体呈深红色的, 新版本则为墨绿。

每一个差异的可视化的显示方式都可以通过 版本→显示 或者快捷键  (旧),  (新) and  (兼)更改。也可以用在这些方式间轮换, 此即所谓结构化变元。通过选中文本, 可以扩大以上快捷键的作用域, 特别地, 选中全文, 则所有差异的显示方式同步更改。

保留特定版本.

我们经常需要在两个版本之间斟酌推敲，逐步对每一个差异的选择保留其中之一。假定当前光标是在某一个差异之中，那么此时可用子菜单版本→保留或者快捷键 **^1**, **^2** 和 **?** 分别保留旧版本，新版本和当前版本。如果当前是新旧版本兼显示，那么按下 **?** 会保留新版本。选定版本后，光标会自动跳到下一个差异处，这样我们可以继续进行操作。

粒度控制和差异重现.

子菜单版本→粒度用于控制版本间差异的计算。默认地，我们使用最细的粒度详细。使用Block，则版本间的差异将以段落为基础计算。也就是说，在比较中，只要两个版本在段落上有任何差异，就会高亮显示。最粗糙的粒度是Rough，只要两个版本间存在差异，全文就会高亮显示。

粒度选项在使用版本→文件→比较时产生作用，但也可以为选中文本更改比较的粒度，只要在选中文本后，再更改版本→粒度子菜单即可。特别地，整个缓冲区的比较粒度也可以这样更改。类似地，如果你在一个差异中更改了粒度值，那么差异将以新的粒度重新计算并显示。

注意您还可以更改文本后，再改变比较粒度。效果将是重新显示选中部分的差异或者是光标处差异。这在特定情境下非常实用。有如下情境，旧版本中有一个引理，而您在新版本中将其改成了定理并在其中做了少量更改。此时，若显示差异，整个定理都将高亮显示。这其实是由 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 文档内部的结构决定的。如果要显示其中内容的差异，您只需将旧版本中的引理改成定理即可。

使用外部版本控制工具.

如果你正在编辑的文件所在的目录在某个版本控制系统下，那么版本下拉菜单中分隔符以上的部分会被激活。目前 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 只支持Subversion。

首先，如果当前的缓冲区在版本控制下，那么你可以通过版本→History浏览该文件的历史版本。该子菜单的下拉菜单是一系列旧版本的超链接及相关信息，比如何时何人更改。历史版本是只读的，但你可以使用版本→比较→With current user version将它们和当前的版本比较。

对版本控制下的文本更改后，编辑器中的版本不再和版本库中的版本相对应。此时，点击版本→Commit以提交更新。提交之后，系统会显示您已经提交的更改。通过版本→Register，您可以把不在版本控制下的文件添加到版本控制系统中。只是注册文件并不会将文件提交到版本库中，您仍然需要点击版本→Commit以提交您的更改。

如果你编辑某文件时，其对应的版本库中的文件正好被更改，此时，你需要点击版本→更新，将两者合并。现在还没有对完成针对冲突的解决方案，不过，不久您就能轻松解决冲突了。

章 9

LAPTOP PRESENTATIONS

$\text{\TeX}_{\text{MACS}}$ features a “presentation mode”, for making presentations from a laptop. The presentation mode is activated/deactivated using 查看→演讲模式 or \textasciixF9 . In this chapter, we describe several dedicated style packages and markup elements which can be used for making slick presentations.

Several types of remote controllers are supported for laptop presentations. Some of them (such as Apple infrared controllers) should work out of the box (at least for the Qt version). Others map the buttons on the remote controller to certain keys on your keyboard, and you will need to toggle 查看→远程控制 in order to remap these keys to the right actions during presentations. If necessary, the appropriate mappings may be specified in 编辑→首选项→键盘→远程控制. By activating the debugging tool 工具→调试工具 and 调试→键盘, you may find out the particular mappings used by your remote control.

9.1. BEAMER STYLES

In order to start writing a laptop presentation, you should first select the **beamer** style using 文档→样式→演示. There are several main “themes” for presentations, such as Bluish, Ice, Metal, Reddish, Ridged paper, which can be selected from the 文档→Beamer theme menu.

The presentation style includes the **alt-colors** style package, which features the use of colors for mathematical formulas, enunciations, etc. Additional customizations of the presentation style are available in the 焦点→首选项 menus for the various tags to which they apply. For instance, the rendering of slide titles and theorems can be further customized.

9.2. TRAVERSAL OF A PRESENTATION

One major family of markup tags for presentations concerns the traversal of the document during a presentation. The keys \textasciixF10 and \textasciixF11 are used respectively for going back and forth in the presentation. The keys \textasciixF9 and \textasciixF12 are used to go to the start resp. end of the presentation. When using the **beamer** style or when enabling the “presentation tool” in the 工具 menu, a 演示 menu and additional icons will appear, which can also be used for the traversal of your presentation.

The most basic traversal tag is called a “switch”, and allows the user to show different pieces of text in successive and mutually exclusive manner. The entire presentation itself usually consists of a **screens** switch, where the pieces are the successive “slides” of the presentation. After selection of the **beamer** style, this switch can be inserted using 焦点→屏幕 or 插入→折叠→Switch→屏幕. You may jump from one screen to another one using \textasciixF10 and \textasciixF11 .

Inside a switch, new “branches” can be inserted after or before the currently visible branch using 焦点→在后面插入参数 or 焦点→在前面插入参数. Besides the **screens** switch, you may use 插入→折叠→Switch→标准 to insert paragraph-wide switches, and 插入→折叠→Switch→标准 to insert inline switches (similarly to displayed and inline formulas).

Another popular way to traverse is presentation is to progressively unroll content. This can be done by inserting an `unroll` tag using 插入→折叠→Unroll. Using a “hack” this tag can be combined with the `itemize` and `enumerate` tags: first create the list environment, but remove the first (automatically inserted) `item` tag. Next insert the unroll tag. When pressing `enter` inside the unroll tag, new items are created; you still have to use 焦点→在后面插入参数 for inserting new branches to the unroll structure (in particular, several items could be unrolled at once).

A variant of unrolling is unfolding. This is basically an unroll tag with exactly two branches, but different variants are available in 插入→折叠→Folded depending on the desired rendering. In particular, some of the renderings display a button which may be pushed in order to fold or unfold some content. The input-output fields inside computer algebra sessions are also foldable. Similarly, the tags in 插入→折叠→Summarize are switches with two branches, again with different kinds of rendering.

When using $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ in combination with an external plug-in, such as a computer algebra system, you will notice that all input-output fields in `sessions` are foldable. In addition, you can create so called “executable switches” using the items in the 插入→折叠→可运行 submenu. This allows you to switch back and forth between a given input to the system and the corresponding output.

All markup for the traversal of presentations may be nested in a natural way. In the 插入→折叠→Traversal menu, you may specify whether unrolled and folded structures should be folded back after traversal.

9.3. OVERLAYS

The standard “fold”, “unroll” and “switch” tags implement the most frequent kinds of traversal of a slideshow. However, there are cases in which more complex successions are needed.

<p>Elimination of x from</p> $x \sin y - 3 x y = \alpha$ <p>yields</p> $x = \frac{\alpha}{\sin y - 3 y}.$	<p>Elimination of x from</p> $x \sin y - 3 x y = \alpha$ <p>yields</p> $x = \frac{\alpha}{\sin y - 3 y}.$
--	--

图 9.1. Example of highlighting a variable x when switching from one slide to a next one.

For instance, imagine that we are given a slide, and that we wish to highlight all occurrences of some variable x in red on the next slide (see figure 9.1). This could be achieved by using a switch tag: we just copy the whole slide both in the first and in the second branch of the switch, and next color all instances of x red in the second branch. However, this solution has the disadvantage that any *a posteriori* modification on the slide has to be made both in the first and in the second branch.

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ provides a so called “overlay” mechanism for this kind of more complex successions of slides. You may insert a pile of overlays using 插入→折叠→Overlays→标准. At the start, the pile contains a unique overlay, but new overlays can then be added using the standard keyboard shortcuts `^→` and `^←` for `structured insertion`. When applied to overlays, the standard keys `F10` and `F11` for traversing the presentation have the effect of going up and down in the pile of overlays.

By default, all text which is typed by the user will be visible on all overlays. But, using the filters in the menu 插入→折叠→Overlay, it is also possible to create text which is only visible on specified overlays of the pile. There are four basic types of filters:

Visible from here on. Text that will be visible on this and all subsequent overlays.

Visible until here. Text that will be visible on this and all previous overlays.

Visible only here. Text that will be visible only on this overlay.

Visible except here. Text that will be visible on all but the current overlays.

In a similar way, $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ provides tags for alternate views: depending on whether a certain predicate is met, we show the “main content” on certain overlays and the “alternate content” on the remaining overlays. We use the same four types of predicates:

Alternate from here on. The alternate text will be used on this and all subsequent overlays.

Alternate until here. The alternate text will be used on this and all previous overlays.

Alternate only here. The alternate text will only be used on this overlay.

Alternate except here. The alternate text will be used on all but the current overlays.

$\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ finally provides a means of customizing the way that “hidden” and “shown” content should be rendered: user determined coloring schemes can be used using the [alter-colors](#) tag which can be inserted from 插入→折叠→Overlay→Specify color alternation. For instance, in order to achieve the effect mentioned in the example from Figure 9.1, you may specify a “black to red” color alternation, and then use a Visible from here on type of overlay.

9.4. DECORATIONS

In order to decorate your laptop presentations, $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ provides a few extra markup elements: [granite](#), [manila-paper](#), [metal](#), [pine](#), [ridged-paper](#) and [rough-paper](#). These tags will put your content on a nice, natural background, as illustrated in the figure below. You may also use the [tit](#) tag for giving individual slides a title.

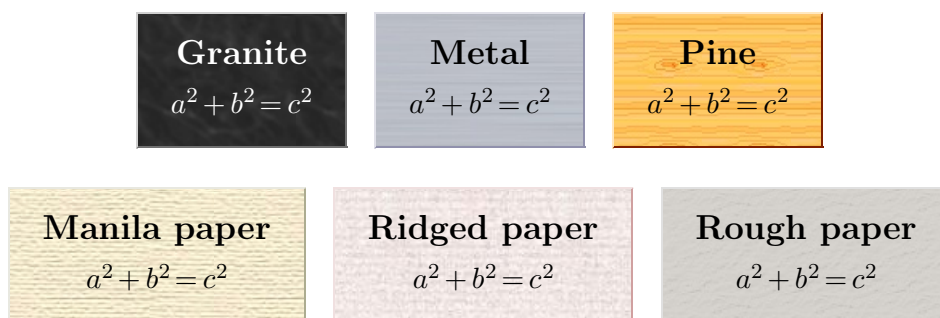


图 9.2. Some standard ornaments for decorating your presentations.

9.5. ANIMATIONS

$\text{\TeX}_{\text{MACS}}$ provides some rudimentary support for animations inside laptop presentations. This support is likely to be further improved in future $\text{\TeX}_{\text{MACS}}$ distributions.

The simplest animations are available from the menus 插入→动画→平移 and 插入→动画→渐开. Using the first menu, it is possible to create moving content: you first specify a duration for the full animation and then enter the content which has to be moved. The different kinds of moving content are illustrated in figure 9.3. Similarly, using the second menu, it is possible to create content which only progressively appears on the screen. The various kinds of progressive content are illustrated in figure 9.4. The duration of the animations can be *modified a posteriori* by putting your cursor inside them and using the shortcuts $\text{\%}\text{\textasciitilde}\leftarrow$ and $\text{\%}\text{\textasciitilde}\rightarrow$.

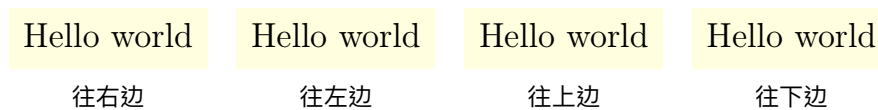


图 9.3. Moving content, as inserted from 插入→动画→平移.

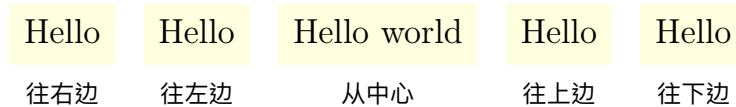


图 9.4. Progressive content, as inserted from 插入→动画→渐开.

