# Ändern: GNU TeXmacs user manual

# Inhaltsverzeichnis

# Kapitel 1
# Erste Schritte

## 1.1. Typografische Konventionen in diesem Handbuch

In diesem TeXmacs Handbuch werden Menü-Einträge in der Schriftart *sans serif* und in der deutschen Version grün dargestellt, z.B. Datei → Laden oder Text → Schriftform → Kursiv. Tastatur-eingaben werden in der Schriftart *typewriter* im Kasten dargestellt, wie z.B. `C-s`. Rechts neben Menü-Einträgen sind, wenn vorhanden, äquivalente Tastenkombinationen angeführt. Darin werden die folgenden Abkürzungen benutzt:

- `S-`. Mit gedrückt gehaltener Umschalttaste
- `C-`. Mit gedrückt gehaltener Control/Steuerung-Taste
- `A-`. Mit gedrückt gehaltener Alt-Taste
- `M-`. Mit gedrückt gehaltener Meta-Taste
- `H-`. Mit gedrückt gehaltener Hyper-Taste

Beispielsweise steht `A-C-b` für `Mit Alt und Strg gedrückt Taste b`. Leerzeichen in den Tastenkombinationen stehen für mehrfache Tastenanschläge. Beispielsweise steht `M-t N b` für `M-t` `N` `b`.

Die Tasten `Alt`, `Meta` und `Hyper` sind nicht auf allen Tastaturen vorhanden. Auf neueren Tastaturen ist die `Meta`-Taste oft durch die `windows`-Taste ersetzt worden. Wenn einer oder verschiedene Modifiziertasten auf Ihrer Tastatur fehlen können Sie die Escapetaste (Esc-Taste) `Esc` anstelle von `M-` verwenden. Entsprechend `Esc Esc` anstelle von `A-` und schließlich eine von den drei Kombinationen `F5`, `Esc Esc Esc` oder `A-C-` anstelle von `H-`. Beispielsweise ist `Esc w` äquivalent zu `A-w`. Um die vollen Vorteile der TeXmacs-Kurzbefehle nutzen zu können, können Sie auch die Modifiziertasten umkonfigurieren. Wie das geht, ist hier beschrieben.

Beachten Sie bitte, dass die TeXmacs-Menüs und das Verhalten der Tastatur vom Kontext abhängig ist, d.h. ihr Verhalten hängt davon ab, in welchem Modus Sie sich gerade befinden (z.B. Text-Modus oder Mathematik-Modus) und wo der Text-Cursor im Dokument steht.

## 1.2. TeXmacs-Konfiguration

Wenn Sie zum allerersten Mal TeXmacs starten, konfiguriert sich das Programm automatisch so, wie es glaubt, dass es das Beste für Sie sei. Beispielsweise versucht das Programm die Ihre Sprache und die Papiergröße Ihres Druckers herauszufinden und richtig einzustellen. Naturgemäß wird ein solcher Automatismus gelegentlich falsch liegen, so dass sie die Ausgangs-Konfiguration abändern wollen. In diesem Fall sollten Sie im Menü Bearbeiten → Einstellungen passende Einstellungen auswählen.

Vor allem empfehlen wir, das gewünschte „Aussehen und Verhalten" von $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ einzustellen. Voreingestellt ist das Aussehen und Verhalten von EMACS, was eine begrenzte Kompatibilität der $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ Tastatur-Kurzbefehle mit denjenigen von EMACS herstellt. $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ hat ein mächtiges System von Tastatur-Kurzbefehlen, das die Verwendung der `Umschalt`- und `Steuerung`-Modifikator-Tasten auf Ihrer Tastatur zu optimieren versucht. Auf vielen X Window Systemen sind diese Tasten aber schlecht konfiguriert, so dass Sie das ändern möchten. Dazu finden Sie detailliertere Informationen im Abschnitt $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ konfigurieren.

## 1.3. Erzeugen, laden und sichern von Dokumenten.

Wenn Sie $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ starten ohne irgendwelche Befehlszeilenoptionen, erstellt der Editor automatisch ein neues leeres Dokument. Sie können jederzeit selbst ein solches neues Leerdokument erstellen mit dem Befehl Datei → Neu. Neue Dokumente tragen noch keinen Namen. Um sie mit einem Namen zu versehen, speichern Sie mit Datei → Sichern als.

Wir empfehlen neuen Dokumenten sofort einen aussagekräftigen Namen zu geben, denn so vermeiden Sie Datenverlust und finden Ihre Datei wieder. Außerdem empfehlen wir, wenn notwendig, sofort die globalen Einstellungen für Ihr Dokument durchzuführen und es danach abzuspeichern. Zuerst sollten Sie den Basis-Stil, Artikel, Buch, Brief usw. mit Dokument → Stil spezifizieren. Wenn Sie Dokumente in verschiedenen Sprachen schreiben, sollten Sie auch die Sprache mit Dokument → Sprache festlegen. Analog können Sie auch die Papiergröße mit Dokument → Seite → Größe spezifizieren.

Nachdem Sie Ihr Dokument geändert haben, speichern Sie mit Datei → Sichern. Vorhandene Dokumente können in den Editor mit Datei → Laden laden. Beachten Sie, dass Sie unter $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ mehrere Dokumente im Editor geladen haben können, zwischen denen Sie mit Nach wechseln können. Die geladenen Dokumente werden auch als *Puffer* bezeichnet.

## 1.4. Dokumente drucken

Sie können das aktuelle Dokument mit Datei → Drucken → Alles drucken als Ganzes ausdrucken. vollständig. Die Drucker-Voreinstellung von $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ sind 600dpi Drucker für A4 Papier. Diese Einstellungen können im Menü Bearbeiten → Einstellungen → Drucker angepasst werden. Sie können auch in eine Postscript-Datei mit Datei → Drucken → Alles in Datei drucken. In diesem Fall werden die Drucker-Voreinstellungen für die Erstellung der Datei benutzt. Sie können alternativ auch Datei → Exportieren → Postscript benutzen, in diesem Fall werden die Voreinstellungen ignoriert.

Sie können die aktuelle Datei in PDF umwandeln mit Datei → Exportieren → PDF. Beachten Sie bitte, dass Sie den Schrifttyp TRUE TYPE mit Bearbeiten → Einstellungen → Drucker → Schrift-Typ → True Type, wenn Sie haben wollen, dass die so erzeugte Postscript- bzw. PDF-Datei TRUE TYPE-Schriften verwendet. Allerdings lassen nur die CM-Schriftarten TRUE TYPE zu. Diese CM-Schriftarten haben eine etwas schlechtere Qualität als die EC-Schriftarten hauptsächlich bei Buchstaben mit Akzenten. Deshalb möchten Sie vielleicht lieber EC-Schriftarten verwenden, solange Sie nicht PDF-Dateien erzeugen wollen, die im ACROBAT READER gut ausschauen.

Wenn Sie den TeXmacs richtig konfiguriert haben, dann ist die Darstellung im Editor garantiert *wysiwyg*: das Druckresultat ist genau das, was Sie auf dem Bildschirm sehen. Um das zu erreichen, müssen Sie vor allem die richtige Papiergröße mit Dokument → Seite → Typ → Papier und Dokument → Ansicht → Seitenlayout → Ränder wie auf dem Papier einstellen. Sie sollten unbedingt sicherstellen, dass die mit Dokument → Schriftart → Punkte pro Zoll (dpi) festzulegende Auflösung genau derjenigen Ihres Druckers entspricht. Momentan können geringfügige Unterschiede im Schriftsatz auftreten, wenn Sie die Auflösung ändern. Leider können diese geringen Unterschiede durch Zeilen- und Seitenumbrüche das ganze Dokument verändern. Dies wollen wir in Zukunft ändern.

# Kapitel 2

# Einfache Dokumente schreiben

## 2.1. Allgemeines

Sobald Sie die vorgehend beschriebenen vorbereitenden Aktionen durchgeführt haben, können Sie Ihren Text eingeben. Die üblichen englischen Buchstaben und Satzzeichen sind auf den meisten Tastaturen vorhanden. Buchstaben mit anderer Sprachen Akzenten können mit der `A-`-Taste systematisch erzeugt werden, z.B. "é" mit `A-' e`, bzw. "à" mit `` A-` a `` usw.. Längere Worte werden, wenn erforderlich, automatisch umgebrochen. Da die Trennungsregeln sprachabhängig sind, sollte die Sprache im Menü Dokument → Sprache korrekt spezifiziert sein.

Auf der linken Seite der Fußzeile sehen Sie den Basis-Stil und die Text-Eigenschaften an der aktuellen Cursorposition. Die Voreinstellung von TeXmacs ist „Allgemein Roman 10 pt", d.h., dass diese Einstellungen gelten, wenn Sie Text eintippen. Sie können die Text-Eigenschaften im Menü Formate einstellen: Schrifttyp, Schriftart, Farbe, Sprache. Die Einstellungen für bereits vorhandenen Text lassen sich ändern, indem Sie ihn markieren und das Formate-Menü benutzen. Einige Text-Eigenschaften auch für das ganze Dokument mit dem Dokument-Menü geändert bzw. eingestellt werden.

Auf der rechten Seite der Fußzeile sehen Sie den Buchstaben oder das Objekt, beispielsweise ein Wechsel in den Text-Eigenschaften an der Stelle unmittelbar vor der Cursorposition. Wir zeigen auch alle Umgebungen, die an der Cursorposition aktiv sind. Diese Informationen sollen Ihnen helfen, sich im Text zurechtzufinden.

## 2.2. Strukturierte Texte schreiben

Lange Dokumente sind normalerweise strukturiert: sie sind in Kapitel, Abschnitte, Unterabschnitte gegliedert und sie enthalten verschiedene Arten von Text, wie normalen Text, Fußnoten, Zitate usw.. Wenn Sie im Menü Dokument → Stil einen geeigneten *Stil* ausgewählt haben, sorgt TeXmacs für das spezielle Dokument-Layout wie beispielsweise Nummerierung von Abschnitten, Seiten, Formeln, Zitaten, Fußnoten usw..

Zur Zeit sind vier Basis-Stile implementiert: Brief, Artikel, Buch und Seminar. Der Seminarstil ist für Dias/Folien gedacht. Sobald sie einen Basis-Stil ausgewählt haben, können Sie Ihren Text in Abschnitte (siehe Text → Abschnitt) gliedern und spezifische *Textmodi*, wie Bemerkung, Satz usw., zu wählen, (siehe Text → Umgebung). Andere Gliederungen sind Auflistungen (siehe Text → Auflistung) oder Auflistungen (siehe Text → Aufzählung).

Wer etwas besser mit TeXmacs vertraut ist, kann auch seine eigenen Textmodi entwerfen und als eigene Stil-Definitionen abspeichern. Wenn Sie z.B. öfter Zitate schreiben und diese gesperrt mit linken und rechten Rändern von jeweils 1 cm darstellen wollen, dann spart es Arbeitszeit, einen eigenen Textmodus zu schreiben, anstatt jedes mal neu die Formatierungen von Hand einzugeben. Es ist nicht nur schneller. Sie können damit auch für das ganze Dokument die Formatierung nachträglich ändern, wenn Sie *a posteriori* feststellen, dass sie beispielsweise die Zitate in einer kleineren Schriftgröße darstellen müssen.

## 2.3. Inhaltliche Auszeichnung

Das einfachste von strukturiertem Text sind Inhaltliche Auszeichnung. Im Menü Text → Inhaltliche Auszeichnung sehen ein Untermenü mit einer Reihe solcher Optionen. Inhaltliche Auszeichnung zeigen, dass ein Stück Text einen bestimmen Zweck erfüllt oder von einer bestimmten Art ist. Beispielsweise sollte wichtiger Text **Stark** hervorgehoben werden. Wie der Text dargestellt wird, hängt vom gewählten Basis-Stil ab. Beispielsweise kann ***Stark*** in einer anderen Farbe dargestellt werden, wenn eine anderer Basis-Sitz benutzt wird.

Hier ist eine Liste der wichtigsten Optionen:

| Option | Beispiel | Zweck |
|---|---|---|
| Stark | das ist **wichtig** | Ein Textstück hervorheben |
| Hervorheben | ein *reelles* Bild | Ein Textstück anders hervorheben |
| Definition | *Mann* = Mensch, männlich | Definition |
| Beispiel | Beispiel-Ligatur æ | Eine Folge von Buchstaben |
| Name | das LINUX System | Die Bezeichnung von etwas |
| Person | Ich bin JORIS | Der Name einer Person |
| Zitat* | Melville's *Moby Dick* | Ein bibliographisches Zitat |
| Abkürzung | Ich arbeite am C.N.R.S. | Eine Abkürzung |
| Akronym | das HTML Format | Ein Akronym |
| Wörtlich | das Programm sagt `hallo` | Wörtlicher Text z.B. Computerausgabe |
| Tastatur | Bitte `return` eingeben | Text, der auf der Tastatur eingegeben werden sollte |
| Quellcode | `cout << 1+1;` yields 2 | Code eines Computer-Programms |
| Variable | `cp` *`src-file dest-file`* | Variablen in einem Computer-Programm |

**Tabelle 2.1.** Die wichtigsten Optionen von Inhaltliche Auszeichnung.

## 2.4. AUFLISTUNG, AUFZÄHLUNG

Mit Text → Auflistung starten Sie eine nicht nummerierte Aufzählung, eine Auflistung. Mit der Taste `Entf` beenden Sie sie. Zur Kennzeichnung der einzelnen Einträge können Sie außer der Voreinstellung noch • (große Punkte), – (Spiegelstrich) oder → (Pfeile) wählen. Aufzählungen können *verschachtelt* werden, wie das folgende Beispiel zeigt:

- Erster Punkt.
- Nun kommt eine weitere Aufzählung:
  - Ein Unterpunkt.
  - Ein weiterer Unterpunkt.
- Der letzte Punkt.

Für die Beendigung innerer Listen gilt das gleiche wie bei der äußeren; `Entf` führt zur nächst äußeren zurück. Die voreingestellte Version verändert, wie oben gezeigt, die Darstellung der Aufzählungsmarke. Die äußerste Aufzählung benutzt •, die nächst innere ∘, usw.. Wenn Sie sich innerhalb einer Aufzählung befinden, startet die Eingabetaste `return` automatisch einen neuen Eintrag. Wenn Sie Einträge haben, die über mehrere Absätze gehen, können Sie `S-return` benutzen, um einen neuen Absatz im gleichen Eintrag zu beginnen.

Nummerierte Aufzählungen, die mit Text → Aufzählung gestartet werden, verhalten sich entsprechend, nur dass die Einträge fortlaufend nummeriert werden. Es folgt ein Beispiel, dass mit Text → Aufzählung → Roman gestartet wurde:

I. Ein erster Punkt.

II. Ein zweiter Punkt.

III. Und schließlich der letzte Punkt.

Der letzte Typ sind beschreibende Auflistungen, bei denen Sie einen beschreibenden Text selbst eingeben können. Sie starten diese mit einem Befehl aus dem Menü Text → Beschreibende Auflistungen z.B. Text → Beschreibende Auflistungen → Dicht, wie im Beispiel unten gezeigt. Sie können dann den beschreibenden Text eingeben und mit linken Pfeiltaste `links` bzw. der Pfeiltaste unten `unten` zur Eingabe des Eintrags wechseln:

**Gnu.** Ein haariges aber friedliches Tier

**Mücke.** Weibliche stechen

Mit Text → Beschreibende Auflistungen → Lang erhalten Sie:

**Gnu.**
Ein haariges aber friedliches Tier

**Mücke.**
Weibliche stechen

## 2.5. Umgebung

*Umgebung* und *Inhaltliche Auszeichnung* sind nahe verwandt. Beide stellen Text in einer Weise dar, die dem Inhalt in besonderer Weise angemessen ist. Während die Befehle im Menü Inhaltliche Auszeichnung jedoch dazu benutzt werden, kurze Textstücke speziell hervorzuheben (mehr hier) dienen die Optionen in Umgebung gewöhnlich zur systematischen speziellen Gestaltung länger Textteile, die oft mehrere Absätze umfassen können. Wichtig ist dabei, dass jede dieser *Umgebungen* seine eigene automatische Zählung vornimmt, sofern eine Nummerierung vorgesehen ist. Beispielsweise werden in der Mathematik die Befehle Satz und Beweis aus dem Menü Umgebung häufig verwendet:

SATZ 2.1. *Es existieren keine positiven ganzen Zahlen $a$, $b$, $c$ und $n$ mit $n \geqslant 3$, so dass $a^n + b^n = c^n$ gilt.*

BEWEIS. Ich habe hier nicht den Platz, um dies zu zeigen. □

Sie finden alle Befehle für Umgebung unter Text → Umgebung: Satz, Beweis, Proposition, Lemma, Folgerung, Axiom und Definition. Innerhalb *Umgebung* können Sie das dueto-Makro benutzen und `\ d u e t o return` eintippen, dann die Person(en), von der die Aussage stammt, und schließlich `Entf` , um das Makro zu verlassen. So erhalten Sie z.B.:

SATZ 2.2. (PYTHAGORAS) *In einem rechtwinkligen Dreieck gilt $a^2 + b^2 = c^2$.*

Andere häufig gebrauchte Optionen, die aber den Text nicht hervorheben, sind Bemerkung, Anmerkung , Beispiel, Warnung und Aufgabe. Die verbleibenden Wörtlich, Quellcode , Zitieren, Zitat und Vers dienen zur Eingabe von wörtlichem Wörtlich, Quellcode, Zitieren, Zitat und Vers in Textteilen mit mehreren Paragraphen.

## 2.6. Anmerkungen zum Layout

Es gilt allgemein, „TeXmacs sorgt für das Layout Ihres Textes". Deshalb sollten Sie vermeiden, Ihr Dokument manuell zu gestalten, indem Sie Leerzeichen oder Leerzeilen über die Tastatur eingeben, um horizontale und vertikale Abstände zu erzeugen, obwohl dies nicht verboten ist und oft der Gewohnheit entspricht. Sie sollten dafür die Befehle aus dem Menü Einfügen → Abstände verwenden. Das hat den Vorteil, dass bei kleinen Änderungen, die den Zeilen- oder Seitenumbruch verändern, und auch bei großen wie einem Wechsel des Dokumentstiles, die Darstellung des Textes unverändert bleibt. Wenn Sie sich daran halten, müssen Sie nach solchen Änderungen nicht den ganzen Text kontrollieren und anpassen.

Mehrere Typen von expliziten Abstandsbefehlen wurden implementiert. Einmal gibt es *fixe* Abstände, die eine vorgegebene Größe haben und behalten. Sie müssen ihre Länge in Standard Längeneinheiten eingeben. Mehr dazu finden Sie unter Standard Längeneinheiten. Zum anderen gibt es variable Abstände, deren Länge vom Zeilenumbruch abhängt. Weiterhin wird unterschieden zwischen horizontalen und vertikalen Abständen. Horizontale Abstände haben Breite aber keine Höhe. Sie können variabel oder fix sein. Vertikale Abstände können vor oder nach einem Absatz eingefügt werden. Der Abstand zwischen zwei Absätzen ist das Maximum von zwei Abständen und zwar dem nach dem vorderen Absatz spezifizierten Abstand einerseits und dem vor dem folgenden Absatz spezifizierten Abstand andererseits. Er ist also **nicht** die Summe beider im Gegensatz zu TeX.

Außerdem können Sie mit der Option Tab Tabulatoren einfügen. Die Funktion der Tabulatoren ist anders als gewöhnlich. Fügt man nur einen Tabulator ein, so wird der nachfolgende Text rechtsbündig an dem rechten Rand eingeführt. Wird ein zweiter Tabulator hinzugefügt, so entsteht eine zweigeteilte Zeile usw., z.B.:

| | | | |
|---|---|---|---|
| kein Tab | | | zwei |
| kein Tab | ein | | zwei |
| kein Tab | ein | zwei | drei |

Absätze können linksbündig, rechtsbündig, zentriert oder als Blocksatz formatiert werden. Außerdem können für jeden Absatz der linke und der rechte Seitenrand und der Erstzeileneinzug sowie die Zeilenabstände innerhalb des Absatzes spezifiziert werden. Wenn für einen größeren Textbereich ein Erstzeileneinzug formatiert wurde, dann kann durch Einzugsmarken der Einzug ab- bzw. angeschaltet werden. Dabei gibt es zwei verschiedene Marken, die auf den vorgehenden Absatz bzw. den nachfolgenden Absatz wirken. Genaueres finden Sie Hier.

Im Menü Dokument → Seite können Sie das Seitenlayout festlegen. Die Art und Weise der Bildschirmdarstellung können Sie mit den Optionen unter Dokument → Seite → Typ beeinflussen. Wenn Sie z.B. Papier wählen, werden die Seitenumbrüche explizit dargestellt. Die Voreinstellung ist Papyrus. Damit werden Seitenumbrüche nicht auf dem Bildschirm gezeigt und die Darstellung ist schneller. Die Einstellung Automatisch unterstellt, dass das Bildschirmfenster exakt der Papiergröße entspricht. Mehr dazu finden Sie Hier. Die Ränder (oben, unten, links, rechts) von Druckseiten werden im Menü Dokument → Seite → Ränder festgelegt. Oft ist es jedoch erwünscht, den Text auf dem Bildschirm anders darzustellen, z. B. die Ränder auf dem Bildschirm zu verkleinern. Das kann man mit den Optionen von Dokument → Ansicht → Seitenlayout.

## 2.7. DAS AUSWAHLSYSTEM FÜR SCHRIFTARTEN

In TeXmacs haben die Schriftarten folgende Haupt-Eigenschaften:

- einen Namen (roman, pandora, concrete, etc.).
- eine Familie (roman, `typewriter` oder sans serif).
- eine Schriftgröße (eine Basisgröße (in Punkten) und eine relative Größe (normal, klein, usw.)).
- eine Schriftstärke (**fett**, normal, dünn).
- eine Form (aufrecht, *kursiv*, *rechts geneigt*, links geneigt, KAPITÄLCHEN, usw.)

Man sollte beachten, dass LaTeX $2\varepsilon$ nicht zwischen Familie und Name unterscheidet. Wichtig ist auch, dass eine Basis-Schriftgröße für das ganze Dokument im Menü Dokument → Schriftart → Größe voreingestellt wird.

## 2.8. DIE TASTATUR MEISTERN

### 2.8.1. Allgemeine Regeln für Tastatur-Kurzbefehle

Weil es eine große Anzahl von Tastatur-Kurzbefehlen in TeXmacs gibt, existieren allgemeine Klassifizierungs-Regeln, die das Erlernen und Behalten dieser Befehle erleichtern sollen. Generell beginnen Tastaturbefehle, die zu einer bestimmten Kategorie gehören, mit der gleichen Modifiziertaste. Die wichtigsten dieser Tasten sind die folgenden:

`C-x`. Dies ist die Steuerungstaste `Strg` auf manchen (englischen) Tastaturen auch mit `Ctrl` beschriftet. Kurzbefehle auf Basis der Steuerungstaste werden für häufig benutzte Editorbefehle benutzt. Es hängt von den Einstellungen im Menü Bearbeiten → Einstellungen → Aussehen und Verhalten ab, welche Kurzbefehle in einzelnen zur Verfügung stehen. Wenn Sie beispielsweise *Emacs* gewählt haben, dann entsprechen Kurzbefehle der Form `C-x` EMACS-Kurzbefehlen, z.B. `C-y` dem Befehl *Text einfügen*.

`A-x`. Die `Alt`-Taste wird für Befehle verwendet, die davon abhängen, in welchem Modus Sie sich gerade befinden. Beispielsweise erzeugt `A-s` **fetten** Text im Textmodus aber das Quadratwurzel-Zeichen im Mathematik-Modus. Beachten Sie bitte, dass zweimaliges Tippen der `Esc`-Taste, also `escape escape` die gleiche Funktion wie `A-` hat.

`M-x`. Die Meta-Taste wird für allgemeine TeXmacs-Befehle genutzt, die in allen möglichen Moden verwendet werden können. Beispielsweise erzeugt `M-!` ein Kennzeichen (label). Diese Taste wird auch für zusätzliche Editierbefehle genutzt, wenn die Einstellungen in Bearbeiten → Einstellungen → Aussehen und Verhalten dies vorsehen. Beachten Sie bitte, dass einmaliges Tippen der `Esc`-Taste, also `escape`, gleichbedeutend ist mit `M-`.

`H-x`. Diese Modifiziertaste wird zur Erzeugung spezieller Symbole, wie z.B. griechischer Buchstaben, benutzt. Sie können Ihre Tastatur so konfigurieren, dass die Hochstelltaste die Rolle der Hypertaste spielt (Bearbeiten → Einstellungen → Tastatur). Die Taste `F5` ist der Taste `H-` äquivalent.

Es sei daran erinnert, dass die Modifiziertasten, die zur Erzeugung von `M-` und `H-` benutzt werden, im Menü Bearbeiten → Einstellungen → Tastatur konfiguriert werden können.

### 2.8.2. Wichtige Tastatur-Kurzbefehle

Die folgenden Kurzbefehle haben in allen Moden die gleiche Funktion:

`S-return`. beginnt einen neuen Absatz.

`C-backspace`. entfernt das TEXMACS-Objekt, die TEXMACS-Umgebung, die den dargestellten Text enthält. Der eingeschlossene Text bleibt erhalten.

`A-Abstand`. füge einen kurzen Abstand ein.

`A-S-Abstand`. füge einen kurzen negativen Abstand ein.

`M-A-Heimverzeichnis`. setze manuell den Beginn einer Auswahl (Markierung).

`M-A-Ende`. setze manuell das Ende einer Auswahl (Markierung).

`M-<`. gehe an den Anfang des Dokuments.

`M->`. gehe an das Ende des Dokuments.

### 2.8.3. Einige Tastaturbefehle für den Textmodus

Um Texte europäischer Sprachen mit einer Tastatur zu schreiben, der bestimmte sprachtypische Akzentuierungs-Tasten fehlen, kann man die folgenden Kurzbefehle im Textmodus verwenden. Beachten Sie bitte, dass diese Befehle unabhängig von der Sprach-Einstellung aktiv sind.

| Kurzbefehl | | Beispiel | | Kurzbefehl | | Beispiel | |
|---|---|---|---|---|---|---|---|
| `A-'` | Akut ´ | `A-' e` | é | `` A-` `` | Gravis ` | `` A-` e `` | è |
| `A-^` | Zirkumflex ˆ | `A-^ e` | ê | `A-"` | Umlaut (Trema) ¨ | `A-" e` | ë |
| `A-~` | Tilde ˜ | `A-~ a` | ã | `A-C` | Cedille ¸ | `A-C c` | ç |
| `A-U` | Brevis ˘ | `A-U g` | ğ | `A-V` | Tschechisch ˇ | `A-V s` | š |
| `A-O` | Nordischer Akzent ˚ | `A-O a` | å | `A-.` | Hochpunkt ˙ | `A-. z` | ż |
| `A-H` | Ungarische Tilde ˝ | `A-H o` | ő | | | | |

**Tabelle 2.2.** Akzentuierte (diakritische) Buchstaben.

Bestimmte spezielle Buchstaben lassen sich sprachunabhängig durch Kurzbefehle erzeugen:

| Kurzbefehl | | | | | | | |
|---|---|---|---|---|---|---|---|
| `S-F5 a` | æ | `S-F5 A` | Æ | `S-F5 a e` | æ | `S-F5 A E` | Æ |
| `S-F5 o` | ø | `S-F5 O` | Ø | `S-F5 o e` | œ | `S-F5 O E` | Œ |
| `S-F5 s` | ß | `S-F5 S` | SS | | | | |
| `S-F5 !` | ¡ | `S-F5 ?` | ¿ | `S-F5 p` | § | `S-F5 P` | £ |

**Tabelle 2.3.** Spezielle Buchstaben.

Wenn man die `"`-Taste drückt, wird automatisch ein Anführungszeichen entsprechend der aktuellen Sprache und dem umgebenden Text eingesetzt. Wenn es nicht Ihren Vorstellungen entspricht können Sie im Menü Bearbeiten → Einstellungen → Tastatur → Anführungszeichen ändern. Sie können auch Anführungszeichen fix eingeben:

| Kurzbefehl | | | |
|---|---|---|---|
| `S-F5 "` | " | `, ,` | „ |
| `< Tab` | ‹ | `> Tab` | › |
| `< <` | « | `> >` | » |

**Tabelle 2.4.** Fixe Anführungszeichen.

“Englische” Anführungszeichen sind Ligaturen zweier Apostrophen. Sie können mit `‘ ‘` bzw. `’ ’` erzeugt werden. Das sind in Wirklichkeit Kurzbefehle: es handelt sich um zwei getrennte Buchstaben nicht um ein spezielles Zeichen.

Einige Kurzbefehle existieren nur in bestimmten Sprachkontexten, die Sie für das ganze Dokument im Menü Dokument → Sprache setzen können oder auch nur lokal im Menü Formate → Sprache.

| Ungarisch | | Spanisch | | Polnisch | | | |
|---|---|---|---|---|---|---|---|
| `S-F5 o` | ő | `! Tab` | ¡ | `S-F5 a` | ą | `S-F5 o` | ó |
| `S-F5 O` | Ő | `? Tab` | ¿ | `S-F5 A` | Ą | `S-F5 O` | Ó |
| `S-F5 u` | ű | `! ‘` | ¡ | `S-F5 c` | ć | `S-F5 s` | ś |
| `S-F5 U` | Ű | `? ‘` | ¿ | `S-F5 C` | Ć | `S-F5 S` | Ś |
| | | | | `S-F5 e` | ę | `S-F5 x` | ź |
| | | | | `S-F5 E` | Ę | `S-F5 X` | Ź |
| | | | | `S-F5 l` | ł | `S-F5 z` | ż |
| | | | | `S-F5 L` | Ł | `S-F5 Z` | Ż |
| | | | | `S-F5 n` | ń | `S-F5 z Tab` | ź |
| | | | | `S-F5 N` | Ń | `S-F5 Z Tab` | Ź |

**Tabelle 2.5.** Sprachspezifische Kurzbefehle.

Sprachspezifische Kurzbefehle haben Vorrang vor den allgemeinen Kurzbefehlen. deshalb ist es beispielsweise schwer in einem ungarischen Kontext ein “ø” zu schreiben.

### 2.8.4. Hybridbefehle und LATEX-Simulation

In TEXMACS können Sie LATEX-Befehle direkt von der Tastatur eingeben. Das geht wie folgt: Sie tippen auf die `\`-Taste, um in den LATEX/TEXMACS-Befehl-Modus zu gelangen. Dann schreiben Sie den Befehl, den Sie eingeben möchten. Danach zeigt die linke Fußleiste in etwa folgendes an:

`return:` *`Befehl`*

Wenn Sie `return` eingeben, wird Ihr Befehl ausgeführt. Beispielsweise können Sie mit `\ f r a c return` einen Bruch erzeugen und werden anschließend zur Eingabe von Zähler und Nenner über die Fußzeile aufgefordert.

Wenn der Befehl, den Sie eingegeben haben, keinem LaTeX-Befehl entspricht, der TeXmacs bekannt ist, dann wird geprüft, ob es sich um ein Makro, eine Funktion oder ein spezielles Layout/Hervorhebung (aus der Stil-Definitions-Datei/einer Paket-Datei) handelt. Ist dies der Fall, dann wird eine entsprechende TeXmacs-Befehlsumgebung mit den notwendigen Argumenten erzeugt. Andernfalls wird unterstellt, dass es sich um eine Umgebungsvariable handelt, deren Wert anschließend erfragt wird. Die `\`-Taste ist einer der folgenden Tastenkombinationen äquivalent `M-i l`, `M-i e`, `M-i a`, `M-i #` oder `M-i v`.

Um den Buchstaben \ (backslash) einzufügen, können Sie `S-F5 \` benutzen.

### 2.8.5. Dynamische Objekte

Bestimmte etwas komplexere Objekte können verschiedene Zustände während der Editiervorgänge einnehmen. Beispiele für *dynamischen Objekte* sind Textmarken und Referenzen, weil ihre Darstellung von Zahlen abhängt, die sich laufend ändern können. Weitere Beispiele dafür finden sich hier.

Wenn man ein dynamisches Objekt, wie z.B. eine Textmarke (label) mit `M-!` einfügt, ist der Vorgabe-Zustand „*inaktiv*". In diesem inaktiven Zustand können notwendige Parameter eingegeben werden, wie in unserem Fall die kennzeichnende Zeichenkette. Manche dynamischen Objekte können eine beliebige Anzahl Parameter aufnehmen. In diesem Fall können neue `Tab`-Taste eingefügt werden.

⟨label|pythagoras⟩

**Abbildung 2.1.** Inaktive Textmarke

Wenn alle relevanten Informationen in das inaktive dynamische Objekt eingegeben sind, dann können sie es mit Wagenrücklauf-Taste `return` *aktivieren*. Ein aktives dynamisches Objekt kann deaktiviert werden, indem man den Cursor unmittelbar dahinter positioniert und dann die Rücktaste `backspace` betätigt.

### 2.8.6. Die Tastatur anpassen

Der Benutzer kann das Tastatur-Verhalten ändern. Wenn Sie das wünschen, dann empfehlen wir Ihnen zuerst die Dateien anzuschauen, die im Ordner `$TEXMACS_PATH/progs/keyboard` stehen, wo das Standard-Verhalten definiert ist. Dann können Sie Ihre eigenen Initialisierungs-Dateien schreiben und damit das Verhalten steuern.

# Kapitel 3

# Mathematische Formeln

Um mathematische Formeln eingeben zu können, müssen Sie erst in den Mathematik-Modus wechseln. Das ist eine spezielle Text-Eigenschaft, die in Objekten aktiviert wird, die mit den Befehlen in dem Menü Einfügen → Mathematik aktiviert wird. Wenn Sie im Mathematik-Modus sind und Bearbeiten → Einstellungen → Ansicht → Kontextabhängige Symbole aktiviert haben, sehen Sie eine größere Anzahl von Icons, die Ihnen Zugang zu den meisten mathematischen Optionen verschaffen.

**Formel** `$`. wird benutzt, wenn in Fließtext kleinere mathematische Formeln eingefügt werden sollen. Dazu dient der Menübefehl Einfügen → Mathematik → Formel.

Formeln werden in einem speziellen Schriftsatz gesetzt, damit sie nicht zu viel vertikalen Platz einnehmen und dennoch lesbar bleiben. Beispielsweise werden obere und untere Schranken deshalb immer links platziert und nicht ober- bzw. unterhalb. Das kann aber erzwungen werden, indem der Stil für eigenständige Formeln aktiviert wird. Dazu dient der Menübefehl Formate → Formelstil → An. Für Formeln im Fließtext ist dieser Stil vorgabemäßig deaktiviert.

**Gleichung** `A-$`. ist die Struktur für eigenständige mathematische Formeln, die in einen eigenen Block gesetzt werden. Sie werden mit dem Menübefehl Einfügen → Mathematik → Gleichung eingefügt.

**Gleichungen** `A-&`. erzeugt einen Block für mehrere mathematische Ausdrücke, eine dreispaltige Tabelle, eqnarray*. Zur Erzeugung dient der Menübefehl Einfügen → Mathematik → Gleichungen. Siehe auch Tabellen erzeugen.