Other basic animations are “animated gif pictures”, which can be inserted from 插入→动画→动画, and sounds, which can be inserted from 插入→动画→声音. Support for movies should be added later.

It is also possible to combine animation, so as to form larger animations. For instance, using 插入→动画→组合 you can play several animations one after another. Often the individual elements of a composed animations are fixed animation of a given duration, which can be inserted using 插入→动画→固定. Of course, you may also use moving or progressive content or even composed animations as building blocks. An animation can be repeated indefinitely using 插入→动画→重复. This may for instance be used to create a blinking effect. Some examples of the various possibilities can be found in figure 9.5.



图 9.5. Different kinds of composed animations.

9.6. EXPORTING BEAMER PRESENTATIONS

Once you have created a file with the `beamer` style you may want to export it to PDF in order to be able to give your presentation without using $\text{\TeX}_{\text{MACS}}$. There are two ways in which this can be done: *expanded* and *unexpanded*.

1. *Expanded* means that `fold`s of all kinds are “flattened” out before they are exported. This is useful if you intend the resulting PDF file to be distributed and printed, since it will have exactly as many pages as slides your presentation.

2. *Unexpanded* means that the PDF file will have as many pages as *steps* your presentation, which depending on your use of [fold](#), [switch](#), [overlay](#), etc. will typically result in many more pages.

You can select which of these methods will be used with 首选项→转换器→
TeXmacs-> df/Postscript→Expand beamer slides.

章 10


USING GNU T_EX_{MACS} AS AN INTERFACE

An important feature of T_EX_{MACS} is its ability to communicate with external systems. For computer algebra systems or other scientific computation systems, this is typically done in shell-like sessions, in which it is possible to evaluate commands and display the results in a nice, graphical way. Some systems can also be used more in the background as scripting languages.

See 帮助→插件 for a list of existing plug-ins and more documentation on these systems.


10.1. CREATING SESSIONS

A session can be started from the 插入→进程 menu. Since T_EX_{MACS} is based on the SCHEME language, it is always possible to start a SCHEME session using 插入→进程→Scheme. On UNIX systems, it is usually also possible to start BASH shell sessions using 插入→进程→Shell. The remainder of the items in the 插入→进程 menu depend on the plug-ins which are installed on your system.

A session consists of a sequence of input and output fields and possible text between them. When pressing  inside an input field of a session, the text inside the environment is evaluated and the result is displayed in an output field.

When entering a command in a session, the application attempts to execute it. Several commands may be launched concurrently in the same document, but the output will only be active in the session where the cursor is and at the place of the cursor. Therefore, we recommend to use different buffers for parallel executions.

For each type of external application, one may choose between sharing a single process by different sessions, or launching a separate process for each different session. More precisely, when inserting a session using 插入→进程→其它, you may specify both a “session type” (Shell, Pari, Maxima, etc.) and a “session name” (the default name is “default”). Sessions with different names correspond to different processes and sessions with the same name share a common process.

In order to finish the process which underlies a given session, you may use 进程→关闭进程. When pressing  in the input of a non-connected system, the system will be restarted automatically. You may also use 进程→中断执行 in order to interrupt the execution of a command. However, several applications do not support this feature.

In order to evaluate all fields of e.g. a previously created session, you may use 进程→求值→全部求值. Similarly, 进程→求值→求上面各值 and 进程→求值→求下面各值 allow you to evaluate all field above or below the current field.

10.2. EDITING SESSIONS

Inside input fields of sessions, the cursor keys have a special meaning: when moving upwards or downwards, you will move to previous or subsequent input fields. When moving to the left or to the right, you will never leave the input field; you should rather use the mouse for this.

Some facilities for editing input, output and text fields are available in the 进程→区域 menu. Keyboard shortcuts for inserting fields are \uparrow (insert above) and \downarrow . Keyboard shortcuts for removing matching text/input/output fields are \leftarrow (remove backwards) and \rightarrow (remove current fields).

It is possible to create “subsessions” using 进程→进程→创建子进程 or \rightarrow . In that case, the current input-output field becomes the body of an unfolded subsession. Such a subsession consists of an explanatory text together with the subsession body. Subsessions can be folded and unfolded using \uparrow F10 resp. \uparrow F11. Subsessions have a nice rendering on the screen when using the `framed-session` package in 文档→Use package→程序.

Notice that input/output fields and subsessions are foldable: when clicking on the prompt with the mouse, you may fold or unfold the entry to hide or show the output. For laptop presentations, this folding and unfolding process is done automatically when traversing your presentation. It is also possible to fold or unfold all fields in a session using 进程→进程→折叠全部区域 and 进程→进程→展开全部区域.

Other useful editing operations are 进程→进程→清除全部区域, which is useful for creating a demo session which will be executed later on, and 进程→分割进程, which can be used for splitting a session into parts for inclusion into a paper.

例 10.1. A typical MAXIMA session is given below. If MAXIMA is present on your system, then you may put your cursor in one of the inputs, perform some edits, and try to reexecute it.

```
Maxima 5.25.1 http://maxima.sourceforge.net
using Lisp SBCL 1.0.51
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
```

```
(%i1) diff (x^x^x, x)
```

```
(%o1)  $x^{x^x} (x^x \log(x) (\log(x) + 1) + x^{x-1})$ 
```

```
(%i2) integrate (%o1, x)
```

```
(%o2)  $e^{e^{x \log(x)} \log(x)}$ 
```

```
(%i3) integrate (x^5 / (x^2 - x + 17), x)
```

```
(%o3)  $\frac{239 \log(x^2 - x + 17)}{2} + \frac{1361 \arctan\left(\frac{2x-1}{\sqrt{67}}\right)}{\sqrt{67}} + \frac{3x^4 + 4x^3 - 96x^2 - 396x}{12}$ 
```

10.3. SELECTING THE INPUT METHOD

By default, T_EX_{MACS} will attempt to evaluate the input field when pressing \rightarrow . Multiline input can be created using \uparrow \rightarrow . Alternatively, when selecting the multiline input mode using 进程→Input mode→多行输入, the \rightarrow key will behave as usual and \uparrow \rightarrow may be used in order to evaluate the input field. Notice finally that certain systems admit built-in heuristics for testing whether the input has been completed; if not, then the \rightarrow may behave as usual.

Certain applications allow you to type the mathematical input in a graphical, two dimensional form. This feature can be used by selecting 进程→Input mode→数学输入. If this feature is available, then it is usually also possible to copy and paste output back into the input. However, it depends on the particular application how well this works. Keep in mind that some key combinations may be used by the 数学输入 mode: for instance the key $\$$ is usually redefined inside math mode, so if you want to input it you'll have to type $\uparrow F5 \$$. You can read more about the prefix key $\uparrow F5$ in “Keyboard shortcuts for text mode”.

例 10.2. Below, you will find the [previous example session](#), but now using mathematical input:

```
Maxima 5.25.1 http://maxima.sourceforge.net
using Lisp SBCL 1.0.51
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
```

(%i1) $\text{diff}(x^{x^x}, x)$

(%o1) $x^{x^x} (x^x \log(x) (\log(x) + 1) + x^{x-1})$

(%i2) $\int \%o1 dx$





(%o2) $e^{e^{x \log(x)} \log(x)}$


(%i3) $\int \frac{x^5}{x^2 - x + 17} dx$

(%o3) $\frac{239 \log(x^2 - x + 17)}{2} + \frac{1361 \arctan\left(\frac{2x-1}{\sqrt{67}}\right)}{\sqrt{67}} + \frac{3x^4 + 4x^3 - 96x^2 - 396x}{12}$

10.4. PLUG-INS AS SCRIPTING LANGUAGES

TeX_{MACS} provides a few other kinds of additional interfaces to external systems in addition to shell-like interfaces. First of all, it is possible to insert a so called “executable switch” anywhere in the document using 插入→折叠→可运行.

For instance, if MAXIMA is installed on your system, then 插入→折叠→可运行→Maxima should yield something like . You may enter a MAXIMA expression in the yellow part of this markup, say  `diff(x^x,x)`. Using , you may now switch back and forth between the unevaluated input and the evaluated output $x^x (\log(x) + 1)$. Using , you enable multi-line input. This kind of executable switches are very useful for plug-ins such as DRA_{TeX} , EUKLEIDES , FEYNMF , etc., which are mainly used for the efficient computation and insertion of special graphics inside TeX_{MACS} documents.

Some plug-ins such as MAXIMA can even be selected as a *scripting language* using 文档→脚本→Maxima. When doing so, a special Maxima menu will appear, which allows for many useful operations directly on formulas. For instance, when putting the cursor inside the formula $1 + 1$ and pressing  or 求值, the formula gets evaluated automatically to yield 2.

If a plug-in can be used as a scripting language, then it is possible to create executable switches with links between them. More precisely, assuming that you selected a scripting language from 文档→脚本, you may insert a new *executable input field* using `\!` or 插入→链接→Executable input field. As before, when pressing `↵`, the current input is evaluated and you will see the corresponding output; you may switch back to the input by pressing `↵` once more.

Contrary to executable switches, you may attach an identifier to the executable input field by disactivating the field or by editing the Ref field in the focus bar. Inside other executable input fields, you may then refer to the value of the field by inserting a *field reference* using `\?` or 插入→链接→Field reference. As a variant to executable input fields, you may sometimes prefer to insert plain *input fields* using `\` or 插入→链接→Input field. These fields can only be used as inputs and pressing `↵` inside such a field will only recompute those other fields which depend on it.

例 10.3. The executable input fields may for instance be nice in pedagogic documents in which parts of the document may be modified and recomputed by the reader. For instance, evaluation of the input fragment

The derivative of x^x equals `diff(function,x)`.

The second derivative is given by `diff(derivative,x)`.

yields

The derivative of x^x equals $x^x (\log(x) + 1)$.

The second derivative is given by $x^x (\log(x) + 1)^2 + x^{x-1}$.

Of course, if the reader changes the input function x^x into something else and presses `↵`, then the first and second derivatives will be updated automatically.

10.5. SPREADSHEETS

T_EX_{MACS} provides rudimentary spreadsheet-like facilities with the advantage that the computations can be carried out using any of the plug-ins that can be used as a scripting language. In order to use the spreadsheet facilities, you should therefore start with the selection of a scripting language in the menu 文档→脚本.

As soon as you have selecting a scripting language, such as MAXIMA, then you may enter a new spreadsheet using 插入→表格→Textual spreadsheet or 插入→表格→Numeric spreadsheet. You may edit the spreadsheet as an ordinary table, except that the `↵` key will attempt to reevaluate the cells of the table.

In addition, when preceding the contents of a cell by `=`, then cell will be considered as an input-output switch. More precisely, the input is a formula which will be evaluated using the current scripting language. After the evaluation, only the result of the evaluation is shown in the cell. After pressing `↵` a second time in the cell, it will be possible switch back and edit the input. In the formulas, one may refer to the others using names such as `c5` for the third row and the fifth column.

例 10.4. On the left-hand side of the figure below, we have displayed a simple table with formulas for evaluating the sums of the first two items of each row. On the right-hand side, we have shown the result after evaluation.

1	10	=a1+b1	1	10	11
100	1000	=a2+b2	100	1000	1100

图 10.1. Evaluation of a simple spreadsheet.

例 10.5. The cells may contain mathematical formulas and the spreadsheet may take advantage of any of the capacities of the scripting language. For instance, the figure below demonstrates another possible use of MAXIMA.

$\sin(x^2)$	$\sin(x^2)$
=diff(a1,x)	$2x \cos(x^2)$
=diff(a2,x)	$2 \cos(x^2) - 4x^2 \sin(x^2)$
=diff(a3,x)	$-12x \sin(x^2) - 8x^3 \cos(x^2)$

图 10.2. Computation of successive derivatives using MAXIMA.

TeX_{MACS} supports a few special notations for applying operations on all cells in a subtable. For instance, as in EXCEL, one may use the notation **c3:d5** for indicating all cells **c3**, **c4**, **c5**, **d3**, **d4**, **d5** in the block from **c3** to **d5**. An alternative notation **,...**, for **:** can be entered by typing **,.**. In a similar way, one may enter the special notation **+...+** by typing **++**. For instance, **c3+...+d5** stands for the sum of all cells between **c3** and **d5**.