Dieser Kontext wurde für mehrstufige Berechnungen konzipiert, bei denen eine Relation der anderen folgt. Die erste Spalte ist für die linke Seite, die mittlere für das Relationssymbol, z.B. „=", die rechte Spalte für die rechte Seite der Relation.

Im Mathematik-Modus gibt es spezielle Befehle und Kurzbefehle, um mathematische Formeln einzugeben. Z.B. kann die Sondertaste `H-` zur Eingabe griechischer Buchstaben benutzt werden. Denken Sie bitte daran, dass `H-` äquivalent zu `F5` ist und auch mit `escape escape escape` oder `A-C-` eingegeben werden kann.

Der Editor favorisiert mathematische Eingabe mit einer bestimmten Bedeutung. Diese Funktion soll in Zukunft weiter ausgebaut werden, denn es erleichtert die Kommunikation mit Computer-Algebra-Paketen. Momentan sollten Sie das Multiplikationssymbol `*` zwischen Symbolen explizit eingeben, denn die Eingabe von `a b` wird vom Editor als „ab" und nicht als „$a\,b$" interpretiert.

## 3.1. Die wichtigsten mathematischen Konstrukte

Die wichtigsten mathematischen Konstrukte werden mit dem Präfix `A-` erzeugt:

| Kurzbefehl | Zweck | Beispiel |
|---|---|---|
| A-$ | Text | $L=\{x\vert x \text{ is sufficiently large}\}$ |
| A-f | Brüche | $\frac{a}{b+c}$ |
| A-s | Quadratwurzeln | $\sqrt{x+y}$ |
| A-S | $n$-te Wurzeln | $\sqrt[3]{x^3+y^3}$ |
| A-n | Negationen | $\cancel{\frac{a}{b+c}}$ |

**Tabelle 3.1.** Erzeugung der wichtigsten mathematischen Objekte.

Akzente, untere und obere Indices, links und rechts, erzeugt man, wie folgt:

| Kurzbefehl | Zweck | Beispiel |
|---|---|---|
| ' | Accent aigu | $f'$ oder $(g+h)'''$ |
| ` | Accent grave | $`f$ |
| _ | rechter unterer Index | $x_n$ or $x_{i_3}$ |
| ^ | rechter oberer Index | $x^2$, $x_n^2$ or $e^{e^x}$ |
| A-l _ | linker unterer Index | ${}_2x$ |
| A-l ^ | linker oberer Index | ${}^{\pi}x$ or ${}^*_*\mathrm{He}^*_*$ |

**Tabelle 3.2.** Erzeugung von Akzenten und Indices.

| Kurzbefehl | Zweck | Beispiel |
|---|---|---|
| @@ | Unendlich | $\infty$ oder $-\infty$ |

**Tabelle 3.3.** Erzeugung des Symbols für Unendlich.

Einige wichtige mathematische Konstrukte sind in Wirklichkeit tabellarische Konstrukte und werden deshalb getrennt behandelt.

## 3.2. Mathematische Symbole eingeben

Die griechischen Buchstaben können in TeXmacs eingegeben werden, indem die *Hyper-Modifiertaste* H- mit einem Buchstaben kombiniert wird. Beispielsweise ergibt H-a $\alpha$ und H-G gibt $\Gamma$. Erinnern Sie sich, dass F5 zu H- äquivalent ist, so dass man $\rho$ mit F5 r erhalten kann.

In ähnlicher Weise können F6, F7, F8 und S-F6 benutzt werden, um in dieser Reihenfolge, ***fetten Text***, $\mathcal{KALLIGRAFISCHEN}$ $\mathcal{TEXT}$, $\mathfrak{Fraktur}$ und $\mathbb{FETTE}$ $\mathbb{TAFELSCHRFT}$ zu erzeugen. F8 m gibt $\mathfrak{m}$, S-F6 R gibt $\mathbb{R}$ und F6 F7 Z ergibt $\boldsymbol{\mathcal{Z}}$.

Griechische Buchstaben können auch als „Varianten" von lateinischen Buchstaben mit der Tab-Taste erzeugt werden. Beispielsweise liefert p Tab im Mathematik-Modus $\pi$. Die Tab-Taste kann auch dazu gebraucht werden, Varianten von griechischen Buchstaben zu erzeugen. Z.B. H-p tab Tab und p Tab Tab tab ergeben $\varpi$, H-p Tab und p Tab tab $\pi$, H-p und p tab $\pi$.

Viele andere Buchstaben erhält man durch „nahe liegende“ Tastenkombinationen. Beispielsweise ergibt `- >` $\rightarrow$, `- - >` $\longrightarrow$ und `> =` $\geqslant$. Oder `| -` ergibt $\vdash$, `| - >` $\mapsto$ und `- > < -` $\rightleftarrows$.

Es gibt einige Regeln:

- `tab`. ist die wichtigste Taste zur Erzeugung von Varianten. Z.B. `> =` yields $\geqslant$, aber `> = Tab` erzeugt $\geq$. `< Tab` gibt $\prec$, `< Tab =` gibt $\preccurlyeq$ und `< Tab = Tab` gibt $\preceq$. Also, `P Tab tab` erzeugt $\wp$ und `e Tab tab` ergiebt die Konstante $\mathrm{e} = \exp(1)$. Sie könne mit `S-Tab` wieder zurückgehen.

- `@`. liefert Zeichen in Kreisen und Quadraten. Z.B. `@ +` ergibt $\oplus$, `@ x` $\otimes$ und `@ Tab +` gibt $\boxplus$.

- `/`. wird für Negatioen benutzt. `= /` ergibt $\neq$ und `< = /` $\nleqslant$. Beachte, dass `< = Tab Tab /` $\nleqq$ ergibt, während `< = Tab Tab / Tab` $\lneqq$ erzeugt.

- `!`. wird nach Pfeilen benutzt, um untere und obere Indices direkt ober- bzw. unterhalb der Pfeile zu plazieren: `- - > ^ x` ergibt $\longrightarrow^{x}$, aber `- - > ! ^ x` ergibt $\xrightarrow{x}$.

Einige andere Symbole, die nicht durch „nahe liegende“ Tastenkombinationen erzeugt werden können, werden mit dem Präfix `S-F5` eingegeben:

| Kurzbefehl | Symbol | Kurzbefehl | Symbol |
|---|---|---|---|
| `S-F5 a` | $\amalg$ | | |
| `S-F5 n` | $\cap$ | `S-F5 u` | $\cup$ |
| `S-F5 v` | $\vee$ | `S-F5 w` | $\wedge$ |

**Tabelle 3.4.** Einige Symbole, die nicht auf naheliegende Weise mit Tastenkombinationen erzeugt werden können.

## 3.3. Grosse Operatorsymbole

Die folgenden Tastenkombinationen erzeugen große mathematische Symbole:

| Kurzbefehl | Resultat | Kurzbefehl | Resultat |
|---|---|---|---|
| `S-F5 I` | $\displaystyle\int$ | `S-F5 O` | $\displaystyle\oint$ |
| `S-F5 P` | $\displaystyle\prod$ | `S-F5 A` | $\displaystyle\coprod$ |
| `S-F5 S` | $\displaystyle\sum$ | `S-F5 @ +` | $\displaystyle\bigoplus$ |
| `S-F5 @ x` | $\displaystyle\bigotimes$ | `S-F5 @ .` | $\displaystyle\bigodot$ |
| `S-F5 U` | $\displaystyle\bigcup$ | `S-F5 N` | $\displaystyle\bigcap$ |
| `S-F5 V` | $\displaystyle\bigvee$ | `S-F5 W` | $\displaystyle\bigwedge$ |

**Tabelle 3.5.** Große mathematische Symbole.

Das große Integralzeichen erlaubt zwei Varianten, die die oberen und unteren Grenzen des Integrals verschieden darstellen. Die Voreinstellung ist die folgende:

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

Die Alternative:

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

erhält man mit `S-F5 L I`. Das gilt auch für $\oint$ mit der Tastenkombination `S-F5 L O`.

## 3.4. Grosse Klammern

Große Klammern und - eine Spezialität von TeXmacs - große innere Trennzeichen- werden so erzeugt:

| Kurzbefehl | Resultat | Kurzbefehl | Resultat |
|---|---|---|---|
| `A-(` | $\big($ | `A-)` | $\big)$ |
| `A-[` | $\big[$ | `A-]` | $\big]$ |
| `A-{` | $\big\{$ | `A-}` | $\big\}$ |
| `A-<` | $\big\langle$ | `A->` | $\big\rangle$ |
| `A-/` | $\big/$ | `A-\` | $\big\backslash$ |

**Tabelle 3.6.** Kurzbefehle für große Klammern.

In TeXmacs können große Klammern linke Klammern, rechte Klammern oder mittlere Trennzeichen sein. Gemäß Vorgabe sind (, [, { und ⟨ linke Klammern sowie ), ], } und ⟩ rechte Klammern. |, / und \ sind mittlere Trennzeichen. Aber ihr Status kann mit den Tastenkombinationen `A-l`, `A-r` und `A-m` beliebig eingestellt werden. Z.B. erzeugt `A-l )` ), welches aber als linke Klammer betrachtet wird.

TeX und LaTeX kennt keinen mittleren Trennzeichen. Sie sind aber sehr praktisch, um z.B. die senkrechten Linien im unteren Beispiel zu erzeugen.

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

Es können beliebig viele mittlere Trennzeichen zwischen den äußeren Klammern stehen.

## 3.5. Breite mathematische Akzente

Die unten stehende Tabelle zeigt wie Formeln akzentuiert werden können. Einige Akzente können sich an nämlich an die Länge der Formel anpassen:

| Kurzbefehl | Beispiel normal | Beispiel breit | Kurzbefehl | Resultat |
|---|---|---|---|---|
| A-~ | $\tilde{x}$ | $\widetilde{x+y}$ | A-' | $\acute{x}$ |
| A-^ | $\hat{x}$ | $\widehat{x+y}$ | A-` | $\grave{x}$ |
| A-B | $\bar{x}$ | $\overline{x+y}$ | A-. | $\dot{x}$ |
| A-V | $\vec{x}$ | $\overrightarrow{\mathrm{AB}}$ | A-" | $\ddot{x}$ |
| A-C | $\check{x}$ | $\widecheck{x+y}$ | | |
| A-U | $\breve{x}$ | $\widebreve{x+y}$ | | |

**Tabelle 3.7.** Kurzbefehle zur Erzeugung breiter Akzente.

# Kapitel 4
# Tabellen

## 4.1. Tabellen erzeugen

Um Tabellen zu erzeugen kann man Befehle aus dem Einfügen → Tabelle verwenden. Dabei erzeugen Einfügen → Tabelle → Kleiner eigenständiger Tabellenblock und Einfügen → Tabelle → Große Tabelle nur Umgebungen mit Beschriftung und Zählern für die eigenständige Tabellen. In diese müssen die eigentlichen Tabellen noch, wie im Folgenden beschrieben, eingefügt werden. Dazu dienen Befehle aus dem Einfügen → Tabelle oder die folgenden Kurzbefehle. Wenn der Cursor sich in einer solchen Tabelle befindet, sind Sie im Tabellen-Modus. Sofern Sie Bearbeiten → Einstellungen → Ansicht → Kontextabhängige Symbole aktiviert haben, sehen Sie eine größere Anzahl von Icons, die Ihnen Zugang zu den vielen Tabellen-Optionen verschaffen.

Befehle zur Erzeugung von Tabellen:

- `M-t N t`. Eine normale Tabelle erzeugen. Zellen sind linksbündig. Keine sichtbaren Ränder. Menü-Befehl: Einfügen → Tabelle → Normaler Tabulatormodus.
- `M-t N T`. Eine normale Tabelle mit zentrierten Zellen. Menü-Befehl: Einfügen → Tabelle → Zentrierter Tabulatormodus.
- `M-t N b`. Eine Tabelle mit sichtbaren Umrandungen. Zellen linksbündig. Menü-Befehl: Einfügen → Tabelle → Normaler Block.
- `M-t N B`. Eine Tabelle mit sichtbaren Umrandungen. Zellen zentriert. Menü-Befehl: Einfügen → Tabelle → Zentrierter Block.

Im Mathematik-Modus, in Kontexten für eigenständige Formeln, können noch einige weitere tabellenartige Strukturen erzeugt werden:

- `M-t N m`. Erzeuge eine Matrix: $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$
- `M-t N d`. Erzeuge eine Determinante: $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$
- `M-t N c`. Erzeuge eine Auswahlliste: $\begin{cases} x \geqslant 0, f = 0 \\ x = 0, f = 1 \\ x \leqslant 0, f = 0 \end{cases}$

Der Kontext eqnarray* ist eine spezielle tabellenartige Struktur, die sich über eine ganze Zeile erstreckt. Man erzeugt diese mit dem Menübefehl Text → Mathematik → Gleichungen.

Wenn man eine neue Tabelle erzeugt hat, hat dies die Minimal-Größe und enthält normalerweise nur eine einzige Zelle, die leer ist. Neue Zeilen und Spalten können mit den Kurzbefehlen `A-links`, `A-rechts`, `A-hoch` und `A-runter` erzeugt werden. Beispielsweise erzeugt `A-rechts` eine neue Spalte rechts von der Cursorposition. Neue Zeilen unterhalb kann man auch mit `return` erzeugen. Man verlässt die Tabellen mit den Pfeiltasten oder der Maus.

## 4.2. Den Formatier-Modus auswählen

In TeXmacs können beliebige Blöcke von Zellen auf bestimmte Weise formatiert und angepasst werden. Z.B. können Sie einzelnen Zellen ein Hintergrundfarbe geben, Sie können aber beispielsweise auch eine ganze Spalte zentriert darstellen. Gemäß Vorgabe operieren die individuellen Formatierbefehle mit einzelnen Zellen. Das kann aber mit dem Menü-Befehl Tabelle → Zellenbearbeitungsmodus oder den folgenden Kurzbefehlen geändert werden:

`M-t m c`. Operiere auf einzelnen Zellen.

`M-t m h`. Operiere auf Zeilen.

`M-t m v`. Operiere auf Spalten.

`M-t m t`. Operiere auf der ganzen Tabelle.

Man kann aber auch einen Block von Zellen mit der Maus auswählen und eine Formatierung des ganzen Blocks auf einmal durchführen.

## 4.3. Ausrichtung von Tabellen und Zellen

Die häufigste Formatierungs-Operation die vertikale oder horizontale Ausrichtung von einzelnen Zellen oder Zellblöcken. Sie können `M-←`, `M-→`, `M-↑` und `M-↓` verwenden, um schnell nach links, rechts, oben oder unten auszurichten.

Spezifische Ausrichtungen können auch mit den Menü-Befehlen Tabelle → Horizontale Zellenausrichtung und Tabelle → Vertikale Zellenausrichtung gewählt werden. Alternativ können Sie dafür auch die Kurzbefehle der Form `M-t h x` und `M-t v x` für horizontale bzw. vertikale Ausrichtung benutzen. `x` kann für die folgenden Tasten stehen: `l` für „links", `c` für „zentriert", `r` für „rechts", `b` für „unten", `t` für „oben", `B` für „vertikale Basislinie", `R` für „horizontale Basislinie", `.` für „zum Dezimalpunkt" und `,` für „zum Dezimalkomma".

Auf ähnliche Weise können Sie beeinflussen, wie die ganze Tabelle zum umgebenden Text ausgerichtet sein soll. das kann entweder mit den Menü-Befehlen aus den Menüs Tabelle → Horizontale Tabellenausrichtung und Tabelle → Vertikale Tabellenausrichtung oder mit den Kurzbefehle der Form `M-t H x` oder `M-t V x` verwenden. Hier kann `x` für die folgenden Tasten stehen: `l` für „links", `c` für „zentriert", `r` für „rechts", `b` für „unten", `t` für „oben" und für die vertikale Ausrichtung noch `f` für „Bruchstrich-Höhe".

## 4.4. Grösse von Tabellen und Zellen

Mit Befehlen aus den Menüs Tabelle → Zellenbreite → Breite einstellen bzw. Tabelle → Zellenhöhe → Höhe einstellen kann man Höhe und Breite der Zellen festlegen. Dafür gibt es drei verschiedene Weisen:

**Minimum Modus.** Die aktuelle Breite/Höhe ist das Minimum der festgelegten Breite/Höhe und der Breite/Höhe der Box in der Zelle.

**Fix-Modus.** Die Breite/Höhe ist genau die vorgegebene.

**Maximum Modus.** Die aktuelle Breite/Höhe ist das Maximum der festgelegten Breite/Höhe und der Breite/Höhe der Box in der Zelle.

Die Breite der sichtbaren Umrandung der Zelle und das Padding, die Breite des Leerraums um die Zeichen, werden bei der Berechnung der Boxgröße berücksichtigt.

Man kann auch Höhe und Breite der gesamten Tabelle festlegen mit den Befehlen aus dem Menü Tabelle → Spezielle Tabelleneigenschaften. Insbesondere kann man die Tabelle über die gesamte Absatzbreitel laufen lassen. Wenn man eine Tabellen-Breite/Höhe festlegt, kann man mit den Befehlen aus Tabelle → Spezielle Zelleneigenschaften → Den nicht benutzten Freiraum verteilen spezifizieren, wie der unbenutzte Raum zu verteilen ist. Die Vorgabe ist die Gleichverteilung.

## 4.5. Sichtbare Zellränder, Padding und Hintergrundfarbe

Sie können die Breite der sichtbaren Umrandung von Zellen und die Breite des Paddings, der Leerraum um die Zeichen in der Zelle, in alle vier Richtungen festlegen: links, rechts oben und unten. Dazu können sie Befehle aus dem Menü Tabelle → Zellenumrandung benutzen oder Kurzbefehle von der Form `M-t b` *x* für die Breite und `M-t p` *x* für das Padding. Dabei kann *x* für die folgenden Tasten stehen: `l` für „links", `r` für „rechts", `b` für „unten" und `t` für „oben".

Die Vorgabebreite von Umrandungen ist `1ln`, mit anderen Worten die Standard Linienbreite in der aktuellen Schriftart z.B. bei einem Bruchstrich. Die Umrandung erscheint rechts und unten, außer in der ersten Zeile oder ersten Spalte. Die horizontale Paddingvorgabe ist `1spc`: die Standardbreite eines Leerzeichens in der aktuellen Schriftart. Die vertikale Paddingvorgabe ist `1sep`: der Standard-Minimal-Abstand zwischen zwei nahen Boxen.

Zellen kann eine Hintergrundfarbe mit Befehlen aus Tabelle → Hintergrundfarbe gegeben werden.

Der gesamten Tabelle kann auch eine Umrandung und ein Tabellenpadding gegeben werden. Dazu dienen die Befehle aus dem Menü Tabelle → Spezielle Tabelleneigenschaften → Sichtbarer Rand. In diesem Fall erfolgt das Padding außerhalb der Umrandung.

## 4.6. Erweiterte Tabellen-Eigenschaften

In den Menüs finden Sie einige speziellere Tabellen-Funktionen. U.a. sind dies folgende:

- Sie können eine Zelle vergrößern. Diese hat dann die Höhe und Breite mehrerer Zellen (rechts bzw. unten von der Ursprungszelle). Menübefehl: Tabelle → Spezielle Zelleneigenschaften → N-fache Großzelle.

- Einfügung von ganzen Unter-Tabellen in einzelnen Zellen. Menübefehl: Tabelle → Spezielle Zelleneigenschaften → Untertabelle.

- Korrektur der vertikalen und horizontalen Ausrichtung von Text, damit die Basislinien übereinstimmen. Menübefehl: Tabelle → Spezielle Zelleneigenschaften → Texthöhenkorrektur.

- Horizontaler Umbruch des Zelleninhalts. Menübefehl: Tabelle → Spezielle Zelleneigenschaften → Trennung.
- Vertikaler Seiten-Umbruch von ganzen Tabellen.
- Mehrere Spalten und/oder Zeilen können verbunden werden. Die verbundenen Zellen werden Teil des sichtbaren Randes der verbleibenden Zellen. Menübefehl: Tabelle → Spezielle Zelleneigenschaften → Zellen verbinden.
- Desaktivierung der Tabelle, um ihren Quellcode sehen zu können. Menübefehl: Tabelle → Spezielle Tabelleneigenschaften → Deaktivieren.
- Eine Referenzzelle definieren. Danach werden alle neu geschaffenen Nachbarzellen gleichartig formatiert. Menübefehl: Tabelle → Spezielle Tabelleneigenschaften → Referenzzelle festlegen.
- Minimale und maximale Tabellen-Größe in Zellzahlen angeben. Dies wird beim Editieren berücksichtigt und ist vor allem bei der Definition von Makros wichtig. Menübefehl: Tabelle → Spezielle Tabelleneigenschaften → Größenbegrenzungen.

Derzeit sind alle Tabellen Inhalte eines Kontexts: tabular, block, matrix, usw.. Wenn Sie eigenen Makros schreiben wollen, können Sie mit dem Befehl Tabelle → Spezielle Tabelleneigenschaften → Format herausziehen das formatierende Konstrukt einer existierenden Tabelle aus seinem Kontext extrahieren.

# Kapitel 5

# Verknüpfungen, Hyperlinks und automatisch erzeugte Verzeichnisse

## 5.1. Label, Verknüpfungen, Hyperlinks und Referenzen erzeugen

Sie können ein neues inaktives Label erzeugen mit dem Kurzbefehl `M-!` oder dem Menübefehl Einfügen → Verknüpfung → Textmarke. Label sind Marken, die mit einer Referenz verbunden sind und die später durch eine Zahl, die auf die Referenz verweist, ersetzt wird. Man erzeugt die Referenz `M-?` oder mit dem Menübefehl Einfügen → Verknüpfung → Verweis. Die Plazierung eines Labels sollte mit Bedacht erfolgen, um eine korrekte Nummerierung zu erreichen. Wenn z.B. Abschnitte einen Label erhalten sollen, dann setzt man am besten das Label direkt hinter den Abschnitts-Namen. Wenn man „Gleichungen“ mit einem Label versieht, sollte man einen Punkt innerhalb der Gleichung verwenden.

Man kann Hyperlinks zu anderen Dokumenten mit dem Kurzbefehl `M-i >` oder mit Einfügen → Verknüpfung → Hyperlink erzeugen. Der Hyperlink hat zwei Felder, zwei Variable. Die erste Variable ist der mit der Verknüpfung assoziierte Text, der, wenn der Link aktiviert ist, blau im Text erscheint. Die zweite Variable ist der Name des Dokuments. Wie bei Hyperlinks üblich, kann mit *`#Label`* auf ein Label im gleichen Dokument verwiesen werden. Sonst ist die allgemeine Form *`url#label`* (oder ohne Label: *`url`*) und verweist auf Label *`#Label`* in *`url`*. *`url`* kann dabei ein Dokument in Netz sein.

Ganz ähnlich kann eine Aktion mit Text oder Graphik verknüpft werden. Man benutzt dazu den Kurzbefehl I `M-i *` oder Einfügen → Verknüpfung → Aktion. Das zweite Feld enthält dann ein ausführbares Guile/Scheme-Skript, das ausgeführt wird, wenn man auf den Text oder die Graphik mit der Maus doppelklickt. Aus Sicherheitsgründen werden solche Skripts nicht immer akzeptiert. In der Voreinstellung werden Sie gefragt, ob Sie das Skript ausführen wollen. Dieses Verhalten kann in Optionen → Sicherheit geändert werden. Beachten sie, dass Sie mit dem Guile/Scheme-Befehl:

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

einen Befehl Ihrer Systemumgebung (shell) ausführen können.

Schließlich kann man mit `M-i i` bzw. Einfügen → Verknüpfung → Einfügen ganze Dokumente in den Text einfügen. Damit können Sie z.B. den Quellcode eines Programms in den Text einfügen und zwar so, dass jegliche Änderung am Quellcode sich automatisch im Textdokument wiederfindet.

## 5.2. Bilder einfügen

Man kann Bilder mit den Befehlen aus dem Menü Einfügen → Bild in den Text einfügen. Zur Zeit erkennt $\TeX_{\text{MACS}}$ `ps`, `eps`, `tif`, `pdf`, `pdm`, `gif`, `ppm`, `xpm` und `fig` Datei-Formate. $\TeX_{\text{MACS}}$ benutzt selbst das gs-Format (mit anderen Worten ghostscript) zur Darstellung von Postscript-Bildern. Wenn Sie ghostscript noch nicht auf ihrem Rechner haben, dann können Sie es von `www.cs.wisc.edu/~ghost/index.html`

herunterladen. Zur Zeit werden die anderen Formate in Postscript umgewandelt. Dazu werden die Skripte `tiff2ps`, `pdf2ps`, `pnmtops`, `giftopnm`, `ppmtogif` und `xpmtoppm` verwendet. Wenn diese nicht auf Ihrem Rechner sind, sollten Sie sich diese besorgen.

Gemäß Vorgabe werden die Bilder in Ihrer ursprünglichen Größe dargestellt. Die folgenden Operationen können vorgenommen werden:

- Ausschneiden eines Rechtecks. Die untere linke Ecke ist der Bezugspunkt für die Rechteck-Größe.
- Größen-Änderung. Wenn nur die Höhe oder nur die Breite spezifiziert wird, bleibt das Seitenverhältnis bei der Änderung erhalten.
- Vergrößerung des Maßstabs. Höhe und Breite werden mit dem gleichen Multiplikator versehen.

TeXmacs enthält außerdem ein Skript zur Konvertierung von Bildern nach eps (encapsulated PostScript). Die Bilder dürfen LaTeX-Formeln enthalten. Es sei daran erinnert, dass man LaTeX-Formeln in `xfig`-Bilder einfügen kann, indem man sie als Text, mit der Schriftart (font) LaTeX und das spezielle Flag in den Textflags setzt.

## 5.3. Inhaltsverzeichnis erzeugen

Es ist sehr leicht ein Inhaltsverzeichnis für ein Dokument zu erstellen. Sie brauchen nur den Cursor auf die Stelle in Ihrem Dokument zu setzen, wohin Sie das Inhaltsverzeichnis haben wollen und dann den Befehl Text → Automatisch → Inhaltsverzeichnis auszuführen.

Wenn Sie ein Inhaltsverzeichnis erzeugen, sollten Sie in einem Modus sein, in dem die Seitenumbrüche sichtbar sind. Wählen Sie dafür Dokument → Seite → Typ → Papier, damit korrekte Seitenzahlen berechnet werden können. Danach sollten Sie Dokument → Aktualisieren → Inhaltsverzeichnis oder Dokument → Aktualisieren → Alles ausführen, um das Inhaltsverzeichnis zu erstellen. Wiederholen Sie den letzten Schritt solange, bis sich das Verzeichnis nicht mehr ändert. Durch Veränderungen im Inhaltsverzeichnis können sich die Seitenzahlen ändern.

## 5.4. Literaturverzeichnis erstellen

Im Moment benutzt TeXmacs `bibtex`, um Literaturverzeichnisse zu erstellen. Deshalb sind folgende Schritte dazu notwendig:

- Zunächst muss eine `.bib`-Datei mit allen erforderlichen Zitaten erstellen. Diese Datei sollte das Standard LaTeX-Zitate-Format haben.
- Mit Befehlen aus dem Menü Einfügen → Verknüpfung → Zitat fügen Sie dann ein Zitat mit einem Label, ein das einer Referenz in der `.bib`-Datei entspricht und aktivieren mit der `enter`-Taste.

- An der Stelle, an der das Literaturverzeichnis eingefügt werden soll, klicken Sie auf Text → Automatisch-erzeugte-Listen → Literaturverzeichnis. An der Eingabeaufforderung geben Sie einen `bibtex`-Stil ein und ihre `.bib`-Datei. Es gibt sehr viele `bibtex`-Stile. Standard-Stile sind z.B.: **plain** (Die Einträge sind alphabetisch sortiert und mit Zahlen gekennzeichnet), **unsr**t (wie plain, nur erscheinen die Einträge in der Reihenfolge, in der sie erstmalig im Text auftreten), **alpha** (wie plain, nur wird mit einem Buchstaben-Nummern-Code gekennzeichnet, der aus den Autoren-Namen und dem Jahr der Publikation gebildet wird), **abbrv** ( wie plain, die Vornamen der Autoren, Monate, Name der Zeitschrift usw. werden abgekürzt.).

- Benutzen Sie Dokument → Aktualisieren → Literaturverzeichnis, um das Literaturverzeichnis zu erstellen. Wiederholen Sie bitte den Vorgang mehrmals, bis sich nichts mehr ändert.

## 5.5. STICHWORTVERZEICHNIS ERZEUGEN

Um ein Stichwortverzeichnis zu erstellen, müssen Sie erst einmal Stichwort-Einträge (Index-Einträge) in Ihrem Dokument machen. Dazu dient die Befehle aus dem Menü Einfügen → Verknüpfung → Indexeintrag. Als zweites müssen Sie an der Stelle, an der das Stichwortverzeichnis erscheinen soll, den Menübefehl Text → Automatisch-erzeugte-Listen → Index ausführen. Das Stichwortverzeichnis wird dann in ähnlicher Weise erstellt wie das Inhaltsverzeichnis.

Im Menü Einfügen → Verknüpfung → Indexeintrag finden Sie verschiedene Arten von Stichwort-Einträgen. Die einfachsten sind „Haupt-", „Unter-" und „Unter-Unter-", welche Makros mit ein, zwei bzw. drei Argumenten sind. Einträge der Formen „Unter-" und „Unter-Unter-" werden benutzt um sie als Unter-Stichworte zu anderen einzuordnen.

Ein *komplexer Index-Eintrag* hat vier Argumente. Das erste ist ein Schlüssel, der die Sortierung angibt. Er muss ein Tupel sein und wird mit `M-i <` erzeugt. Die erste Komponente des Tupels ist die Haupt-Kategorie, die zweite eine Unter-Kategorie usw.. Das zweite Argument eines *komplexen Index-Eintrags* ist entweder leer oder es ist „strong". In diesem Fall wird der Eintrag **fett** dargestellt. Das dritte Argument ist meist leer. Wenn sie dagegen zwei Index-Einträge mit dem gleichen nicht leeren dritten Argument benutzen, dann erzeugt TeXmacs einen Seitenummern-Bereich. Das vierte Argument ist wieder ein Tupel; es ist der Eintrag selbst.

Man kann auch Einträge in das Stichwortverzeichnis vornehmen, ohne dass eine Zeilennummer erscheint. Benutzen Sie dazu Einfügen → Verknüpfung → Indexeintrag → Ohne Seitenzahl. Das erste Argument diese Makros ist ein Schlüssel zur Einsortierung. Das zweite Argument enthält den eigentlichen Text. Damit kann man beispielsweise Kennzeichen für Abschnitte in das Stichwortverzeichnis einbringen, z.B. „A", „B" usw..

## 5.6. GLOSSAR ERSTELLEN

Glossare werden ähnlich wie Stichwortverzeichnisse mit Befehlen aus dem Menü Einfügen → Verknüpfung → Glossareintrag erstellt. Sie sind aber nicht sortiert. Ein Eintrag vom Typ „normal“ enthält etwas Text und erhält eine Seitennummer. Der Typ „mit Erklärung“ besitzt ein zweite Argument, der den Eintrag erklärt. Der Typ „doppelt" , wenn ein Eintrag an einer anderen Stelle noch einmal gebraucht wird. Er erhält dann eine zweite Seitenzahl. Der Typ „ohne Seitenzahl“ liefert einen Eintrag ohne Seitenzahl.

## 5.7. Bücher, aus mehreren Dateien bestehende Dokumente

Wenn ein Dokument wirklich groß wird, ist es sinnvoll, es in mehrere Teile zu zerlegen. Das macht es leichter, die einzelnen Teile an anderer Stelle weiter zu verwenden und macht zugleich den Editor schneller. Eine ganze Datei kann in eine andere mit dem Befehl Einfügen → Verknüpfung → Einfügen einfügen. Um den Zugriff auf die einzelnen Dokumente zu beschleunigen, werden die eingefügten Dokumente in einen Puffer geladen. Um alle eingefügten Dokumente zu aktualisieren, benutzt man Werkzeuge → Aktualisieren → Eingefügte Dokumente.

Wenn man ein Buch schreibt, nimmt man üblicherweise für jedes Kapitel eine Datei, z.B. `c1.tm`, `c2.tm` bis `cn.tm`. Dann erzeugt man eine Datei book.tm, in die man die Dateien `c1.tm`, `c2.tm` bis `cn.tm` auf die oben beschriebene Weise einfügt. Auch das Inhaltsverzeichnis, Literaturverzeichnis usw. werden normalerweise in `book.tm` eingefügt.

Um Referenzen zu anderen Kapiteln zu sehen, wenn man ein bestimmtes Kapitel `ci.tm` editiert, kann man `book.tm` zur Steuerdatei für die Dateien `c1.tm` bis `cn.tm` Dokument → Steuerdatei → Beifügen machen. Im Moment existiert in diesem Fall noch keine automatische Kapitel-Nummerierung. Deshalb muss man die Kapitel-Nummern mit Hand über die Kontext-Variable *chapter-nr* während der Editierung einfügen.

# Kapitel 6

# Werkzeuge zum Editieren

## 6.1. Auswählen, Ausschneiden und Einfügen

Sie können Text und Formel markieren, indem Sie die linke Maustaste gedrückt halten und die Maus bewegen. Mit Bearbeiten → Ausschneiden oder mit der `entf`-Taste schneiden Sie das markierte Textstück aus, das Sie mit Bearbeiten → Einfügen an der Cursor-Position einfügen können. Um ein markiertes Textstück zu kopieren, benutzen Sie Bearbeiten → Kopieren mit nachfolgendem Bearbeiten → Einfügen. Alternativ können Sie auch das zu kopierende Textstück markieren, den Cursor neu positionieren und durch Drücken der mittleren Maustaste die Auswahl an die Cursorposition kopieren.

Wenn Sie ein Textstück durch Markieren ausgewählt haben, können Sie die Eigenschaften dieser Auswahl ändern. Z.B. können Sie sie rot färben mit Text → Farbe → Rot. Oder, wenn Sie einen Teil einer Formel auswählen und dann auf Einfügen → Mathematik → Bruch, wird aus der Auswahl der Zähler eines Bruchs.

Wenn man den Ausschneiden-und-Einfügen-Mechanismus zur Kommunikation mit anderen Anwendungen benutzt, wird das TeXmacs-Daten-Format verwandt. Mit Bearbeiten → Importieren bzw. Bearbeiten → Exportieren kann man andere Datenformate erzeugen. Ausschneiden und Einfügen benutzt vorgabemäßig den aktuellen Textpuffer (aktuelle geladene Datei). Mit Bearbeiten → Kopieren nach und Bearbeiten → Einfügen aus können beliebige Textpuffer verwendet werden.

## 6.2. Suchen und Ersetzen

Man kann Text suchen mit den Befehlen `C-s` oder Bearbeiten → Suchen. Der zu suchende Text ist in die Fuß-Zeile auf der linken Seite hinter dem Text „suchen“ einzugeben. Während der Suche bleibt er dort sichtbar. Jeder Buchstabe, den sie eingeben wird angehängt und im Text wird das jeweilige erstmalige Auftreten mit einer roten Box markiert. Wenn Sie `C-s` erneut eingeben, wird weiter gesucht. Ein Piep zeigt, dass die Suche beendet ist. Mit `C-s` beginnt die Suche erneut am Beginn des Dokuments. Man kann `⟵` benutzen, um Eingaben während der Suche zurückzunehmen. Die Suche unterscheidet zwischen Klein- und Groß-Buchstaben.

Normalerweise wird Text von der Cursorposition ab in vorwärts Richtung durchsucht. Sie können mit `C-r` auch rückwärts suchen. Während der Suche wird Text nur im gleichen Modus und in der Sprache,die an der Startposition aktiv war, gefunden. Mit anderen Worten, wenn Sie von einer Gleichung aus, als im Mathematik-Modus etwa nach x suchen, finden sie kein x, das sich im normalen Text befindet. Weiterhin kann zur Zeit nur nach einfachen Textstücken gesucht werden, nicht nach mathematischen Symbolen oder einem kompliziert strukturierten Text.

Gleichzeitig Suchen und Ersetzen kann man mit `C-=` oder Bearbeiten → Ersetzen. Sie werden zur Eingabe eines zu suchenden Textstücks aufgefordert und zur Eingabe des Textes, der an seiner Stelle eingefügt werden soll. Wird ein zu ersetzendes Textstück gefunden, werden Sie aufgefordert, durch Eingabe von y, n oder a anzugeben, ob sie ersetzen wollen (y), nicht ersetzen wollen (n) alle alle Vorkommen ohne Rückfrage ersetzt haben wollen (a). Es gelten ansonsten die gleichen Beschränkungen wie bei der Suche.

## 6.3. RECHTSCHREIBPRÜFUNG

Wenn Sie das Programm `ispell` installiert haben, dann können sie es zur Rechtschreibprüfung benutzen, indem Sie `M-$` oder den Menübefehl Bearbeiten → Rechtschreibprüfung ausführen. Sie sollten sich aber vergewissern, dass die Wörterbücher, der von Ihnen benutzten Sprachen vorhanden sind.

Wenn Sie die Rechtschreibprüfung gestartet haben, entweder für den gesamten Text oder für eine Auswahl, werden Sie bei jedem falsch geschriebenen Wort zu einer Aktion aufgefordert. Die möglichen Aktionen finden Sie in der Fußzeile:

**a).** Akzeptiert das Wort und jedes weitere Auftreten.

**r).** Ersetzt das falsch geschriebene Wort durch eine Korrektur, die Sie einzugeben haben.

**i).** Besagt, dass das „Falsch geschriebene Wort“ korrekt geschrieben ist und so in das Wörterbuch aufgenommen werden soll.

**1-9).** Mehrere Vorschläge, die sie verwenden können.

Beachten Sie, dass `ispell` nur nach falsch geschriebenen Worten sucht. Es werden keine grammatikalischen Fehler gefunden.

Wenn Sie die Rechtschreibprüfung starten, wird das Wörterbuch derjenigen Sprache verwendet, die an der Cursorposition bzw. am Beginn der Auswahl aktiv war. Nur Text in dieser Sprache wird geprüft. Wenn Ihr Text mehrere verschiedene Sprachen benutzt, d.h. das Textteile mit Befehlen des Menüs Formate → Sprache explizit als Text einer bestimmten Sprache formatiert wurden, dann müssen Sie die Rechtschreibprüfung für jede dieser Sprachen einzeln durchführen.

## 6.4. ÄNDERUNGEN ZURÜCKNEHMEN ODER WIEDERHOLEN

Man kann die Änderungen, die man in einem Dokument vorgenommen hat, schrittweise zurücknehmen und zwar von dem Moment an, an dem Sie $\mathrm{T_EX_{MACS}}$ gestartet haben. Man erreicht das mit Bearbeiten → Zurück bzw. `M-[` oder `C-_`. Nach dem Zurücksetzen kann man mit Zurückkehren in den ursprünglichen Zustand wieder zurück: Bearbeiten → Wieder oder `M-]`.

Die Anzahl gespeicherter Änderungen ist standardmäßig auf 100 beschränkt. Man kann die Zahl aber in der persönlichen Initialisierungs-Datei z.B. auf 1000 erhöhen, indem man die folgende Zeile einfügt:

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

Wenn Sie eine negative Zahl angeben, gibt es kein obere Schranke außer Ihrem Speicher.

# Kapitel 7
# Fortgeschrittene Layout-Eigenschaften

## 7.1. Ströme

Komplexe Dokumente enthalten oft Fußnoten oder Bewegliche Objekte, die anders dargestellt werden als der Haupt-Text. Komplexere Dokumente bestehen aus mehreren *Strömen*, aus denen sich das endgültige Dokument zusammengesetzt wird, einem für den Haupt-Text, einen für die Fußnoten, ein anderer für bewegliche Objekte und noch eine anderer für mehrspaltigen Text. Die verschiedenen Ströme verteilen sich über die Seiten ziemlich unabhängig von einander.

Um eine Fußnote einzufügen, benutzen Sie Einfügen → Seiteneinfügung → Fußnote. Die Anzahl der Spalten kann im Menü Absatz → Spaltenanzahl eingestellt werden.

## 7.2. Bewegliche Objekte

Bewegliche Objekte dürfen auf der Seite unabhängig vom Haupt-Text positioniert werden. Normalerweise enthalten sie Abbildungen oder Tabellen, die zu groß sind, um leicht in den Haupt-Text zu passen. Ein solches Objekt wird mit Einfügen → Seiteneinfügung → Bewegliches Objekt erzeugt.

Sie können aber auch ein bewegliches Objekt, erstellen, das eine Abbildung oder eine Tabelle, enthält. Dazu dienen die Befehle Einfügen → Seiteneinfügung → Bewegliche Abbildung bzw. Einfügen → Seiteneinfügung → Bewegliche Tabelle. Manchmal will man aber mehrere kleine Abbildungen oder Tabellen in einem beweglichen Objekt unterbringen. Man kann dies mit den Befehlen Einfügen → Bild → Kleine Abbildung bzw. Einfügen → Tabelle → Kleiner eigenständiger Tabellenblock.

Nachdem Sie ein bewegliches Objekt erzeugt haben, können Sie im gewissen Umfang steuern, wo dieses Objekt eingefügt werden soll. Dazu dient das Menü Einfügen → Seiteneinfügung → Bewegliches Objekt positionieren. Dazu müssen Sie sich aber innerhalb des beweglichen Objektes befinden. Sie können festlegen, dass das bewegliche Objekt „ganz oben“ auf der Seite, „ganz unten“, „hier“ oder auf „anderen Seiten“ erscheint. Die Voreinstellung ist überall. Ein bewegliches Objekt wird, sofern es nicht alleine auf einer Seite steht, nie weniger als drei Zeilen von der oberen oder der unteren Seitengrenze entfernt sein.

## 7.3. Seitenumbruch

Der Seitenumbruch kann sehr genau mit den Befehlen aus dem Menü Dokument → Seite → Umbruch gesteuert werden. In dem Untermenü Algorithmus können Sie einen Algorithmus auswählen. „professionell“ bringt die besten Ergebnisse, kann aber das Editieren verlangsamen, wenn man ihn im interaktiven „Dokument → Seite → Typ → Papier“-Modus benutzt. Dokument → Seite → Algorithmus → Nachlässig ist der schnellste. Dokument → Seite → Algorithmus → Mittel entspricht „professional“ außer im Fall von Mehrspalten-Satz. Dann ist er deutlich schneller.

Sie können außerdem dem Seitenumbruch-Algorithmus erlauben, die Seitenlänge etwas zu verkleinern oder zu vergrößern, wenn es sonst zu Problemen kommen könnte. Dazu dient das Untermenü Grenzen. Die Dehnbarkeit von vertikalem Leerraum zwischen Absätzen kann im Untermenü Flexibilität festgelegt werden. Der Faktor 1 ist die Vorgabe, ein kleinerer Faktor sorgt für strengere Einhaltung des Satzspiegels, die Qualität der Seitenumbrüche kann aber schlechter werden.

# Kapitel 8
## TeXmacs Plugins

Es gibt viele Wege, TeXmacs anzupassen, zu erweitern und zu verbessern: Anwender können ihre eigenen Stile entwerfen, die TeXmacs-Oberfläche anpassen und Verknüpfungen mit externen Programmen schreiben. Das Plugin-System bietet einen universellen Mechanismus, um mehrere solche Erweiterungen in ein einziges Paket zusammenzufassen. Plugins sind einfach zu schreiben, zu installieren und zu pflegen.

### 8.1. Ein Plugin installieren und benutzen

Anwender finden meist Plugins auf irgendwelchen Webseiten. Ein solches Plugin, z.B. *`myplugin`* liegt normalerweise als „getarte” Version

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

vor. Wenn Sie Ihr TeXmacs in das Verzeichnis `$TEXMACS_PATH` installiert haben, dann sollten Sie diese Datei in dem Verzeichnis `$TEXMACS_PATH/plugins` auspacken mit dem Befehl

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

Damit wird ein Unterverzeichnis *`myplugin`* unter `$TEXMACS_PATH/plugins` angelegt. Wenn Sie nun TeXmacs starten, sollte Ihr Plugin automatisch erkannt werden. Bitte lesen Sie die Dokumentation, die mit dem Plugin kommt, um den Gebrauch des Plugins zu erlernen.

Bemerkung 8.1. Wenn Sie TeXmacs nicht selbst installiert haben, oder wenn Sie keinen Zugriff auf `$TEXMACS_PATH` haben, dann können Sie auch in das Verzeichnis `$TEXMACS_HOME_PATH/plugins` gehen und dort Ihren „Tarball” auspacken. Erinnern Sie sich, dass `$TEXMACS_HOME_PATH` gemäß Vorgabe das Verzeichnis `$HOME/.TeXmacs` ist. Nachdem Sie Ihr Plugin so installiert haben, sollte TeXmacs nach dem Neustart auch hier Ihr Plugin automatisch erkennen.,

Bemerkung 8.2. Wenn Ihr Plugin in Form des Quellcodes geliefert wird, z.B. als mit einem Namen der folgenden Art: *`myplugin-version`*`-src.tar.gz`, dann müssen Sie den Code noch kompilieren, bevor Sie TeXmacs starten. Sie müssen zuerst in das myplugin-Verzeichnis wechseln (Befehl: cd myplugin) und dann je nach Plugin, lesen Sie die Anweisungen, den Befehl

`make`

oder

`zuerst: ./configure anschließend: make`

ausführen.

Bemerkung 8.3. Um eine neue Version des Plugins die Dateien im entsprechenden Plugin-Verzeichnis `$TEXMACS_PATH/plugins` oder `$TEXMACS_HOME_PATH/plugins` mit

```
rm -rf myplugin
```

und installieren die neue Version, wie oben beschrieben.

## 8.2. Eigene Plugins schreiben

Um ein Plugin *myplugin* zuschreiben, sollten Sie zuerst ein Verzeichnis erstellen

`$TEXMACS_HOME_PATH/plugins/myplugin`

das Sie alle notwendigen Dateien enthalten wird. Denken Sie bitte daran, dass der `$TEXMACS_HOME_PATH` gemäß Vorgabe `$HOME/.TeXmacs` ist. Zusätzlich können Sie die folgenden Verzeichnisse erzeugen, falls Sie sie brauchen sollten:

`bin` — Für Binärdateien.

`doc` — Für die Dokumentation (noch nicht unterstützt).

`langs` — Für Sprach-Dateien wie z.B. Wörterbücher (noch nicht unterstützt).

`lib` — Für Bibliotheken.

`packages` — Für Stil-Pakete.

`progs` — Für Scheme-Programme.

`src` — Für Quellcode.

`styles` — Für Stil-Definitionen.

Generell gilt, Dateien, die sich in diesen Verzeichnissen befinden, werden automatisch erkannt, wenn TeXmacs startet. Wenn z.B. ein `bin` Unterverzeichnis existiert, dann wird

`$TEXMACS_HOME_PATH/plugins/myplugin/bin`

automatisch zu der `PATH`-Kontext-Variablen hinzugefügt. Beachten Sie, dass die Verzeichnisstruktur eines Plugins derjenigen von `$TEXMACS_PATH` ähnelt.

Beispiel 8.4. Der simpelste Typ von Plugins besteht nur aus Daten-Dateien, wie z.B. einer Sammlung von Stil-Definitionen und Stil-Paketen. Dazu genügt es, die Verzeichnisse

`$TEXMACS_HOME_PATH/plugins/myplugin`

`$TEXMACS_HOME_PATH/plugins/myplugin/styles`

`$TEXMACS_HOME_PATH/plugins/myplugin/packages`

herzustellen und die Dateien in die entsprechenden Verzeichnisse zu kopieren. Danach werden diese automatisch nach einem Neu-Start von TeXmacs, in den Menüs Dokument → Stil bzw. Dokument → Paket benutzen erscheinen.

Komplexere Plugins, wie z.B. Plugins mit zusätzlichem Scheme oder C++ Code muss man meist noch eine Scheme-Konfigurations-Datei erstellen

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

Diese Konfigurations-Datei sollte eine Anweisung der folgenden Form

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

enthalten. Darin beschreiben ***configuration-options*** die Tätigkeiten, die beim Start durchzuführen sind, einschließlich der Frage, ob der Code in Ordnung ist. In den nachfolgenden Abschnitten werden wir anhand von einfachen Beispielen dir Arbeitsweise und die Programmierung von Plugins erläutern. Viele weitere Beispiele finden Sie unter

`$TEXMACS_PATH/examples/plugins`

`$TEXMACS_PATH/plugins`

Einige werden eingehender im Kapitel über Schnittstellen beschrieben.

## 8.3. Beispiel für ein Plugin mit Scheme-Code

**Das `world` plugin.**

Betrachten wir das `world` Plugin im Verzeichnis

`$TEXMACS_PATH/examples/plugins`

Es zeigt wie man $\mathrm{T_EX_{MACS}}$ erweitert mit ein wenig zusätzlichen Scheme-Code, den Sie in der Datei

`world/progs/init-world.scm`

finden. Um das Plugin zu testen müssen Sie sie das Verzeichnis world/progs/init-world.scm rekursiv in das Verzeichnis `$TEXMACS_PATH/plugins` bzw. `$TEXMACS_HOME_PATH/plugins` kopieren. Wenn Sie dann $\mathrm{T_EX_{MACS}}$ erneut starten, sollte das Plugin automatisch erkannt werden und eine entsprechendes Menü eingerichtet werden.

**Wie es funktioniert..**

Die Datei `init-world.scm` enthält den folgenden Code:

```
(define (world-initialize)
  (menu-extend texmacs-extra-menu
    (=> "World"
        ("Hello world" (insert-string "Hello world")))))

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

Die Konfigurations-Option `:require` spezifiziert eine Bedingung, die erfüllt sein muss, damit das Plugin von $\mathrm{T_EX_{MACS}}$ wird. Später werden wir damit überprüfen, ob bestimmte Programme vorhanden sind oder nicht. Hier ist sie, da auf wahr gesetzt, praktisch unwirksam. Wenn die Bedingung nicht erfüllt wäre, würde die Konfiguration abgebrochen.

Die Option `:initialize` gibt eine Anweisung, die durchgeführt werden soll, wenn die Bedingung erfüllt ist. In unserem Beispiel erzeugen wir im Hauptmenü einen Menü-Eintrag World und einen Menü-Punkt World → Hello world, der dazu benutzt werden kann, den Text „Hello world" in das Dokument einzufügen. Im allgemeinen sollte eine solche Routine kurz sein und ein Modul laden, das die wirkliche Initialisierung durchführt. Das hat den Vorteil, dass kleine `init-`***`myplugin`***`.scm` Dateien die Zeit, die $\mathrm{T_EX_{MACS}}$ zum Hochfahren braucht, kurz hält.

## 8.4. 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 TEXMACS, the plug-in should now be automatically recognized.

**How it works.**

The `minimal` plug-in demonstrates a minimal interface between TEXMACS 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 Text → Sitzung → Minimal.

## 8.5. Zusammenfassung der Konfigurations-Optionen für Plugins

Wie bereits gesagt wurde, sollte die SCHEME-Konfigurations-Datei *`myplugin`*`/progs/init-`*`myplugin`*`.scm` eines Plugins mit dem Namen ***`plugin`*** eine Anweisung der Form

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

enthalten. Hier folgt eine Liste von vorhandenen `configuration-options`:

`(:require condition)` — Diese Option spezifiziert eine Bedingung `condition`, die erfüllt sein muss, damit das Plugin funktioniert. Normalerweise wird überprüft, ob bestimmte Dateien vorhanden sind. Wenn die Bedingung nicht wahr ist, dann wird TeXmacsso weitermachen, als ob das Plugin gar nicht existiert. Die Konfiguration wird abgebrochen. Die `:require` Option ist normalerweise die allererste.

`(:version version-cmd)` — Diese Option führt einen Scheme-Ausdruck `version-cmd` aus, der zu der Version des Plugins evaluiert.

`(:setup cmd)` — Diese Anweisung wird nur ausgeführt, wenn die Version des Plugins sich seit dem letzten Aufruf von TeXmacs geändert hat. Das passiert eigentlich nur, wenn Sie neue Versionen von TeXmacs oder Hilfsanwendungen neu installiert haben..

`(:initialize cmd)` — Diese Option führt den Scheme-Ausdruck `cmd` aus. Er folgt normalerweise unmittelbar auf `:require` option, so dass das Plugin nur konfiguriert wird, wenn es auch existiert. Auch große Plugins sollten eine kleine `myplugin/progs/init-myplugin.scm` Datei haben, damit der TeXmacs-Start nicht zu sehr verzögert wird, denn diese Routine muss bei jedem Start abgearbeitet werden. darum sollte der Großteil der Scheme-Anweisungen in einer eigenen Datei untergebracht werden, die von dem Initialisierungs-Befehl geladen wird.

`(:launch shell-cmd)` — Diese Option erklärt, dass das Plugin Ausdrücke über eine Pipeline evaluieren kann, die eine Hilfsanwendung benutzt, welche mit dem System-Befehl `shell-cmd` gestartet wird..

`(:link lib-name export-struct options)` — Diese Option entspricht weitgehend :launch, nur wird die externe Anwendung damit dynamisch eingebunden. Weitere Informationen finden Sie im Abschnitt Dynamisch ladbare Bibliotheken.

`(:session menu-name)` — Diese Option erklärt, dass das Plugin ein Anwendung unterstützt, die in interaktiven Sitzungen Eingaben evaluieren kann. Ein Menü-Punkt `menu-item` wird in das Menü **Einfügen → Sitzung** eingetragen, um damit solche Sitzungen zu starten.

`(:serializer ,fun-name)` — Wenn das Plugin zur Evaluierung taugt, dann dient diese Option dazu die Scheme-Funktion `fun-name` zu benennen, die TeXmacs-Bäume ind Zeichenketten umformt.

`(:commander ,fun-name)` — Diese Anweisung entspricht weitgehend `:serializer`, nur dass :commander benutzt wird, um spezielle Befehle in Zeichenketten umzuwandeln .

`(:tab-completion #t)` — Diese Anweisung erklärt, dass das Plugin Text-Ergänzungen unterstützt.

`(:test-input-done #t)` — Diese Anweisung erklärt, dass das Plugin ein Routine hat, die testen kann, ob die Eingabe vollständig ist.

# Kapitel 9

## GNU TeXmacs als Schnittstelle benutzen

TeXmacs kann mit externen Programmen kommunizieren und zwar in einer Form, die der Kommunikation mit der Systemumgebung ähnelt. Man kann in einer „Sitzung“ Befehle auf externen „Computer Algebra Systemen“ (CAS) ausführen und die Resultate in einer ansprechenden graphischen Weise darstellen lassen. Innerhalb solcher Sitzungen kann man auch Befehle der Systemumgebung und Scheme-Programme ausführen..

### 9.1. Sitzungen erzeugen

Eine Sitzung kann mit Befehlen aus dem Menü Text → Sitzung gestartet werden. Sie besteht aus einer Folge von Eingabe- und Ausgabe-Feldern. Wenn man `return` innerhalb eines Eingabefeldes benutzt, dann wird der Text innerhalb dieses Feldes von dem externen Programm evaluiert und in einem Ausgabefeld gezeigt.

Wenn man einen Befehl in einer Sitzung eingibt, so versucht die externe Anwendung, diesen auszuführen. mehrere Befehle können innerhalb eines Dokuments auf einmal gestartet werden, die Ausgabe ist aber nur dort aktiv, wo sich der Cursor befindet und auch nur dort wird ausgegeben. Deshalb empfehlen wir, unterschiedliche Puffer zu verwenden, wenn mehrere Befehle parallel ausgeführt werden sollen.

Für jede externe Anwendung können Sie wählen, ob Sie einen Prozess für mehrere Sitzungen gemeinsam haben wollen, oder ob jede Sitzung seinen eigen Prozess haben soll. Genauer gesagt, wenn Sie eine Sitzung mit dem Befehl Einfügen → Sitzung → Anders in ein Dokument einfügen, dann können Sie die „Sitzungsart” (Shell, Pari, Maxima usw.) angeben und einen „Sitzungsnamen” (Vorgabe ist „default”). Sitzungen mit unterschiedlichen Namen entsprechen verschiedenen Prozessen und Sitzungen, die einen gemeinsamen Namen haben, gehören zum gleichen Prozess.

Um den Prozess, der einer Sitzung zugrunde liegt, zu beenden, benutzen Sie Sitzung → Sitzung schließen. Die Sitzung-Befehle erscheinen automatisch, wenn Sie sich in einer Sitzung befinden.Wenn man `return` in einer Sitzung drückt, die nicht mehr verbunden ist, dann wird diese wieder automatisch gestartet. Sie können Sitzung → Ausführung abbrechen benutzen, um die Ausführung eines Befehls in der Sitzung zu unterbrechen, wenn die Anwendung dies erlaubt.

### 9.2. Sitzungen editieren

Innerhalb der Eingabefelder von Sitzungen haben die Cursor-Tasten eine besondere Bedeutung. Die Tasten ↑ und ↓ führen zu den vorgehenden bzw. nachfolgenden Feldern. Die Tasten ← und → verlassen nie das Eingabefeld. Dazu sollten Sie die Maus benutzen.

Die Menüs Sitzung → Feld einfügen und Sitzung → Entferne Felder, die zugänglich werden, wenn Sie sich in einer Sitzung befinden, stellen einige Optionen zum Editieren bereit. Die meisten Operationen wirken auf zugehörige Ein- bzw. Ausgabe-Felder. Mit dem Befehl Sitzung → Feld einfügen → Text Feld kann zu einem Eingabefeld ein erklärendes Textfeld eingefügt werden. Kurzbefehle zum Einfügen von Eingabefeldern sind `A-hoch`, ( `A-↑`, Sitzung → Feld einfügen → Insert field above) und `A-runter`, ( `A-↓`, Sitzung → Feld einfügen → Insert field below). Mit dem Kurzbefehl `A-löschen`, ( `Entf`, Sitzung → Entferne Felder → Remove inputfield ) wird das aktuelle Eingabefeld entfernt. `A-backspace`, ( `A-⟵`, Sitzung → Entferne Felder → Remove inputfield above ) entfernt das vorgehende Eingabefeld. Mit Sitzung → Entferne Felder → Remove all outputfields werden alle Ausgabefelder entfernt.

Mit Sitzung → Feld einfügen → Eingabe-Feld zusammenfalten oder `A-rechts` ( `→` ) können Sie eine „Unter-Sitzung" starten. Die aktuellen Eingabe-, Ausgabe- und Text-Felder werden zum Rumpf einer neuen nicht verborgenen „Unter-Sitzung" . Diese bestehen aus einem erklärenden Text und einer Folge von Text-, Eingabe- und Ausgabe-Feldern. Man kann verborgene Unter-Sitzungen mit `M-A-↑` sichtbar machen und mit `M-A-↓` verbergen. Unter-Sitzungen werden besonders gut dargestellt, wenn man das `varsession`-Paket benutzt: Dokument → Paket zufügen → Programm → Varsession.

Nützlich ist auch der bereits erwähnte Befehl Sitzung → Entferne Felder → Entferne alle Ausgabe-Felder und zwar besonders für Demonstrationszwecke, wenn man eine Sitzung erzeugt, die erst später ausgeführt werden soll. Ein anderer nützlicher Befehl ist Sitzung → Die Sitzung aufteilen, mit dem eine Sitzung geteilt werden kann. Meist wird das benötigt, um die einzelnen Teile in Veröffentlichungen einzufügen.

## 9.3. EINE EINGABEMETHODE AUSWÄHLEN

In der Grundeinstellung versucht TeXmacs das Eingabefeld zu evaluieren, wenn `return` gedrückt wird. Mehrzeilige Eingaben können mit `S-return` erzeugt werden. Alternativ können Sie auch den Mehrzeilen-Modus mit Sitzung → Eingabe Modus → Mehrzeilen-Modus auswählen. Dann wirkt die `return`-Taste wie sonst die `S-return`-Tastenkombination. Beachten Sie aber bitte, dass einige Anwendungen eigene heuristische Systeme besitzen, um herauszufinden, ob die Eingabe beendet wurde oder nicht. Wenn das nicht der Fall ist, kann `return`, wie gewohnt, reagieren

Bei bestimmten Anwendungen können Sie mathematische Eingaben in einer graphischen zweidimensionalen Form eingeben. Diese Form der Eingabe können Sie mit Sitzung → Eingabe Modus → Mathematik auswählen. Wenn diese Möglichkeit besteht, dann können Sie meist auch Ausgaben in die Eingabe kopieren. Das hängt aber ganz von der entsprechenden Anwendung ab.

## 9.4. UNTERSTÜTZTE ANWENDUNGEN

Wenn Sie das Einfügen → Sitzung-Menü betrachten, werden Sie feststellen, dass nur diejenigen Anwendungen gezeigt werden, die tatsächlich auf Ihrem System vorhanden sind. Die einzigen Ausnahmen sind System-Umgebungs-Sitzungen und SCHEME-Sitzungen.

Unten finden Sie eine kurze Liste von Computer Algebra Systemen, für die eine TeXmacs-Schnittstelle existiert. Es gibt auch eine Reihe von anwendungseigenen Schnittstellen, dafür sollten Sie aber die Dokumentation dieser Systeme studieren.

### 9.4.1. System-Umgebungs- und Scheme-Sitzungen

In einer „System-Umgebungs-Sitzung" können Sie System-Befehle ausführen. Jegliche Ein- und Ausgabe ist „verbatim" (wörtlich). Es wurden bislang keinerlei spezielle System-Shell-Eigenschaften implementiert. Die Ausgabe vollzieht sich genauso wie das Programm abarbeitet und ausgibt.

In einer„Scheme-Sitzung" können Sie Guile/Scheme-Programme laufen lassen. Die Eingabe sollte „wörtlich" sein. Die Eingabe wird evaluiert und das Resultat angezeigt.

### 9.4.2. Giac

Giac ist ein Computer Algebra System, das von

`http://www-fourier.ujf-grenoble.fr/~parisse/english.html`

heruntergeladen werden kann.

### 9.4.3. GTybalt

GTybalt ist eine freies Computer Algebra System, dass auf GiNaC, CLN und einem Interpreter für C und C++ basiert. Weiter Informationen finden Sie unter:

`http://www.fis.unipr.it/~stefanw/gtybalt.html`

### 9.4.4. Macaulay 2

Macaulay 2 ist eine neues Programm, zur Unterstützung von wissenschaftlichen Untersuchungen in algebraischer Geometrie und kommutativer Algebra. Die Software ist jetzt als Quellcode vorhanden, sowie kompiliert für Linux, Suna OS, Solaris, Windows und einigen anderen UNIX-Maschinen. Zu erhalten von

`http://www.math.uiuc.edu/Macaulay2`

### 9.4.5. Maxima

Maxima ist nicht nur eines der ältesten und besten Computer Algebra Systeme. Es ist auch ein wirkliches Universal-System mit einer freien Implementation. Zu erhalten von:

`http://www.ma.utexas.edu/users/wfs/maxima.html`

Die unterstützte Version ist das GCL-basierte Maxima 5.6. Für CLisp-basiertes Maxima 5.6, müssen Sie `tm_maxima` editieren und `-load` durch `-i` ersetzen. Für Maxima 5.9-pre muss `-load` durch `-p` ersetzt werden. Bekannte Probleme:

- Wenn man `return` drückt und der Befehl ist nicht komplett (weil z.B. ; oder `$` fehlt), dann hängt sich die Schnittstelle auf.

- Wenn man einen Lisp break Prompt (Fehlerausgabe) erzeugt, hängt sich die Schnittstelle auf.

- Das Kommando `info` wird nicht unterstützt, da es in dem darunter liegenden Lisp implementiert ist und nicht Maxima selbst. Daher ist die Portierung schwierig.

- Einige Befehle des Debuggers arbeiten korrekt, aber einige, einschließlich :c, arbeiten nicht und keiner weiß warum.
- Der `load`-Befehl benimmt sich manchmal eigenartig.

### 9.4.6. Pari

PARI ist ein Programm für Computer-unterstützte Zahlentheorie. Es enthält eine C-Library, `libpari` (mit optionalen Assembler-Cores für einige häufige Computer-Architekturen) sowie einen programmierbaren interaktiven `gp` Rechner. Weitere Informationen unter

`http://www.math.u-psud.fr/~belabas/pari/`

Herunterzuladen von

`ftp://megrez.math.u-bordeaux.fr/pub/pari`

Sie brauchen eine Version, die neuer ist als PARI-2.1.0, wenn sie sie mit TEXMACS benutzen wollen. (Zur Ermittlung der Version eines bereits installierten System benutzen Sie: `gp --version`).

### 9.4.7. Qcl

QCL ist eine höhere Programmiersprache für Quanten-Computer, die Architektur unabhängig ist. Sie wurde von klassischen Prozedursprachen wie C oder PASCAL abgleitet. Damit erhält man eine komplette Implementierung und Simulation von Quanten-Algorithmen, einschließlich klassischer Komponenten in einem in sich konsistenten Formalismus. Die TEXMACS-Schnittstelle dient vorwiegend dazu, die Quantenzustände in einer lesbaren Form darzustellen. Weitere Informationen finden Sie unter

`http://tph.tuwien.ac.at/~oemer/qcl.html`

Beginnend mit Version 1.0.0.8 unterstützt TEXMACS QCL 0.4.3 oder neuer. Anwender von älteren Versionen sollten auf eine neuere umstellen.

### 9.4.8. Yacas

YACAS ist, wie der Name suggerieren soll, noch ein anderes Computer Algebra System. Implementiert sind u.a. beliebige Genauigkeit, Rationalzahlen, Vektoren, komplexe Zahlen und Matrizenrechnung einschließlich Inversen, Determinanten und der Lösung von Matrizengleichungen. Außerdem Ableitungen, Lösung von Gleichungen, Taylorreihen, numerische Lösung von Gleichungen nach der Newton-Methode und eine weiterer nicht mathematischer Algorithmen. Die Sprache unterstützt Variablen und Anwender-definierte Funktionen. Es gibt Basis-Unterstützung für univariate Polynome, Integrale und Tensorrechnung. Yacas gibt es bei

`http://www.xs4all.nl/~apinkus/yacas.html`

# Kapitel 10

## So schreiben Sie Ihre erste $\mathrm{T_EX_{MACS}}$-Stil-Definition

Eine der grundlegenden Stärken von TEXMACS ist, dass Sie Ihre eigenen Stile definieren und vorhandene freizügig anpassen können. Dazu können Sie Stil-Definitions-Dateien und Stil-Pakete schreiben. Diese erfüllen gleichzeitig mehrere Aufgaben:

- Sie dienen zur abstrakten Definition von repetitiven Elementen in Texten wie z.B. Abschnitte, Nummerierungen usw..

- Sie bieten Mechanismen zur Text-Strukturierung. Man kann z.B. ein Textstück als Abkürzung, als Zitat oder als „wichtig" definieren.

- Mit den Standard-Basis-Stilen können Sie professionell gestaltete Dokumente schreiben, denn die zugehörigen Stil-Definitionen wurden mit großer Sorgfalt von Leuten geschrieben, die viel von Typographie und Ästhetik verstehen.

Jedes Dokument kann mit mehreren Stilen assoziiert werden, die sowohl Standard-Stile sein können als auch vom Anwender selbst definiert sein können. Der Haupt-Stil, der Basis-Stil, wird im Menü Dokument → Stil ausgewählt. Er entspricht in der Regel dem Dokument, das Sie schreiben wollen: Brief, Buch, Veröffentlichung usw. oder eine bestimmende Layout-Politik, wie Sie z.B. viele Verlage vorgeben. Dazu können weitere Stil-Pakete hinzugefügt werden, die im Menü Dokument → Paket zufügen auszuwählen sind. Diese Stil-Pakete modifizieren den Basis-Stil. Z.B. dient das Paket `number-europe` Paket dazu, die europäische Art der Nummerierung von Abbildungen, Beweisen usw. einzuführen, bei jeder einzelne Typ auch einzeln gezählt wird. Das Paket `maxima` enthält Makros, um die Ausgabe des Computer Algebra Systems MAXIMA ansprechend zu formatieren, wenn man TEXMACS als Oberfläche dafür benutzt. Mehrere Pakete können gleichzeitig verwendet werden.

Wenn Sie eigenes Layout schreiben wollen oder wenn Sie vorhandenes Layout Ihren Bedürfnissen anpassen wollen, dann müssen Sie sich entscheiden, ob Sie einen vollständig neuen Basis-Stil schreiben wollen oder ein Stil-Paket. In den meisten Fällen werden Sie möglicherweise es vorziehen, ein Stil-Paket zu schreiben, denn dann können Sie dieses mit beliebigen anderen Paketen kombinieren. In einigen Fällen ist es vorteilhafter einen neuen Basis-Stil zu schaffen, meist, indem Sie einen vorhandenen verändern. Das ist hauptsächlich dann der Fall, wenn man die Layout-Politik eine bestimmten Zeitschrift nachäffen will. In diesem Kapitel werden wir erklären, wie man Basis-Stile und Stilpakete selbst schreibt oder vorhandene anpasst.

### 10.1. Ein einfaches Stil-Paket schreiben

Lassen Sie uns an einem einfachen Beispiel erklären, wie man ein einfaches Stil-Paket schreibt.

Wenn Sie das Beispiel direkt am Rechner nachvollziehen wollen, erleichtern Sie sich die Arbeit, wenn Sie ein zweites TeXmacs parallel eventuell in einem anderen virtuellen Fenster starten und zwischen den beiden Instanzen wechseln.

Zuerst einmal müssen Sie einen neuen Puffer, d.h. eine neue leere Textdatei, erzeugen. Dazu wählen Sie das Menü Datei → Neu und wählen den `Quellcode` Basis-Stil unter Dokument → Basis-Stil → Quellcode-Menü. Dann speichern Sie mit einem aussagekräftigen Namen und der Datei-Ergänzung `.ts` in Ihr Paket-Verzeichnis:

`$HOME/.TeXmacs/packages`

Beachten Sie bitte, dass der Knopf *Texte* im Datei-Browser dem Verzeichnis

`$HOME/.TeXmacs/texts`

entspricht. Daher können Sie durch Doppelklick auf .. und danach auf `packages` schnell in dieses Verzeichnis wechseln. Ganz entsprechend enthält das Verzeichnis

`$HOME/.TeXmacs/styles`

Ihre persönlichen Basis-Stil-Dateien. Nach dem Sichern mit der Dateiergänzung `.ts` sollte das leere Stil-Paket automatisch in dem Menü Dokument → Paket einfügen erscheinen. Wenn Sie ein Unterverzeichnis im Verzeichnis `$HOME/.TeXmacs/packages` erstellen, erzeugen Sie automatisch ein neues Untermenü und wenn Sie da hinein ein Stilpaket speichern einen neuen Menüpunkt in dem entsprechenden Untermenü.

Lassen Sie uns nun ein einfaches Makro hi erzeugen, das „Hello world” auf dem Bildschirm ausgibt. Zuerst tippen Sie `A-=` oder `M-i =`, um eine Zuordnung, engl. „assignment”, zu erzeugen. Sie sollten nun auf dem Bildschirm Folgendes sehen

⟨assign||⟩

Geben Sie nun “hi” als erstes Argument ein, gehen zum zweiten Argument und tippen `A-m` oder `M-i m` um ein Makro einzufügen. Jetzt sollte es so aussehen:

⟨assign|*hi*|⟨macro|⟩⟩

Schließlich schreiben Sie „Hello world” in den Rumpf des Makros. Ihr Dokument sollte nun aus folgender Zeile bestehen:

⟨assign|*hi*|⟨macro|Hello world⟩⟩

Nachdem Sie Ihr Stil-Paket unter einem Namen gespeichert haben, können Sie das Makro verwenden, z.B., indem Sie ein neues Dokument erstellen und es mit Ihrem Stil-Paket mit Dokument → Paket hinzufügen verbinden. Sie benutzen das Makro hi durch Eintippen von `\ h i` mit nachfolgendem Drücken der Eingabetaste, `return`.

Analog können Sie Makros mit Argumenten erzeugen, die Sie zur Laufzeit eingeben und im Makro auswerten können. Wenn Sie z.B. in gleicher Weise ein Makro hello erzeugt haben, können Sie mit der Tastenkombination `A-links` oder `M-i links` im Makrorumpf ein zusätzliches Argument auf der linken Seite des Cursors einfügen. „links“ steht dabei für die linke Pfeiltaste. Nachdem Sie mit dem Cursor im Makrorumpf `A-links` oder `M-i links` eingetippt haben, geben Sie dem Argument einen Namen, z.B. „name”, um anschließend darauf zugreifen zu können. Sie sollten nun Folgendes sehen:

⟨assign|*hello*|⟨macro|*name*|⟩⟩

In die zweite Argumentposition des Makrorumpfes tippen Sie nun Ihren Text z.B. „Hallo“, dann um das mit dem Namen „name“ bezeichnete erste Argument einzusetzen, drücken Sie die Kombinationen `A-#` oder `M-i #` tippen dann schließlich „name”, drücken `rechts`, das ist die rechte Pfeiltaste und geben weiter Text ein z.B. „, wie geht es Ihnen?”. Das sieht dann so aus:

⟨assign|*hello*|⟨macro|*name*|Hallo *name*, wie geht es Ihnen?⟩⟩

Die Kurzbefehlkombination `A-#` bzw. `M-i #` wird zum Zugriff auf das Makroargument, hier *name*, verwendet. Anstatt `A-#` bzw. `M-i #` zu benutzen, dann „name” and `rechts` einzutippen, können Sie auch die `\`-Taste benutzen und `\ n a m e` gefolgt von der Eingabetaste `return` eintippen. Nachdem Sie Ihr Stil-Paket gesichert haben, können Sie Ihr neues Makro in jedem Dokument, dem Sie dieses Paket zugefügt haben, benutzen, indem Sie `\ h e l l o` eingeben und die `return`-Taste betätigen.

Intern werden alle Makrodefinitionen in der „TEXMACS typesetter“-Umgebung gespeichert. Daneben werden dort auch normale Kontextvariablen wie Absatzzähler (section counters) oder Schriftgröße (font size) abgelegt. Die Kontextvariablen können global mit dem assign-Konstrukt oder lokal mit dem with-Konstrukt gesetzt werden. Wenn z.B. die folgenden Zeile

⟨assign|*section-nr*|-1⟩

in Ihrem Paket enthalten ist und Sie als Basis-Stil `Artikel` verwenden, dann erhält der erste Abschnitt die Abschnittnummer 0.

Die folgende Variante

⟨assign|*hello*|⟨macro|*name*|Hello ⟨with|*font-shape*|small-caps|*name*⟩!⟩⟩

hello-Makros bringt den Namen in in KLEINEN GROSSBUCHSTABEN auf den Bildschirm. Beachten Sie, daß Sie mit dem with-Konstrukt auch ein Makro lokal umdefinieren können. Dies wird beispielsweise in den Standardumgebungen für Listen benutzt, wo das Makro, das die Listensymbole liefert innerhalb des Listenrumpfes modifiziert wird.

Eine weitere Variante des hello-Makros benutzt das person-Makro des Standard-Stils:

⟨assign|*hello*|⟨macro|*name*|Hello ⟨person|*name*⟩!⟩⟩

Um in die Makrodefinition ⟨person|*name*⟩ einzufügen, müssen Sie zuerst an seiner Stelle ein Leerkonstrukt (compound) erzeugen. Dazu benutzen Sie `A-c` oder `M-i c`, tippen dann „person”, fügen ein Argument mit `A-rechts` oder `M-i rechts` hinzu, und tippen schließlich den Namen des Arguments *name*. Schhließlich drücken Sie `return`, um das unbenanntes Makro in ein person-Makro umzuwandeln. Alternativ können Sie `\`, “person”, `A-rechts` und „name” tippen.

Durch Kombination der vorgehend beschriebenen Vorgehensweisen sollte der Durchschnittsanwender bereits Stil-Pakete für alle häufig vorkommenden Anwendungsfälle zu schreiben können.

Ein interessane Technik, mit der sich Makros schreiben lassen, die komplizierte mathematische Formeln enthalten, die wiederum von variablen Formeln abhängen, ist die Folgende:

1. Schreibe die Formel z.B. $(a_1, \ldots, a_n)$ in ein gewöhnliches Dokument.

2. Erzeuge ein Makroskelett in ihrem Stil-Paket:

```
⟨assign|n-tuple|⟨macro|a|⟩⟩
```

3. Kopiere die Formel und füge sie in den Rumpf des Makros ein:

```
⟨assign|n-tuple|⟨macro|a|(a⟨rsub|1⟩,<ldots>,a⟨rsub|n⟩)⟩⟩
```

4. Ersetze die die Variablen, die parametrisiert werden sollen, durch Makro-Argumente:

```
⟨assign|n-tuple|⟨macro|a|(a⟨rsub|1⟩,<ldots>,a⟨rsub|n⟩)⟩⟩
```

5. Nach dem Speichern können Sie das neue Makro in Dokumenten einsetzen, die Ihr Paket verwenden, z.B.:

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

## 10.2. Die Darstellung von Basis-Stil-Dateien und Paketen

### 10.2.1. ASCII-basierte oder Baum-basierte Editierung: problematische Alternativen

Die meisten Nutzer sind daran gewöhnt, Quellcode mit einem konventionellen Editor wie Emacs zu editieren, die den Quellcode im ASCII-Format darstellen. TeXmacs-Dokumente werden aber als Bäume (tree) gespeichert. Es ist daher eine interessante aber sehr komplizierte Frage, welches Format für die Editierung am besten geeignet ist. Eine Option wäre, ein ASCII-basiertes Format wie XML oder Scheme zu verwenden oder auch das ASCII-basierte Format, in dem Dateien auf einer Diskette oder Festplatte gespeichert werden. Die andere Option besteht darin die Bäume als solche zu verwenden und damit Textdokumente und Quellcode gleich zu behandeln.

Wir haben uns entschlossen in TeXmacs die zweite Alternative zu verwenden. Genauer gesagt, jedes Dokument kann im „Quellmodus" editiert werden. Das ist nur eine spezielle Form der Darstellung, die die Baumstruktur besser erkennbar macht. Es ist sehr instruktiv, irgendein Dokument zu nehmen und es im Quellmodus zu betrachten. Dazu wählen Sie Dokument → Ansicht → Quellmodus. Auf diese Weise können Sie vorallem bei Verweisen jeder Art das Ziel erkennen.

Die Wahl zwischen ASCII-basiertem Editieren und Baum-basierten Editieren ist nicht trivial, denn TeXmacs Stil-Dateien und Pakete haben eine Doppelnatur. Sie sind einerseits Programme, die spezifizieren, wie Makros dargestellt werden, andererseits enhalten sie normalen Text. Es gibt eine Reihe von Gründen, warum Nutzer ein ASCII-basiertes Format vorziehen:

1. Der Code kann leicht so formatiert werden, daß man ihn besser lesen kann.

2. Kommentare können leicht hinzugefügt werden.

3. Standard Editoren wie Emacs stellen Werkzeuge für automatische Hervorhebungen, Einzüge usw. bereit.

4. Man wird nicht durch irgendwelche „Strukturen" in der Entwicklungsphase behindert.

Wir versuchen möglichst viele dieser Vorteile in unser Konzept einer a strukturierten Dokument- Darstellung einfließen zu lassen, obwohl es offensichtlich schwierig ist, Punkt 4 angemessen zu berücksichtigen. Wir glauben, daß die in ersten drei Punkten genannten Vorteile in einer solchen strukturierten Umgebungen sogar deutlicher ausgeprägt sein werden. Jedoch erfordert die Verwirklichung dieses Konzeptes ein tiefes Verständnis davon, wie Nutzer Quellcode tatsächlich formatieren und editieren.

Lassen Sie uns beispielsweise dieses Stück formatierten Code betrachten:

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

Man erkennt, daß dieser Code mit in spezieller Weise formatiert ist. Die Formatierungsrichtlinien, die derjenige, der den Code formatierte, im Kopf hatte sind, im Code nicht enthalten. Wenn etwa die Variable cond in c umbenannt wird, oder wenn die Variable holala in hola umbenannt wird, muß der Code manuell neu formatiert werden.

Zum jetzigen Zeitpunkt stellt TeXmacs keine Werkzeuge bereit, die das Problem dieses Beispiels automatisch lösen könnten, aber einige wichtige Werkzeuge sind schon vorhanden. So hat der Anwender viele Möglichkeiten, Quellcode durch Einzüge zu gestalten und vernünftige Standardformatierungen sind vorhanden. Weitere Werkzeuge sollen in Zukunft entwickelt werden. Wir sind für Anregungen dankbar.

### 10.2.2. Anpassung der globalen Darstellung

In der Codeformatierung-Gruppe des Dokument → Ansicht-Menüs finden Sie verschiedene Möglichkeiten die Darstellung von Quellcode-Bäumen in Ihrem Dokument an Ihre Bedürfnisse azupassen. Wir empfehlen, daß Sie mit den verschiedenen Möglichkeiten an einem eigenen Dokument zu experimentieren, um die Vor- und Nachteile kennen zu lernen, nachdem Sie den Quellmodus mit Dokument → Ansicht → Quellcode eingestellt haben.

Zuerst einmal können Sie zwischen verschiedenen Basis-Stilen wählen: „Angular", „Scheme", „Functional" und „LaTeX". Die verschiedenen Darstellungsweisen werden in Graphiken unten beispielhaft gezeigt:

Angular

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body⟩⟩⟩
```

Scheme

```
(assign "quick-theorem"
    (macro "body"
        (surround (no-indent)"Theorem. " ""
            (arg "body"))))
```

Functional

```
assign (quick-theorem,
    macro (body,
        surround (no-indentTheorem. , ,
            body)))
```

LaTeX

```
assign{quick-theorem}{
    macro{body}{
        surround{no-indentTheorem. }{}{
            body}}}
```

**Abbildung 10.1.** Verschiedene Basis-Stile zur Darstellung von Quellcode-Bäumen.

Möglicherweise wollen Sie, daß bestimmte Quellcode-Konstrukte wie z.B. concat und document in besonderer Weise dargestellt werden. Im Menü Dokument → Ansicht → Speziell können Sie festlegen, in welchem Ausmaß Sie solche speziell zulassen wollen:

**Keine.** Kein Quellcode-Konstrukt erhält eine Sonderbehandlung.

**Formatierung.** Nur die Formatierungskonstrukte concat and document werden ausgeführt.

**Normal.** Zusätzlich zu den oben genannten Formatierungskonstrukten werden werden einige andere Quellcode-Konstrukte wie compound, value und arg ausgeführt.

**Maximal.** Diese Option ist noch nicht implementiert. Sie soll den Anwender in die Lage versetzen, spezielle Darstellungen von Konstrukten wie plus zu programmieren.

Die verschiedenen Optionen sind unten dargestellt:

Keine

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨document|
            ⟨surround | ⟨concat | ⟨no-indent⟩ |
            Theorem. ⟩||
                ⟨arg|body⟩⟩⟩⟩⟩
```

Formatierung

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            ⟨arg|body⟩⟩⟩⟩
```

Normal

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body⟩⟩⟩
```

Maximal

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body⟩⟩⟩
```

**Abbildung 10.2.** Verschiedene Optionen zur Darstellung von Quellcodekonstrukten.

Darüber hinaus kann der Anwender noch kontrollieren, wie verdichtet die Darstellung von Quellcode-Konstrukten sein soll, wie stark also Konstrukte durch Zeilenumbrüche gegliedert werden sollen. Das Ausmaß kann im Menü Dokument → Ansicht → Verdichtungsgrad eingestellt werden:

**Minimal.** Alle Konstrukte werden durch Zeilenumbrüche gegliedert.

**Nur Zeilenbefehle.** Alle Konstrukte außer Zeilenbefehle werden durch Zeilenumbrüche gegliedert.

**Normal.** Alle Zeilen-Argumente am Anfang des Konstrukts werden verdichtet dargestellt. Wenn ein Block-Argument angetroffen wird, wird der Rest der Argumente durch Zeilenumbrüche gegliedert.

**Zeilenargumente.** Alle Zeilen-Argumente werden verdichtet dargestellt. Nur Block-Konstrukte werden durch Zeilenumbrüche gegliedert.

**Maximal.** Der ganze Quellcode wird verdichtet dargestellt.

Die Optionen Normal und Zeilenargumente unterscheiden sich nur unwesentlich. Beispiele für den Effekt der verschiedenen Optionen sind unten zu sehen:

Minimal

```
⟨assign|
    quick-theorem|
    ⟨macro|
        body|
        ⟨surround|
            ⟨concat|
                ⟨no-indent⟩|
                Theorem. ⟩|
            |
            body⟩⟩⟩
```

Nur Zeilenbefehle

```
⟨assign|
    quick-theorem|
    ⟨macro|
        body|
        ⟨surround|
            ⟨no-indent⟩Theorem. |
            |
            body⟩⟩⟩
```

Normal

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body⟩⟩⟩
```

Maximal

```
⟨assign|quick-theorem|⟨macro|body|⟨document|
⟨surround|⟨no-indent⟩Theorem. ||body⟩⟩⟩⟩
```

**Abbildung 10.3.** Die verschiedenen Optionen für den Verdichtungsgrad.

Schließlich kann der Anwender im Menü Dokument → Ansicht → Befehlsabschluss die Darstellung der Stoptags von Quellcode-Befehlen einstellen, wenn sie mehrzeilig gegliedert dargestellt werden. Die folgenden Optionen sind verfügbar: minimal, Dicht, Gespreizt und Rekursiv. Diese Optionen werden unten in Beispielen gezeigt:

Minimal

```
assign quick-theorem
    macro body
        surround ⟨no-indent⟩Theorem. |
            body
```

Dicht

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body⟩⟩⟩
```

Gespreizt

```
⟨assign|quick-theorem|
    ⟨macro|body|
        ⟨surround|⟨no-indent⟩Theorem. ||
            body
        ⟩
    ⟩
⟩
```

Rekursiv

```
⟨\assign|quick-theorem⟩
    ⟨\macro|body⟩
        ⟨\surround|⟨no-indent⟩Theorem. |⟩
            body
        ⟨/surround⟩
    ⟨/macro⟩
⟨/assign⟩
```

**Abbildung 10.4.** Die verschiedenen Optionen zur Darstellung von Stoptags.

### 10.2.3. Lokale Anpassung

Auch wenn TeXmacs versucht, den Quellcode mit den globalen Darstellungsoptionen übersichtlich zu präsentieren, muß die Lesbarkeit des Codes lokal verbessert werden. Im Quellmodus kann man das mit den Optionen in den Menüs Quellcode → Aktivierung und Quellcode → Darstellung erreichen. Alle lokalen Anweisungen, die die Darstellung des Quellcodes betreffen, werden automatisch entfernt, wenn die Datei als Basis-Stil oder als Stil-Paket verwendet wird.

Insbesondere wenn es um ganz bestimmte Inhalte wie mathematische Symbole oder um eingebettete Bilder geht, wird man lieber das Ergebnis des Konstrukts, also die „aktivierte“ Form, als das inaktive Quellcode-Konstrukt sehen wollen. Beispielsweise wollen Sie möglicherweise lieber in Ihrem Quellcode

```
⟨assign|R|⟨macro|ℝ⟩⟩
```

als den inaktiven Code

```
⟨with|mode|math|⟨assign|R|⟨macro|<bbb-R>⟩⟩⟩
```

sehen.

### Beispiel

Besonders in komplizierteren Makros wie

```
⟨with | mode | math | ⟨assign | diag | ⟨macro | var | dim |
<left-(-1> var_1 <b-0> <ddots> <b-0> var_dim <right-)-1>⟩⟩⟩
```

kann eine teilweise Aktivierung des Codes beispielsweise zu folgender Darstellung

$$\langle\mathsf{assign}|\mathit{diag}|\langle\mathsf{macro}|\mathit{var}|\mathit{dim}|\begin{pmatrix} \mathit{var}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathit{var}_{\mathit{dim}} \end{pmatrix}\rangle\rangle$$

die Lesbarkeit des Codes sehr verbessern.

Teile des Codes können aktiviert werden, indem man es mit der Maus auswählt und dann das Menü Quellcode → Aktivierung → Aktivieren oder die Tastenkombination `M-+` verwendet. Entsprechend kann Code nach Markierung mit `M--` deaktiviert werden. Im obigen Beispiel haben wir das für die Darstellung der Argumente benutzt und zwar zur Darstellung der Variablen *var* and *dim* in beiden gezeigten Codefragmenten. Aktivierung und Deaktivierung können sich auf den ganzen (markierten) Baum Quellcode → Aktivierung → Aktivieren beziehen oder auch nur auf die Wurzel Quellcode → Aktivierung → Aktiviere die Wurzel.

Ein anderer Weg , um die Darstellung an spezielle Bedürfnisse anzupassen, liegt darin, globale Darstellungsoptionen lokal zu ersetzen. Das ist vor allem dort von Interesse, wo es darum geht, die Gliederung durch Zeilenumbrüche zu verändern. Beispielsweise kann das concat-Konstrukt dazu benutzt werden, Textinhalte aneinanderzufügen oder auch dazu einen Block bestehend aus einer Folge von Anweisungen/Befehlen zu erzeugen - oder auch einer Kombination aus beiden. So haben wir in folgendem Beispiel

```
⟨assign|my-section|
   ⟨macro|title|
      ⟨concat|
         ⟨header-hook|
            title⟩|
         ⟨toc-hook|
            title⟩|
         ⟨my-section-title|
            title⟩⟩⟩⟩
```

das Konstrukt concat mit seinen Argumenten derartig dargestellt, daß er sich über mehrere Zeilen erstreckt. dazu haben wir den Befehl Quellcode → Darstellung → Gespreizt benutzt. Das setzt übrigens voraus, daß das Konstrukt concat explizit auftaucht, da sonst eine Verwechslung mit dem Konstrukt document möglich wäre. Wenn dagegen ein Teil dieses Konstrukts wie üblich dargestellt werden sollen, kann man Quellcode → Darstellung → Dicht benutzen:

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

Zur Zeit ist es noch nicht vorgesehen, Argumente als *inline* or *block* zu markieren. Möglicherweise tun wir das aber noch.

Schließlich kann die Darstellung von Code aus dem Menü Quellcode → Darstellung → Apply macro oder Quellcode → Darstellung → Apply macro once mit einem beliebigen Makro angepasst werden. Solche Makros werden automatisch entfernt, wenn das Dokument als Basis-Stil oder als Stil-Paket genutzt wird.



## 10.3. Die Sprache der TeXmacs-Stil-Definitionen

Im Abschnitt So schreiben Sie ihre erste TeXmacs-Stil-Definition haben Sie bereits einen ersten Eindruck von der Sprache erhalten, in der TeXmacs-Stil-Definition geschrieben werden. In diesem Abschnitt werden wir Ihnen einen vollständigeren Überblick über die vorhandenen Sprachelemente geben. Noch detailliertere Informationen finden Sie in dem Kapitel über TeXmacs-Konstrukte.

Die meisten Stil-Definitionen betreffenden Konstrukte finden Sie im Quellcode-Menü, wenn Sie sich im Quellmodus befinden. Sie können sie auch im Textmodus erreichen über die Menüs Einfügen → Makro und Einfügen → Ausführbar. Alternativ können Sie im Quellmodus auch die Tastenkombination mit `A-` und `M-e` benutzen bzw. in den anderen Moden mit `M-i` und `M-e`. Schließlich erinnern wir daran, dass die `\`-Taste benutzt werden kann, um kontextabhängig Makros oder Argumente zu erzeugen und dass `A-rechts` und `A-links` Argumente einfügen.

### 10.3.1. Zuweisungen

Alle vom Benutzer definierten TeXmacs-Makros und Stil-Variablen werden in der „aktuellen Schriftsatzumgebung" gespeichert. Da alle Dokumente in der Form von Bäumen gespeichert werden, assoziiert diese Umgebung mit jeder Zeichenketten-Variablen einen zugehörigen „Baum-Wert". Variablen, deren Werte Makros sind, entsprechen neuen Konstrukten. Alle anderen sind normale TeXmacs-Umgebungsvariablen. Die Konstrukte, die auf der aktuellen Schriftsatzumgebung arbeiten, können über das Menü Quellcode → Definition erreicht werden.

Sie können den Wert einer Umgebungsvariablen global ändern, indem Sie das Konstrukt assign benutzen, wie z.B. in:

```
⟨assign|hi|⟨macro|Hallo da!⟩⟩
```

Sie können auch nur lokal Werte ändern mit dem with Konstrukt:

```
⟨with|font-series|bold|color|red|Fetter roter Text⟩
```

Der Wert der Umgebungsvariablen kann mit dem value-Konstrukt geholt werden, was z.B. in einem Zähler verwendet werden könnte:

```
⟨assign|mein-zaehler|⟨plus|mein-zaehler|1⟩⟩
```

Schließlich können Sie Umgebungsvariablen Logik-Eigenschaften verleihen, indem Sie das Konstrukt drd-props benutzen. Das ist in Makro-Konstrukte genauer erklärt.

### 10.3.2. Makro Expansion

Das Interessanteste an der TeXmacs-Stil-Definitions-Sprache ist die Möglichkeit, Makros selbst zu definieren. Dabei gibt es drei verschiedene Makro-Typen: gewöhnliche Makros, Makros mir einer unbestimmten Anzahl von Argumenten und externe Makros, die durch SCHEME oder ein Plug-In ausgewertet werden. Die Makro-bezogenen Konstrukte sind über das Quellcode → Makro-Menü erreichbar. Anschließend werden wir nur die gewöhnlichen Makros beschreiben. Weitere Details finden Sie hier.

Gewöhnliche Makros werden üblicherweise mit einer Zuweisung (*assign*) definiert:

```
⟨assign|mein-makro|⟨macro|x1|···|xn|Rumpf⟩⟩
```

Nach solch einer Zuweisung wird mein-makro ein neues Konstrukt mit $n$ Argumenten, das mit:

```
⟨mein-makro|y1|...|yn⟩
```

aufgerufen werden kann.

Innerhalb des Rumpfes kann das arg-Konstrukt benutzt werden, um auf die Werte von Argumenten zuzugreifen, z.B.:

```
⟨assign|hello|⟨macro|name|Hello name, Sie sehen heute aber gut aus!⟩⟩
```

Man kann ein Makro mit mehr oder weniger Argumenten aufrufen als eigentlich vorgeshen sind. Überflüssige Argumente werden einfach ignoriert. Fehlende Argumente werden durch eine Nullgröße, das uninit-Konstrukt ersetzt:

```
⟨assign|hallo-hallo|
    ⟨macro|erster|zweiter|
        ⟨if|
            ⟨equal|zweiter|?⟩|
            Hallo  erster, Sie sehen heute einsam aus...|
            Hallo erster und zweiter, Sie sind ein hübsches Paar!⟩⟩⟩
```

Angemerkt sei, dass Sie mit Makros ähnlich wie beim funktionalen Programmieren rechnen können, nur sind TeXmacs-Makros keine *Closures*, noch nicht. Beispielsweise:

```
<assign|meine-makro-kopie|mein-makro>
```

Das compound-Konstrukt kann zur Anwendung von Makros dienen, die das Resultat einer Berechnung sind:

```
<assign|overloaded-hi|
    <macro|name|
        <compound|
            <if|<gutes-wetter>|fröhliches-hi|trauriges-hi>|
            name>>>
```

### 10.3.3. Formatier-Konstrukte

Dieser Abschnitt enthält einige wichtige Anmerkungen über Formatier-Konstrukte, die nicht wirklich Teil der Stil-Definitions-Sprache sind, aber damit in einem engen Zusammenhang stehen.

Erstens gehören fast alle TeXmacs-Makro-Befehle, die die Darstellung von Text betreffen, in eine von zwei Klassen. Es handelt sich entweder um Fließtext-Konstrukte oder um Block-Konstrukte, deren Argument also *Fließtext* oder ein *Block* ist. Der Unterschied wird in Basis-Konstrukte genauer erklärt. Ein *Block* ist dabei ein Textelement, dass aus mehreren Absätzen bestehenden Text, *Multi-Absatz-Text*, enthält. Dagegen kann Fließtext zwar aus mehreren Textelementen bestehen, enhält selbst aber keine weiteren Absätze. Beispielsweise ist frac ein typisches Fließtext-Konstrukt, wohingegen Satz ein typisches Blockkonstrukt ist. Einige Konstrukte richten sich nach ihren Argumenten, sie sind Fließtext-Konstrukte, wenn ihr Argument Fließtext oder Fließtext-Konstrukte sind bzw. Blockkonstrukte, wenn ihr Argument ein Block oder ein Blockkonstrukt ist. Ein Beipiel dafür ist stark. Wenn man Makros schreibt, ist es sehr wichtig, dass man sich darüber im Klaren ist, ob ein Konstrukt ein Fließtext– oder ein Blockkonstrukt ist, denn Blockkonstrukte innerhalb einer horizontalen Verkettung werden nicht richtig dargestellt. Wenn man ein Blockkonstrukt mit Fließtext umgeben muß, dann muß man das surround-Konstrukt einsetzen:

```
<assign|mein-theorem|
    <macro|Rumpf|
        <surround|<no-indent><with|font-series|bold|Satz>|<right-flush>|
            Rumpf>>>
```

In diesem Beispiel wurde der Rumpf mit dem **fett** dargestellten Text „Satz” links und einem „rigth-flush” rechts umgeben (surround). Dieser Flushing „rigth-flush” dient zur besseren Darstellung der blauen Hilfslinien, wenn der Cursor sich innerhalb des Wirkungsbereichs eines Kontextes befindet. Eer dehnt den Wirkungsbereich (für die Hilfslinien) bis an den rechten Rand aus.

In den meisten Fällen findet TeXmacs selbständig heraus, welche Befehle Zeilen-Befehle und welche Block-Befehle sind. Manchmal möchte man aber einen Befehl zwangsweise zum Block-Befehl erklären. Beispielsweise kann ein Konstrukt sehr-wichtig, das folgendermaßen definiert ist:

```
<assign|sehr-wichtig|<macro|body|<with|font-series|bold|color|red|body>>>
```

sowohl als Block-Befehl als auch als Zeile-Befehl dienen. Wenn Sie nun den Cursor genau vor den with-Tag plazieren und `Wagenrücklauf` mit nachfolgender `Rücktaste` eingeben, dann erhalten Sie:

```
⟨assign|sehr-wichtig|
    ⟨macro|body|
        ⟨with|font-series|bold|color|red|body⟩⟩⟩
```

Weil der Rumpf, *body* nun ein Block ist, wird das Makro sehr-wichtig automatisch zu einem Block-Befehl. In Zukunft wird sich mit dem drd-props-Konstrukt noch besser steuern lassen, welche Befehle Zeilen-Befehle und welche Block-Befehle sind.

Eine weitere wichtige Eigenschaft von Tags ist es, ob sie normalen Text als Inhalt haben, oder ob der Inhalt tabellarisch ist. Nehmen wir einmal die folgende Definition eines Standard-Formel-Blocks, eqnarray*, allerdings ohne einen Teil der Darstellungs-Tags:

```
⟨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⟩⟩⟩⟩⟩
```

Die Verwendung von surround zeigt, dass eqnarray* ein Block-Befehl ist und die Verwendung von tformat spezifiziert ein Tabellen-Konstrukt. Außerdem wurden twith und cwith benutzt, um zusätzliche Formatierinformationen festzulegen. Weil es sich um einen Block-Kontext handelt, ermöglichen wir Zeilenumbruch und lassen für die Tabelle die gesamte Absatzbreite zu (freier Leerraum wird auf die erste und letzte Spalte gleich verteilt). Außerdem haben wir festgelegt, dass es genau drei Spalten gibt.

Schließlich sollte man daran denken, dass Stildefinitionen nicht nur die Darstellung des fertigen Dokuments festlegen, sonder dass sie auch die Darstellung während der Erstellung regeln. Oben haben wir bereits den right-flush-Befehl erwähnt, der die Darstellung von sichtbaren Hinweisen von sichtbaren Inhalten verbessert. Ganz ähnlich kann auf unsichtbarere Inhalte durch Flags hingewiesen werden:

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

ganz allgemein kann mit dem specific-Befehl, dem als erstes Argument „screen” übergeben wird, ein sichtbarer Hinweis eingefügt werden, der im Druck nicht erscheint.

### 10.3.4. Steuerung der Evaluierung

Das im Quellcode-Modus erscheinende Menü Quellcode → Auswertung enhält eine Anzahl von Konstrukten, mit denen sich die Evaluierung von Ausdrücken der Stil-Definitions-Sprache steuern lässt. Die wichtigste Anwendung solcher Konstrukte ist der Gebrauch in „Meta-Makros”, die Makros als Argumente nehmen, um diese zu ändern wie z.B. das unten definierte new-theorem:

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

Mit dem Aufruf ⟨new-theorem | theorem | Theorem⟩, werden zuerst alle unquote-Befehle innerhalb des quasi-Konstrukts ausgeführt, was zuu folgendem Ausdruck führt:

```
⟨assign|theorem|
   ⟨macro|body|
      ⟨surround|⟨no-indent⟩⟨strong|Theorem. ⟩|⟨right-flush⟩|
         body⟩⟩⟩
```

Dann wird dieser Ausdruck evaluiert. Das definiert das Makro theorem.

Man beachte, dass TeXmacs-Evaluierungsregeln sich etwas von den Scheme-Regeln unterscheiden. Diese geringfügigen Unterschiede sollen es dem Anwender besonders leicht machen, Makros für den Schriftsatz zu schreiben.

Beipielsweise, wenn TeXmacs ein Makro ⟨macro|$x_1$|⋯|$x_n$|body⟩ mit den Argumenten $y_1$ bis $y_n$ aufruft, bleiben die Argumentvariablen $x_1$ bis $x_n$ an die unevaluierten Ausdrücke $y_1$ bis $y_n$ gebunden bis der Rumpf, body, mit diesen Bindungen evaluiert worden ist. Die Evaluierung von $y_i$ wird jedesmal durchgeführt, wenn das Argument $x_i$ aufgerufen wird. Wenn also ein Makro ⟨macro|$x$|$x$ und nochmal $x$⟩ auf den Ausdruck y angewandt wird, wird der Ausdruck y genau zweimal ausgewertet.

In Scheme werden die Rümpfe von Scheme-Makros zweimal evaluiert, während bei Funktionen die Argumente evaluiert werden. Dagegen wird, wenn auf eine Variable zugegriffen wird, der Wert nicht evaluiert, gleichgültig, ob es sich um ein Argument oder eine Kontext-Variable handelt. Daher entspricht das TeXmacs-Makro

```
⟨assign|foo|⟨macro|x|⟨blah|x|x⟩⟩⟩
```

dem folgenden Scheme-Makro:

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

Umgekehrt gehören zu dem folgenden Scheme-Makro bzw. -Funktion

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

die folgenden TeXmacs-Analoga:

```
⟨assign|foo|⟨macro|x|⟨eval|⟨blah|⟨quote-arg|x⟩|⟨quote-arg|x⟩⟩⟩⟩⟩
⟨assign|fun|⟨macro|x|⟨with|x*|x|⟨blah|⟨quote-value|x*⟩|⟨quote-value|x*⟩⟩⟩⟩⟩
```

Hier wurden die Konstrukte quote-arg und quote-value dazu benutzt Argumente bzw. Kontext-Variable aufzurufen. Die TeXmacs-Konstrukte eval, quote, quasiquote und unquote verhalten sich wie ihre Scheme-Analoga. Das quasi-Konstrukt ist eine Abkürzung für quasiquote gefolgt von Evaluierung.

### 10.3.5. Steuerung des logischen Ablaufs

Neben Folgen von Ausdrücken, die mit dem concat-Konstrukt erzeugt werden könne, und dem Mechanismus für die Makro-Expansion besitzt die TeXmacs-Stil-Definitions-Sprache noch einige andere Konstrukte, um den logischen Ablauf zu steuern: if, case, while und foreach. Diese Konstrukte können mit dem Menü Quellcode → Ablaufsteuerung erreicht werden. Wir müssen aber den Anwender warnen, diese Konstrukte sind nicht sehr sicher: sie können nur auf Zeileninhalte angewendet werden und ihr Ergebnis hängt von der Erreichbarkeit der Makroargumente ab. Die Erreichbarkeit kann aber nicht vorab geprüft werden!

Das wichtigste Konstrukt ist if, das mit dem Kurzbefehl `A-?` erzeugt werden kann. Damit kann man bedingten Schriftsatz erreichen:

```
⟨assign|appendix|
    ⟨macro|title|body|
        ⟨compound|
            ⟨if|⟨long-document⟩|chapter-appendix|section-appendix⟩|
            title|
            body⟩⟩⟩
```

In diesem Beispiel ist appendix ein Block, der aus einem Titel und einem Rumpf besteht. Dieser wird als Kapitel in langen Dokumenten und als Abschnitt in anderen Dokumenten gesetzt. Im folgenden ist eine unzulässige Inplementierung gezeigt. Sie funktioniert nicht, weil if zur Zeit nur mit Zeileninhalten arbeitet:

```
So gehts nicht:

⟨assign|appendix|
    ⟨macro|title|body|
        ⟨if|
            ⟨long-document⟩|
            ⟨chapter-appendix|title|body⟩|
            ⟨section-appendix|title|body⟩⟩⟩⟩
```

Das if-Konstrukt kann auch benutzt werden um optionale Argumente einzuführen:

```
⟨assign|hey|
   ⟨macro|first|second|
      ⟨if|
         ⟨equal|second|?⟩|
         Hey first, you look lonely today…|
         Hey first and second, you form a nice couple!⟩⟩⟩
```

TeXmacs kann aber nicht entscheiden, welche Argumente optional sind und welche Argumente erreichbar (vom Anwender editierbar) sind. Deshalb muß diese Information von Hand mit dem drd-props-Konstrukt festgelegt werden. Die Konstrukte case, while und for-each werden in Steuerung des logischen Ablaufs im Abschnitt Konstrukte für Stil-Definitionen eingehender behandelt.