例 10.6. The figure below shows an example on how to use taking sums of cells. Notice that empty cells count for zero.

15.10	15.10	30.20	15.10	15.10	30.20
100	125	75	100	125	75
28.50		14.25	28.50		14.25
12	16	20	12	16	20
=a1+...+a4	=b1+...+b4	=c1+...+c4	155.6	156.1	139.45

图 10.3. Evaluation of a simple spreadsheet.

Notice that copying and pasting of subtables works in the same way as for ordinary tables, with the additional features that the names of the cells and references to cells in the formulas are renumbered automatically. Similarly, automatic renumbering is used when inserting new columns or rows, or when removing existing columns or rows.

We also notice that field references can be used inside spreadsheet cells in order to refer to some computational markup outside the table. Inversely, each spreadsheet also carries an invisible Ref field which can be edited by deactivating the spreadsheet or from the focus bar (when selecting the entire spreadsheet). The Ref field of the spreadsheet is used as a prefix for referring to the contents of cells outside the table or from within other spreadsheets. For instance, if Ref equals **sheet**, then **sheet-c4** will refer to the field **c4** inside the spreadsheet.

10.6. REMOTE PLUG-INS

It sometimes happens that certain plug-ins are only installed on a remote computer. In many cases, it will still be possible to use such a plug-in inside TeX_{MACS} over an **SSH** connection.

In order to make this work, you first have to make sure that SSH is installed on both computers and that connecting by SSH to the remote machine can be done automatically, without having to type a password. This can be done by copying your public identity on the local server to the remote server into the file `~/.ssh/authorized_keys`; see the documentation on SSH for more information.

As the next step, you have to make sure that T_EX_{MACS} has been installed on both computers. The remote T_EX_{MACS} installation will mainly be used in order to detect which plug-ins can be used on the remote computer.

When everything has been set up correctly in this way, select 插入→进程→远程 in order to open the remote plug-in selector. Add the name of the remote server by typing its name or IP address and clicking on Add. After a small pause, the remote server should appear in the list together with the remote plug-ins which are supported. You may now simply select the plug-in you want to use from the list. Notice that remote plug-ins may take a few seconds in order to boot. Please be patient while booting is in progress.

Servers which have been added to the list of remote plug-in servers will be remembered at the next time when you start T_EX_{MACS}. You may use the buttons 删除 and 更新 in order to remove a server from the list and to redetermine the list of supported remote plug-ins.

章 11

WRITING T_EX_{MACS} STYLE FILES

One of the fundamental strengths of T_EX_{MACS} is the possibility to write your own style files and packages. The purpose of style files is multiple:

- They allow the abstraction of repetitive elements in texts, like sections, theorems, enumerations, etc.
- They form a mechanism which allow you to structure your text. For instance, you may indicate that a given portion of your text is an abbreviation, a quotation or “important”.
- Standard document styles enable you to write professionally looking documents, because the corresponding style files have been written with a lot of care by people who know a lot about typography and aesthetics.

The user may select a major style from the 文档→样式 menu. The major style usually reflects the kind of document you want to produce (like a letter, an article or a book) or a particular layout policy (like publishing an article in a given journal).

Style packages, which are selected from the 文档→样式 menu, are used for further customization of the major style. For instance, the **number-europe** package enables European-style theorem numbering and the **maxima** package contains macros for customizing the layout of sessions of the MAXIMA computer algebra system. Several packages may be used together.

When you want to add your own markup to T_EX_{MACS} or personalize the layout, then you have to choose between writing a principal style file or a style package. In most cases, you will probably prefer to write a style package, since this will allow you to combine it arbitrary other styles. However, in some cases you may prefer to create a new principal style, usually by personalizing an existing style. This is usually the case if you want to mimic the layout policy of some journal. In this chapter, we will both explain how to write your own style packages and how to customize the standard styles.

11.1. WRITING A SIMPLE STYLE PACKAGE

Let us explain on an example how to write a simple style package. First of all, you have to create a new buffer using 文件→新建 and select the **source** document style using 文档→样式→源码. Now save your empty style package in your personal style package directory

`$HOME/.TeXmacs/packages`

Notice that the button Texts in the file browser corresponds to the directory

`$HOME/.TeXmacs/texts`

Consequently, you can go to the style package directory from there, by double clicking on `..` and next on **packages**. Similarly, the directory

`$HOME/.TeXmacs/styles`

contains your personal style files. After saving your empty style package, it should automatically appear in the 文档→Package menu. Notice that style files must be saved using the `.ts` file extension. If you save the style file in a subdirectory of `$HOME/.TeXmacs/packages`, then it will automatically appear in the corresponding submenu of 文档→Package. Let us now create a simple macro `hi` which displays “Hello world”. First type `⌘=`, so as to create an assignment. You should see something like

```
<assign|>
```

Now enter “hi” as the first argument and type `⌘M` inside the second argument in order to create a macro. You should now see something like

```
<assign|hi|<macro|>
```

Finally, type the text “Hello world” in the body of the macro. Your document should now consist of the following line:

```
<assign|hi|<macro|Hello world>
```

After saving your style package, opening a new document and selecting your package in the 文档→Use package menu, you may now use the macro `hi` in your document by typing `\HI` and hitting `↵`.

In a similar way, you may create macros with arguments. For instance, assume that we started entering a macro `hello` in a similar way as above. Instead of typing “Hello world”, we first type `⌘←` inside the macro body so as to create an additional argument on the left hand side of the cursor. We next enter the name of the argument, say “name”. You should now see something like below:

```
<assign|hello|<macro|name|>
```

In the second argument of the body, we now type “Hello ”, `⌘#`, “name”, `→` and “, how are you today?”. After this you should see

```
<assign|hello|<macro|name|Hello name, how are you today?>
```

The `⌘#` shortcut is used to retrieve the macro argument `name`. Instead of typing `⌘#`, “name” and `→`, you may also use the hybrid `\`-key and type `\NAME` followed by `↵`. After saving your style package, you may again use the macro in any document which uses your package by typing `\HELLO` and hitting `↵`.

From the internal point of view, all macro definitions are stored in the environment of the T_EX_{MACS} typesetter. Besides macros, the environment also contains normal environment variables, such as section counters or the font size. The environment variables can either be globally changed using the `assign` primitive, or locally, using the `with` primitive. For instance, when including the line

```
<assign|section-nr|-1>
```

in your package, and using `article` as your major style, then the first section will be numbered 0. Similarly, the variant


```
<assign|hello|<macro|name|Hello <with|font-shape|small-caps|name|!>>
```

of the `hello` macro displays the name of the person in SMALL CAPITALS. Notice that the `with` primitive can also be used to locally redefine a macro. This is for instance used in the definitions of the standard list environments, where the macro which renders list icons is changed inside the body of the list. Yet another variant of the `hello` macro relies on the standard `person` macro:

```
<assign|hello|<macro|name|Hello <person|name|!>>
```

In order to produce the macro application `<person | name>`, you first have to start a compound tag using `⌘^C`, type the name “person”, insert an argument `^→`, and enter the argument `name` as before. When you are done, you may press `⌘` in order to change the `compound` tag into a `person` tag. Alternatively, you may type `\`, “person”, `^→` and “name”.

By combining the above constructs, an ordinary user should already be able to produce style packages for all frequently used notations. An interesting technique for writing macros which involve complex formulas with some subformulas which may change goes as follows:

1. Type the formula, say (a_1, \dots, a_n) , in an ordinary document.
2. Create the skeleton of your macro in your style package:

```
<assign|n-tuple|<macro|a|>>
```

3. Copy the formula and paste it into the body of your macro:

```
<assign|n-tuple|<macro|a|(a<rsup|1>,...,a<rsup|n>)>>
```

4. Replace the subformulas you want to parameterize by macro arguments:

```
<assign|n-tuple|<macro|a|(a<rsup|1>,...,a<rsup|n>)>>
```

5. You may now use the macro in documents which use your package:

$$(a_1, \dots, a_n) = (b_1, \dots, b_n).$$

11.2. RENDERING OF STYLE FILES AND PACKAGES

11.2.1. ASCII-based or tree-based editing: an intricate choice

Most users are used to edit source code using a conventional editor like EMACS, while presenting the source code in ASCII format. Since all $\text{\TeX}_{\text{MACS}}$ documents are stored as *trees*, an interesting but complicated question is which format is most suitable for editing such documents. One option is to represent the tree using an ASCII-based format, such as XML, Scheme, or the native format for storing files on a disk. The other option is to edit the trees as such, making no fundamental distinction between source code and normal documents.

In T_EX_{MACS} we have chosen to implement the second option. More precisely, any document can be edited in “source mode”, which is merely a mode for rendering the document in a way which makes its tree structure particularly apparent. It may be instructive to take an arbitrary document of yours and to take a look at it in “source mode” by enabling 文档→源码→编辑源码树.

The choice between ASCII-based editing and tree-based editing is non-trivial, because T_EX_{MACS} style files and packages have a double nature: they may be seen as programs which specify how to render macros, but these programs naturally contain ordinary content. There are several reasons why users often prefer to edit source code in an ASCII-based format:

1. It is easy to manually format the code so as to make it more readable.
2. In particular, it is easy to add comments.
3. Standard editors like EMACS provide tools for automatic highlighting, indentation, etc.
4. One is not constraint by any “structure” during the editing phase.

Our approach is to reproduce as much of the above advantages in a structured document environment. Although point 4 will obviously be hard to meet when following this approach, we believe that the first three advantages might actually become greater in a structured environment. However, this requires a more profound understanding of how users format and edit source code.

For instance, consider a piece of manually formatted code like

```
if (cond) hop    = 2;
else        holala= 3;
```

Clearly, the user had a particular formatting policy when writing this code. However, this policy does not appear in the document: manual intervention will be necessary if the variable `cond` is renamed `c`, or if the variable `holala` is renamed `hola`.

At the moment, T_EX_{MACS} provides no tools for dealing with the above example in an automatic way, but a few tools are already provided. For instance, the user is given a great amount of control on how to indent source code and reasonable defaults are provided as a function of the structure. We also provide high level environments for comments and structured highlighting. Further tools will be developed later and we are open for any suggestions from our users.

11.2.2. Global presentation

In the 代码标记 group of the 文档→源码 menu, you find several ways to customize the rendering of source trees in your document. We recommend you to play around with the different possibilities in a document of your own (after enabling 文档→源码→Source tree) or a standard style package in \$TEXMACS_PATH/packages.

First of all, you may choose between the different major styles “angular”, “scheme”, “functional” and “L^AT_EX” for rendering source trees, as illustrated in the figure below:

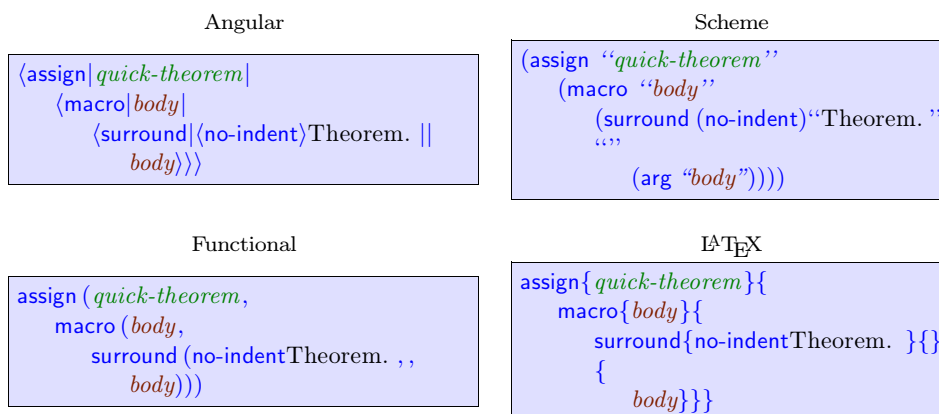


图 11.1. Different styles for rendering the same source tree.

Secondly, you may wish to reserve a special treatment to certain tags like `concat` and `document`. In the menu 文档→源码→特定 you may specify to which extent you want to treat such tags in a special way:

None. No tags receive a special treatment.

Formatting. Only the formatting tags `concat` and `document` are represented as usual.

Normal. In addition to the formatting tags, a few other tags like `compound`, `value` and `arg` are represented in a special way.

Maximal. At the moment, this option is not yet implemented. The intention is to allow the user to write his own customizations and to allow for special rendering of basic operations like `plus`.