### 10.3.6. Funktionelle Befehle

In den Menüs Quellcode → Arithmetisch, Quellcode → Text, Quellcode → Tupel und Quellcode → Bedingung, die im Quellcode-Modus erreichbar sind, finden Sie eine Anzahl verschiedener Konstrukte zum Operieren mit Ganzzahlen, Zeichenketten, Tupel und boolschen Werten bzw. Variablen. Im folgenden finden sie ein Beipiel, in dem mit einem Makro new-important ein neuer „important" -Befehl definiert werden kann gleichzeitig mit einer Variante in Rot:

```
⟨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⟩⟩⟩⟩⟩⟩⟩
```

Hier wurde das merge-Konstrukt benutzt, um zwei Zeichenketten zusammenzufügen. Die verschiedenen berechnenden Konstrukte finden sich im Abschnitt Funktionelle Operatoren von Konstrukte für Stildefinitionen genauer erläutert.

## 10.4. Standard-TeXmacs-Stildefinitionen anpassen

Immer, wenn die Standard-TeXmacs-Stil-Definitionen sich für einen bestimmten Zweck als schlecht geeignet erweisen, können Sie einen eigenen Stil definieren. Nun ist es aber nicht sehr leicht, eine Stildefinition ganz neu zu schreiben, deshalb ist es meist einfacher, existierende Stil-Definitionen anzupassen. Das erfordert ein gewisses Verständnis der globalen Architektur der Standard-Definitonen und ein genaueres Verständnis der Teile, die sie anpassen möchten. In diesem Abschnitt werden wir die allgemeinen Prinzipien erläutern. Genaueres finden Sie unter dem Abschnitt die Basis der TeXmacs-Stile.

### 10.4.1. Organization of style files and packages

Each standard TeXmacs 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:

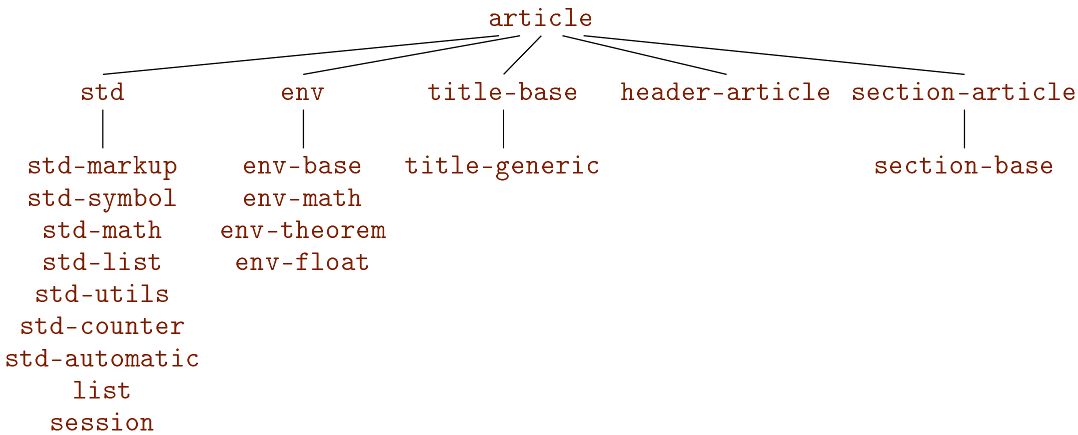



**Abbildung 10.5.** The tree with the packages from which the `article` style has been built up. In order to save space, we have regrouped the numerous children of `std` and `env` in vertical lists.

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.

Bemerkung 10.1. 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 `C-x C-f`, 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 `C-x C-f`, followed by the file name `std-markup.ts` and `return`.

BEMERKUNG 10.2. 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.

### 10.4.2. Anpassung, Allgemeines

Stil-Definitionen und -Pakete bereichern den Kontext mit einer Kombination von

- Kontextvariablen.
- Anwenderbefehlen.
- Anpassungsmakros.

Außerdem können sie einige interne Befehle definieren, die nicht in diesem Handbuch behandelt werden. Auch können sie einige logische Eigenschaften von Befehlen definieren, indem sie das drd-props-Konstrukt benutzen.

Kontext-Varaible sind fast immer Attribute, die die Darstellung irgendwelcher Inhalte steuern, oder die Zähler enthalten für Abschnitte, Gleichungen usw.. Obwohl viele einfache Anwender-Befehle wie z.B. stark in eignen Stil-Definitionen geändert werden können, wird diese Praxis nicht empfohlen für kompliziertere Befehle, wie z.B. Abschnitt. In der Tat beinhaltet der Abschnitt-Befehl eine Menge verschiedener Operationen wie das Setzen von Unterzählern, den Titel in das Inhaltsverzeichnis eintragen usw.. Darum gibt es spezielle zusätzliche Makros, zur leichteren Anpassung, in diesem Fall z.B.: section-title, section-clean und section-toc.

### 10.4.3. Das allgemeine Layout anpassen

Muß noch geschrieben werden.

### 10.4.4. Listenkontexte anpassen

Listen bestehen aus zwei verschiedenen Bestandteilen: der äußeren Listenstruktur z.B. Ränder und der inneren Struktur, den einzelnen Punkten. Die Listenkontexte können durch Um- oder Neudefinition der Darstellungsmakros oder durch Definition zusätzlicher Makros, die zur gleichen abstrakten Schnittstelle (D.T.D.) passen, geändert werden.

Die Darstellung der äußeren Listenstruktur wird durch das render-list-Makro gesteuert, das den Rumpf der Liste als Argument übernimmt. Betrachten Sie die folgenden Umdefinition von 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⟩⟩⟩⟩
```

Diese Umdefinition verändert die Darstellung aller Listen (Auflistung, Aufzählung, usw.), indem der rechte Rand um `3fn` reduziert wird:

- Dieser Text, der zu lang ist, um auf eine einzelne Zeile zu passen, wird auf der rechten Seite um `3fn` eingezogen.

    1. Dieser Text wird zusätzlich um `3fn` eingezogen, weil er sich in einer Unterliste befindet.

- Und wiederum: Dieser Text, der zu lang ist, um auf eine einzelne Zeile zu passen, wird auf der rechten Seite um `3fn` eingezogen.

In ähnlicher Weise kann man die innere Listenstruktur, die einzelnen Punkte, konfigurieren, indem man die Makros aligned-item und compact-item benutzt. Beide Makros haben ein Argument, das die Kennzeichnung (kennzeichnenden Text) des Listenpunktes übergibt. aligned-item für „ausgerichtete" Punkte, setzt an einer festen Stelle in Bezug auf den Seitenrand die Trennmarke, stellt die Kennzeichnung rechtsbündig von der Trennmarke dar. Dadurch ist der Platz für die Kennzeichnung beschränkt. Dann positioniert es den Text rechts von der Trennmarke. Dagegen positioniert compact-item die Kennzeichnung linksbündig an den Seitenrand, dann die Trennmarke und fügt den Textinhalt daran rechts anschließend ein. So sind lange Kennzeichnungen möglich. Die folgende Umdefinition von aligned-item

```
⟨assign|aligned-item|
    ⟨macro|x|
        ⟨concat|
            ⟨vspace*|0.5fn⟩|
            ⟨with|par-first|-3fn|⟨yes-indent⟩⟩|
            ⟨resize|⟨with|color|red|x⟩|r-2.5fn||r+0.5fn|⟩⟩⟩⟩
```

stellt die Kennzeichnung aller Listen-Kontexte mit Punkten des „kompakten“ Typs rot dar:

- Diese Liste und alle Listen mit „ausgerichteten“ Punkten haben rote Kennzeichnung.

    **C1.** Erste Bedingung.

    **C2.** Zweite Bedingung.

- Die folgenden Punkte mit „kompakten“ Punkten benutzen compact-item und bleiben unverändert.

    **Pferde und Hunde.** Liebe Tiere.

    **Mücken und Fliegen.** Nicht so nett.

Bemerkung 10.3. Die Makros aligned-item und compact-item müssen Zeileninhalt produzieren, damit man sie benutzen kann, um damit Blockinhalte zu umgeben. Eine Reihe von anderen internen Makros (aligned-space-item, long-compact-strong-dot-item, usw.) basieren auf aligned-item und compact-item, und werden für eine große Zahl verschiedener Arten von Listen benutzt (itemize-arrow, description-long, usw.). Für die Zukunft planen wir, item und item* mit einem notwendigen Rumpfargument, *body*, zu versehen. Man sollte das berücksichtigen, wenn man Listen-Kontexte entwirft, um den Code aufwärts-kompatibel zu halten.

Die `std-list` D.T.D. stellt ein Makro new-list bereit, mit dem neue Listen definiert werden können. Seine Syntax ist ⟨new-list|*name*|*item-render*|*item-transform*⟩. *name* ist der Name des neuen Listen-Konstrukts, *item-render* ein (Zeilen)-Makro zur Darstellung und *item-transform* eine zusätzliche Transformation, die auf den zu diesem Punkt gehörigen Text angewendet wird. So kann man z.B. einen Kontext enumerate-roman, wie folgt, definieren:

```
⟨new-list|enumerate-roman|aligned-dot-item|⟨macro|x|⟨number|x|roman⟩⟩⟩
```

### 10.4.5. Nummerierte Text-Kontexte

TEXMACS verfügt über drei verschiedene nummerierte Standard-Kontexte für Text: Satz-ähnliche, Bemerkung-ähnliche und Aufgabe-ähnliche. Die folgenden Aspekte können leicht angepasst werden:

- Neue Konstrukte erzeugen.
- Die Darstellung ändern.
- Die Nummerierung ändern.

**Neue Konstrukte erzeugen.**

Mit den Meta-Makros new-theorem, new-remark und new-exercise können neue nummerierte Kontexte erzeugt werden. Sie alle haben zwei Argumente, den Namen des neuen Konstrukts und die Bezeichnung, die in der Darstellung auf dem Bildschirm verwendet werden soll. Wenn Sie beispielsweise einen Kontext für Experimente erzeugen wollen, der mit Experiment 1 usw. fortlaufend beschriftet wird, dann definieren Sie experiments mit

```
⟨new-theorem|experiments|Experiment⟩
```

Wenn Experiment in dem geeigneten TEXMACS-Wörterbuch enthalten ist, wird der Text „Experiment" automatisch übersetzt. Im Abschnitt Definition neuer Kontexte wird u.a. beschrieben, wie man neue nummerierte Kontexte schreiben kann, die nicht zu den Satz-ähnlichen, Bemerkung-ähnlichen und Aufgabe-ähnlichen gehören.

**Die Darstellung ändern.**

Die Darstellung dieser Kontexte kann mit den Konstrukten render-theorem, render-remark und render-exercise beeinflusst werden. Diese Makros haben als Argumente den Namen des Kontexts (z.B. „theorem 1.2") und seinen Rumpf. Eine Bemerkung wird gemäß Vorgabe genau wie ein Satz gesetzt allerdings in der Schriftform „senkrecht". Daher beeinflussen Änderungen der Definition von render-theorem auch die Darstellung einer Bemerkung. Wenn man beipielsweise von einem Satz verlangt, dass er etwas eingerückt und geneigt dargestellt wird, dann kann man render-theorem so umdefinieren:

```
⟨assign|render-theorem|
    ⟨macro|which|body|
        ⟨surround|
            ⟨vspace*|1fn⟩⟨no-indent⟩⟨theorem-name|which⟨theorem-sep⟩⟩|
            ⟨right-flush⟩⟨vspace|1fn⟩|
            ⟨with|font-shape|slanted|par-left|⟨plus|par-left|1.5fn⟩|body⟩⟩⟩⟩
```

Das führt zu folgender Darstellung:

Satz 10.4. *Das ist ein Satz, der geneigt und eingerückt dargestellt wird.*

Bemerkung 10.5. Die Darstellung einer Bemerkung basiert auf der Darstellung des Satz-Kontexts nur, dass die Schriftform „senkrecht" verwendet wird.

Manchmal möchte man aber nur die Darstellung der Bezeichnung oder des Trennzeichens zwischen Bezeichnung und Text-Rumpf ändern. Wie man im vorstehenden Beispiel erkennen kann, werden diese Aspekte durch die Makros theorem-name und theorem-sep gesteuert. Beispielsweise wird mit

```
⟨assign|theorem-name|⟨macro|name|⟨with|color|dark red|font-series|bold|
name⟩⟩⟩
⟨assign|theorem-sep|⟨macro|: ⟩⟩
```

eine Proposition, wie folgt, dargestellt:

**Proposition 10.6:** *Diese Proposition hat eine ungewöhnliche Art der Darstellung.*

**Die Nummerierung ändern.**

In den Abschnitten über Zähler und Zählergruppen wird erklärt, wie man Zähler zu einen bestimmten Zweck ändern kann. Beispielsweise können Sie für den Kontext „Folgerung" den Zähler zurücksetzen, indem Sie inc-theorem neu definieren:

```
⟨quasi|
    ⟨assign|
        inc-theorem|
        ⟨macro|⟨compound|⟨unquote|inc-theorem⟩⟩⟨reset-corollary⟩⟩⟩⟩
```

Beachten Sie den Trick mit quasi und unquote, um alle Aktionen zu berücksichtigen, die von den früheren Werten des Makros inc-theorem stammen können.

Der folgende Code von `number-long-article.ts` dient dazu, allen Standard-Kontexten die Nummer des aktuellen Abschnitts als Praefix voranzustellen.

```
⟨assign|section-clean|⟨macro|⟨reset-subsection⟩⟨reset-std-env⟩⟩⟩
⟨assign|display-std-env|⟨macro|nr|⟨section-prefix⟩nr⟩⟩
```

Beachten Sie auch, dass mit den Paketen `number-europe.ts`, `number-long-article.ts`, `number-us.ts`, structured-list.ts (Zahl-europäisch, Zahl-lang-Artikel, Zahl-US Stil, structured-list und structured-Abschnitt) die Nummerierung im Menü Ansicht → Paket zufügen → Nummerierung angepasst werden kann.

## 10.4.6. Anpassung von Abschnitt-Formatierungen

Muß noch geschrieben werden.

# Kapitel 11

## Customizing TeXmacs

One major feature of TeXmacs is that it can be highly customized. First of all, the most important aspects of the program can be configured in Bearbeiten → Einstellungen. Most other parts of TeXmacs can be entirely adapted or reprogrammed using the Guile/Scheme extension language. In the sequel, we give a short overview of how this works in simple cases.

### 11.1. Introduction to the Guile extension language

Like Emacs, TeXmacs comes with a Lisp-like extension language, namely the Guile Scheme dialect from the Gnome project. For documentation about Guile Scheme, we refer to

```
http://www.gnu.org/software/guile/guile.html
```

Scheme has the advantage that it may be extended with extern C and C++ types and routines. In our case, we have extended Scheme with routines which you can use to create your own menus and key-combinations, and even to write your own extensions to TeXmacs.

If you have downloaded the source files of TeXmacs, then it may be interesting for you to take a look at the files

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

These three "glue" files contain the C++ routines, which are visible within Scheme. In what follows, we will discuss some of the most important routines. We plan to write a more complete reference guide later. You may also take a look at the scheme `.scm` files in the directory `$TEXMACS_PATH/progs`.

### 11.2. Writing your own initialization files

When starting up, TeXmacs executes the file

```
$TEXMACS_PATH/progs/init-texmacs.scm
```

as well as your personal initialization file

```
$TEXMACS_HOME_PATH/progs/my-init-texmacs.scm
```

if it exists. By default, the path `$TEXMACS_HOME_PATH` equals `.TeXmacs`. Similarly, each time you create a new buffer, the file

```
$TEXMACS_PATH/progs/init-buffer.scm
```

is executed, as well as

```
$TEXMACS_HOME_PATH/progs/my-init-buffer.scm
```

if it exists.

## 11.3. Creating your own dynamic menus

You may define (or modify) a (part of a) menu with name `name` using

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

and append new entries to an existing (part of a) menu with name `name` using

```
(menu-extend name . prog)
```

Here `prog` is a program which represents the entries of the menu. In particular, you may take a look at the files in the directory

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

in order to see how the standard $\mathrm{T_EX_{MACS}}$ menus are defined.

More precisely, the program `prog` in `menu-set` or `menu-append` is a list of entries of one of the following forms:

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

The constructors `=>` and `->` are used to create pulldown or pullright menus and `menu-definition` should contain a program which creates the submenu. The constructor `("entry" action)` creates an ordinary entry, where `action` will be compiled and executed when you click on `entry`. Items of a menu may be separated using `---`. The constructor `if` is used for inserting menu items only if a certain `condition` is satisfied (for instance, if we are in math mode).

Finally, if you declared a menu `name`, then you may use this menu indirectly using the `link` constructor. This indirect way of declaring submenus has two advantages

- An "indirect" submenu may be linked to as many menus as we like.
- New items may be added to "indirect" submenus *a posteriori* using `menu-append`.

The main $\mathrm{T_EX_{MACS}}$ menus are `texmacs-menu`, `texmacs-popup-menu`, `texmacs-main-icons`, `texmacs-context-icons` and `texmacs-extra-icons`. Other standard indirect menus are `file-menu`, `edit-menu`, `insert-menu`, `text-menu`, `paragraph-menu`, `document-menu`, `options-menu` and `help-menu`.

## 11.4. Creating your own keyboard shortcuts

Keymaps are specified using the command

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

The predicate specifies under which circumstances the keymaps are valid. Examples of predicates are `always?`, `in-math?` and `in-french?`, but the user may define his own predicates. Each item in `keymaps` is of one of the following forms:

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

In the first case, the `action_i` are SCHEME commands associated to the string `key-combination`. In the second and third case, `result` is a string which is to be inserted in the text when the `key-combination` has been completed. An optional `help-message` may be displayed when the `key-combination` is finished.

## 11.5. OTHER INTERESTING FILES

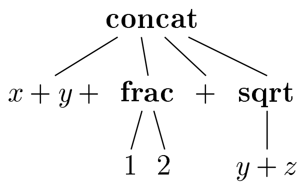
Some other files may also be worth looking at:

- `$TEXMACS_PATH/fonts/enc` contains encodings for different TEX fonts.
- `$TEXMACS_PATH/fonts/virtual` contains definitions of virtual characters.
- `$TEXMACS_PATH/langs/natural/dic` contains the current dictionaries used by TEXMACS.
- `$TEXMACS_PATH/langs/natural/hyphen` contains hyphenation patterns for various languages.
- `$TEXMACS_PATH/progs/fonts` contains SCHEME programs for setting up the fonts.

# Kapitel 12
# Das TeXmacs-Format

## 12.1. TeXmacs-Bäume

TeXmacs repräsentiert alle Dokumente oder Teile von Dokumenten, also alle Formen von Text, durch Bäume. Zum Beispiel repräsentiert der Baum




die Formel

$$x+y+\frac{1}{2}+\sqrt{y+z}$$

Die Knoten eines solchen Baum sind Standard-*Operatoren* des Typs `tree_label` (siehe `Basic/Data/tree.gen.h`), z.B.: **concat**, **frac**, **sqrt**. Der Inhalt der "Blätter" des Baums sind Zeichenketten (Strings). Diese sind entweder auf dem Bildschirm sichtbar und können auch gedruckt werden, dies ist der eigentliche Text, - oder sie sind auf dem Bildschirm nicht sichtbar wie z.B. Längen- oder Makro-Definitionen und werden daher auch nicht ausgedruckt. Die Baumdarstellung von Dokumenten in TeXmacs spiegelt ihren logischen Aufbau wieder. Dateien sind jedoch linear aufgebaut, sodass Bäume zur Speicherung in einen fortlaufenden Text umgeformt werden müssen (serialization, deutsch: *Linearisierung*), der von einem sogenannten Parser zurückübersetzt werden kann. Dies ist auf verschiedene Weise möglich. Der oben gezeigte Beispielbaum kann in der Notation von Scheme geschrieben werden als:

```
(concat
  "x+y"
  (frac "1" "2")
  "+"
  (sqrt "y+z"))
```

Wie ein solcher Text interpretiert wird und wie das Schriftsatz-Programm TeXmacs den Text schließlich setzt, hängt vom aktuellen Kontext ab. Dieser Kontext ist im wesentlichen eine Hash-Tabelle, welche die Kontextvariable mit den Bauminhalten verknüpft. Die aktuelle Sprache, die aktuelle Schrift und die aktuelle Farbe sind Beispiele für Kontextvariable; Variablen können durch den Nutzer geändert definiert werden. Zum Beispiel erzeugt der Scheme Ausdruck

```
(concat
  "Ein "
  (with "color" "blue" "blauer")
  " Text.")
```

das folgende Textfragment

> Ein blauer Text.

Der TEXMACS-Befehl `with` setzt eine Kontextvariable neu. Diese Änderung ist lokal, d.h., sie bezieht sich nur auf Ausdrücke in der Klammer (variable). Daher ist auch „blauer" blau.

Im nachfolgenden beschreiben wir im Detail, wie die verschiedenen Standard TEXMACS-Konstrukte (Befehle) und Kontextvariablen funktionieren. Es sollte erwähnt werden, dass an dem TEXMACS Datenformat noch gearbeitet wird. Im letzten Abschnitt werden diese Änderungen beschrieben. Für gewöhnlich wird der Anwender von einer Erweiterung des Datenformats nichts bemerken, da solch eine Änderung immer zusammen mit einem Konvertierungs-Programm entwickelt wird, das die bestehenden Dokumente automatisch auf das neue Format ergänzt. Die meisten Änderungen bestehen im Hinzufügen von neuen Konstrukten. Für Entwickler kann die Kenntnis solcher Änderungen dennoch wichtig sein.

## 12.2. TEXMACS-Dokumente

TEXMACS-Dokumente sind natürlich TEXMACS-Bäume, diese haben allerdings eine spezielle Form, die im folgenden beschrieben wird. Unter einem TEXMACS-Dokument wird hier eine Datei bezeichnet, die unter TEXMACS erstellt, sich im Speicher (Puffer) befindet oder auf einem Speichermedium gespeichert wurde. Textstücke, also Teile von Dokumenten, sind zwar auch TEXMACS-Bäume, aber normalerweise keine Dokumente.

Die Wurzel eines Dokuments muss der Standard-Operator document sein. Er hat notwendigerweise die beiden Kinder

⟨TeXmacs|*Version*⟩ (TEXMACS Version)

Dieser Operator ist notwendig und spezifiziert die Version von TEXMACS, die zum abspeichern des Dokuments benutzt wurde.

⟨body|*Inhalt*⟩ (Dokumenten-Rumpf )

Dieser notwendige Operator spezifiziert den Rumpf des Dokuments, also im Endeffekt den Inhalt.

Zu diesen beiden notwendigen Operatoren können weitere Operatoren/Kinder hinzutreten, sofern diese aus den folgenden ausgewählt werden:

⟨style|*Version*⟩
⟨style|⟨tuple|*style*|*pack-1*|···|*pack-n*⟩⟩ (Stil und Stilpakete)

Ein Stil und zusätzliche Stilpakete.

⟨project|*Verweis*⟩ (Projekthinweis)

Damit kann ein Verweis auf ein Projekt hinzugefügt werden, zu dem das Dokument gehört.

⟨initial|*Tabelle*⟩ (Start-Kontext)

Hiermit kann ein Kontext festgelegt werden, mit dem das Dokument gestartet wird: z.B. Seitengröße, Randmaße, usw.. Die *Tabelle* hat die Form ⟨collection|*Bindung-1*|···|*Bindung-n*⟩. Jede *Bindung-i* hat die Form ⟨associate|*Variable-i*|*Wert-i*⟩ und setzt die Kontextvariable *Variable-i* auf den Startwert *Wert-i*. Startwerte für Kontextvariable, die nicht in der Tabelle enthalten sind, werden durch den Basis-Stil und die Stilpakete festgelegt.

⟨references|*Tabelle*⟩ (Verweise)

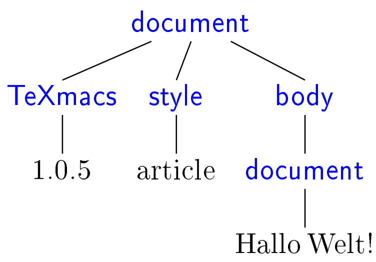
Wahlweise eine Liste aller gültigen Verweise auf eine Marke „Label” in dem Dokument. Auch wenn diese Information von TeXmacs automatisch erzeugt werden kann, ist es sinnvoll eine solche Tabelle abzuspeichern, denn es braucht mehrere Durchgänge, um eine solche Tabelle automatisch zu erzeugen. Um anwenderfreundliches Verhalten des Editors zu erreichen, werden Verweise zusammen mit dem Dokument gespeichert.

Die *Tabelle* hat eine ähnliche Form wie in der vorgehenden Operation. Hier wird ein „Tuple“ mit jedem Verweis verbunden. Dieses Tuple hat entweder die Form ⟨tuple | *Inhalt* | *Seiten-Nr*⟩ oder ⟨tuple | *Inhalt* | *Seiten-Nr* | *Datei*⟩. Der *Inhalt* enthält den Text, der gezeigt wird, wenn der Verweis angesprochen wird, und *Seiten-Nr* enthält die dazugehörige Nummer der Seite. Die Option „*Datei*” gibt die Datei an, in welcher die Marke „Label” definiert wurde. Das ist nur notwendig, wenn das Dokumente zu einem Projekt gehört, das aus mehreren Dateien besteht.

⟨auxiliary|*Tabelle*⟩ (zusätzliche Hilfsdaten zu einem Dokument)

Dies ist eine wahlweise Tabelle, die zusätzliche Hilfsdaten abspeichert. Wie oben können diese Daten automatisch aus dem Dokument selbst berechnet werden, diese Berechnungen können aber aufwändig sein und unter Umständen auch Werkzeuge benötigen, die möglicherweise auf Ihrem System nicht vorhanden sind. Die *Tabelle*, wird ähnlich wie oben definiert, assoziiert die Hilfsdaten mit einem Schlüssel. Standard-Schlüssel sind u.a. `bib`, `toc`, `idx`, `gly` usw..

Beispiel 12.1. Ein Beispiel für ein Dokument, das nur den einfachen Text „Hallo Welt!” enthält, liefert den folgenden Baum,




Wie Sie sehen, kann unterhalb der Dokument-Wurzel mit Standard-Operator document der Standard-Operator document erneut auftreten. Diese Äste sind in der Regel keine Dokumente, das die Kinder ⟨TeXmacs|*Version*⟩ und ⟨body|*Inhalt*⟩ fehlen.

## 12.3. Normale Linearisierung

Dokumente werden in der Standard-TeXmacs-Sprache geschrieben. Dies entspricht den Datei-Endungen `.tm` und `.ts`. Die Standard-TeXmacs-Syntax ist relativ leicht zu verstehen und zu lesen, so dass der Inhalt eines Dokuments, ähnlich wie bei LaTeX, schon mit einem einfachen Texteditor verstanden werden kann. Beispielsweise wird die Formel

$$x+y+\frac{1}{2}+\sqrt{y+z}$$

durch

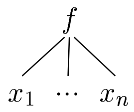
```
<with|mode|math|x+y+<frac|1|2>+<sqrt|y+z>>
```

dargestellt.

Dagegen ist die TeXmacs-Sprache, in der TeXmacs-Stile definiert werden, schwer zu lesen und nicht dazu gedacht, von Hand editiert zu werden: Die Semantik einer Sprache, mit der Leerräume (Weißraum) beschrieben werden kann, ist sehr komplex und schwer darzustellen. Deshalb sollte man es vermeiden, TeXmacs-Dokumente und vor allen Stil-Definitionen direkt von Hand zu editieren, es sei denn, man ist mit alle Details vertraut und weiß genau, was man tut.

### Das Prinzip der Linearisierung.

Das TeXmacs-Format benutzt die Sonderzeichen `<`, `|`, `>`, `\` und `/`, um Bäume zu linearisieren. Ein Baum wie



(12.1)

wird linearisiert zu

```
<f|x1|...|xn>
```

Wenn eines der Argumente $x_1$, ..., $x_n$ aus mehreren Absätzen besteht, was in diesem Zusammenhang heißt, dass dieses Argument den Standard-Operator `document` oder den Standard-Operator `collection` enthält, dann wird eine andere Form der Linearisierung verwendet. Wenn beispielsweise `f` nur Argumente mit mehreren Absätzen enthält, (Polyabsätze), dann würde der Baum (12.1) folgendermaßen linearisiert werden:

```
<\f>
  x1
<|f>
  ...
<|f>
  xn
</f>
```

Generell werden Argumente, die nicht Polyabsätze sind, in der Kurzform linearisiert und Polyabsätze in der Langform. Beispielsweise, wenn `n=5` ist, und `x3` sowie `x5` Polyabsätze sind, `x1`, `x2` und `x4` dagegen nicht, dann wird (12.1) linearisiert zu

```
<\f|x1|x2>
  x3
<|f|x4>
  x5
</f>
```

Die *Escape-Sequenzen* `\<`, `\|`, `\>` und `\\` können zur Darstellung der Zeichen `<`, `|`, `>` und `\` benutzt werden. Beispielsweise wird $\alpha + \beta$ zu `\<alpha\>+\<beta\>` linearisiert.

### Formatierung und Leerraum.

Die Grundoperationen `document` und `concat` werden auf spezielle Weise linearisiert. Die Linearisierung der Grundoperation `concat` besteht in der üblichen Verkettung von Zeichenketten. Beispielsweise wird der Text "an *important* note" linearisiert zu:

```
an <em|important> note
```

Die Grundoperation document wird linearisiert, indem Absätze durch zwei „neue Zeile"-Sonderzeichen, dies entspricht einer Leerzeile, getrennt werden. So wird das Zitat

> Dies ist der erste Absatz.
>
> Das ist der zweite Absatz.

zu

```
<\quote-env>
  Dies ist der erste Absatz.

  Das ist der zweite Absatz.
</quote-env>
```

linearisiert.

Beachten Sie bitte, dass Leerraum am Anfang und am Ende von Absätzen ignoriert wird. Innerhalb von Absätzen wird Leerraum beliebiger Länge als ein einziges Leerzeichen interpretiert. Entsprechend wird eine Folge von mehr als zwei „neue Zeile"-Zeichen als genau zwei „neue Zeile"-Zeichen interpretiert. Wenn also die linearisierte Form des obigen Beispiels folgendermaßen auf der Festplatte gespeichert wäre:

```
<\quote-env>
  Dies ist der           erste Absatz.