These different options are illustrated below:

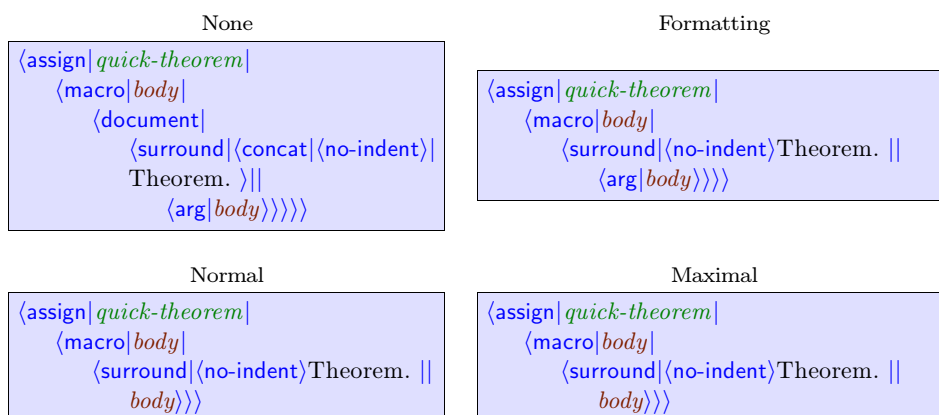


图 11.2. Different ways to render special tags.

Another thing which may be controlled by the user is whether the presentation of tags should be compact or stretched out across several lines. Several levels of compactification may be specified in the 文档→源码→紧化 menu:

Minimal. The tags are all stretched out across several lines.

Only inline tags. All non-inline tags are stretched out across several lines.

Normal. All inline arguments at the start of the tag are represented in a compact way. As soon as we encounter a block argument, the remainder of the arguments are stretched out across several lines.

Inline arguments. All inline arguments are represented in a compact way and only block tags are stretched out across several lines.

Maximal. All source code is represented in a compact way.

The “normal” and “inline arguments” options rarely differ. The visual effect of the different options is illustrated below:

Minimal	Only inline tags
<pre> <assign quick-theorem <macro body <surround <concat <no-indent> Theorem. > body>>> </pre>	<pre> <assign quick-theorem <macro body <surround <no-indent>Theorem. body>>> </pre>
Normal	Maximal
<pre> <assign quick-theorem <macro body <surround <no-indent>Theorem. body>>> </pre>	<pre> <assign quick-theorem <macro body <document <surround <no-indent>Theorem. body>>>> </pre>

图 11.3. Different levels of compactification.

Finally, the user may specify the way closing tags should be rendered when the tag is stretched out across several lines. The rendering may either be minimalistic, compact, long, or recall the matching opening tag. The different options are illustrated below:

Minimal	Compact
<pre> assign quick-theorem macro body surround <no-indent>Theorem. body </pre>	<pre> <assign quick-theorem <macro body <surround <no-indent>Theorem. body>>> </pre>
Stretched	Repeat
<pre> <assign quick-theorem <macro body <surround <no-indent>Theorem. body > > > </pre>	<pre> <\assign quick-theorem> <\macro body> <\surround <no-indent>Theorem. body </surround> </macro> </assign> </pre>

图 11.4. Different ways to render closing tags.

11.2.3. Local customization

Even though $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ tries hard to render source code in a nice way following the global rendering options that you specified, the readability of the source code often needs to be further enhanced locally. In source mode, this can be done using the menus 源码→激活 and 源码→Presentation. Any local hints on how to render source code are automatically removed from the document when it is being used as a style file or package.

First of all, for certain pieces of content the user may prefer to see them in their “activated” form instead as dead source code. This may for instance be the case for embedded images, or for mathematical symbols, like in

```
<assign|R|<macro| $\mathbb{R}$ >>
```

Such an active presentation may also be preferred for certain more complex macros:

```
<assign|diag|<macro|var|dim>>

$$\begin{pmatrix} \textcolor{brown}{var_1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \textcolor{brown}{var_{dim}} \end{pmatrix}$$

```

A piece of code can be activated by selecting it and using 源码→激活→激活 or \textasciix26^+ . Similarly, a piece of content may be deactivated using \textasciix26^- (we used this in the second example above for the rendering of the arguments *var* and *dim*). Activation and deactivation either apply to the whole tree, or to the root only (e.g. 源码→激活→仅激活一次).

Another way to customize the rendering is to override some of the global rendering options. This is mainly interesting for controlling more precisely which tags have to be stretched across several lines and which tags have to be represented in a compact fashion. For instance, the *concat* tag can be used in order to concatenate content, as well as for specifying a block of sequential statements, or a combination of both. For instance, in the piece of code

```
<assign|my-section|
  <macro|title|
    <concat|
      <header-hook|title>|
      <toc-hook|title>|
      <my-section-title|title>>>>
```

we have stretched the *concat* tag along several lines using 源码→Presentation→拉伸 (notice that this implies the *concat* tag to appear explicitly, so as to avoid confusion with the *document* tag). Similarly, if a part of the concatenation were to be displayed as usual, then one may use 源码→Presentation→紧凑:

```
<assign|my-section|
  <macro|title|
    <concat|
      <header-hook|title>|
      <toc-hook|title>|
      <with|font-series|bold|Section:> title>>>
```

At present, we did not implement a way to mark arguments as inline or block, but we might do this later.

A final way to customize the rendering of source code is to apply an arbitrary macro using 源码→Presentation→使用宏 or 源码→Presentation→仅这次使用宏. This macro will be automatically removed when you use your document as a style file or package.

11.3. THE STYLE-SHEET LANGUAGE

In the section about [writing a simple style package](#) we already gave you a first impression about the style-sheet language of T_EX_{MACS}. In this section, we will give a more complete survey of the available features. For more detailed descriptions, we refer to the chapter about the T_EX_{MACS} primitives.

The style-sheet primitives can be obtained from the 源码 menu when you are in source mode. In all other modes, the 源码 menu becomes visible after enabling the 源码菜单 in the 工具 menu. Alternatively, you may use the `⌘` and `⌘^E` keyboard prefixes in source mode and the `⌘^I` and `⌘^E` prefixes otherwise. Furthermore, we recall that the hybrid `\`-key may be used for creating macro-applications or arguments, depending on the context. Finally, the `^→` and `^←` keys are used for inserting arguments.

11.3.1. Assignments

All user defined T_EX_{MACS} macros and style variables are stored in the “current typesetting environment”. This environment associates a tree value to each string variable. Variables whose values are macros correspond to new primitives. The others are ordinary environment variables. The primitives for operating on the environment are available from 源码→Define.

You may permanently change the value of an environment variable using the [assign](#) primitive, as in the example

```
<assign|hi|<macro|Hi there!>>
```

You may also locally change the values of one or several environment variables using the [with](#) primitive:

```
<with|font-series|bold|color|red|Bold red text>
```

The value of an environment variable may be retrieved using the [value](#) primitive. This may for instance be used in order to increase a counter:

```
<assign|my-counter|<plus|my-counter|1>>
```

Finally, you may associate logical properties to environment variables using the [drd-props](#) primitive. This is explained in more detail in the section about [macro primitives](#).

11.3.2. Macro expansion

The main interest of the T_EX_{MACS}’ style-sheet language is the possibility to define macros. These come in three flavours: ordinary macros, macros which take an arbitrary number of arguments and external macros, whose expansion is computed by SCHEME or a plug-in. The macro-related primitives are available from the 源码→宏命令 menu. Below, we will only describe the ordinary macros. For more details, we refer to the section about [macro primitives](#).

Ordinary macros are usually defined using

```
<assign|my-macro|<macro|x1|...|xn|body>>
```

After such an assignment, `my-macro` becomes a new primitive with n arguments, which may be called using

```
<my-macro|y1|...|yn>
```

Inside the body of the macro, the `arg` primitive may be used to retrieve the values of the arguments to the macro.

```
<assign|hello|<macro|name|Hello name, you look nice today!>>
```

It is possible to call a macro with less or more arguments than the expected number. Superfluous arguments are simply ignored. Missing arguments take the nullary `uninit` primitive as value:

```
<assign|hey|
  <macro|first|second|
    <if|
      <equal|second|?>|
      Hey first, you look lonely today...|
      Hey first and second, you form a nice couple!>>>
```

We finally notice that you are allowed to compute with macros, in a similar way as in functional programming, except that our macros are not closures (yet). For instance:

```
<assign|my-macro-copy|my-macro>
```

The `compound` tag may be used to apply macros which are the result of a computation:

```
<assign|overloaded-hi|
  <macro|name|
    <compound|
      <if|<nice-weather>|happy-hi|sad-hi|
      name>>>
```

11.3.3. Formatting primitives

This section contains some important notes on formatting primitives which are not really part of the style-sheet language, but nevertheless very related.

First of all, most $\text{\TeX}_{\text{MACS}}$ presentation tags can be divided in two main categories: inline tags and block tags. For instance, `frac` is a typical inline tag, whereas `theorem` is a typical block tag. Some tags, like `strong` are inline if their argument is and block in the contrary case. When writing macros, it is important to be aware of the inline or block nature of tags, because block tags inside a horizontal concatenation are not rendered in an adequate way. If you need to surround a block tag with some inline text, then you need the `surround` primitive:

```

<assign|my-theorem|
  <macro|body|
    <surround|<no-indent><with|font-series|bold|Theorem. >|<right-flush>|
      body>>

```

In this example, we surrounded the body of the theorem with the bold text “Theorem.” at the left hand side and a “right-flush” at the right-hand side. Flushing to the right is important in order to make the blue visual border hints look nice when you are inside the environment.

In most cases, T_EX_{MACS} does a good job in determining which tags are inline and which ones are not. However, you sometimes may wish to force a tag to be a block environment. For instance, the tag `very-important` defined by

```

<assign|very-important|<macro|body|<with|font-series|bold|color|red|body>>

```

may both be used as an inline tag and a block environment. When placing your cursor just before the `with`-tag and hitting `↵` followed by `⌘`, you obtain

```

<assign|very-important|
  <macro|body|
    <with|font-series|bold|color|red|body>>

```

Since the body of the macro is now a block, your tag `very-important` will automatically become a block environment too. In the future, the `drd-props` primitive will give you even more control over which tags and arguments are inline and which ones are block.

Another important property of tags is whether they contain normal textual content or tabular content. For instance, consider the definition of the standard `eqnarray*` tag (with a bit of presentation markup suppressed):

```

<assign|eqnarray*|
  <macro|body|
    <with|par-mode|center|mode|math|math-display|true|par-sep|0.45fn|
      <surround|<no-page-break*><vspace*|0.5fn>|<vspace|0.5fn><no-indent*>|
        <tformat|
          <twith|table-hyphen|y>|
          <twith|table-width|1par>|
          <twith|table-min-cols|3>|
          <twith|table-max-cols|3>|
          <cwith|1|-1|1|1|cell-hpart|1>|
          <cwith|1|-1|-1|-1|cell-hpart|1>|
            body>>>>

```

The use of `surround` indicates that `eqnarray*` is a block environment and the use of `tformat` specifies that it is also a tabular environment. Moreover, the `twith` and `cwith` are used to specify further formatting information: since we are a block environment, we enable hyphenation and let the table span over the whole paragraph (unused space being equally distributed over the first and last columns). Furthermore, we have specified that the table contains exactly three columns.

Finally, it is important to bear in mind that style-sheets do not merely specify the final presentation of a document, but that they may also contain information for the authoring phase. Above, we have already mentioned the use of the `right-flush` tag in order to improve the rendering of visual border hints. Similarly, visual hints on invisible arguments may be given in the form of flags:

```
<assign|labeled-theorem|
  <macro|id|body|
    <surround|
      <concat|
        <no-indent|
          <flag|Id: id|blue|id|
            <with|font-series|bold|Theorem. >>|
          <right-flush|
            body>>>
```

More generally, the `specific` tag with first argument “screen” may be used to display visual hints, which are removed when printing the document.

11.3.4. Evaluation control

The 源码→赋值 menu contains several primitives to control the way expressions in the style-sheet language are evaluated. The most frequent use of these primitives is when you want to write a “meta-macro” like `new-theorem` which is used for defining or computing on other macros. For instance:

```
<assign|new-theorem|
  <macro|name|text|
    <quasi|
      <assign|<unquote|name>|
        <macro|body|
          <surround|<no-indent><strong|<unquote|text>. >|<right-flush>|
            body>>>>>>
```

When calling `<new-theorem|theorem|Theorem>` in this example, we first evaluate all `unquote` instructions inside the `quasi` primitive, which yields the expression

```
<assign|theorem|
  <macro|body|
    <surround|<no-indent><strong|Theorem. >|<right-flush>|
      body>>>
```

Next, this expression is evaluated, thereby defining a macro `theorem`.

It should be noticed that the $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ conventions for evaluation are slightly different than those from conventional functional languages like SCHEME. The subtle differences are motivated by our objective to make it as easy as possible for the user to write macros for typesetting purposes.

For instance, when T_EX_{MACS} calls a macro $\langle \text{macro} | x_1 | \dots | x_n | \text{body} \rangle$ with arguments y_1 until y_n , the argument variables x_1 until x_n are bound to the unevaluated expressions y_1 until y_n , and the body is evaluated with these bindings. The evaluation of y_i takes place each time we request for the argument x_i . In particular, when applying the macro $\langle \text{macro} | x | x \text{ and again } x \rangle$ to an expression y , the expression y is evaluated twice.

In SCHEME, the bodies of SCHEME macros are evaluated twice, whereas the arguments of functions are evaluated. On the other hand, when retrieving a variable (whether it is an argument or an environment variable), the value is not evaluated. Consequently, a T_EX_{MACS} macro

```
 $\langle \text{assign} | \text{foo} | \langle \text{macro} | x | \langle \text{blah} | x | x \rangle \rangle \rangle$ 
```

would correspond to a SCHEME macro

```
(define-macro (foo x)
  '(let ((x (lambda () ,x)))
    (blah (x) (x))))
```

Conversely, the SCHEME macro and function

```
(define-macro (foo x) (blah x x))
(define (fun x) (blah x x))
```

admit the following analogues in T_EX_{MACS}:

```
 $\langle \text{assign} | \text{foo} | \langle \text{macro} | x | \langle \text{eval} | \langle \text{blah} | \langle \text{quote-arg} | x \rangle | \langle \text{quote-arg} | x \rangle \rangle \rangle \rangle \rangle$ 
 $\langle \text{assign} | \text{fun} | \langle \text{macro} | x | \langle \text{with} | x^* | x | \langle \text{blah} | \langle \text{quote-value} | x^* \rangle | \langle \text{quote-value} | x^* \rangle \rangle \rangle \rangle \rangle$ 
```

Here the primitives `quote-arg` and `quote-value` are used to retrieve the value of an argument resp. an environment variable. The T_EX_{MACS} primitives `eval`, `quote`, `quasiquote` and `unquote` behave in the same way as their SCHEME analogues. The `quasi` primitive is a shortcut for quasi-quotation followed by evaluation.

11.3.5. Flow control

Besides sequences of instructions, which can be achieved using the `concat` primitive, and the mechanism of macro expansion, the T_EX_{MACS} style-sheet language provides a few other primitive for affecting the control flow: `if`, `case`, `while` and `for-each`. These primitives are available from the 源码→流程控制 menu. However, we have to warn the user that the conditional constructs are quite fragile: they only apply to inline content and the accessibility of macro arguments should not to much depend on the conditions.

The most important primitive `if`, which can be entered using `⌘^?`, allows for basic conditional typesetting:

```
 $\langle \text{assign} | \text{appendix} |$ 
 $\quad \langle \text{macro} | \text{title} | \text{body} |$ 
 $\quad \quad \langle \text{compound} |$ 
 $\quad \quad \quad \langle \text{if} | \langle \text{long-document} \rangle | \text{chapter-appendix} | \text{section-appendix} \rangle |$ 
 $\quad \quad \quad \text{title} |$ 
 $\quad \quad \quad \text{body} \rangle \rangle \rangle$ 
```

In this example, `appendix` is a block environment consisting of a title and a body, and which is rendered as a chapter for long documents and as a section for short ones. Notice that the following implementation would have been incorrect, since the `if` primitive currently only works for inline content:

```
<assign|appendix|
  <macro|title|body|
    <if|
      <long-document>|
      <chapter-appendix|title|body>|
      <section-appendix|title|body>>>>
```

The `if` primitive may also be used in order to implement optional arguments:

```
<assign|hey|
  <macro|first|second|
    <if|
      <equal|second|?>|
      Hey first, you look lonely today...|
      Hey first and second, you form a nice couple!>>>
```

However, $\text{\TeX}_{\text{MACS}}$ is not clever enough to detect which arguments are optional and which arguments are accessible (i.e. which arguments can be edited by the user). Therefore, you will have to manually give this information using the `drd-props` primitive. The `case`, `while` and `for-each` primitives are explained in more detail in the corresponding section on the $\text{\TeX}_{\text{MACS}}$ primitives.

11.3.6. Computational markup

In the menus `源码→算术`, `源码→文本`, `源码→元组` and `源码→条件` you will find different primitives for computing with integers, strings, tuples and boolean values. For instance, in the following code, the `new-important` tag defines a new “important tag” as well as a variant in red:

```
<assign|new-important|
  <macro|name|
    <quasi|
      <concat|
        <assign|
          <unquote|name>|
          <macro|x|<with|font-series|bold|x>>>|
        <assign|
          <unquote|<merge|name|-red>>|
          <macro|x|<with|font-series|bold|color|red|x>>>>>>>
```

Here we use the `merge` primitive in order to concatenate two strings. The different computational primitives are described in more detail in the corresponding section on the $\text{\TeX}_{\text{MACS}}$ primitives.

11.4. CUSTOMIZING THE STANDARD $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ STYLES

Whenever the standard $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ style files are inadequate for a given purpose, it is possible to write your own style files. However, designing your own style files from scratch may be a complex task, so it is usually preferable to customize the existing styles. This requires some understanding of the global architecture of the standard style files and a more precise understanding of the parts you wish to customize. In this section, we will explain the general principles. For more details, we refer to the chapter on the principal $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ tags.

11.4.1. Organization of style files and packages

Each standard $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ style file or package is based on a potentially finite number of subpackages. From an abstract point of view, this organization may be represented by a labeled tree. For instance, the tree which corresponds to the `article` style is represented below:

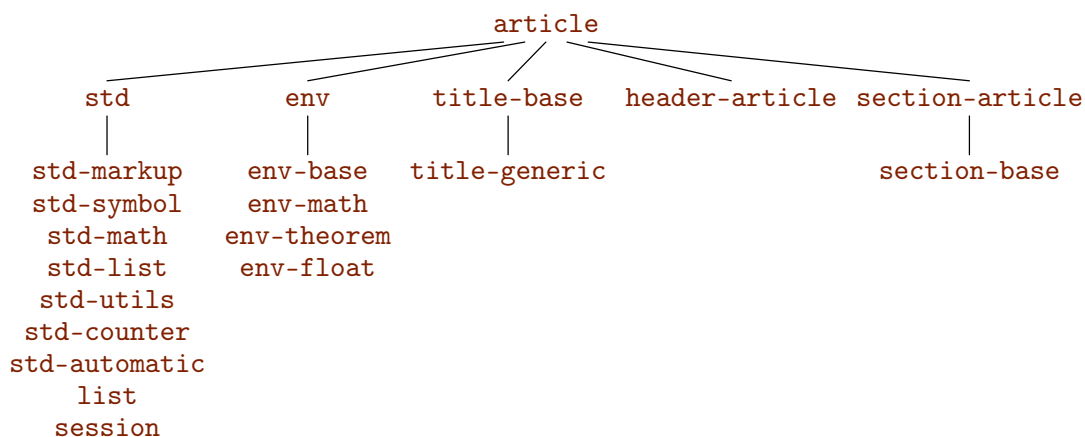


图 11.5. The tree with the packages from which the `article` style has been built up. In order to save space, we have regrouped the numerous children of `std` and `env` in vertical lists.

Most of the style packages correspond to a d.t.d. (data type definition) which contains the “abstract interface” of the package, i.e. the exported tags. For instance, the package `std-markup` corresponds to the d.t.d. `std-markup`. Sometimes however, several style packages match the same d.t.d.. For instance, both `header-article` and `header-book` match the d.t.d. `header`, since they merely implement different ways to render the same tags.

When building your own style files or packages, you may use the `use-package` primitive in order to include other packages. For instance, the `article` style essentially consists of the line

```
<use-package|std|env|title-generic|header-article|section-article>
```

More precisely, the `use-package` package sequentially includes the style packages corresponding to its arguments. The packages should be in `$TEXMACS_PACKAGE_PATH`, which contains `.`, `~/TeXmacs/packages` and `$TEXMACS_PATH/packages` by default. Furthermore rendering information for the source code like `style-with` tags are discarded before evaluation of the files.

注意 11.1. We strongly recommend the user to take a look at some of the standard style files and packages which can be found in

`$TEXMACS_PATH/styles`

`$TEXMACS_PATH/packages`

When loading using `\usepackage{}`, these paths are in the standard load path. For instance, if you want to take a look at the `std-markup` package, then it suffices to type `\usepackage{std-markup}`, followed by the file name `std-markup.ts` and `\end{document}`.

注意 11.2. It is also possible to customize the presentation of the source code of the style files and packages themselves, by using other packages in addition to `source` or by using another major style file based on `source`. In that case, the extra markup provided by such packages may be used for presentation purposes of the source code, but it is not exported when using your package in another file.

11.4.2. General principles for customization

Style files and packages basically enrich the current typesetting environment with a combination of

- Environment variables.
- Tags for the end-user.
- Customizable macros.

Furthermore, they may define some tags for intern implementation purposes, which will not be documented in this manual. They may also specify some logical properties of tags using the `dird-props` primitive.

Environment variables are almost always attributes for controlling the rendering of content, or counters for sections, equations, etc.. Although several simple tags for the end-user like `strong` may be redefined in your own style files, this practice is not recommended for more complex tags like `section`. Indeed, the `section` tag involves many things like resetting subcounters, entering the title into the table of contents and so on. Therefore, special additional macros are provided the customization of such tags, like `section-title`, `section-clean` and `section-toc`.

11.4.3. Customizing the general layout

The general layout of a document is mainly modified by setting the appropriate environment variables for `page layout` and `paragraph layout`. For instance, by including the following lines in your style file, you can set the page size to `letter` and the left and right margins to `2in`:

```
\assign{page-type}{letter}
\assign{page-odd}{2in}
\assign{page-even}{2in}
\assign{page-right}{2in}
```

It should be noticed that the environment variables for page layout are quite different in T_EX_{MACS} and T_EX/L^AT_EX. In order to make it easier to adapt L^AT_EX style files to T_EX_{MACS}, we have therefore provided the `std-latex` package, which emulates the environment variables from T_EX/L^AT_EX. Typically, this allows you determine the global layout by lines like

```
<assign|tex-odd-side-margin|<macro|20pt>>
<assign|tex-even-side-margin|<macro|20pt>>
<assign|tex-text-width|<macro|33pc>>
```

We notice that macros which return lengths are considered as `lengths` themselves. In the case of the T_EX/L^AT_EX emulation package, we actually *require* all lengths to be macros.

The page headers and footers are usually not determined by global environment variables or macros, since they may change when a new chapter or section is started. Instead, T_EX_{MACS} provides the call-back macros `header-title`, `header-author`, `header-primary` and `header-secondary`. These macros are called when the document title or author are specified or when a new primary or secondary section is started (primary sections are typically chapters in books, or sections in articles). For instance, the following redefinition makes the principal section name appear on even pages, together with the current page number and a wide underline.

```
<assign|header-primary|
  <macro|title|nr|type|
    <assign|page-even-header|
      <quasiquote|
        <wide-std-underlined|
          <concat|
            <page-the-page>|
            <htab|5mm>|
            <unquote|title>>>>>>>>>>
```

11.4.4. Customizing list environments

Lists are made up of two principal ingredients: the outer list environment and the inner items. List environments may either be customized by customizing or redefining the rendering macros for these environments, or defining additional list environments which match the same abstract interface.