  Das ist der zweite          Absatz.
</quote-env>
```

würde das Zitat trotzdem unverändert dargestellt.

Ein Leerzeichen kann explizit durch die Zeichenkombination „\ " erzeugt werden und ein leerer Absatz durch „\;".

**Rohdaten.**

Die Grundoperation raw-data (Rohdaten) dient in $\mathrm{T_EX_{MACS}}$ zur Darstellung von Binärdaten wie beispielsweise Bilddateien innerhalb eines Dokuments. Solche Binärdaten werden folgendermaßen linearisiert:

```
<#binary-data>
```

Darin ist „*binary-data*" eine Zeichenkette von Hexadezimalzahlen, eine Kette von Bytes.

## 12.4. XML-Linearisierung

Um Kompatibilität mit XML zu erreichen, unterstützt $\mathrm{T_EX_{MACS}}$ auch die Linearisierung ins XML-Format. Allerdings ist das XML-Format wortreicher und schwerer lesbar als das normale $\mathrm{T_EX_{MACS}}$-Format. Um eine Datei im XML-Format (mit Datei-Erweiterung: `.tmml`) zu speichern oder zu laden, können Sie das Menü Datei → Exportieren → XML bzw. Datei → Importieren → XML.

$\mathrm{T_EX_{MACS}}$-Dokumente entsprechen keinem vordefiniertem DTD, denn das für ein Dokument zutreffende DTD hängt von seinem Stil ab. Das XML-Format stellt daher nur eine XML-Darstellung von $\mathrm{T_EX_{MACS}}$-Bäumen bereit. Die Syntax wurde mit dem Ziel, eine möglichst Baum-ähnliche Struktur zu erreichen, unter Verwendung konventioneller XML-Syntax entwickelt, die von den üblichen Standard-Werkzeugen unterstützt wird.

**die Codierung von Zeichenketten.**

Die Blätter der $\mathrm{T_EX_{MACS}}$-Bäume werden von der universellen $\mathrm{T_EX_{MACS}}$-Kodierung in Unicode übertragen. Zeichen ohne Unicode-Entsprechung werden als Dateneinheit dargestellt (für die Zukunft planen wir eine `tmsym`-Operation zur Darstellung solcher Zeichen).

**XML-Darstellung von normalen Operationen.**

Bäume mit einem einzigen Kind werden einfach durch die entsprechende XML-Operation ersetzt. Hat der Baum mehrere Kinder, wird jedes Kind in eine `tm-arg` -Markierung (tag) eingeschlossen. $\sqrt{x+y}$ wird so zu

```
<sqrt>y+z</sqrt>
```

linearisiert wird, während der Bruch $\frac{1}{2}$ durch

```
<frac>
  <tm-arg>1</tm-arg>
  <tm-arg>2</tm-arg>
</frac>
```

dargestellt wird.

In dem Beispiel oben wurde Leerraum ignoriert. Indem man die Standard-Variable `xml:space` auf `preserve` setzt, kann man Leerraum erhalten.

**Spezielle tags (Markierungen).**

Einige $\mathrm{T_EX_{MACS}}$-Operationen werden auf eine spezielle Weise nach XML übertragen. Die Grundoperation `concat` wird einfach durch die Verkettung von Zeichenketten ersetzt. So wird aus $\frac{1}{2}+\sqrt{x+y}$ einfach:

```
<frac><tm-arg>1</tm-arg><tm-arg>2</tm-
arg></frac>+<sqrt>y+z</sqrt>
```

Auch die `document` Grundoperation wird nicht explizit exportiert. Dafür wird jedes Argument, das ein Absatz ist, in `tm-par` Markierungen eingeschlossen. Z.B. wird das Zitat

> Dies ist der erste Absatz.
>
> Das ist der zweite Absatz.

linearisiert zu

```
<quote-env>
  <tm-par>
    Dies ist der erste Absatz.
  <tm-par>
    Das ist der zweite Absatz.
</quote-env>
```

Eine with Grundoperation, die nur Zeichenketten-Attribute und Werte enthält, wird durch die normalen XML-Attribut-Anweisungen ersetzt. „ein blauer Text” würde zu

```
ein <with color="blue">blauer</with> Text
```

Umgekehrt stellt TEXMACS die Grundoperation attr bereit, um XML Markierungen in TEXMACS verwenden zu können. So würde das XML Fragment

```
ein <mytag Tier="haarig">special</mytag> Text
```

importiert zu „ein ⟨my-tag | ⟨attr | Tier | haarig⟩ | special⟩ Text”. Deshalb ist es prinzipiell möglich, TEXMACS als Editor für normale XML-Dateien zu verwenden.

## 12.5. SCHEME-LINEARISIERUNG

In der Sprache SCHEME kann man leicht TEXMACS-Erweiterungen schreiben. In diesem Kontext werden TEXMACS-Bäume üblicherweise als SCHEME-Ausdrücke dargestellt. Die SCHEME-Syntax wurde geschaffen, um eine Sprache zu haben, die leicht von Hand zu editieren ist, voraussagbare Ergebnisse liefert und bei dem die interne Struktur von Dokumenten vollständig erkennbar ist. Beispielsweise wird die Formel

$$x+y+\frac{1}{2}+\sqrt{y+z}$$

in SCHEME dargestellt durch

```
(with "mode" "math" (concat "x+y+" (frac "1" "2") "+" (sqrt
"y+z")))
```

Die Darstellung in SCHEME kann sehr nützlich sein, wenn komplexe Makros mit hohem Programmieraufwand geschrieben werden sollen. Schließlich ist SCHEME das sicherste Format, wenn Textstücke aus TEXMACS per Email versandt werden sollen, da sowohl das Standard-TEXMACS-Format und als auch die XML-Linearisierung empfindlich auf Leerraum reagieren kann.

Um ein Dokument im SCHEME-Format zu sichern oder zu laden, können Sie die Menüs Datei → Exportieren → Scheme bzw. Datei → Importieren → Scheme verwenden. Dateien, die im SCHEME-Format gespeichert wurden, können in der Regel ohne weiteres von externen SCHEME-Programmen verarbeitet werden, ganz so wie Dateien im XML-Format von XML-Werkzeugen, wie z.B. XSLT.

Um einen Teil eines Dokuments im SCHEME-Format in eine Email zu kopieren, können Sie das Menü Bearbeiten → Kopieren nach → Scheme benutzen. Entsprechend können Sie externen SCHEME-Code in einen TEXMACS-Text einfügen, indem Sie Bearbeiten → Einfügen aus → Scheme benutzen. Das SCHEME-Format kann auch mit interaktiven Kurzbefehlen verwendet werden. Z.B. führt die Eingabe von `M-x` mit nachfolgendem

```
(insert '(frac "1" "2"))
```

zur Einfügung von $\frac{1}{2}$ an der Cursorposition.

Schließlich ist dieses Format zur interaktiven Eingabe geeignet, wenn TEXMACS als Oberfläche für SCHEME-Sitzungen eingesetzt wird.

## 12.6. Der Schriftsatz-Prozess

Um das TeXmacs-Format zu verstehen, sollte man wissen, wie der Schriftsatz von Dokumenten im Editor durchgeführt wird. Der Prozess setzt die logischen TeXmacs-Baum-Strukturen in physikalische *Boxen* um, die auf dem Bildschirm dargestellt werden können. Man muss sich darüber klar sein, dass diese Boxen außer den Informationen, die zur Darstellung auf dem Bildschirm erforderlich sind, weitere Informationen enthalten, so z.B. wie der Cursor innerhalb der Box zu positionieren ist oder wie wie Text ausgewählt wird.

Der Prozess, der den Schriftsatz durchführt, der *Schriftsetzer*, führt zwei verschiedene Abläufe aus: die Auswertung des TeXmacs-Baums und den eigentlichen Schriftsatz. Diese beiden Vorgänge laufen zur Zeit in einem Schritt ab, das kann sich aber in Zukunft ändern.

Die Schriftsatz-Konstrukte sind in den Editor integriert und auf Schnelligkeit optimiert. So gibt es z.B. Konstrukte zur horizontalen Verkettung (concat), für den Seitenumbruch (page-break), für mathematische Brüche (frac), für Hyperlinks (hlink) usw.. Die genaue Darstellungsweise vieler Schriftsatz-Konstrukte kann mittels vordefinierter Kontextvariablen gesteuert werden. Zum Beispiel spezifiziert die Kontextvariable *color* die aktuelle Farbe eines Objekts, *par-left* dagegen den linken Seitenabstand von Absätzen usw..

Die Sprache für Stil-Definitionen erlaubt dem Benutzer, neue Konstrukte (Makros) zu definieren und vordefinierte zu modifizieren. Sie enthält die Konstrukte zur Makrodefinition, Ablaufsteuerung, Berechnungen, bedingte und verzögerte Ausführung usw.. Die Sprache für Stil-Definitionen besitzt außerdem den besonderen Befehl extern, mit dem sich Scheme-Programme einbinden lassen.

Man beachte, dass benutzerdefinierte Makros zwei verschiedene Aufgaben durchführen.Eine Aufgabe besteht in einfachen Ersetzungen. Zum Beispiel ist das folgende Makro eine Kurzform von so etwas wie $a_1, \ldots, a_n$.

⟨assign|*seq*|⟨macro|*var*|*from*|*to*|$var_{from}, \ldots, var_{to}$⟩⟩

Wenn Makros nur aus Ersetzungen bestehen, dann sind sie und ihre Kinder „erreichbar“, *accessible* im Editor. Im Beispiel oben sind z.B. die Argumente *var*, *from* und *to* die Kinder von seq. Sie können also den Cursor hinein setzen und diese ändern. Die andere Aufgabe sind Berechnungen und synthetische Aufgaben. Das ist beispielsweise bei den Punkten im obigen Beispiel der Fall. Diese sind nicht erreichbar. Sie können sie nicht verändern. Das Makro

⟨assign|*square*|⟨macro|$x$|⟨times|$x$|$x$⟩⟩⟩

ist ein reines Rechenmakro. Generell gilt rein, dass synthetische Makros leichter zu schreiben sind, dass aber die Editierbarkeit von Makros leichter und natürlicher wird, je mehr man auf den Erhalt der Erreichbarkeit achtet.

TeXmacs produziert eine Reihe von Hilfsdaten während des Schriftsetzens, beispielsweise die Werte von Verweisen, Seitenzahlen, Inhaltsverzeichnisse, Indexe usw.. Diese werden im Zuge des Schriftsetzens ermittelt und gespeichert. Diese Hilfsdaten können prinzipiell aus dem Dokument berechnet werden. Das kann aber zeitaufwendig sein und im Fall, dass ein externes Plugin die Berechnungen durchführt, kann es passieren, dass dieses Plugin auf anderen Rechnern nicht zur Verfügung steht, die Berechnung also nicht durchführbar ist. Deshalb werden die gespeicherten Hilfsdaten mit dem Dokument zusammen abgespeichert, sobald sie es auf der Festplatte sichern.

# 12.7. Daten-Beschreibungen (D.R.D.) (data relation descriptions)

### der sinn von Daten-Beschreibungen, D.R.D.s.

Einer der Hauptvorteile von TeXmacs ist die Verwendung von Baumstrukturen als allgemeines Datenformat des Editors. Wie bei XML hat dies den Vorteil, dass diese Strukturen übersichtlich und leicht verständlich sind, sodass es einfach ist, Dokumente mit den eingebauten Werkzeugen zu ändern und anzupassen. Wenn jedoch der Editor für einen ganz bestimmten, besonderen Zweck eingesetzt werden soll, dann muss in aller Regel das Datenformat auf eine bestimmte Teilmenge der möglichen Bäume beschränkt werden.

In XML benutzt man Data Type Definitions (D.T.D.s), um eine spezielle Teilmenge des allgemeinen XML-Formats für einen bestimmten Zweck zu definieren: für Web-Dokumente „XHTML", für Mathematik „MathML", für 2-dimensionale Graphik „SVG" usw.. Außerdem erlaubt XML, solche D.T.D.s in einem gewissen Umfang miteinander zu kombinieren. Schließlich enthalten D.T.D.s. normalerweise ein Referenz-Handbuch.

In TeXmacs sind wir mit der Einführung von D.R.D.s noch einen Schritt weiter gegangen. Neben der Entscheidung, ob ein gegebenes Dokument syntaktisch korrekt ist, können wir damit auch formal bestimmte Eigenschaften des Dokuments beschreiben. Beispielsweise können Textfragmente erreichbar (*accessible*) sein oder eben nicht. So ist typischerweise der Zähler in einem Bruch erreichbar und kann daher jederzeit im Editor geändert werden, während beispielsweise die URL eines Hyperlinks nicht erreichbar ist und daher erst nach Aufhebung des Schutzes veränderbar wird. Ganz ähnlich implizieren bestimmte Konstrukte wie z.B. itemize Blockinhalte, während andere wie z.B. sqrt Zeileninhalt impliziert. Schließlich verhalten sich bestimmte verwandte Konstrukte wie chapter, section, subsection, usw. bei bestimmten Operationen wie z.B. Konversionen gleichartig.

Eine Daten-Beschreibung (D.R.D.) besteht aus einer „Data Type Definition" und zusätzlichen logischen Eigenschaften von Marken, Konstrukten und Textfragmenten. Diese logischen Eigenschaften werden in sogenannten *Horn clauses* definiert, die auch in anderen Programmiersprachen wie z.B. Prolog Verwendung finden. Es sollte damit relativ leicht sein, die Eigenschaften von Marken, Konstrukten und Textfragmenten zu bestimmen, so dass bestimmte Datenbanktechniken zur effektiven Implementierung genutzt werden können. Momentan haben wir damit gerade erst begonnen und benutzen zur Zeit noch eine Menge C++ Code im Widerspruch zu dem, was wir oben beschrieben haben. Daher können wir eine bessere formale Beschreibung von D.R.D.s jetzt noch nicht liefern.

Einer der wichtigsten Vorteile von D.R.D.s ist, dass anders als in objektorientierten Sprachen keine festen hierarchischen Strukturen vorgegeben werden müssen. Das ist deshalb so nützlich, weil Eigenschaften wie Veränderbarkeit (accessability), Blockinhalt, Zeileninhalt von einander unabhängig sind. Es ist in der Tat so, dass D.T.D.s meist zur Beschreibung passiver Dokumente ausreichend sind, dass aber, wenn es darum geht, Dokumente interaktiv zu editieren, eine feinere Beschreibung von Eigenschaften günstiger ist.

### Derzeitige Eigenschaften und Anwendungen von D.R.D.s.

Derzeit enthält die D.R.D. eines Dokuments die folgenden Informationen:

- Die erlaubten Anzahlen von Variablen (arity).

- Die Erreichbarkeit (accessability) eines Konstrukts und seiner Kinder.

In nächster Zukunft sollen die folgenden Eigenschaften eingeführt werden:

- Ist_Zeileninhalt (inline-ness) eines Konstrukts und seiner Kinder.
- Ist_Tabelle (tabular-ness) eines Konstrukts und seiner Kinder.
- Zweck eines Konstrukts und seiner Kinder.

Diese Informationen werden neben anderen für folgende Aktionen verwandt:

- Natürliches Vorgabeverhalten, wenn Konstrukte oder ihre Kinder erzeugt oder vernichtet werden, z.B. automatische Ergänzung von fehlenden Argumenten und/oder Löschung von Konstrukten mit zu geringer Argumentanzahl.
- Beschränke Suchoperationen auf erreichbare Knoten z.B. bei der Rechtschreibprüfung.
- Automatische Einfügung der Konstrukte document bzw. table, wenn Blöcke oder Tabellen erzeugt werden.
- Syntaktische Hervorhebungen im Quellmodus in Abhängigkeit von dem Zweck des Konstrukts und seiner Argumente.

**Erzeugung der Daten-Beschreibung (D.R.D.) eines Dokuments.**

$\TeX_{MACS}$ erzeugt für jedes Dokument eine eigene Daten-Beschreibung, D.R.D.. Diese wird in zwei Stufen erzeugt. Zunächst versucht $\TeX_{MACS}$ heuristisch die Eigenschaften von Konstrukten, die der Benutzer definiert hat, und die Konstrukte von assoziierten Stildefinitionen auszuwerten. Ist zum Beispiel folgendes Makro

```
⟨assign|hi|⟨macro|name|Hello name!⟩⟩
```

vorhanden, dann erkennt $\TeX_{MACS}$ automatisch, dass hi ein Makro mit einem Argument ist, und nimmt an, dass die zulässige Argumentanzahl 1 ist. Beachten Sie, dass die heuristische Erzeugung der Daten-Beschreibung interaktiv ist. Wenn Sie ein Makro in Ihrem Dokument erzeugen, werden seine Eigenschaften automatisch in die Daten-Beschreibung aufgenommen, sofern Sie $\TeX_{MACS}$ dafür etwas Zeit lassen (ca. 1 s).

Manchmal sind die heuristisch definierten Eigenschaften nicht zutreffend. Für diesen Fall besitzt $\TeX_{MACS}$ das Konstrukt drd-props, mit der eine manuelle Korrektur dieser Eigenschaften durchgeführt werden kann.

## 12.8. STANDARD LÄNGENEINHEITEN

Die Blätter von $\TeX_{MACS}$ - Bäumen enthalten entweder normalen Text oder spezielle Daten. $\TeX_{MACS}$ kennt die folgenden atomaren Datentypen:

**Boolesche Zahlen.** Entweder `true` oder `false` (richtig, falsch - ja, nein).

**Ganzzahl.** Folge von Ziffern, vor denen ein Plus- oder Minus-Zeichen stehen darf.

**Gleitpunktzahlen.** Zahlen mit Dezimalpunkt in der üblichen wissenschaftlichen Darstellung.

**Längen.** Gleitpunktzahl gefolgt von einer Längeneinheit z. B. `29.7cm` oder `2fn`.

In diesem Abschnitt besprechen wir die $\mathrm{T_EX_{MACS}}$ - Längeneinheiten.

Es gibt zwei prinzipiell verschiedene Arten von Längeneinheiten/Längen: fixe Längeneinheiten und Kontext-abhängige Längeneinheiten. Fixe Längeneinheiten sind die üblichen Längeneinheiten: ihre Länge auf dem Bildschirm oder dem Papier ist vorgegeben und unveränderlich. Die Länge Kontext-abhängiger Einheiten wird dagegen den vorliegenden Gegebenheiten angepasst. Solche können beispielsweise Schriftart und Schriftgröße sein.

Einige der variablen Längeneinheiten sind *dehnbar*. Drei Kennzahlen charakterisieren eine *dehnbare* Länge: die minimale Länge, die Vorgabelänge und die maximale Länge. Wenn Zeilen oder Seiten im Blocksatz umgebrochen werden, werden *dehnbare Längen* so angepasst, dass ein optimales Druckbild entsteht.

Im Fall des Seitenumbruchs erlaubt die *page-flexibility* - Umgebung eine zusätzliche Steuerung der Dehnbarkeit von Leerräumen. Setzt man die *page-flexibility* auf 1, so verhält sich dehnbarer Leerraum wie gewohnt. Wird die *page-flexibility* dagegen auf 0 gesetzt, dann werden dehnbare Längen ***fix***. Andere Werte beeinflussen das Verhalten linear.

**Fixe Längeneinheiten.**

`cm`. 1 Zentimeter.

`mm`. 1 Millimeter.

`in`. 1 inch (Zoll).

`pt`. 1 typographischer Punkt: 1/72 inch = 0.353 mm.

**kontext-abhängige Längeneinheiten.**

`fn`. Die Nenngröße der Schrift. Typischerweise sind die Basislinien zweier aufeinander folgender Zeilen durch den Abstand `1fn` getrennt (in $\mathrm{T_EX_{MACS}}$ und LaTeX wird ein geringfügig größerer Abstand benutzt, um obere und untere Indices besser darstellen zu können. Die Dehnbarkeit liegt für 1fn zwischen `0.5fn` und `1.5fn`.

`fn*`. Ist eine Variante von `fn`, mit der Vorgabelänge Null, die aber bis `1fn` gedehnt werden kann.

`spc`. Die dehnbare Breite eines Leerzeichens in der aktuellen Schrift.

`ex`. Die Höhe des Buchstabens "x" in der aktuellen Schrift.

`ln`. Die Breite eines gut aussehenden Bruchstrichs in der aktuellen Schrift.

`yfrac`. Der Abstand eines Bruchstrichs von der Basislinie in der aktuellen Schrift. (ungefähr `0.5ex`).

`sep`. Ein typischer Abstand zwischen Text und Graphik in der aktuellen Schrift, der benötigt wird, um den Text lesbar darzustellen. Beispielsweise wird der Zähler eines Bruchs um den Betrag `1sep` nach oben gesetzt.

**weitere Längeneinheiten.**

`par`. Die zulässige Text-breite in einem Absatz. Sie hängt von der Papiergröße, den Rändern, der Spaltenzahl, dem Spaltenabstand usw. ab.

`pag`. Die Länge des Haupt-Textes einer Seite. Ähnlich wie `par`, wird diese Längeneinheit von der Papiergröße, den Rändern usw. beeinflusst.

`px`. 1 Bildschirm-Pixel. Diese Längeneinheit hängt von der Bildschirmauflösung ab, die TeXmacs vom X Server beim Start mitgeteilt bekommt.

`unit`. px/256. Diese Längeneinheit wird von TeXmacs intern für Längenberechnungen benutzt.

# Kapitel 13

## Vordefinierte Kontextvariable

Die Art und Weise wie $\text{T}_\text{E}\text{X}_\text{MACS}$ den Schriftsatz von Dokumenten durchführt, wird durch sogenannte *Kontextvariablen* beeinflusst. Die Stildefinitions-Sprache benutzt einen sogenannten *Kontext*, um dort Kontextvariable und Makros zu speichern. Kontextvariablen lassen sich in zwei Klassen einteilen: vordefinierte Variablen und zusätzliche Variablen, die durch Stildefinitionen bereitgestellt werden. Vordefinierte Variablen beeinflussen in der Regel das eigentliche Layout, während die zusätzlichen Variablen mehr zu Berechnungen dienen. In den nächsten Abschnitten werden wir alle vordefinierten Kontextvariablen beschreiben.

Eine typische vordefinierte Kontextvariable ist *color*. Der Wert dieser Variablen kann dauerhaft geändert werden, mit dem Befehl assign und vorübergehend (lokal) mit dem Konstrukt with:

Roter Text.

⟨with|*color*|dark red|Roter⟩ Text.

Zähler sind typische Kontextvariablen von Stildefinitionen:

1. eine verrückt

4. nummerierte Liste ...

```
⟨enumerate|
    ⟨item⟩Eine verrückt
    ⟨assign|item-nr|3⟩⟨item⟩nummerierte Liste ...⟩
```

Die Schriftsatz-sprache benutzt *dynamische Kontextbereiche* für ihre Variablen. Das bedeutet, dass Makros auf Kontextvariablen, die den Kontext betreffen, indem sie aufgerufen wurden, zugreifen können und diese modifizieren dürfen. Im obigen Beispiel hat das enumerate-Makro lokal die Variable *item-nr* auf 0 gesetzt (dabei hat es with benutzt) dann inkrementiert das item-Makro um 1 und zeigt den Wert. Im folgenden wird durch assign auf 3 gesetzt und durch item-Makro um 1 inkrementiert und angezeigt. Der Original-Wert von *item-nr* wird beim Verlassen von enumerate wiederhergestellt.

Jede Dokument kommt mit einem Startkontext mit vorgegebenen Werten für die Kontextvariablen, das sind also Werte, die gesetzt werden, bevor mit dem Schriftsetzen begonnen wird. Wenn eine Kontextvariable in diesem Start-kontext noch nicht vorhanden ist, dann wird sie auf ihren Vorgabewert gesetzt, nachdem der Dokument-Stil gesetzt wurde und ggfs. weitere Stilpakete geladen wurden. Der Start-kontext selbst ist Teil des Editors.

Einige Variablen wie Kopf- und Fußzeilen, müssen innerhalb des Dokuments gesetzt werden. Ihre Startwerte werden ignoriert. Sie sollten generell immer mit den Fuß- und Kopfzeilen-Befehlen gesetzt werden.

* Allgemeine Kontextvariable

## 13.1. Festlegung der aktuellen Schriftart

In diesem Abschnitt werden wir die Kontextvariablen beschreiben, die die Darstellung von Schriften steuern. Vier Parameter bestimmen die Schrift (Name, Variante, Stärke, Form). Einige Kontextvariablen steuern das Verhalten unabhängig von diesen Parametern, während andere für davon abhängig sind. Schrift-Eigenschaften können global über das Menü Dokument → Schriftart und lokal über das Menü Formate → Schriftart gesetzt werden.

Aus einem abstrakten Gesichtspunkt ist eine Schriftart eine in sich konsistente graphische Darstellungsweise von Zeichen wie z.B. „x", „ffi", „$\alpha$", „$\sum$", usw.. Wenn eine Zeichenkette dargestellt werden soll, dann wird sie zuerst in Zeichen zerlegt, um z.B. Ligaturen wie fi, fl, ff, ffi, ffl zu berücksichtigen. Dann werden die einzelnen Zeichen positioniert, wobei die individuellen Eigenschaften der einzelnen Zeichen berücksichtigte werden. So wird z.B. in „xo" das Zeichen „o" ein wenig nach links gerückt, um das „Loch" in „x" zu berücksichtigen. Im Fall von mathematischen Schriftarten stellt TeXmacs eine kohärente Darstellung von größenveränderlichen Zeichen bereit, wie z.B. Klammern:

$$\left(\left(\Bigg(\Big(\Big)\Bigg)\right)\right).$$

Eine Schriftfamilie ist eine Anzahl von Schriftvarianten mit verschiedenen Charakteristiken wie Schriftstärke, Neigung usw., die aber alle gemeinsame, in sich konsistente typographische Regeln besitzen. Die Schriftarten einer Familie passen gut zusammen und werden deshalb im gemeinsam im gleichen Dokument verwendet. Dabei hat oft jede Schriftart seine spezielle Aufgabe. So passen die Schriftarten der Familie „Roman" z.B. die Varianten **fett** und *italic* gut zueinander, während die Schriftart Avant Garde nicht dazu passt.

Bemerkung 13.1. Für die Zukunft planen wir die Variablen Variante und Form durch eine größere Zahl von Variablen zu ersetzen, um Eigenschaften wie Neigung, Serifen, Kapitälchen usw. individuell zu steuern. Es ist außerdem geplant Unicode Schriftarten, möglicherweise mit zusätzlichen mathematischen Zeichen, zu verwenden. Dies sollte automatisch zu landesspezifisch korrekten Schriften führen, so dass z.B. kyrillische Schriftzeichen in russischen Texten zur Verfügung stehen.

*font* := `roman`
*math-font* := `roman`
*prog-font* := `roman` (Schrift-Name)

Diese Variablen setzen die Schriftfamilie. Beispiele sind:

Roman, Pandora, *Chancery*, Palatino

Genauso unterstützt TeXmacs verschiedenen mathematische Schriftarten:

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

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

New roman: $a^2 + b^2 = c^2$

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

*font-family* := `rm`
*math-font-family* := `mr`
*prog-font-family* := `tt` (Schriftvariante)

Diese Variable wählt eine Variante aus der Schriftfamilie aus, wie z.B: Sans-Serif, Schreibmaschine usw.. Wie bereits erklärt, passen Varianten einer Schriftfamilie gut zu einander. Aber nicht alle Schriftfamilien haben alle möglichen Varianten. Wenn eine Variante gewählt wird, die nicht vorhanden ist, dann versucht TeXmacs eine passende Alternative zu finden.

Typisch Varianten für Text-Schriftarten sind `rm` (Roman), `tt` (Schreibmaschine) und `ss` (Sans- Serif):

> Roman, `Schreibmaschine` und Sans-Serif

Die Schriftvarianten des Mathematik-Modus `mr` (Roman), `mt` (Schreibmaschine) und `ms` (Sans-Serif) unterscheiden sich von ihren Text-Entsprechungen `rm` (Roman), usw.. In der Mathematik-Variante haben Variablen und Operatoren in der Regel unterschiedliche Neigungen, was in der Textversion fehlt.

> `ms`: $\sin(x+y) = \sin x \cos y + \cos x \sin y$
>
> `ss`: $\sin(\mathrm{x}+\mathrm{y}) = \sin \mathrm{x} \cos \mathrm{y} + \cos \mathrm{x} \sin \mathrm{y}$

*font-series* := `medium`
*math-font-series* := `medium`
*prog-font-series* := `medium` (Schriftstärke)

Diese Kontextvariablen regeln die Schriftstärke. Mögliche Werte sind: light, medium, bold (mager, mittel, fett). Die meisten Schriften besitzen nur die Varianten mittel und fett.

> medium, **bold**

*font-shape* := `right`
*math-font-shape* := `normal`
*prog-font-shape* := `right` (Schriftform)

Diese Parameter bestimmen die Form, d.h. solche Eigenschaften wie Neigung, Kapitälchen, Proportionalschrift usw., wie in den folgenden Beispielen:

> senkrecht = right, *geneigt* = slanted, *kursiv* = italic, nach links geneigt = left-slanted, Kapitälchen = small-caps, proportionale Schreibmaschinenschrift = proportional, **fett dicht** = condensed, Sans-Serif flach = flat, lang = long

*font-base-size* := `10` (Basis-Schriftgröße)

Die Basis-Schriftgröße wird in `Punkten, pt` festgelegt und ist normalerweise für das ganze Dokument fest eingestellt. Üblicherweise ist die Basisgröße `9pt`, `10pt`, `11pt` oder `12pt`. Andere Größen werden normalerweise durch Festlegung des *Vergrößerungsfaktor* oder des Größenverhältnisses font-size erzeugt.

9pt, 10pt, 11pt, 12pt

*font-size* := 1 (Schriftgröße)

Die aktuelle Schriftgröße wird aus der Basis-Schriftgröße *font-base-size* durch Multiplikation mit dem Größenverhältnis *font-size* ermittelt. Die folgenden Standard-Schriftgrößen sind im Menü Formate → Größe einzustellen:

| Schriftgröße | Multiplikator | Schriftgröße | Multiplikator |
|---|---|---|---|
| Tiny | 0.59 | Very small | 0.71 |
| Small | 0.84 | Normal | 1 |
| Large | 1.19 | Very large | 1.41 |
| Huge | 1.68 | Really huge | 2 |

**Tabelle 13.1.** Standard Schriftgrößen.

Die Multiplikatoren bilden eine geometrische Folge mit dem Faktor $\sqrt[4]{2}$. Beachten Sie bitte, dass die Schriftgröße auch noch von der Kontextvariablen index level abhängt.

*dpi* := 600 (Auflösung)

Die Auflösung von Rasterschriftarten (auch Typ 3 fonts genannt), wie sie beispielsweise von dem METAFONT Programm erzeugt werden, ist abhängig von der Präzision der Rasterung in Punkten pro Zoll, dots per inch, dpi. Heutzutage liefern die meisten Laserdrucker eine Auflösung von wenigstens `600 dpi`, was auch die Vorgabe für TeXmacs ist. Für professionellen Hochqualitätsdruck werden heutzutage meist `1200 dpi` benutzt. Die Auflösung wird normalerweise nur einmal für das gesamte Dokument eingestellt.

## 13.2. Mathematischer Schriftsatz

*math-level* := 0 (Indexhöhe)

In geschachtelten Indizierungen (Potenzen) oder Brüchen kennzeichnet die *Indexhöhe*, *index level*, das Ausmaß der Verschachtelung. Sie nimmt innerhalb bestimmter mathematischer zu. Ein hoher Wert führt zur Darstellung in einer kleineren Schriftgröße. Das gilt allerdings nur bis zur Indexhöhe 2. Ab 2 bleibt die Größe gleich, damit die Formeln lesbar bleiben:

$$\mathrm{e}^{\mathrm{e}^{\mathrm{e}^{\mathrm{e}^{x}}}} = \frac{1+\frac{1}{x+\mathrm{e}^{x}}}{1+\frac{1}{\mathrm{e}^{x}+\frac{1}{\mathrm{e}^{\mathrm{e}^{x}}}}}$$

Die Indexhöhe kann im Menü Formate → Indexhöhe manuell verändert werden, um beispielsweise $x^{y^z}$ so darzustellen

$$x^{y^{z}}$$

x⟨rsup|⟨with|*math-level*|0|y⟨rsup|⟨with|*math-level*|0|z⟩⟩⟩⟩

anstatt so

$$x^{y^z}$$

*math-display* := `false` (eigenständige Formel)

Diese Variable steuert den Stil der Formel nämlich, ob die Formel wie eine eigenständige Formel oder wie eine Formel im laufenden Text gesetzt wird. Eigenständige Formeln wie

$$\frac{n}{H(\alpha_1, \ldots, \alpha_n)} = \frac{1}{\alpha_1} + \cdots + \frac{1}{\alpha_n}$$

sind anders gesetzt als solche Formeln $\frac{n}{H(\alpha_1, \ldots, \alpha_n)} = \frac{1}{\alpha_1} + \cdots + \frac{1}{\alpha_n}$ im laufenden Text. der Stil für eigenständige Formeln erzeugt weitere Abstände. Er wird in manchen mathematischen Konstrukten wie z.B. in Brüchen, Indices, Binomialkoeffizienten usw. automatisch abgeschaltet. Daher haben im folgenden Beispiel die Brüche im Nenner eine kleinere Schriftgröße:

$$H(\alpha_1, \ldots, \alpha_n) = \frac{n}{\frac{1}{\alpha_1} + \cdots + \frac{1}{\alpha_n}}$$

Sie könne die Vorgaben im Menü Formate → Eigenständige Formel ändern.

*math-condensed* := `false` (Dichte eigenständige Formeln)

Formeln wie $a + \cdots + z$ werden mit weiten Abständen zum + Symbol gesetzt. In Formeln mit Indices wie $\mathrm{e}^{a+\cdots+z} + \mathrm{e}^{\alpha+\cdots+\zeta}$ wird die Lesbarkeit erhöht, indem innerhalb der Indices kürzere Abstände verwendet werden. Das hilft, Symbole in Indices von den Basis-Zeichen zu unterscheiden ganz besonders dann, wenn die Indices sehr lang sind. Dieses vorgegebene Verhalten kann im Menü Formate → Dicht angepasst werden.

*math-vpos* := 0 (Indexposition in Brüchen)

Um Brüche in hoher Qualität zu setzen, muss Vorsorge getroffen werden, dass untere Indices in Zählern nicht zu weit nach unten reichen, und dass obere Indices in Nennern nicht zu weit nach oben kommen. Deshalb gibt es in TeXmacs eine weitere Variable *math-vpos*, die den Wert 1 in Zählern annimmt und −1 in Nennern. Außerhalb von Brüchen bleibt sie 0. Das folgende Beispiel zeigt diesen Effekt. Die Unterschiede sind sehr gering aber im qualitativ hochwertigem Drucksatz sichtbar.

$$a_1^2 + a_1^2 + a_1^2$$

```
⟨with|math-vpos|-1|⟨group|a⟨rsub|-1⟩⟨rsup|2⟩⟩⟩+⟨with|math-vpos|0|
⟨group|a⟨rsub|-1⟩⟨rsup|2⟩⟩⟩+⟨with|math-vpos|1|⟨group|a⟨rsub|1⟩⟨rsup|
2⟩⟩⟩
```

Die Gruppierung ist in diesem Beispiel notwendig, damit sich die vertikalen Effekte auf die ganze Gruppe ausdehnen.

## 13.3. Absatz-Layout

*par-mode* := `justify` (Absatzausrichtung)

Diese Kontextvariable legt fest wie Zeilen ausgerichtet werden, also wie sie in Bezug auf die Absatzränder gesetzt werden. Es gibt vier mögliche Werte: `left`, `center`, `right` und `justify`, die den Ausrichtungen linksbündig, zentriert, rechtsbündig und Blocksatz entsprechen:

| | |
|---|---|
| Dieser Absatz ist linksbündig ausgerichtet. Dieser Absatz ist linksbündig ausgerichtet. Dieser Absatz ist linksbündig ausgerichtet. Dieser Absatz ist linksbündig ausgerichtet.Dieser Absatz ist linksbündig ausgerichtet. | Dieser Absatz ist zentriert ausgerichtet. Dieser Absatz ist zentriert ausgerichtet. Dieser Absatz ist zentriert ausgerichtet. Dieser Absatz ist zentriert ausgerichtet. |
| Dieser Absatz ist rechtsbündig ausgerichtet. Dieser Absatz ist rechtsbündig ausgerichtet. Dieser Absatz ist rechtsbündig ausgerichtet. Dieser Absatz ist rechtsbündig ausgerichtet. | Dieser Absatz ist im Blocksatz gesetzt. Dieser Absatz ist im Blocksatz gesetzt. Dieser Absatz ist im Blocksatz gesetzt. Dieser Absatz ist im Blocksatz gesetzt. Blocksatz ist die Vorgabe. |

**Tabelle 13.2.** Unterstützte Absatzausrichtungen.

*par-hyphen* := `normal` (Qualität der Trennungen)

Dieser Parameter steuert die Qualität der Trennungs-Algorithmen. Mögliche Werte sind `normal` und `professional`. Der professionelle Trennungs-Algorithmus verwendet einen Algorithmus der den ganzen Absatz umfasst. Der normale ist schneller.

| | |
|---|---|
| Die Unterschiede zwischen den verschiedenen möglichen Trennungs-Algorithmen sieht man in der Regel erst bei längeren Absätzen und zwar dann, wenn die Absätze in schmale Kolumnen gesetzt werden. | Die Unterschiede zwischen den verschiedenen möglichen Trennungs-Algorithmen sieht man in der Regel erst bei längeren Absätzen und vor allem dann, wenn die Absätze in schmale Kolumnen gesetzt werden. |

**Tabelle 13.3.** Vergleich zwischen verschiedenen Trennungs-Algorithmen. Links der normale Algorithmus, rechts der professionelle. Auch wenn auf der rechten Seite noch einige unschöne Lücken verbleiben, so wurden von dem professionellen Algorithmus doch die unschönen Lücken in der vorletzten Zeile um das Wort „Absätze“ vermieden.

*par-width* := `auto` (Absatzbreite)

Diese Kontextvariable steuert die Breite der Absätze. Normalerweise wird die Absatzbreite automatisch aus der Papierbreite und den Breiten der Ränder berechnet.

*par-left* := `0cm`
*par-right* := `0cm` (Linker und rechter Rand)

Diese Kontextvariablen bestimmen die Breite des linken bzw. rechten Randes von Absätzen und zwar in Bezug auf die Vorgabewerte, die von dem Seitenlayout bestimmt werden.

Ein Beispiel:

> Dieser Text benutzt die Vorgabe.
>
> Dieser Text hat einen linken Rand von `1cm`.
>
> Dieser Text hat einen linken Rand von `2cm`.
>
> Dieser Text hat einen linken Rand von `3cm`.
>
> Die linken und die rechten Ränder dieses Textes sind beide auf `3cm` gesetzt worden.

Layoutelemente wie Auflistungen, itemize, oder quote-env, die verschachtelt werden können, berechnen in der Regel neue Randabstände als Funktion der alten, indem sie vorgegebene Abstände hinzufügen oder abziehen. Das verdeutlicht die typischen Makrodefinition für quote-env des folgenden Beispiels:

```
⟨assign|quote-env|
  ⟨macro|body|
    ⟨surround|
      ⟨vspace*|0.5fn⟩|
      ⟨right-flush⟩⟨vspace|0.5fn⟩|
      ⟨with|par-left|⟨plus|par-left|3fn⟩|par-right|⟨plus|par-right|
      3fn⟩|par-first|0fn|par-par-sep|0.25fn|body⟩⟩⟩⟩
```

*par-first* := `1.5fn` (Erstzeileneinzug)

Der *par-first*-Parameter spezifiziert den Erstzeileneinzug. Ein solcher Einzug hat den Sinn, den Beginn eines neuen Absatzes zu kennzeichnen. Eine andere Alternative ist ein erhöhter Zeilenabstand.

| | |
|---|---|
| In den TeXmacs-Basis-Stilen `Artikel` und `Buch` wird der Anfang eines neuen Absatzes durch den Einzug der ersten Zeile gekennzeichnet.<br>Der TeXmacs-`allgemein`- und der `Brief`-Basis-Stil benutzen dagegen senkrechten Abstand. | Der TeXmacs-`allgemein`- und der `Brief`-Basis-Stil benutzen einen zusätzlichen senkrechten Abstand zur Kennzeichnung des Beginns von neuen Absätzen.<br>In den TeXmacs-Basis-Stilen `Artikel` und `Buch` wird der Anfang eines neuen Absatzes dagegen durch den Einzug der ersten Zeile gekennzeichnet.. |

**Tabelle 13.4.** Zwei klassische Weisen zur Kennzeichnung eines neuen Absatzes.

*par-par-sep* := `0.5fn*` (Zusätzlicher Abstand zwischen Absätzen)

Der *par-par-sep*-Parameter legt den Abstand zwischen zwei auf einander folgenden Absätzen fest. Der Abstand wird in kontextabhängigen Längeneinheiten gemessen. In der Regel erzeugt TeXmacs keinen vergrößerten Zeilenabstand zwischen auf einander folgenden Absätzen, es sei denn, es würde kein vernünftiger Seitenumbruch gefunden. Darum wird die kontextabhängige Längeneinheiten `fn*` benutzt. Normalerweise wird der Erstzeileneinzug benutzt (Tabelle 13.4).

*par-line-sep* := `0.025fn*` (Leerraum zwischen Zeilen)

Dieser Parameter legt die *Breite des Leerraums* zwischen den Zeilen in einem Absatz fest. Dies entspricht nicht dem, was man normalerweise mit Zeilenabstand bezeichnet. Die übliche Definition des Zeilenabstands ist die Summe des Leerraums und der Schrifthöhe.

> Ein doppelter „Zeilenabstand" entspricht *par-line-sep* := `1fn`. Dieser wird oft von faulen Menschen verwendet, die vorgeben wollen, viele Seiten geschrieben zu haben, die sich aber um das Wohlergehen der Wälder nicht kümmern..

*par-sep* := `0.2fn` (Minimaler vertikaler Abstand zwischen Zeilen)

Diese Variable definiert einen Mindestabstand zwischen Boxen in den einzelnen Zeilen. Das verhindert Kollisionen von besonders großen Kästen mit solchen in vorausgehenden bzw. den nachfolgenden Zeilen.

*par-hor-sep* := `0.5fn` (Minimaler horizontaler Abstand)

Wenn ein Absatz mehrere besonders große Boxen enthält, dann versucht TeXmacs die auf einander folgenden Zeilen in einander zu schieben, solange Boxen nicht mit einander kollidieren:

> Betrachten Sie einen Bruch, der sich tiefer als die Unterlängen der normalen Schrift erstreckt wie beispielsweise den Bruch $\frac{1}{x+1}$ und einen Ausdruck, der höher als normal ist wie $\mathrm{e}^{\mathrm{e}^x}$. Wie Sie sehen versucht TeXmacs eine kompakte Darstellung zu erreichen.
>
> Wenn der Bruch $\frac{1}{x+1}$ und der besonders hohe Ausdruck aber an der falschen Stelle liegen, wie $\mathrm{e}^{\mathrm{e}^x}$ hier, dann bleiben die Boxen im Abstand *par-sep*.

Wenn der horizontale Abstand zwischen zwei großen Boxen kleiner ist als *par-hor-sep* dann wird das als Kollision betrachtet.

*par-fnote-sep* := `0.2fn` (Minimaler Abstand zwischen Fußnoten)

Dieser Parameter definiert den Abstand zwischen auf einander folgenden Fußnoten.

*par-columns* := `1` (Spaltenanzahl)

Diese Variable legt die Anzahl der Spalten fest. Innerhalb eines Dokuments können unterschiedliche Spaltenanzahlen verwendet werden.

*par-columns-sep* := `2fn` (Spaltenabstand)

Die Variable definiert die horizontale Breite des Leerraums zwischen Spalten.

## 13.4. Seitenlayout

In diesem Abschnitt befassen wir uns damit, wie TeXmacs Seiten mit formatiertem Text füllt. Der Benutzer kann nicht nur festlegen, wie der Text gedruckt werden soll. Er kann auch bestimmen, wie der Text auf dem Bildschirm dargestellt werden soll. Man sollte sich darüber im Klaren sein, dass die Anzahl der Dokument-Variablen redundant ist, da einige sich aus anderen errechnen lassen. So wird beispielsweise die Absatzbreite gemäß Voreinstellung aus der Papiergröße und der Breite des linken und rechten Randes berechnet.

**Papier spezifische Variablen.**

*page-type* := `a4` (die Papiergröße)

Gibt die Papiergröße beim Druck an. Im Menü Dokument → Seite → Größe können fast alle üblichen Papierformate ausgewählt werden. Als Voreinstellung dient die Papiergröße des Druckers, die im Menü Bearbeiten → Einstellungen → Drucker eingestellt werden kann. Setzt man *page-type* :=`user`, dann wird die Papiergröße durch die Variablen *page-width* und *page-height* bestimmt.

*page-orientation* := `portrait` (Seitenausrichtung)

Die Seitenausrichtung kann entweder `portrait` (Längsformat) oder `landscape` (Querformat) sein.

*page-nr* := `0` (aktuelle Seitennummer)

Die aktuelle Seitennummer. Diese Variable sollte mit großer Vorsicht behandelt werden, da sie zur Entwurfszeit noch nicht bekannt ist. Für eine zuverlässige Ermittlung der aktuellen Seitennummer sollte man die Grundformen label und page-ref geeignet kombinieren. Jedoch kann die *page-nr*-Variable in Makros zur Erzeugung von Seitenkopf und -fuß benutzt werden.

*page-the-page* (die Seitennummer ausgeben)

Diese Variable enthält in Wirklichkeit ein Makro, das die Seitennummer ausgibt. Als Vorgabe gibt es den Inhalt von *page-nr* aus. Dieses Makro nimmt keine Argumente an. Um ein Dokument zu simulieren, dessen Seitenzahl mit 123 beginnt, kann man das Makro z.B. folgendermaßen umdefinieren:

```
⟨assign|page-the-page|⟨macro|⟨plus|page-nr|122⟩⟩⟩
```

*page-breaking* := `optimal` (Seitenumbruch-Algorithmus)

Dieser Parameter wählt den Seitenumbruch-Algorithmus. Die Vorgabe ist „`optimal`“. Dieser Algorithmus berücksichtigt die globalen gesetzten Dokument-Optionen und versucht schlechte Seitenumbrüche zu vermeiden. Die Alternative ist „`sloppy`“, ein Algorithmus, der schnell ist aber schlechte Seitenumbrüche mit erhöhter Wahrscheinlichkeit liefert. Ein weiterer Algorithmus „`medium`“ entspricht dem „`optimal`“-Algorithmus, außer für zwei-spaltigen Text.

*page-flexibility* := `1.0` (Dehnungsflexibilität)

Dieser Parameter spezifiziert wie stark dehnbarer Leerraum gedehnt oder komprimiert werden darf, um Seiten zu füllen, die an sich zu kurz oder zu lang sind. Eine Dehnungsflexibilität von 1.0 erlaubt Änderungen bis zu den Minimal- bzw. Maximalwerten. Eine Dehnungsflexibilität von 0.0 verhindert die Anpassung. Andere Werte der *page-flexibility* verhalten sich linear.

*page-shrink* := `1fn` (maximal zulässige Seitenlängenreduzierung)

In Fällen, in denen es schwer ist, gute Seitenumbrüche zu finden, spezifiziert dieser Parameter eine maximal zulässige Verkürzung der Seitenlänge.

*page-extend* := `0fn` (maximal zulässige Seitenverlängerung)

In Fällen, in denen es schwer ist, gute Seitenumbrüche zu finden, spezifiziert dieser Parameter eine maximal zulässige Vergrößerung der Seitenlänge.

**bildschirmspezifische Variablen.**

*page-medium* := `papyrus` (Bildschirmdarstellung)

Diese Variable, die im Menü Dokument → Seite → Typ gesetzt wird, regelt, wie Seiten auf dem Bildschirm dargestellt werden. Es gibt folgende Werte:

`paper`. Dokument → Seite → Typ → Papier Seitenumbrüche werden explizit auf dem Bildschirm gezeigt. Diese Darstellung ist vor allem nützlich bei der letzten Korrektur vor dem Ausdruck oder der Veröffentlichung. Sie ist allerdings langsam, da jede Veränderung den Seitenumbruch neu berechnet.

Beachten Sie auch, dass das Setzen dieser Option allein noch nicht gewährleistet, dass alles auf dem Bildschirm so erscheint wie auf dem Papier. Um sicherzustellen, dass der Bildschirm exakt mit dem Druckbild übereinstimmt, müssen Sie noch folgende Optionen setzen: Dokument → Ansicht → Seitenlayout → Seitenkopf und Seitenfuß zeigen und Dokument → Ansicht → Seitenlayout → Ränder wie auf dem Papier.

`papyrus`. Dokument → Seite → Typ → Papyrus Die Absatzbreite auf dem Bildschirm entspricht dem Druckbild. Ein Seitenumbruch wird jedoch nicht durchgeführt. Dieser Modus ist besonders geeignet, um ein Dokument zu erstellen. Es ist ein guter Kompromiss zwischen Geschwindigkeit und realistischen Zeilenumbrüchen.

`automatic`. Dokument → Seite → Typ → Automatisch Die größte Absatzbreite, die noch in das Darstellungs-Fenster passt, wird ausgewählt. Der Seitenumbruch wird ausgeschaltet. Diese Option nutzt das Darstellungs-Fenster optimal. Sie ist dann nützlich, wenn ein Dokument zur Darstellung auf dem Rechner und nicht zum Druck bestimmt ist. Beispielsweise kann diese Option gewählt werden, wenn TeXmacs als Browser oder als Oberfläche für ein Computer-Algebra-System verwendet wird.

*page-screen-width* := `10cm` (Fensterbreite)

Im „`automatic`" Modus ist dies die Breite des darstellenden Fensters.

*page-screen-height* := `10cm` (Fensterhöhe)

Im „`automatic`" Modus ist dies die Höhe des darstellenden Fensters.

*page-screen-margin* := `true` (Sind für die Bildschirmdarstellung eigene Ränder festgelegt?)

Dieses Flag zeigt an, ob vom Benutzer für die Bildschirmdarstellung Ränder speziell festgelegt wurden oder nicht, ob also die Darstellung dem Druckbild entspricht.

*page-screen-left* := `5mm`
*page-screen-right* := `5mm`
*page-screen-top* := `15mm`
*page-screen-bot* := `15mm` (Breite der Fensterränder)

Wenn *page-screen-margin* auf „`true`" gesetzt ist, dann bestimmen diese Variablen die Breiten der Fensterränder.

*page-show-hf* := `false` (Kopf- und Fußzeilen auf dem Bildschirm anzeigen?)

Dieses Flag bestimmt, ob Kopf- und Fußzeilen auf dem Bildschirm dargestellt werden sollen. Wenn es auf „`true`" gesetzt ist, müssen die Kopf- und Fußzeilen nicht immer korrekt dargestellt werden. Wenn Sie vermuten, dass die Darstellung falsch sein könnte, können Sie die Bildschirm-anzeige erneuern, indem Sie den Text im Fenster verschieben. Es sollten dann korrekte Werte erscheinen.

**Ränder für den Druck festlegen.**

Die Parameter, die die Ränder von Druckseiten bestimmen, sind in der Abbildung 13.1 schematisch dargestellt. Man kann die Breite eines Absatzes entweder durch den linken und den rechten Rand bestimmen oder die Breite der Ränder durch die Absatzbreite. Linker und rechter Seitenrand können davon abhängen, ob die Seitenzahl gerade oder ungerade ist.

*page-width-margin* := `false` (Ränder aus der Absatzbreite berechnen?)

Wenn dieses Flag auf „`false`" gesetzt ist, wird aus der linken und rechten Seitenrand sowie der Papiergröße die Absatzbreite *par-width* berechnet. Wenn dieses Flag auf „`true`" gesetzt ist, werden die linken und rechten Ränder aus der Papiergröße und der Absatzbreite berechnet, indem zusätzlich die Randverschiebungs-Parameter *page-odd-shift* und *page-even-shift* berücksichtigt werden, um unterschiedliche Ränder für gerade und ungerade Seiten zu erzeugen.

*page-width* := `auto`
*page-height* := `auto` (Seitengröße)

Als Vorgabe werden Breite und Höhe von Seiten automatisch mit Hilfe der Variablen *page-type* bestimmt. Wenn diese auf „`user`" gesetzt ist, dann kann man die Seitengröße manuell mit den Variablen *page-width* und *page-height* festlegen.

*page-odd* := `auto`
*page-even* := `auto` (linker Rand)

Wenn *page-width-margin* auf „`false`" gesetzt ist, dann spezifizieren *page-odd* und *page-even* die linken Ränder von ungeraden bzw. geraden Seiten. Wenn *page-width-margin* „`true`" ist, dann werden diese Werte aus der Papiergröße, Absatzbreite, *page-odd-shift* und *page-even-shift* berechnet. Wenn *page-odd* und *page-even* auf „`auto`" gesetzt sind, dann wird ein passender linker Rand auf Grund der Variable *page-type* erzeugt.

*page-right* := `auto` (rechter Rand)

Wenn *page-width-margin* auf „`false`" gesetzt ist, dann spezifiziert die Variable *page-right* den rechten Rand von ungeraden Seiten. Der rechte Rand ist durch die folgende Formel gegeben:

$$\textit{page-right} + \textit{page-even} - \textit{page-odd}$$

Wenn *page-width-margin* „`true`“ ist oder wenn *page-right* auf „`auto`“ gesetzt ist, dann wir der rechte Rand analog zum linken berechnet.

*page-odd-shift* := `0mm`
*page-even-shift* := `0mm` (Randverschiebung)

wenn *page-width-margin* „`true`“ ist, dann werden die linken Ränder für ungerade und gerade Seiten aus der Papiergröße, der Absatzbreite und den Randverschiebungen nach folgenden Formeln berechnet:

$$\begin{aligned} \textit{page-even} &= \frac{\textit{page-width} - \textit{par-width}}{2} + \textit{page-odd-shift} \\ \textit{page-odd} &= \frac{\textit{page-width} - \textit{par-width}}{2} + \textit{page-even-shift} \end{aligned}$$

Die rechte Rand wird so gewählt, dass die Absatzbreite und die linken und rechten Ränder zusammen die Seitenbreite ergeben.

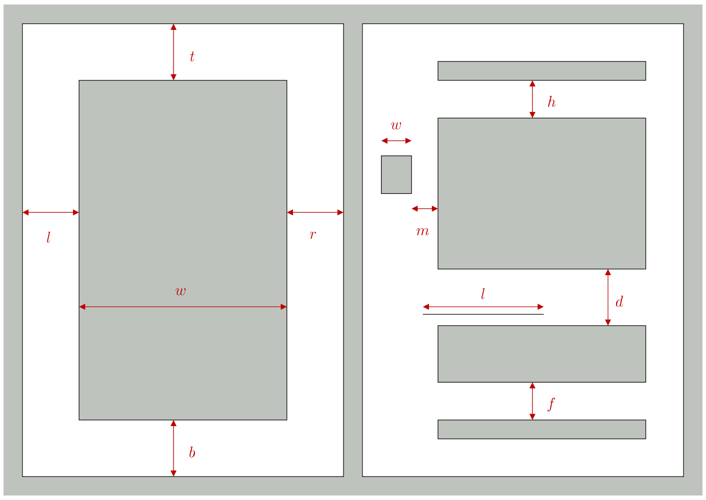



**Abbildung 13.1.** Schematische Darstellung des Seitenlayouts. Links entsprechen die Parameter $l$, $r$, $t$ und $b$ dem linken, rechten, oberen und unteren Rand. $w$ ist die Absatzbreite. Rechts entsprechen $h$, $f$, $d$ und $m$ den Abständen für Kopfzeile, Fußzeile, Fußnote und Randnotiz. $l$ ist die Länge des Fußnotentrennstrichs.

### Kopf- und Fußzeilen, Fußnoten, Randnotizen.

*page-odd-header* :=
*page-odd-footer* :=
*page-even-header* :=
*page-even-footer* := (Kopf- und Fußzeilentexte)

Diese Variablen enthalten die Texte der Kopf- und Fußzeilen für gerade und ungerade Seiten.

*page-head-sep* := `8mm`
*page-foot-set* := `8mm` (Abstand von Kopf- und Fußzeilen zum Text)

Diese Parameter bestimmen den Abstand zwischen dem eigentlichen Text und den Kopf- und Fußzeilen. Sie entsprechen den Größen $h$ und $f$ des rechten Teilbildes von Abbildung 13.1.

*page-fnote-sep* := `1.0fn` (Abstand von Fußnote und Text)

Dies ist der Abstand zwischen dem eigentlichen Text und den Fußnoten also $d$ in Abbildung 13.1.

*page-fnote-barlen* := `7.5fn` (Länge des Fußnotentrennstrichs)

Die Länge des Fußnotentrennstrichs.

*page-float-sep* := `1.5fn` (Der Abstand zwischen Text und beweglichen Objekten)

Der Abstand zwischen Text und beweglichen Objekten.

*page-mnote-sep* := `5mm` (Der Abstand von Randnotizen und Text)

Der Abstand von Randnotizen und dem eigentlichen Text (noch nicht implementiert!).

*page-mnote-width* := `15mm` (Die Breite von Randnotizen)

Die Breite von Randnotizen (noch nicht implementiert!).

## 13.5. TABELLEN-LAYOUT

Die Kontextvariablen für Tabellen können in zwei Klassen eingeteilt werden. Solche die die ganze Tabelle betreffen, die beginnen mit dem Präfix *table-*, und solche die nur eine einzelne Zelle betreffen, diese beginnen mit dem Präfix *cell-*. Während normale Kontextvariablen mit den Konstrukten assign und with gesetzt werden, benutzt man dazu bei den Tabellenvariablen das Konstrukt tformat. Mit diesem Konstrukt können bestimmte Vorgaben auf alle rechteckigen Unter-Tabellen übertragen werden. Das gilt vor allem für Reihen und Spalten. Mehr Informationen finden Sie hier für die Konstrukte twith und cwith.

**Layout der ganzen Tabelle.**

*table-width* :=
*table-height* := (Minimale Tabellendimensionen)

Diese Parameter geben einen Hinweis auf die ungefähren Dimensionen einer Tabelle. Die Parameter *table-hmode* und *table-vmode* bestimmen, wie sie auszuwerten sind.

*table-hmode* :=
*table-vmode* := (Berechnungsweise der Tabellendimensionen)

*table-hmode* und *table-vmode* werden zur Zeit ignoriert. Momentan werden *table-width* und *table-height* als minimale Tabellendimensionen interpretiert. In Zukunft sollen *table-hmode* und *table-vmode* aber steuern wie genau *table-width* und *table-height* auszuwerten sind.

*table-halign* := `l`
*table-valign* := `f` (Einpassung in den Text)

Die vorstehenden Parameter bestimmen, wie die Tabelle in den umgebenden Text eingefügt werden soll. Mögliche Werte für *table-halign* sind `l` (links), `c` (zentriert) und `r` (rechts); mögliche Werte für *table-valign* sind `t` (ganz oben), `f` (auf Bruchlinienhöhe), `c` (zentriert) und `b` (ganz unten).

Außer den oben genannten Werten sind noch weitere möglich, die die Tabelle in Bezug zu der Basislinie bestimmter Zellen positioniert. Für *table-halign* sind dies `L` (nach der linken Spalte ausrichten), `C` (nach der mittleren Spalte ausrichten), `R` (nach der rechten Spalte ausrichten) und `O` (nach der Spalte der privilegierten Zelle *table-col-origin* ausrichten , s.u.). Entsprechend kann *table-valign* die zusätzlichen Werte `T` (nach der obersten Zeile ausrichten), `C` (Nach der Mittelzeile ausrichten), `B` (nach der untersten Zeile ausrichten) und `O` (nach der Zeile der privilegierten Zelle *table-row-origin* ausrichten, s.u.).

*table-row-origin* := `0`
*table-col-origin* := `0` (privilegierte Zelle)

Die Tabellen-Koordinaten einer privilegierten Zelle, origin cell, die zur Ausrichtung im umgebenden Text dienen kann (s.o.).

*table-lsep* := `0fn`
*table-rsep* := `0fn`
*table-bsep* := `0fn`
*table-tsep* := `0fn` (Padding um eine Tabelle)

Der kontextabhängige Leerraum, Padding, um eine Tabelle (zusätzlich zum Padding um die einzelnen Zellen).

*table-lborder* := `0ln`
*table-rborder* := `0ln`
*table-bborder* := `0ln`
*table-tborder* := `0ln` (Tabellenränder)

Breite der Tabellenränder zusätzlich zur Breite der Ränder um die einzelnen Zellen.

*table-hyphen* := `n` (Seitenumbruch erlaubt?)

Boolesche Variable, die angibt ob ein Seitenumbruch innerhalb der Tabelle erlaubt ist. Wenn *table-hyphen* auf y gesetzt ist, kann ein Seitenumbruch erfolgen aber nur dann, wenn

1. Die Tabelle innerhalb des selben Absatzes nicht von weiteren Kontexten oder speziellen Hervorhebungen gesteuert wird.

2. Die Zeilen zwischen denen der Seitenumbruch erfolgt, haben kein Gitter.

Ein Beispiel, dass für eine Tabelle mit erlaubtem Seitenumbruch ist `eqnarray*`.

*table-min-rows* :=
*table-min-cols* :=
*table-max-rows* :=
*table-max-cols* := (Minimum bzw. Maximum der Tabellendimensionen)

Man kann die minimale und die maximale Anzahl sowohl von Zeilen wie von Spalten festlegen. Das begrenzt die Möglichkeiten des Editors beim Einfügen und Löschen von Zeilen und Spalten. Dies ist besonders nützlich bei Tabellenmakros. So sind für `eqnarray*` *table-min-columns* und *table-max-columns* fest auf 3 gesetzt.

**Layout individueller Zellen.**

*cell-background* := (Hintergrundfarbe)

Die Hintergrundfarbe einer Zelle.

*cell-width* :=
*cell-height* := (Ungefähre Zelldimensionen)

Diese Variablen sind Vorgaben für die Festlegung von Zellhöhe und -Breite, die außerdem von den Variablen *cell-hmode* und *cell-vmode* sowie *cell-hpart* und *cell-vpart* abhängt.

*cell-hpart* :=
*cell-vpart* := (zusätzlicher Paddinganteil einer Zelle)

Wenn die Summe $s$ der Breiten aller Spalten kleiner ist als die Breite der Tabelle $w$, dann muss man vorgeben, was mit dem leeren Platz geschehen soll. Der Parameter *cell-hpart* legt fest wie viel Platz eine einzelne Zelle von dem leeren Platz bekommen soll. Der horizontale Anteil einer ganzen Spalte ist das Maximum der Anteile aller Zellen der Spalte. $p_i$ sei der so definierte Anteil jeder Spalte $i$, $(i \in \{1, ..., n\})$. Der verbleibende Anteil wird dann verteilt, in dem jeder Spalte Dann wird jeder Spalte $i$ der Anteil $p_i (w - s)/(p_1 + \cdots + p_n)$ zugeteilt wird. Ganz entsprechend wird in jeder Zeile verfahren.

*cell-hmode* := `exact`
*cell-vmode* := `exact` (Berechnungsweise der Zelldimensionen)

Diese Parameter legen fest, auf welche Weise Breite und Höhe der Zelle zu berechnen sind. Wenn *cell-hmode* auf `exact` gesetzt ist, dann ist die Breite *cell-width*. Wenn *cell-hmode* auf `min` oder auf `max` gesetzt ist, dann ist die wirkliche Breite das Minimum von *cell-width* und der Breite des Zellinhalts bzw. das entsprechende Maximum. Die Festlegung der Zellhöhe geschieht analog.

*cell-halign* := `l`
*cell-valign* := `B` (Zellausrichtung)

Diese Parameter bestimmen die Zellausrichtung. Mögliche Werte für *cell-halign* sind `l` (links), `c` (zentriert), `r` (rechts), `.` (Dezimalpunkt), `,` (Dezimalkomma) und `R` (vertikale Basislinie). Mögliche Werte für *cell-valign* sind `t` (ganz oben), `c` (zentriert), `b` (ganz unten) und `B` (Basislinie).

*cell-lsep* := `0fn`
*cell-rsep* := `0fn`
*cell-bsep* := `0fn`
*cell-tsep* := `0fn` (Zellpadding)

Padding einer Zelle (links, rechts, ganz unten und ganz oben).

*cell-lborder* := `0ln`
*cell-rborder* := `0ln`
*cell-bborder* := `0ln`
*cell-tborder* := `0ln` (Zellgitter)

Die Dicke der Linien, die als Ränder die Zelle begrenzen (links, rechts, ganz unten und ganz oben). Die Breite der Randlinie zwischen den Zellen $T_{i,j}$ und $T_{i,j+1}$ an den Positionen $(i, j)$ und $(i, j+1)$ ist das Maximum der rechten Randlinie von $T_{i,j}$ und der linken von $T_{i,j+1}$. Analog werden die oberen und unteren Randlinien-Breiten bestimmt.

*cell-vcorrect* := `a` (vertikale Korrektur von Text)

Wie bereits oben beschrieben können die Dimensionen und die Ausrichtung der Zellen vom Inhalt abhängen. Wenn die Zellen Boxen enthalten dann kann die Höhe der Boxen von deren Inhalt abhängen, denn z.B. hat der Buchstabe „k“ eine größere Oberlänge als „y“, das dafür eine größere Unterlänge besitzt. Solche Unterschiede führen manchmal zu unerwünschten Ungleichmäßigkeiten im Satzbild. Die vertikale Zellenkorrektur *cell-vcorrect* verbessert die Gleich-mäßigkeit von Text in einer einzigen Schriftart, indem Boxen nach oben oder unten verschoben werden in einer Weise, die nur von der gewählten Schriftart abhängt. Mögliche Werte für *cell-vcorrect* sind `n` (keine Höhenkorrektur), `b` (vertikale Korrektur unten), `t` (vertikale Korrektur oben), `a` (vertical correction of bottom and the top).

*cell-hyphen* := `n` (Trennung innerhalb einer Zelle erlaubt?)

Normalerweise wird innerhalb von Zellen nicht getrennt. Durch setzen der Option Tabelle → Spezielle Zelleneigenschaften → Trennung → Multi-Absatz wird die Zelle für die Aufnahme mehrerer Absätze eingerichtet. In diesem Fall kann mit *cell-hyphen* die Art und Weise des Zeilenumbruch bzw. der Trennung bestimmt werden. Mögliche Werte sind `n` (kein Zeilenumbruch), `b` (enable line breaking and align at the bottom), `c` (enable line breaking and align at the center) und `t` (Zeilenumbruch und Ausrichtung an der obersten Zeile).

*cell-row-span* := `1`
*cell-col-span* := `1` (Mehrfachzellen)

Man kann Zellen so konfigurieren, das diese den Platz von mehreren Zellen belegen, die entweder rechts neben ihnen oder unter ihnen liegen. Mit solchen Mehrfachzellen können z.B. Überschriften, die sich über mehrere Zellen erstrecken und sie zusammenfassen, in die Tabelle eingefügt werden. *cell-row-span* und *cell-col-span* legen fest, wie breit bzw. wie hoch eine solche Zelle in Anzahl der Nachbarzellen ist.

*cell-decoration* := (Dekorierende Tabelle für eine Zelle)

Diese Kontextvariable kann eine „dekorierende” Zelle enthalten. Diese fügt der die ursprünglichen Tabelle weitere Spalten und Zeilen außen hinzu. Das Konstrukt tmarker definiert den Ort der ursprünglichen Zelle. Ihre Umgebung wird in der vergrößerten Tabelle mit Dekorationen aufgefüllt. Zelldekorationen werden kaum gebraucht und verschwinden vielleicht in zukünftigen Versionen von TeXmacs.

*cell-orientation* := `portrait` (Zellorientierung)

Bisher ist nur das Längsformat, `portrait`, implementiert.

*cell-row-nr* := `1`
*cell-col-nr* := `1` (Aktuelle Zellposition)

Das ist noch nicht implementiert. In Zukunft sollen diese Variablen die aktuelle Zellposition während des Schriftsetzens enthalten.

## 13.6. Quellcode editieren

Die verschiedenen Darstellungsmöglichkeiten von Quellcode-Bäumen werden hier detailliert geschildert. An dieser Stelle werden nur die zugehörigen Kontextvariablen beschrieben.

*src-style* := `angular` (Darstellungsstil für Quellcode)

Der Basis-Stil, wie er im Menü Dokument → Ansicht → Stil eingestellt werden kann. Mögliche Werte sind `angular` (spitze Klammern), `scheme` (Scheme), `functional`, (functional) und `latex` (latex).

*src-special* := `normal` (Darstellung spezieller Konstrukte)

Wie spezielle Konstrukte wie `concat`, `document`, `compound`, usw. dargestellt werden sollen. Diese kann im Menü Dokument → Ansicht → Speziell eingestellt werden. Mögliche Werte sind `raw` (keine), `format` (Formatierung), `normal` und `maximal`.

*src-compact* := `normal` (Verdichtungsgrad)

Wie verdichtet sollen Quellcode-Konstrukte dargestellt werden. Das kann im Menü Dokument → Ansicht → Verdichtungsgrad eingestellt werden. Mögliche Werte sind `none` (Minimal), `inline` (nur Zeilenbefehle), `normal`, `inline tags` (Zeilenargumente) und `all` (maximal).

*src-close* := `compact` (Darstellung der Stoptags)

Die Darstellung von Stoptags in mehrzeiligen Befehlen kann im Menü Dokument → Ansicht → Befehlsabschluss eingestellt werden. Mögliche Werte sind `repeat` (Rekursiv), `long` (Gespreizt), `compact` (Dicht) und `minimal` (minimal).

## 13.7. Weitere Kontextvariablen

Die folgenden Kontextvariablen sind in erster Linie für den internen Gebrauch gedacht:

*save-aux* := `true` (Zusatzinformationen speichern?)

Diese boolesche Variable bestimmt, ob die Zusatzinformationen mit dem Dokument gespeichert werden sollen.

*sfactor* := `5` (Verkleinerungsfaktor)

Der Verkleinerungsfaktor der bei der Darstellung benutzt werden soll.

*par-no-first* := `false` (Erstzeileneinzug im folgenden Absatz verhindern?)

Diese boolesche Variable bestimmt, ob die erste Zeile des folgenden Absatzes eingezogen wird.

*cell-format* (aktuelles Zellenformat)

Diese Variable speichert während des Schriftsetzens von Tabellen die Informationen, die die aktuelle Zelle betreffen.

*atom-decorations*
*line-decorations*
*page-decorations*
*xoff-decorations*
*yoff-decorations* (Hilfsvariable für Dekorationen)

Diese Variablen speichern zusätzliche Informationen während des Schriftsetzens von Dekorationen.

# Kapitel 14

## Fundamentale TeXmacs-Konstrukte

In diesem Kapitel beschreiben wir die Konstrukte, die in dem TeXmacs-Editor eingebaut sind und die für normale Dokumente gedacht sind. Zusätzliche Primär-Konstrukte, die zur Erzeugung von Stil-Definitionen dienen, werden in einem besonderen Kapitel beschrieben.

### 14.1. Basis-Konstrukte

⟨document|*par-1*|···|*par-n*⟩ (Vertikale Folge von Absätzen)

Dieses Konstrukt wird für Folgen von zusammengehörigen Absätzen benutzt. Ein einfaches Textdokument besteht in der Regel aus einer Folge von Absätzen. Z.B.

> Ein einfaches Dokument.
>
> Das Dokument besteht aus mehreren Absätzen. Da zu lang, muss ein Seitenumbruch durchgeführt werden. Lange Worte am Zeilenende werden getrennt.

wird intern als document mit zwei Unter-Bäumen gespeichert:

> ⟨document|
> Ein einfaches Dokument.|
> Das Dokument besteht aus mehreren Absätzen. Da zu lang, muß ein Seitenumbruch durchgeführt werden. Lange Worte am Zeilenende werden getrennt.⟩

Auf dem Bildschirm und im Druck werden auf einander folgende Absätze oft durch Leerraum oder durch Erstzeileneinzüge markiert. Die Wurzel eines TeXmacs-Dokuments ist normalerweise ein document-Knoten.

Das document-Konstrukt wird häufig für Inhalte benutzt, die aus mehreren Absätzen bestehen, wenn sie innerhalb von anderen Konstrukten wie z.B. Listen oder Theoremen vorkommen. Kontexte, die ein document-Konstrukt benötigen, heißen „Blockkontext".

⟨paragraph|*unit-1*|···|*unit-n*⟩ (Vertikale Folge von Absatzeinheiten)

Dieses noch nicht implementierte Konstrukt ist eine Variante von document. Während ein Dokument eine Folge von Absätzen ist, ist ein paragraph, ein Absatz, eine Folge „Absatzeinheiten", sprich einzelnen Zeilen. Auch sind eigenständige Formeln Absatzeinheiten in einem größeren Absatz.

⟨concat|*item-1*|···|*item-n*⟩ (Horizontale Folge von Zeileninhalt)

Dieses Konstrukt definiert eine horizontale Folge kurzen Textstücken oder von Konstrukten, die kurze Textstücke speziell darstellen, Zeileninhalt. Z.B.

Dies ist Text *hervorgehoben mit der Form Italic*.

wird intern gespeichert als:

⟨concat|Dies ist Text|⟨em| hervorgehoben mit der Form Italic⟩|.⟩

Das concat-Konstrukt wird gebraucht, um Konstrukte in einen Baum einzufügen, die mehrere Parameter haben. Das vorstehende Textfragment soll z.B. in einen Text mit mehreren Absätzen eingefügt werden:

Mehrere Absätze.

Dies ist Text *hervorgehoben mit der Form Italic*.

In diesem Beispiel benötigen wir das concat-Konstrukt, um klarzustellen, dass „Dies ist Text *hervorgehoben mit der Form Italic*." ein einzelner Absatz ist.

⟨document|
    Mehrere Absätze.|
    ⟨concat|Dies ist Text |⟨em| hervorgehoben mit der Form Italic⟩|.⟩⟩

Beachten Sie bitte, dass Block-Konstrukte wie document Zeilen-Konstrukte wie concat als Kinder haben dürfen aber nicht umgekehrt. Um Zeileninhalt vor oder hinter Blockinhalt zu platzieren, muss man den Konstrukt surround benutzen (s.u.).

⟨surround|*left*|*right*|*body*⟩ (Blockinhalt mit Zeileninhalt umgeben)

Obwohl es in TeXmacs nicht möglich ist, Blockinhalt in horizontalen Aufreihungen zu benutzen, kann es manchmal nützlich sein, zusätzlichen Zeileninhalt vor oder hinter Blockinhalten zu platzieren. Dazu dient das surround-Konstrukt, der Zeileninhalt *left* und Zeileninhalt *right* dem Blockinhalt *body* hinzufügt. Beispielsweise produziert

⟨surround|$\lightning$ ||
    ⟨theorem|
        Gegeben $P\in\mathbb{T}\{F\}$ und $f<g\in\mathbb{T}$ mit $P(f)\,P(g)<0$, dann existiert ein $h\in\mathbb{T}$ mit $P(h)=0$.⟩⟩

das folgende

$\lightning$ SATZ 14.1. *Gegeben $P\in\mathbb{T}\{F\}$ und $f<g\in\mathbb{T}$ mit $P(f)\,P(g)<0$, dann existiert ein $h\in\mathbb{T}$ mit $P(h)=0$.*

Gewöhnlich wird surround in Stildefinitionen gebraucht. Gelegentlich ist es in normalen Text auch recht nützlich.

## 14.2. FORMATIER-KONSTRUKTE

### 14.2.1. Leerraum-Konstrukte

⟨vspace|*len*⟩

⟨vspace|*len*|*min*|*max*⟩ (Vertikaler Abstand danach)

Dieses Konstrukt fügt einen vertikalen variablen Abstand nach dem aktuellen Absatz ein. Alle Operanden müssen Längeneinheiten sein. Das *len*-Argument spezifiziert die Vorgabelänge und *min* bzw. *max* die untere und obere Grenze der Dehnbarkeit. Werden *min* und *max* nicht angegeben, dann wir aus *len* nach impliziten Vorgaben eine obere und untere Grenze berechnet.

Beachten Sie, dass die Operanden nicht evaluiert werden. Sie müssen daher Zeichenketten sein.

⟨vspace*|*len*⟩
⟨vspace*|*len*|*min*|*max*⟩ (Vertikaler Abstand davor)

Dieses Konstrukt ähnelt vspace bis auf die Tatsache, dass der vertikale Abstand vor dem aktuellen Absatz eingefügt wird. Der daraus resultierende wirkliche Abstand ist das Maximum der mit vspace und vspace* spezifizierten Abstände zweier auf einander folgender Absätze - nicht die Summe.

⟨space|*len*⟩
⟨space|*len*|*bot*|*top*⟩ (Fixer horizontaler Abstand)

Dieses Konstrukt fügt eine leere Box ein, deren Länge *len* ist und deren Untergrenze bzw. Obergrenze um *bot* und *top* oberhalb der Basislinie liegen.

Wenn *bot* und *top* nicht spezifiziert werden, dann wird eine leere Box eingefügt, deren Untergrenze die Basislinie ist und deren Obergrenze auf der Höhe des Kleinbuchstabens x in der aktuellen Schriftart und -größe ist.

Beachten Sie, dass die Operanden nicht evaluiert werden. Sie müssen daher Zeichenketten sein.

⟨hspace|*len*⟩
⟨hspace|*len*|*min*|*max*⟩ (Variabler horizontaler Abstand)

Dieses Konstrukt fügt einen variablen horizontalen Abstand der Nennlänge *len* ein. *len* muss eine Längeneinheit sein. *min* und *max* spezifizieren Ober- und Untergrenzen der Dehnbarkeit. Wenn *min* und *max* nicht angegeben werden, dann werden sie aus *len* nach impliziten Vorgaben berechnet.

Beachten Sie, dass die Operanden nicht evaluiert werden. Sie müssen daher Zeichenketten sein.

⟨htab|*min*⟩
⟨htab|*min*|*weight*⟩ (Horizontaler Sprung)

Sprünge sind horizontaler Leerraum, der so gedehnt wird, dass er den ganzen verfügbaren horizontalen Raum einnimmt. Wenn ein Absatz umgebrochen wird, also in mehrere sichtbare Zeilen zerlegt wird, dann werden nur Sprünge in der letzten Zeile gedehnt.

Ein Sprung hat eine *Minimallänge* (min)und ein *Gewicht* (weight). Wenn das Gewicht 0 ist, dann handelt sich um einen schwachen Sprung, sonst um einen starken. Wenn eine Zeile sowohl starke wie schwache Sprünge enthält, dann werden nur die starken gedehnt.

Der Bruchteil des vorhandenen horizontalen Leerraums, der jedem starken Sprung zugeteilt wird, ist seinem Gewicht proportional. Wenn nur schwache Sprünge existieren, erhält jeder den gleichen Anteil.

⟨htab | *min*⟩ fügt einen starken Sprung der Minimallänge *min* und Gewicht 1 ein. Der *min* -Operand muss eine Längeneinheit sein.

⟨htab|*min*|*weight*⟩ spezifiziert ein Gewicht, das entweder eine positive Dezimalzahl ist oder eine von den unten angegebenen Werten.

⟨htab|*min*|first⟩ fügt einen *hinten schwachen* Sprung ein, signifikant ist nur der erste in einem Absatz.

⟨htab | *min* | last⟩ fügt einen *vorne schwachen* Absatz ein, nur der letzte in einem Absatz ist signifikant.

Operanden werden nicht evaluiert, sie müssen daher Zeichenketten sein.

Schwache Sprünge sind in Stildefinitionen sehr nützlich. Beispielsweise werden hinten schwache Sprünge benutzt, damit Listen sich über eine vollen Absatz ausdehnen können. Das ist nötig, damit in verschachtelten Listen die vertikalen Abstands-Konstrukte richtig funktionieren. In normalen Dokumenten werden Sprünge oft dazu benutzt, um ein Textstück auf die rechte Seite einer Seite zu platzieren und ein anderes auf die linke Seite.

### 14.2.2. Zeilenumbruch-Konstrukte

Ein einfaches Dokument ist eine Sequenz von *logischen Absätzen*, eine für jeden Unter-Baum eines document- oder paragraph-Knoten. Absätze, die den verfügbaren horizontalen Raum überschreiten, müssen in *physikalische Zeilen* umgebrochen werden. Umgebrochene Zeilen werden als Vorgabe im Blocksatz gesetzt, dabei kann horizontaler Leerraum gedehnt oder gestaucht werden, um ein gutes Schriftbild zu erzeugen.

⟨new-line⟩ (Beginn eines neuen Absatzes)

Dieses Konstrukt ist überholt. Es dient dazu, einen logischen Absatz in mehrere logische Absätze zu unterteilen, ohne explizit neue Unter-Bäume für alle Absätze zu erzeugen.

Wir erinnern daran, dass logische Absätze wichtige Strukturen im Schriftsatz-Prozess sind. Viele Konstrukte und Kontextvariablen (Vertikaler Abstand, Absatz-Stil, Zeileneinzug, Seitenumbruch usw.) operieren mit ganzen Absätzen oder an den Grenzen des umgebenden Absatzes.

⟨next-line⟩ (Beginn einer neuen Zeile)

Dieses Konstrukt wird überholt sein, wenn das paragraph-(Absatz)-Konstrukt korrekt implementiert ist. Sein Gebrauch ähnelt new-line, nur dass hier eine neue logische Absatzeinheit (Zeile) anstelle eines logischen Absatzes erzeugt wird.

Zur Zeit kann next-line benutzt werden, um einen Zeilenumbruch zu erzwingen, bei dem die Zeile vor dem Umbruch genau an dieser Stelle umgebrochen wird und auch nicht in Blocksatz gesetzt wird.

⟨line-break⟩ (Bedingter Trennstrich)

Es wird unsichtbarer Abstand mit Breite 0 und eine Trennstrafe 0. Die verschiedenen Wort-Trennungen besitzen unterschiedliche Trennstrafen. Der Trennungs-Algorithmus sucht nach einem Satz von Trennpunkten, indem er die Summe der Trennstrafen minimiert. Deshalb ist die Trennung an der Stelle des line-break wahrscheinlicher als irgendwo in seiner Nähe.

Im Gegensatz zu next-line erzwingt line-break keinen Umbruch. Es ist ein bedingter Trennstrich.

⟨no-break⟩ (Trennung an dieser Stelle verhindern)

Setze einen Trennpunkt mit einer unendlich hohen Trennstrafe, verbiete also die Trennung an dieser Stelle. Das ist manchmal sehr nützlich. Eine andere Möglichkeit unerwünschte Trennungen zu verhindern, ist das group-Konstrukt.

### 14.2.3. Einzüge, Grundformen

Gewöhnlich wird eine von den folgenden beiden Weisen genutzt, um den Übergang von einem Absatz zum nächsten zu markieren: Die Absätze werden durch einen kleinen zusätzlichen Abstand von einander getrennt oder die erste Zeile des neuen Absatzes wird eingezogen. Der Einzug kann explizit mit den folgenden Marken gesteuert werden: no-indent, yes-indent, no-indent* und yes-indent*. Die Grundformen no-indent und yes-indent werden auf den aktuellen Absatz angewandt, während no-indent* und yes-indent* erst im folgenden Absatz wirksam werden.

⟨no-indent⟩
⟨yes-indent⟩

Aktiviere bzw. deaktiviere den Erstzeileneinzug im aktuellen Absatz. Z.B. erzeugt der folgende Code

> ⟨no-indent⟩Dies ist eine langer Absatz, der zeigt wie der Erstzeilenabzug durch no-indent abgeschaltet wird.
> ⟨yes-indent⟩Dies ist eine langer Absatz, der zeigt wie der Erstzeilenabzug durch yes-indent eingeschaltet wird.

gewöhnlich

> Dies ist eine langer Absatz, der zeigt wie der Erstzeilenabzug durch no-indent abgeschaltet wird.
> Dies ist eine langer Absatz, der zeigt wie der Erstzeilenabzug durch yes-indent eingeschaltet wird.

⟨no-indent*⟩
⟨yes-indent*⟩

Aktiviere bzw. deaktiviere den Erstzeileneinzug im nachfolgenden Absatz. Beispielsweise erzeugt

> Ein erster Absatz.⟨yes-indent*⟩
> Ein zweiter Absatz.

gewöhnlich

> Ein erster Absatz.
>
> Ein zweiter Absatz.

Es bleibt anzumerken, dass no-indent und yes-indent Vorrang vor no-indent* und yes-indent*-Konstrukten im vorgängigen Absatz haben.

Derzeit werden die no-indent* und yes-indent* -Anweisungen vor allem benutzt, um den Einzug nach Abschnitt-Überschriften oder Absätzen mit speziellen Layout zu steuern ( z.B. equation). Es ist geplant, dass zukünftig Abschnitt-Marken den Abschnittsrumpff als Argument erhalten. Wenn das geschehen ist, wird die Absatz-Marke (paragraph tag) korrekt implementiert und no-indent* sowie yes-indent* werden überflüssig.

### 14.2.4. Seitenumbruch-Konstrukte

Ein Dokument wird in ähnlicher Weise in Seiten umgebrochen wie die Absätze in Zeilen. Der Seitenumbruch-Algorithmus erzeugt *Seitenfüllung*, das ähnelt dem Blocksatz, er versucht Seiten gleichmäßig mit Text zu versehen, so dass der Text bis zum Seitenende läuft. Er versucht auch sogenannte *Weisenkinder* und *Witwen* zu vermeiden. Das sind ein oder zwei Zeilen, die vom Rest ihres Absatzes durch einen Seitenumbruch getrennt wurden. Wenn es keine bessere Lösung gibt, können diese dennoch entstehen.

⟨no-page-break⟩ (einen automatischen Seitenumbruch nach dieser Zeile verhindern)

Dies verhindert einen automatischen Seitenumbruch direkt hinter dieser Zeile. Dies setzt die Seitenumbruch-Strafe für diese Zeile auf unendlich, ganz ähnlich wie no-break.

Verbotene Seitenumbruch-Punkte werden durch „new page" und „page break" aufgehoben.

⟨no-page-break*⟩ (einen automatischen Seitenumbruch vor dieser Zeile verhindern)

Ähnlich wie no-page-break. Es setzt aber die Strafe in der vorgehenden Zeile.

⟨new-page⟩ (beginne eine neue Seite nach dieser Zeile)

Sorgt dafür, dass die nächste Zeile auf einer neuen Seite erscheint, ohne das die Seite gefüllt wird. Der Seitenumbruch-Algorithmus versucht also nicht die aktuelle Zeile ganz unten auf der Seite zu setzen.

⟨new-page*⟩ (beginne eine neue Seite vor dieser Zeile)

Ähnlich zu new-page. Aber der Seitenumbruch erfolgt vor der aktuellen Zeile, so dass die aktuelle Zeile auf der neuen Seite erscheint. Dies ist für Kapitel-Überschriften geeignet.

⟨page-break⟩ (einen Seitenumbruch nach dieser Zeile erzwingen)

Dies erzwingt einen Zeilenumbruch hinter der aktuellen Zeile. Im Gegensatz zu new-page wird die Seite gefüllt. Seitenumbruch-Algorithmus versucht die aktuelle Zeile ganz unten auf die Seite zu setzen.

Das sollte man nur benutzen, um den automatischen Seitenumbruch vorsichtig anzupassen. Idealerweise sollte das eigentlich wie line-break wie ein Hinweis arbeiten, da es aber als Befehl implementiert ist, sollte man es mit äußerster Vorsicht einsetzen.

⟨page-break*⟩ (einen Seitenumbruch nach dieser Zeile erzwingen)

Ähnlich wie page-break, bezieht aber auf die vorgängige Zeile.

Wenn mehrere „new page" und „page break" Befehle sich auf die selbe Stelle beziehen, dann wird nur der erste berücksichtigt. Jedes new-page oder page-break nach dem ersten wird ignoriert. Jedes new-page oder page-break in einer Zeile wird new-page* oder page-break* in der folgenden Zeile vorgezogen. Jedes new-page* oder page-break* nach dem ersten Auftreten wird ignoriert.

### 14.2.5. Konstrukte für Boxen

⟨move|*content*|*delta-x*|*delta-y*⟩ (Position verschieben)

Dieses Konstrukt verschiebt die Box mit dem Inhalt *content* um *delta-x* nach rechts und *delta-y* nach oben.

⟨resize|*content*|*left-lim*|*bot-lim*|*right-lim*|*top-lim*⟩ (Größenanpassung)

Verändere die Größe der Box mit dem Inhalt *content* zu neuen Grenzen links, unten, oben und oben: *left-lim*, *bot-lim*, *right-lim* und *top-lim*. Die Grenzen können leere Zeichenketten sein, in diesem Fall werden die alten Grenzen verwendet, absolute Koordinaten oder Grenzen, die aus den alten berechnet werden.

In diesem Fall sollten die Grenzen die Form `<pos><op><len>` haben. Das erste Zeichen `<pos>` verweist auf die Position der ursprünglichen Box und ist entweder `l` (links), `b` (unten), `c` (Mitte), `r` (rechts) oder `t` (oben). Der zweite Buchstabe `<op>` verweist auf die Operation, die an dieser Position durchgeführt werden soll, und der letzte Buchstabe auf eine Längenangabe, die auf die Position mit der gegebenen Operation angewendet werden soll. Mögliche Operationen sind `+`, `-`, `[` und `]`. Die Klammern `[` und `]` stehen für „Minimum" und „Maximum". Beispielsweise verbreitert

(⟨resize|Hopsa|l-5mm||r+5mm||⟩)

die Box „Hopsa" um `5mm` an jeder Seite:

( Hopsa )

⟨if*|*condition*|*content*⟩ (bedingtes Erscheinen einer Box)

Die Box mit dem Inhalt *content* wird normal angezeigt, wenn *condition* wahr ist, sonst als Leerraum. Dieses Konstrukt wird vor allem zur Definition des phantom-Makro verwendet. Beispielsweise wird der Leerraum „           " erzeugt mit ⟨if*|false|phantom⟩.

⟨repeat|*content*|*pattern*⟩ (Überschreibe)

Dieses Konstrukt kann dazu benutzt werden, den Inhalt mit *content* mit einen bestimmten Schriftzug *pattern* zu überschreiben. Beispielsweise erzeugt der Code

⟨assign|*wipe-out*|⟨macro|*x*|⟨repeat|*x*|⟨with|*color*|red|/⟩⟩⟩⟩

⟨wipe-out|obsolete⟩ das Folgende ~~obsolete~~. Das repeat-Konstrukt kann außerdem dazu benutzt werden die aktuelle Zeile mit einem bestimmten Inhalt zu füllen, wie z.B. die Punkte in Inhaltsverzeichnissen.

⟨datoms|*foo*|*content*⟩
⟨dlines|*foo*|*content*⟩
⟨dpages|*foo*|*content*⟩ (Dekorationen)

Diese Konstrukte sind dafür gedacht, Zeilen eines Absatzes, Zeilen eines Dokuments bzw. Zeilen einer Seite nachträglich zu dekorieren. Derzeit sind nur Dekorationen für atomare Zeilenelemente eines Absatzes implementiert.

Das erste Argument *foo* ist ein Makro, dass auf alle Boxen in einer Zeile angewendet wird und das zweite Argument *content* ist der Teil des Absatzes auf die die Dekoration angewendet wird. Beispiel kann