The rendering of the outer list environment is controlled by the `render-list` macro which takes the body of the list as its argument. For instance, consider the following redefinition of `render-list`:

```
<assign|render-list|
  <macro|body|
    <surround|
      <no-page-break*><vspace*|0.5fn>|
      <right-flush><vspace|0.5fn><no-indent*>|
      <with|par-left|<plus|par-left|3fn>|par-right|<plus|par-right|3fn>|body>>>>>>
```

This redefinition affects the rendering of all list environments (itemize, enumerate, etc.) by reducing the right margin with a length of 3fn:

- This text, which has been made so long that it does not fit on a single line, is indented on the right hand side by 3fn.
 1. This text is indented by an additional 3fn on the right hand side, since it occurs inside a second list environment.
- Once again: this text, which has been made so long that it does not fit on a single line, is indented on the right hand side by 3fn.

In a similar way, you may customize the rendering of list items by redefining the macros `aligned-item` and `compact-item`. These macros both take one argument with the text of the item and render it either in a right-aligned way (such that subsequent text is left aligned) or in a left-aligned way (such that subsequent text may not be aligned). For instance, consider the following redefinition of `aligned-item`:

```
<assign|aligned-item|
  <macro|x|
    <concat|
      <vspace*|0.5fn|
      <with|par-first|-3fn|<yes-indent>>|
      <resize|<with|color|red|x|<minus|1r|2.5fn||<plus|1r|0.5fn|>>>>>
```

Then items inside all list environments with compact items will appear in red:

- This list and aligned descriptions have red items.
 - C1.** First condition.
 - C2.** Second condition.
- The items of compact description lists are rendered using `compact-item`.

Gnus and gnats. Nice beasts.

Micros and softies. Evil beings.

注意 11.3. The macros `aligned-item` and `compact-item` are required to produce inline content, so that they may be used in order to surround blocks. In particular, several other internal macros (`aligned-space-item`, `long-compact-strong-dot-item`, etc.) are based on `aligned-item` and `compact-item`, and used for the rendering of the different types of lists (`itemize-arrow`, `description-long`, etc.). In the future, we also plan to extend `item` and `item*` with a compulsory *body* argument. When customizing the list environments, it is important to keep that in mind, so as to make your style-sheets upward compatible.

The `std-list` d.t.d. also provides a macro `new-list` to define new lists. Its syntax is `<new-list|name|item-render|item-transform>`, where *name* is the name of the new list environment, *item-render* an (inline) macro for rendering the item and *item-transform* an additional transformation which is applied on the item text. For instance, the `enumerate-roman` environment is defined by

```
<new-list|enumerate-roman|aligned-dot-item|<macro|x|<number|x|roman>>>
```

11.4.5. Customizing numbered textual environments

T_EX_{MACS} provides three standard types of numbered textual environments: theorem-like environments, remark-like environments and exercise-like environments. The following aspects of these environments can be easily customized:

- Adding new environments.
- Modifying the rendering of the environments.
- Numbering the theorems in a different way.

Defining new environments.

First of all, new environments can be added using the meta-macros `new-theorem`, `new-remark` and `new-exercise`. These environments take two arguments: the name of the environment and the name which is used for its rendering. For instance, you may wish to define the environment `experiment` by

```
<new-theorem|experiment|Experiment>
```

When available in the T_EX_{MACS} dictionaries, the text “Experiment” will be automatically translated when your document is written in a foreign language. In the section about [how to define new environments](#), it is also explained how to define other numbered textual environments (besides theorems, remarks and exercises).

Customization of the rendering.

The principal rendering of the environments can be customized by redefining the `render-theorem`, `render-remark` and `render-exercise` macros. These macros take the *name* of the environment (like “Theorem 1.2”) and its *body* as arguments. For instance, if you want theorems to appear in a slightly indented way, with a slanted body, then you may redefine `render-theorem` as follows:

```
<assign|render-theorem|
  <macro|which|body|
    <padded-normal|1fn|1fn|
      <surround|<theorem-name|which<theorem-sep>>||
        <with|font-shape|slanted|par-left|<plus|par-left|1.5fn|body>>>>>
```

This redefinition produces the following effect:

定理 11.4. *This is a theorem which has been typeset in a slanted font.*

By default, the theorems are rendered as remarks with the only difference that their bodies are typeset in an italic font. Hence, redefining the `render-remark` macro will also affect the rendering of theorems. The default `render-proof` macro is also based on `render-remark`.

Instead of redefining the entire rendering, the user might just wish to customize the way names of theorems are rendered or redefine the separator between the name and the body. As the user may have noticed by examining the above redefinition of `render-theorem`, these aspects are controlled by the macros `theorem-name` and `theorem-sep`. For instance, consider the following redefinitions:


```

\assign\theorem-name\<macro\name\<with\color\dark red\font-series\bold\name\>\>
\assign\theorem-sep\<macro\>

```

Then theorem-like environments will be rendered as follows:

命题 11.5: *This proposition is rendered in is a fancy way.*

Customization of the numbering.

In the sections about `counters` and `counter groups`, it is explained how to customize the counters of numbered environments for particular purposes. For instance, by redefining `inc-theorem`, you may force theorems to reset the counter of corollaries:

```

\quasi
  \assign
    \inc-theorem
      \<macro\>\<compound\<unquote\inc-theorem\>\<reset-corollary\>\>\>

```

Notice the trick with `quasi` and `unquote` in order to take into account additional action which might have been undertaken by the previous value of the macro `inc-theorem`.

The following code from `number-long-article.ts` is used in order to prefix all standard environments with the number of the current section:

```

\assign\section-clean\<macro\>\<reset-subsection\>\<reset-std-env\>\>
\assign\display-std-env\<macro\nr\<section-prefix\nr\>\>

```

11.4.6. Customizing sectional tags

By default, T_EX_{MACS} provides the standard sectional tags from L^AT_EX `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, `subparagraph`, as well as the special tag `appendix`. T_EX_{MACS} also implements the unnumbered variants `part*`, `chapter*`, etc. and special section-like tags `bibliography`, `table-of-contents`, `the-index`, `the-glossary`, `list-of-figures`, `list-of-tables`.

注意 11.6. Currently, the sectional tags take one argument, the section title, but a second argument with the body of the section is planned to be inserted in the future (see the experimental `structured-section` package). For this reason (among others), style files should never redefine the main sectional tags, but rather customize special macros which have been provided to this effect.

From a global point of view, an important predicate macro is `sectional-short-style`. When it evaluates to `true`, then appendices, tables of contents, etc. are considered to be at the same level as sections. In the contrary case, they are at the same level as chapters. Typically, articles use the short sectional style whereas books use the long style.

The rendering of a sectional tag `x` is controlled through the macros `x-sep`, `x-title` and `x-numbered-title`. The `x-sep` macro prints the separator between the section number and the section title. It defaults to the macro `sectional-sep`, which defaults in its turn to a wide space. For instance, after redefining

```
<assign|sectional-sep|<macro| -- >>
```

sectional titles would typically look like

11.1 – HAIRY GNUS

The `x-title` and `x-numbered-title` macros respectively specify how to render unnumbered and numbered section titles. Usually, the user only needs to modify `x-title`, since `x-numbered-title` is based on `x-title`. However, if the numbers have to be rendered in a particular way, then it may be necessary to redefine `x-numbered-title`. For instance, consider the redefinition

```
<assign|subsection-numbered-title|
  <macro|name|
    <sectional-normal|
      <with|font-series|bold|<the-subsection>. >name>>>
```

This has the following effect on the rendering of subsection titles:

2.3. Very hairy GNUs

Notice that the `section-base` package provides several useful helper macros like `sectional-normal`.

注意 11.7. Sectional titles can either be rendered in a “short” or in the “long” fashion. By default, paragraphs and subparagraphs use the short rendering, for which the body starts immediately at the right of the title:

My paragraph. Blah, blah, and more blahs...

All other sectional tags use the long rendering, in which case the section title takes a separate line on its own:

MY SECTION

Blah, blah, and more blahs...

We do not recommend to modify the standard settings (i.e. to render paragraphs in a long way or sections in a short way). If you really want to do so, then we recommend to redefine the corresponding environment variables `enrich-x-long`. This will ensure upward compatibility when sectional tags will take an additional argument (see remark 11.6).

Besides their rendering, several other aspects of sectional tags can be customized:

- The call-back macro `x-clean` can be used for cleaning some counters when a new section is started. For instance, in order to prefix all standard environments by the section counter, you may use the following lines:

```
<assign|section-clean|<macro|<reset-subsection><reset-std-env>>>
<assign|display-std-env|<macro|nr|<section-prefix>nr>>
```

- The call-back macro `x-header` should be used in order to modify page headers and footers when a new section is started. Typically, this macro should call `header-primary`, or `header-secondary`, or do nothing.
- The call-back macro `x-toc` should be used in order to customize the way new sections appear in the table of contents.

11.4.7. Customizing the treatment of title information

$\text{\TeX}_{\text{MACS}}$ uses the `doc-data` tag in order to specify global data for the document. These data are treated in two stages by the `doc-data` macro. *First*, the document data are separated into several categories, according to whether the data should be rendered as a part of the main title or in footnotes or the abstract. *Secondly*, the data in each category are rendered using suitable macros.

Each child of the `doc-data` is a tag with some specific information about the document. Currently implemented tags are `doc-title`, `doc-subtitle`, `doc-author`, `doc-date`, `doc-running-title`, `doc-running-author`, `doc-keywords`, `doc-msc` and `doc-note`. The `doc-author` tag may occur several times. The `author-data` tag is used in order to specify structured data for each of the authors of the document. Each child of the `author-data` tag is a tag with information about the corresponding author. Currently implemented tags with author information are `author-name`, `author-affiliation`, `author-email`, `author-homepage` and `author-note`.

Most of the tags listed above also correspond to macros for rendering the corresponding information as part of the main title. For instance, if the date should appear in bold italic at a distance of at least `1fn` from the other title fields, then you may redefine `doc-date` as

```
<assign|doc-date|
  <macro|body|
    <concat|
      <vspace*|1fn|
      <doc-title-block|<with|font-shape|italic|font-series|bold|body>>|
      <vspace|1fn>>>>
```

The `title-block` macro is used in order to make the text span appropriately over the width of the title. The `doc-title` and `author-name` are special in the sense that they also render possible references to footnotes. For this reason, you should rather customize the `doc-render-title` and `author-render-name` macros in order to customize the rendering of the title and the name themselves.

Notice also that the `doc-running-title` and `author-running-author` macros do not render anything, but rather call the `header-title` and `header-author` call-backs for setting the appropriate global page headers and footers. By default, the running title and author are extracted from the usual title and author names.

In addition to the rendering macros which are present in the document, the main title (including author information, the date, etc.) is rendered using the `doc-make-title` macro. The author information, as part of the main title, is rendered using `render-doc-author` or `render-doc-authors`, depending on whether the document has one or more authors. Footnotes to the title or to one of the authors are rendered using `doc-title-note` resp. `doc-author-note`. These footnote macros always expect a `document` tag on input, because they may compress it into a horizontal concatenation.

The first stage of processing the document data is more complex and the reader is invited to take a look at the [short descriptions](#) of the macros which are involved in this process. It is also good to study the definitions of these macros in the [package itself](#). In order to indicate the way things work, we finish with an example on how the email address and homepage of an author can be rendered in a footnote instead of the main title:

```
<assign|doc-author-main|
  <macro|data|
    <quasi|
      <unquote*|<select|<quote-arg|data|author-name|>>
      <unquote*|<select|<quote-arg|data|author-affiliation|>>|>>>>
    <assign|doc-author-data-note|
      <xmacro|data|
        <quasi|
          <unquote*|<select|<quote-arg|data|author-email|>>
          <unquote*|<select|<quote-arg|data|author-homepage|>>
          <unquote*|<select|<quote-arg|data|author-note|document|<pat-any|>>>>>>>>>>
```

11.5. FURTHER NOTES AND TIPS

11.5.1. Customizing arbitrary tags

Imagine that you want to change the rendering of a given tag, like [lemma](#). As a general rule, T_EX_{MACS} provides a set of well-chosen macros which can be customized by the user so as to obtain the desired effect. For instance, as we have seen [above](#), you should use modify one of the macros [render-theorem](#), [theorem-name](#) or [theorem-sep](#) in order to customize the rendering of [lemma](#) and all other theorem-like environments.

However, in some cases, it may not be clear which “well-chosen” macro to customize. If we just wanted to change the presentation of lemmas and not of any other theorem-like environments, then we clearly cannot modify [render-theorem](#), [theorem-name](#) or [theorem-sep](#). In other cases, the user may not want to invest his time in completely understanding the macro hierarchy of T_EX_{MACS}, and find out about the existence of [render-theorem](#), [theorem-name](#) and [theorem-sep](#).

So imagine that you want all lemmas to appear in red. One thing you can always do is copy the original definition of lemmas in a safe place and redefine the lemma macro on top of the original definition:

```
<assign|orig-lemma|lemma>
<assign|lemma|<macro|body|<with|color|red|<orig-lemma|body|>>>>
```

Alternatively, if only the text inside the lemma should be rendered in red, then you may do:

```
<assign|orig-lemma|lemma>
<assign|lemma|<macro|body|<orig-lemma|<with|color|red|body|>>>>
```

Of course, you have to be careful that the name [orig-lemma](#) is not already in use.

Another frequent situation is that you only want to modify the rendering of a tag when it is used inside another one. On the web, the *Cascading Style Sheet* language (CSS) provides a mechanism for doing this. In $\text{\TeX}_{\text{MACS}}$, you may simulate this behaviour by redefining macros inside a `with`. For instance, imagine that we want the inter-paragraph space inside lists inside theorem-like environments to vanish. Then we may use:

```

<assign|orig-render-theorem|render-theorem>
<assign|render-theorem|
  <macro|name|body|
    <with|orig-render-list|render-list|
      <with|render-list|<macro|x|<orig-render-list|x>>|
        <orig-render-theorem|
          name|
          body>>>>

```

On the one hand side, this mechanism is a bit more complex than CSS, where it suffices to respecify the *par-par-sep* attribute of lists inside theorems. On the other hand, it is also more powerful, since the `render-theorem` macro applies to all theorem-like environments at once. Furthermore, if the above mechanism is to be used frequently, then real hackers may simplify the notations using further macro magic.

11.5.2. Standard utilities

In the package `std-utils`, the user may find several useful additional macros for writing style files. It mainly contains macros for

- Writing block environments which span over the entire paragraph width. Notice that the `title-base` package provides some *additional macros* for wide section titles.
- Writing wide block environments which are underlined, overlined or in a frame box.
- Recursive indentation.
- Setting page headers and footers.
- Localization of text.

It is good practice to use these standard macros whenever possible when writing style files. Indeed, the low-level $\text{\TeX}_{\text{MACS}}$ internals may be subject to minor changes. When building upon standard macros with a clear intention, you increase the upward compatibility of your style-sheets.

章 12

定制 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$

高度可定制性是 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的主要特点之一。一般需求可在编辑→首选项中简单配置，深入定制则需使用 $\text{G}_{\text{U}}\text{I}\text{L}\text{E}/\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 扩展。

12.1. $\text{G}_{\text{U}}\text{I}\text{L}\text{E}$ 扩展语言入门

与Emacs类似， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 使用类Lisp语言作为扩展——来自GNU计划的 $\text{G}_{\text{U}}\text{I}\text{L}\text{E}$ $\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 。关于 $\text{G}_{\text{U}}\text{I}\text{L}\text{E}$ $\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ ，请参考：

<http://www.gnu.org/software/guile/guile.html>

$\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 的优势在于可通过外部的C或C++程序扩展。在 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 中，你可以使用 $\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 来创建你自己的菜单和快捷键，甚至是你自己的 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 扩展。

如果你已经下载了 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 的源代码，不妨看一下这些文件：

```
Guile/Glue/build-glue-basic.scm
Guile/Glue/build-glue-editor.scm
Guile/Glue/build-glue-server.scm
```

这三个胶水代码文件包含了在 $\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 中可见的C++的库函数(routine)。在下面的小节中，我们将讨论其中最重要的部分。我们计划撰写更加完整的参考文档。目前你可以参考 $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{PATH}}/\text{progs}$ 目录中的 $\text{S}\text{C}\text{H}\text{E}\text{M}\text{E}$ 源文件（以scm为后缀名）。

12.2. 撰写初始化配置文件

启动程序时， $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 将读取并执行这个文件：

```
 $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{PATH}}/\text{progs}/\text{init-texmacs.scm}$ 
```

以及用户自己的配置文件（如果存在的话）：

```
 $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{HOME\_PATH}}/\text{progs}/\text{my-init-texmacs.scm}$ 
```

缺省情况下， $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{HOME_PATH}}$ 的值就是 .TeXmacs 。类似地，每次你新建一个buffer，程序将执行：

```
 $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{PATH}}/\text{progs}/\text{init-buffer.scm}$ 
```

以及（如果存在的话）

```
 $\$ \text{T}_{\text{E}}\text{X}_{\text{MACS}}_{\text{HOME\_PATH}}/\text{progs}/\text{my-init-buffer.scm}$ 
```

12.3. 定制动态菜单

定义一个名称为name的菜单，可用

```
(menu-bind name . def)
```

或者

```
(tm-menu (name) . def)
```

其中 `def` 是一个呈现所有菜单条目的程序。下面这个目录下的文件可供参考：

```
$TEXMACS_PATH/progs/menu
```

其中包含了标准 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 菜单的定义方式。对于`tm-menu`，你还可以指定额外的参数，使之根据参数动态构建更加复杂的菜单。

更精确地说，在`menu-bind`或者`tm-menu`中的`def`程序实际上是如下所示形式的程序代码：

```
(=> "pulldown menu name" menu-definition)
(-> "pullright menu name" menu-definition)
("entry" action)
---
(if condition menu-definition)
(link variable)
```

函数`=>`和`->`用来创建下拉或者右拉菜单，`menu-definition`中要包含创建子菜单的程序。使用函数`("entry" action)`可创建一个菜单条目，点击`entry`就会执行`action`。菜单项之间使用`---`分割。函数`if`用于在满足特定的`condition`时插入菜单项。（比如说在数学模式中）

如果你声明了一个菜单`name`，那么你可以使用函数`link`间接引用该菜单。这种间接声明子菜单的方式有两个优势：

- “间接”子菜单可以链接到我们所需的菜单，无论多少
- 使用`menu-append`可以后续添加新条目到“间接”子菜单中

主要的 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 菜单是`texmacs-menu`，`texmacs-popup-menu`，`texmacs-main-icons`，`texmacs-mode-icons`，`texmacs-focus-icons`和`texmacs-extra-icons`。其他一些标准的间接菜单是`file-menu`，`edit-menu`，`insert-menu`，`text-menu`，`paragraph-menu`，`document-menu`，`options-menu`和`help-menu`。

12.4. 定制快捷键

下面这行代码用来指定键盘布局

```
(kbd-map . keymaps)
```

使用`:mode`选项，你可以指定使键盘布局生效的条件。比如，这行代码：

```
(kbd-map (:mode in-math?) . keymaps)
```

指定了只在数学模式下生效的键盘布局。`keymaps` 这个列表的元素有下面三种形式：


```
(key-combination action_1 ... action_n)
(key-combination result)
(key-combination result help-message)
```

第一行中, `action_i`是和`key-combination`相关的Scheme代码。第二行和第三行中的`result`是`key-combination`结束之后插入的字符串。第三行中, `key-combination` 结束之后, 将会显示`help-message`。

12.5. 其它值得一看的文件

其他一些值得一看的文件:

- `$TEXMACS_PATH/fonts/enc`包含了不同 $\text{T}_{\text{E}}\text{X}$ 字体的编码
- `$TEXMACS_PATH/fonts/virtual`包含了虚拟字符的定义
- `$TEXMACS_PATH/langs/natural/dic`包含了 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 当前所使用的字典 (用于本地化菜单)
- `$TEXMACS_PATH/langs/natural/hyphen`包含了不同语言的字符连接模式
- `$TEXMACS_PATH/progs/fonts`包含了设置字体的SCHEME程序

章 13

THE T_EX_{MACS} PLUG-IN SYSTEM

There are many ways in which T_EX_{MACS} can be customized or extended: users may define their own style files, customize the user interface, or write links with extern programs. The plug-in system provides a universal mechanism to combine one or several such extensions in a single package. Plug-ins are both easy to install by other users and easy to write and maintain.

13.1. INSTALLING AND USING A PLUG-IN

From the user's point of view, a plug-in *myplugin* will usually be distributed on some web-site as a binary tarball with the name

```
myplugin-version-architecture.tar.gz
```

If you installed T_EX_{MACS} yourself in the directory \$TEXMACS_PATH, then you should unpack this tarball in the directory \$TEXMACS_PATH/plugins, using

```
tar -zxvf myplugin-version-architecture.tar.gz
```

This will create a *myplugin* subdirectory in \$TEXMACS_PATH/plugins. As soon as you restart T_EX_{MACS}, the plug-in should be automatically recognized. Please read the documentation which comes with your plug-in in order to learn using it.

注意 13.1. If you did not install T_EX_{MACS} yourself, or if you do not have write access to \$TEXMACS_PATH, then you may also unpack the tarball in \$TEXMACS_HOME_PATH/plugins. Here we recall that \$TEXMACS_HOME_PATH defaults to \$HOME/.TeXmacs. When starting T_EX_{MACS}, your plug-in should again be automatically recognized.

注意 13.2. If the plug-in is distributed as a source tarball like *myplugin-version-src.tar.gz*, then you should first compile the source code before relaunching T_EX_{MACS}. Depending on the plug-in (read the instructions), this is usually done using

```
cd myplugin; make
```

or

```
cd myplugin; ./configure; make
```

注意 13.3. In order to upgrade a plug-in, just remove the old version in \$TEXMACS_PATH/plugins or \$TEXMACS_HOME_PATH/plugins using

```
rm -rf myplugin
```

and reinstall as explained above.

13.2. WRITING YOUR OWN PLUG-INS

In order to write a plug-in *myplugin*, you should start by creating a directory

`$TEXMACS_HOME_PATH/plugins/myplugin`

where to put all your files (recall that `$TEXMACS_HOME_PATH` defaults to `$HOME/.TeXmacs`). In addition, you may create the following subdirectories (when needed):

bin — For binary files.

doc — For documentation (not yet supported).

langs — For language related files, such as dictionaries (not yet supported).

lib — For libraries.

packages — For style packages.

progs — For SCHEME programs.

src — For source files.

styles — For style files.

As a general rule, files which are present in these subdirectories will be automatically recognized by T_EX_{MACS} at startup. For instance, if you provide a **bin** subdirectory, then

`$TEXMACS_HOME_PATH/plugins/myplugin/bin`

will be automatically added to the `PATH` environment variable at startup. Notice that the subdirectory structure of a plug-in is very similar to the subdirectory structure of `$TEXMACS_PATH`.

例 13.4. The easiest type of plug-in only consists of data files, such as a collection of style files and packages. In order to create such a plug-in, it suffices to create directories

`$TEXMACS_HOME_PATH/plugins/myplugin`

`$TEXMACS_HOME_PATH/plugins/myplugin/styles`

`$TEXMACS_HOME_PATH/plugins/myplugin/packages`

and to put your style files and packages in the last two directories. After restarting T_EX_{MACS}, your style files and packages will automatically appear in the 文档→样式 and 文档→Use package menus.

For more complex plug-ins, such as plug-ins with additional SCHEME or C++ code, one usually has to provide a SCHEME configuration file

`$TEXMACS_HOME_PATH/plugins/myplugin/progs/init-myplugin.scm`

This configuration file should contain an instruction of the following form

```
(plugin-configure myplugin
 configuration-options)
```

Here the *configuration-options* describe the principal actions which have to be undertaken at startup, including sanity checks for the plug-in. In the next sections, we will describe some simple examples of plug-ins and their configuration. Many other examples can be found in the directories

`$TEXMACS_PATH/examples/plugins`

`$TEXMACS_PATH/plugins`

Some of these are [described](#) in more detail in the chapter about writing new interfaces.

13.3. EXAMPLE OF A PLUG-IN WITH SCHEME CODE

The **world** plug-in.

Consider the world plug-in in the directory

`$TEXMACS_PATH/examples/plugins`

This plug-in shows how to extend $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ with some additional SCHEME code in the file `world/progs/init-world.scm`

In order to test the world plug-in, you should recursively copy the directory

`$TEXMACS_PATH/examples/plugins/world`

to `$TEXMACS_PATH/plugins` or `$TEXMACS_HOME_PATH/plugins`. When relaunching $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$, the plug-in should now be automatically recognized (a World menu should appear in the menu bar).

How it works.

The file `init-world.scm` essentially contains the following code:

```
(plugin-configure world
  (:require #t))

(when (supports-world?)
  (display* "Using world plug-in!\n"))
```

The configuration option `:require` specifies a condition which needs to be satisfied for the plug-in to be detected by $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ (later on, this will for instance allow us to check whether certain programs exist on the system). The configuration is aborted if the requirement is not fulfilled.

Assuming that the configuration succeeds, the `supports-world?` predicate will evaluate to `#t`. In our example, the body of the `when` statement corresponds to some further initialization code, which just sends a message to the standard output that we are using our plug-in. In general, this kind of initialization code should be very short and rather load a module which takes care of the real initialization. Indeed, keeping the `init-myplugin.scm` files simple will reduce the startup time of $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$.

13.4. EXAMPLE OF A PLUG-IN WITH C++ CODE

The **minimal** plug-in.

Consider the example of the minimal plug-in in the directory

```
$TEXMACS_PATH/examples/plugins
```

It consists of the following files:

```
minimal/Makefile
minimal/progs/init-minimal.scm
minimal/src/minimal.cpp
```

In order to try the plug-in, you first have to recursively copy the directory

```
$TEXMACS_PATH/examples/plugins/minimal
```

to `$TEXMACS_PATH/progs` or `$TEXMACS_HOME_PATH/progs`. Next, running the Makefile using

```
make
```

will compile the program `minimal.cpp` and create a binary

```
minimal/bin/minimal.bin
```

When relaunching T_EX_{MACS}, the plug-in should now be automatically recognized.

How it works.

The `minimal` plug-in demonstrates a minimal interface between T_EX_{MACS} and an extern program; the program `minimal.cpp` is explained in more detail in the chapter about writing interfaces. The initialization file `init-minimal.scm` essentially contains the following code:

```
(plugin-configure minimal
  (:require (url-exists-in-path? "minimal.bin")))
(:launch "minimal.bin")
(:session "Minimal"))
```

The `:require` option checks whether `minimal.bin` indeed exists in the path (so this will fail if you forgot to run the Makefile). The `:launch` option specifies how to launch the extern program. The `:session` option indicates that it will be possible to create sessions for the `minimal` plug-in using 插入→进程→最小.

13.5. SUMMARY OF THE CONFIGURATION OPTIONS FOR PLUG-INS

As explained before, the SCHEME configuration file `myplugin/progs/init-myplugin.scm` of a plug-in with name *plugin* should contain an instruction of the type

```
(plugin-configure myplugin
  configuration-options)
```

Here follows a list of the available *configuration-options*:

(:winpath *package-path inner-bin-path*) — Specify where to search for the plug-in under windows. The *package-path* is the usual place where the plug-in is installed. The *inner-bin-path* is the place where to look for the binary executable corresponding to the plug-in, relative to the *package-path*.

- (`:winpath package-path inner-bin-path`) — Analogous to `:winpath`, but under MacOS.
- (`:require condition`) — This option specifies a sanity *condition* which needs to be satisfied by the plug-in. Usually, it is checked that certain binaries or libraries are present on your system. If the condition fails, then `TEXMACS` will continue as whether your plug-in did not exist. In that case, further configuration is aborted. The `:require` option usually occurs first in the list of configuration options.
- (`:versions version-cmd`) — This option specifies a SCHEME expression *version-cmd* which evaluates to a list of available versions of the plug-in.
- (`:setup cmd`) — This command is only executed when the version of the plug-in changed from one execution of `TEXMACS` to another one. This occurs mainly when installing new versions of `TEXMACS` or helper applications.
- (`:launch shell-cmd`) — This option specifies that the plug-in is able to evaluate expressions over a pipe, using a helper application which is launched using the shell-command *shell-cmd*.
- (`:link lib-name export-struct options`) — This option is similar to `:launch`, except that the extern application is now linked dynamically. For more information, see the section about [dynamic linking](#).
- (`:session menu-name`) — This option indicates that the plug-in supports an evaluator for interactive shell sessions. An item *menu-item* will be inserted to the 插入→ 进程 menu in order to launch such sessions.
- (`:serializer ,fun-name`) — If the plug-in can be used as an evaluator, then this option specifies the SCHEME function *fun-name* which is used in order to transform `TEXMACS` trees to strings.
- (`:commander ,fun-name`) — This command is similar to the `:serializer` option except that it is used to transform special commands to strings.
- (`:tab-completion #t`) — This command indicates that the plug-in supports tab-completion.
- (`:test-input-done #t`) — This command indicates that the plug-in provides a routine for testing whether the input is complete.

It should be noticed that the configuration of the plug-in *myplugin* automatically creates a few predicates:

- supports-myplugin?*. Test whether the plug-in is fully operational (all requirements are met).
- in-myplugin?*. Test whether *myplugin* is the current programming language.
- myplugin-scripts?*. Test whether *myplugin* is the current scripting language.

索引

从中心	54	Alternate except here	53
代码标记	66	Alternate from here on	53
删除	62	Alternate only here	53
单元格		Alternate until here	53
操作模式	30	Specify color alternation	53
水平对齐	30	Visible except here	53
竖直对齐	30	Visible from here on	53, 53
背景色	31	Visible only here	53
边界	31	Visible until here	53
单行方程	12	Overlays	
工具	51, 70	标准	52
更新		Summarize	52
包含文件	36	Switch	
调试工具	51	屏幕	51
项目		标准	51, 51
关联主文件...	36	Traversal	52
帮助		Unroll	52
插件	57	描述列表	13
弹性	43	数学	38
往上边	54, 54	公式	19
往下边	54, 54	单行公式	19
往右边	54, 54	多行公式	19, 29
往左边	54, 54	文本	38
插入	12	无序列表	12
几何		有序列表	13
剪裁	37	Roman	13
单位	37	样条曲线	37
大小	37	点	37
放缩	37	环境	13
分式	45	空白	13
分组/打散	41	线	37
动画		网格	37
动画	54	类型	
固定	54	直角坐标	37
声音	54	自动	
平移	54, 54	参考文献	35, 36
渐开	54, 54	目录	34
组合	54	索引	35
重复	54	表格	29
图片	33	大表格标题	29
在选择区域内绘制	37	小表格标题	29, 43
小图片标题	43	Numeric spreadsheet	60
绘制图形	37	Textual spreadsheet	60
圆圈	38	语义	26
圆弧	38	其它	26
多边形	37	说明	11
折叠		进程	57, 57, 95
可运行	52, 59	其它	57
Maxima	59	最小	94
Folded	52	远程	62
Overlay	53		

- Scheme 57
- Shell 57
- 链接
 - 动作 33
 - 包含文件 33, 36
 - 参考 33
 - 引用 35, 36
 - 标签 33
 - 索引项 35, 35, 35
 - 超链接 33
 - Alternate
 - 参考文献 36, 36, 36
 - Executable input field 60
 - Field reference 60
 - Input field 60
 - Invisible citation 35
- 闭合样条曲线 37
- Database entry 34
- Last name 34
- Move objects 41
- Particle 34
- Position float 43
- Resize objects 41
- Rotate objects 41
- Set properties 41
- Title suffix 34
- 文件
 - 保存 8
 - 另存为 8
 - 导入 45
 - 导出 45
 - Postscript 8
 - 打印
 - 全部打印 8
 - 全部打印为文件 8
 - 打开 7, 8
 - 新建 8, 63
- 文档 7, 8, 11
- 字体
 - 大小 14
 - Dpi 9
- 更新
 - 全部 34, 36
 - 参考文献 35
 - 目录 34
- 样式 8, 11, 63, 63, 92
 - 源码 63
 - 演示 51
- 源码 66
 - 特定 67
 - 紧化 67
 - 编辑源码树 66
 - Source tree 66
- 脚本 60, 60
 - Maxima 59
- 语言 8, 11, 16
- 页面 14, 14
 - 大小 8
 - 断行 43
 - 类型 34
- Paper 9
 - 边白 14
- Screen layout
 - 在纸上显示边白 9
- Beamer theme 51
 - Bluish 51
 - Ice 51
 - Metal 51
 - Reddish 51
 - Ridged paper 51
- Package 64, 64
- Use package 64, 92
 - 程序 58
- 更新 62
- 极限 43
- 查看
 - 演讲模式 51
 - 远程控制 51
- 格式 11, 12
 - 变换 49
 - Resize object 21
- 字型
 - 斜体 7
- 数学
 - 展示风格 19
- 空白 17, 48, 49
- 语言 16
- 页面插入
 - 浮动图形 43
 - 浮动对象 43
 - 浮动表格 43
 - 脚注 43
- 颜色
 - 红色 45
- 段落
 - 栏数 43
- 求值 59
- 源码 70, 70
 - 元组 75
 - 宏命令 70
 - 文本 75
 - 条件 75
 - 流程控制 74
 - 激活 69
 - 仅激活一次 69
 - 激活 69
 - 算术 75
 - 赋值 73
 - Define 70
 - Presentation 69
 - 仅这次使用宏 70
 - 使用宏 70
 - 拉伸 69
 - 紧凑 69
- 源码菜单 70
- 演示 51
- 焦点 8, 12
 - 在前面插入参数 51
 - 在后面插入参数 51, 52
- 屏幕 51

- 插入行(在上方) 34
- 插入行(在下方) 34
- 线箭头 40
- 首选项 51
 - Highlight incorrect formulas 24
 - Line dashes 40
- 版本 49, 50
 - 保留 50
 - 文件
 - 比较 50
 - 显示 49
 - 更新 50
 - 比较
 - 与新版 49
 - 与旧版 49
 - With current user version 50
 - 移动 49
 - 粒度 50, 50
 - Commit 50, 50
 - History 50
 - Register 50
- 算法 43
- 编辑
 - 剪切 45
 - 复制 45
 - 复制到 45
 - 替换 46
 - 查找 46
 - 恢复 46
 - 拼写 46
 - 撤销 46
 - 替换 46
 - 查找 45
 - 粘贴 45
 - 粘贴自 45
 - 首选项 8, 14, 87
 - 工具
 - 版本菜单 49
 - 打印机 8
 - 数学
 - 手动纠正 24
 - 自动纠正 25
 - 语义编辑 23
 - 键盘
 - 自动补全引号 15
 - 自动补全括号
 - 禁用 21
 - 远程控制 51
- Correct
 - Correct all 24
 - Correct manually 24
- 表格
 - 单元格宽度
 - Set width 30
 - 单元格高度
 - Set height 30
 - 水平对齐 30
 - 特定
 - Extract format 31
 - 竖直对齐 30
- 补白 31
- 边界 31
 - Special cell properties
 - 分配未使用空隙 31
 - Special table properties 31
- 详细 50
- 调试
 - 键盘 51
- 转到 8
- 进程
 - 中断执行 57
 - 关闭进程 57
 - 分割进程 58
 - 区域 58
 - 求值
 - 全部求值 57
 - 求上面各值 57
 - 求下面各值 57
 - 进程
 - 创建子进程 58
 - 展开全部区域 58
 - 折叠全部区域 58
 - 清除全部区域 58
- Input
 - 数学输入 59
- Input mode
 - 多行输入 58
 - 数学输入 59
- 首选项
 - 安全 33
 - 转换器
 - TeXmacs-> df/Postscript
 - Expand beamer slides 55
- Add 62
- alt-colors 51
- article 11, 64, 76, 76, 76, 76
- beamer 11, 51, 51, 51, 54
- Block 50
- book 11
- env 76, 76
- env-base 76
- env-float 76
- env-math 76
- env-theorem 76
- exam 11
- framed-session 58
- generic 11
- header 76
- header-article 76, 76
- header-book 76
- Homoglyph substitutions 25
- Insert missing invisible operators 25
- letter 11
- list 76
- Maxima 59
- maxima 63
- new-list 79
- number-europe 63
- Pager 34
- Remove superfluous invisible operators 24

Rough	50	std-list	79
section-article	76	std-markup	76, 76, 77
section-base	76, 82	std-markup	76
seminar	11	std-math	76
session	76	std-symbol	76
source	11, 63, 77, 77	std-utils	76, 85
std	76, 76	structured-section	81
std-automatic	76	Texts	63
std-counter	76	title-base	76, 85
std-latex	78	title-generic	76
std-list	76	World	93