```
⟨datoms|
    ⟨macro|x|[x]⟩|
    body⟩
```

benutzt werden, um die Boxen in einem Absatz zu zeigen:

| | |
|---|---|
| Dies ist ein genügend langer Absatz. | Dies ist ein genügend langer Absatz. |
| Dies ist ein genügend langer Absatz. | Dies ist ein genügend langer Absatz. |
| Dies ist ein genügend langer Absatz. | Dies ist ein genügend langer Absatz. |
| Dies ist ein genügend langer Absatz. | |

Wenn man dies in Kombination mit repeat benutzt, kann man die punktierten Linien der Inhaltsverzeichnisse reproduzieren

```
⟨assign|
    toc-dots|
    ⟨macro|
        ⟨datoms|
            ⟨macro|x|⟨repeat|x|⟨space|0.2fn⟩.⟨space|0.2fn⟩⟩⟩|
            ⟨htab|5mm⟩⟩⟩⟩
```

Man beachte, das datoms sehr empfindlich ist, da das *foo*-Makro keinen Zugriff auf den Kontext hat, in dem *content* gesetzt wird.

## 14.3. Mathematik-Konstrukte

⟨left|*large-delimiter*⟩
⟨left|*large-delimiter*|*size*⟩
⟨left|*large-delimiter*|*bottom*|*top*⟩
⟨mid|*large-delimiter*|⋯⟩
⟨right|*large-delimiter*|⋯⟩ (Große Klammern)

Diese Konstrukte erzeugen große Klammern, wie z.B.:

$$\left\langle \frac{1}{a_1} \middle| \frac{1}{a_2} \middle| \cdots \middle| \frac{1}{a_n} \right\rangle.$$

Zu einander passende rechte und linke Klammern werden in ihrer Größe automatisch dem Inhalt angepasst. Dazwischen kann sich eine beliebige Anzahl rechter und linker mittlerer Klammern befinden, die in ähnlicher Weise angepasst werden. Anders als in TEX stehen rechte und linke Klammern nicht notwendiger Weise auf gleicher Höhe. Daher können auch Formeln wie

$$f\left(\frac{1}{x+\frac{1}{y+\frac{1}{z}}}\right)$$

korrekt gesetzt werden. Der Nutzer kann die automatisch bestimmte Größe anders einstellen, indem er zusätzliche Parameter wie *size*, *bottom* und *top* festlegt. So wird aus

```
f⟨left|(|-8mm|4mm⟩x⟨mid|||8mm⟩y⟨right|)|-4mm|8mm⟩
```

$$f\left(x\middle|y\right)$$

*size* kann auch eine ganze Zahl $n$ sein. In diesem Fall wird die $n$-te Größe der vorhandenen Klammergrößen verwendet. Z.B.

```
g⟨left|(|0⟩⟨left|(|1⟩⟨left|(|2⟩⟨left|(|3⟩z⟨right|)|3⟩⟨right|)|2⟩⟨right|)|1⟩⟨right|
)|0⟩
```

liefert

$$g(\bigl(\Bigl(\biggl(z\biggr)\Bigr)\bigr))$$

⟨big|*big-symbol*⟩ (große Symbole)

Dieses Konstrukt erzeugt große mathematische Symbole wie z.B. in

$$\sum_{i=0}^{\infty} a_i z^i \tag{14.1}$$

Die Größe des Operators hängt davon ab, ob es sich um eine eigenständige Formel oder eine Formel in Fließtext handelt. Formeln wie (14.1) heißen eigenständig im Gegensatz zu Formeln wie $\sum_{i=0}^{\infty} a_i z^i$. Im Menü Formate → Eigenständige Formel können die Voreinstellungen angepasst werden.

Beachten Sie, dass die Formel(14.1) intern als

```
⟨big|sum⟩⟨rsub|i=0⟩⟨rsup|<infty>⟩a⟨rsub|i⟩*z⟨rsup|i⟩⟨big|.⟩
```

gespeichert wird.

Das unsichtbare Konstrukt ⟨big|.⟩ dient als Stoptag für den Starttag ⟨big|sum⟩.

⟨frac|*num*|*den*⟩ (Brüche)

Das frac-Konstrukt erzeugt Brüche wie $\frac{x}{y}$. In eigenständigen Formeln wird der Zähler *num* und der Nenner *den* in normaler Größe dargestellt. Während des Schriftsetzens innerhalb des Zählers bzw. innerhalb des Nenners wird der für eigenständige Formeln benutzte Stil ausgeschaltet. Daher werden Argumente innerhalb eigenständiger Formeln in Indexgröße dargestellt, wie z.B. in

```
⟨frac|1|a⟨rsub|0⟩+⟨frac|1|a⟨rsub|1⟩+⟨frac|1|a⟨rsub|2⟩+<ddots>⟩⟩⟩
```

das, wie folgt, aussieht:

$$\frac{1}{a_0+\frac{1}{a_1+\frac{1}{a_2+\ddots}}}$$

⟨sqrt|*content*⟩
⟨sqrt|*content*|*n*⟩ (Wurzeln)

sqrt erzeugt Quadratwurzeln wie $\sqrt{x}$ oder *n*-te Wurzeln wie $\sqrt[3]{x}$. Das Wurzelzeichen wird automatisch der Größe des Inhalts, *content*, angepasst:

$$\sqrt[i+j]{\frac{f(x)}{y^2+z^2}}$$

⟨lsub|*script*⟩
⟨lsup|*script*⟩
⟨rsub|*script*⟩
⟨rsup|*script*⟩ (Indices)

Diese Konstrukte fügen Indices, *script*, einer Box in einer horizontalen Verkettung hinzu und zwar rechte Indices an eine linksstehende Box und umgekehrt. Sie können aber auch allein stehen. Außerdem wird ein oberer Index mit einem unteren Index auf der selben Seite automatisch zusammengefasst. Dies

x⟨rsub|a⟩⟨rsup|b⟩+⟨lsub|1⟩⟨lsup|2⟩x⟨rsub|3⟩⟨rsup|4⟩=y⟨rsub|1⟩+z⟨lsub|c⟩

liefert das

$$x_a^b + {}_1^2x_3^4 = y_1 + z_c$$

Wenn ein rechter Index an ein Symbol oder Konstrukt angebunden wird, welches untere bzw. obere Grenzen akzeptiert, dann wird der Index automatisch als Grenze interpretiert und entsprechend gesetzt:

$$\lim_{n \to \infty} a_n$$

Indices werden im Fließtext in einer kleineren Schriftgröße dargestellt, um die Lesbarkeit zu erhalten jedoch nicht kleiner als doppelte Indexgröße.

⟨lprime|*prime-symbols*⟩
⟨rprime|*prime-symbols*⟩ (hochgestellte Symbole)

Linke und rechte hochgestellte Symbole wie $f'$ ähneln linken oder rechten oberen Indices. Sie benehmen sich aber anders, wenn sie editiert werden. Wenn ihr Cursor hinter dem hochgestellten Symbol in $f'$ steht und sie die `Rücktaste` drücken, dann wird das hochgestellte Symbol entfernt. Wenn sie hinter dem oberen Index in $f^n$ stehen und die `Rücktaste` mehrmals drücken, dann bewegen sie sich zuerst in den Index, dann wird $n$ entfernt und schließlich der obere Index. Beachten Sie bitte, dass *prime-symbols* eine Verkettung von Symbolen ist. So ist $f'^{\dagger}$ die Darstellung von f⟨rprime|'<dag>⟩.

⟨below|*content*|*script*⟩
⟨above|*content*|*script*⟩ (Symbole unter und über Text)

Die Konstrukte below und above erzeugen Symbole, *script*, unter oder über einem Inhalt, *content*. Beide können gleichzeitig verwendet werden wie in:

$$\overset{\infty}{\underset{i=1}{\mathrm{xor}}}\, x_i$$

das durch

```
⟨above|⟨below|xor|i=1⟩|<infty>⟩ x⟨rsub|i⟩
```

erzeugt wird.

⟨wide|*content*|*wide-symbol*⟩
⟨wide*|*content*|*wide-symbol*⟩ (breite Zeichen)

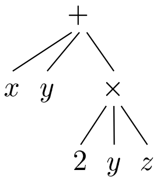
Diese Konstrukte erzeugen breite Zeichen über oder unter einem mathematischen Inhalt *content*. Beispielsweise entspricht $\overline{x+y}$ ⟨wide|x+y|<bar>⟩.

⟨neg|*content*⟩ (Negationen)

Dieses Konstrukt erzeugt verneinten Inhalt wie $\nrightarrow$, $\not a$ und $a\not b c$.

⟨tree|*root*|*child-1*|···|*child-n*⟩ (Bäume)

Dieses Konstrukt erzeugt einen Baum mit der Wurzel, *root*, und Kinder *child-1* bis *child-n*. Er sollte rekursiv zur Erzeugung von Bäumen benutzt werden. Z.B. entspricht




dem Konstrukt

```
⟨tree|+|x|y|⟨tree|<times>|2|y|z⟩⟩
```

Für die Zukunft planen wir weitere Stil-Parameter, um die Darstellung zu steuern.

## 14.4. Tabellen-Konstrukte

Tabellen sind in allen Dokumenten vorhanden, die tformat-Argumente akzeptieren. Alle fundamentalen Tabellen-Konstrukte haben Ränder, die nicht erreichbar sind. Das grundlegende Konstrukt ist tabular.

⟨tformat|*with-1*|···|*with-n*|*table*⟩ (Container zur Tabellen-Formatierung)

Jede Tabellen-Struktur in einem Dokument hat einen tformat-Tag.

⟨tformat|*table*⟩ bedeutet, dass die Tabellen- und Zell-Kontextvariablen unmodifizierte Vorgaben sind. Das Argument, *table*, kann eine Tabelle, table, oder ein verschachtelter tformat-Konstrukt sein. Letzterer erscheint nicht in den Dokumenten, wird aber während der Evaluierung des obersten Konstrukts automatisch erzeugt.

⟨tformat | *with-1* | ··· | *with-n* | *table*⟩ wird benutzt, wenn die Tabelle, *table*, spezielle Formatierungs-Optionen benötigt. Die *with-1* bis *with-n* Argumente müssen alle twith oder cwith-Konstrukte sein.

⟨twith|*var*|*val*⟩ (eine Tabellenvariable setzen)

Die Formatierung einer Tabelle als Ganzes wird von einer Anzahl von *Tabellenvariablen* gesteuert, die nur intern benutzt werden und nicht im Kontext erscheinen wie die normalen den Schriftsatz steuernden Kontextvariablen.

Das twith-Konstrukt setzt die Variable *var* (Zeichenfolge) auf den Wert *val* (nach Evaluierung).

⟨cwith | *top-row* | *bot-row* | *left-col* | *right-col* | *var* | *val*⟩ (Zellvariablen für einen Zellbereich setzen)

Die Formatierung von Zellen wird von einer Anzahl von *Zellvariablen* gesteuert, die nur intern benutzt werden und nicht im Kontext erscheinen wie die normalen den Schriftsatz steuernden Kontextvariablen. Zeilen, Spalten, generell jeder rechteckige Bereich kann mit Hilfe eines einzigen cwith-Konstrukts mit einer Zellvariablen assoziiert werden.

Das cwith-Konstrukt setzt die Zellvariable, *var* (Zeichenfolge) auf de Wert *val* (nach Evaluierung) für den Zeilen-Bereich *top-row* bis *bot-row* und Spalten-Bereich *left-col* bis *right-col* (Zahlzeichen ohne 0).

Die Bereichs-Koordinaten sind ganzzahlige Werte, außer 0, positive Werte werden von links nach rechts und von oben nach unten gezählt, negative Werte entsprechend von rechts nach links und von unten nach oben. 2 bedeutet also die zweite Spalte rechts oder die zweite Reihe nach unten, -1 heißt die Spalte links oder Zeile darüber.

Typische Werte für (*top-row*, *bot-row*, *left-col*, *right-col*) sind $(r, r, 1, -1)$ für „Zeile $r$", $(1, -1, c, c)$ für „Spalte $c$", und $(r, r, c, c)$ für „die Zelle Reihe r, Spalte c". Wenn neue Zellen eingefügt werden, macht es einen Unterschied, ob die Reihen von oben oder von unten und ob die Spalten von links oder von rechts gezählt werden. Wenn $m$ die Anzahl der Zeilen und n die Anzahl de Spalten ist, dann repräsentieren $r$ und $r - m - 1$ dieselbe Zeile nur einmal von oben und einmal von unten gezählt. Ähnlich entsprechen $c$ und $c - n - 1$ dieselbe Spalte einmal von links andermal von rechts.

⟨table|*row-1*|···|*row-n*⟩ (Zeilencontainer)

Der einzige Zweck des table-Konstrukts ist es, row-(Zeilen)-Konstrukte aufzunehmen. Die Anzahl der Zeilen ist die Anzahl der Unter-Bäume.

⟨row|*cell-1*|···|*cell-k*⟩ (Zellencontainer)

Der einzige Zweck des row-Konstrukts ist es, cell-(Zellen)-Konstrukte aufzunehmen. Die Anzahl der Zeilen ist die Anzahl der Unter-Bäume. Alle row-(Reihen)-Konstrukte in einer Tabelle, table, müssen die genau so viele Unter-Bäume, cell-(Zellen)-Konstrukte, haben, wie Spalten in der Tabelle vorhanden sind.

⟨cell|*content*⟩ (Zell-Datencontainer)

Die Zellen von Tabellen können jede Art von Dokument-Fragmenten enthalten. Eine Zelle, cell, kann direkt Zeileninhalt oder ein concat-Konstrukt enthalten, wenn Blockinhalt eingefügt werden soll, muss es in Form eines document-Baumes sein.

Eine Zelle, cell, deren Inhalt ein document ist, ist eine *Multi-Absatz-Zelle*. Weil Tabellen im Zeilen-kontext erlaubt sind, ist dies das einzige Konstrukt, das indirekt die Einfügung von Blockinhalten in Zeilen-kontext erlaubt. Man beachte, dass fast jeder Blockinhalt nur in Zellen, die umgebrochen werden können, korrekt gesetzt werden kann. Dies wird mit der Tabellenvariablen *cell-hyphen* eingestellt.

⟨subtable|*table*⟩ (Unter-Tabellen)

Zellen können Unter-Tabellen, subtable, enthalten. Das Argument von subtable ist ein tformat-Baum, der normalen Tabellen-Inhalt enthält.

Ein ähnlicher Effekt kann erreicht werden, wenn man das Zell-Padding in alle Richtungen auf 0 setzt. Eine Besonderheit von subtable ist aber, dass ihre Ränder nicht erreichbar sind.

⟨tmarker|*table*⟩ (Markierung des Dekorationsurpsrungs)

Dieses Konstrukt wird bei der Definition von Zelldekorationen benutzt, siehe die Dokumentation zu *cell-decoration*.

Es wird außerdem außerhalb von Tabellen zur Markierung der augenblicklich dargestellten Position im switch-Konstrukt verwendet.

⟨tabular|*table*⟩ (fundamentales Tabellenmakro)

Diese Makro definiert links ausgerichtete Standard-Tabellen ohne Gitter. Obwohl tabular in $\text{T}_\text{E}\text{X}_{\text{MACS}}$ eingebaut ist, sollte es eigentlich nicht als ein fundamentales Konstrukt betrachtet werden. Allerdings gehört es auch nicht zu den Stil-Definitionen.

## 14.5. Link-Konstrukte

⟨label|*name*⟩ (Referenzziel)

Der Operand muss zu einer Zeichenkette evaluieren. Dieser wird als Zielname einer der folgenden Referenztypen, reference, pageref und hlink verwendet.

Der Name eines Labels sollte eindeutig innerhalb eines Dokuments sein und darf daher nur einmal vergeben werden.

Beispiele in diesem Abschnitt verweisen auf das Label, label, mit dem Namen „there" .

⟨label|there⟩

⟨reference|*name*⟩ (Verweis auf einen Namen)

Das Argument ist eine Zeichenkette, ein Label, das in einem label-Konstrukt im aktuellen oder einem anderen zum aktuellen Projekt gehörigen Dokument definiert worden ist.

⟨reference|there⟩

Der Verweis, reference, wird beim Schriftsetzen durch den Wert der Variablen *the-label* am Punkt des Ziels label. Die Variable *the-label* wird in vielen unterschiedlichen Strukturen wie Abschnitte, Abbildungen, nummerierten Gleichungen usw. gesetzt.

Eine Referenz, ein Verweis, reference, reagiert auf Mausklicks wie ein Hyperlink.

⟨pageref|*name*⟩ (Seitenzahl)

Das Argument muss zu einer Zeichenkette evaluieren, die als Label mit dem Konstrukt label innerhalb des aktuellen Dokuments oder einem Dokument des aktuellen Projektes definiert wurde.

⟨pageref|there⟩

pageref wird im Schriftsatz durch die Seitenzahl der Seite ersetzt, die das Ziellabel, label, enthält. Man beachte, dass Seitenzahlen nur berechnet werden können, wenn das Dokument mit Seitenumbruch gesetzt wird. Das ist nicht der Fall in den Seitentypen „automatisch" oder „Papyrus".

pageref reagiert auf Mausklicks wie ein Hyperlink.

⟨hlink|*content*|*url*⟩ (Hyperlink)

Dieses Konstrukt erzeugt einen Hyperlinks mit dem sichtbaren Text *content*, der auf *url* zeigt. *url* muss zu einer Zeichenkette in URL-Syntax evaluieren und zu einem lokalen Dokument oder einem Dokument auf einem anderen Rechner verweisen. Positionen innerhalb eines Dokuments können mit Labeln definiert werden.

Die folgenden Beispiele verweisen auf das gleiche Dokument, ein Dokument im gleichen Verzeichnis und einem Web-Dokument.

⟨hlink|Dieses Dokument|`./#there`⟩

⟨hlink|gleiches Verzeichnis|`file.tm#there`⟩

⟨hlink|Im Web|`http://example.org/#there`⟩

Das erste Beispiel wird im laufenden Text so Dieses Dokument dargestellt. Wenn das Dokument nicht editierbar ist, wird es mit einem einfachen Klick erreicht. Für editierbare Dokumente ist dagegen ein Doppelklick erforderlich.

⟨include|*url*⟩ (ein anderes Dokument einbinden)

*url* muss zu einer Zeichenkette in URL-Syntax evaluieren und zu einem lokalen Dokument oder einem Dokument auf einem anderen Rechner verweisen. Er wird als Dateiname interpretiert und der Inhalt der Datei wird anstelle des include-Konstrukts eingefügt. Dafür muss dieser in einem Blockkontext liegen.

⟨action|*content*|*script*⟩ (ein ausführbares Skript einbinden)

Ein ausführbares SCHEME-Skript, *script* einbinden, dass bei einem doppelten Mausklick auf *content* ausgeführt wird. Beispielsweise, wenn Sie auf hier doppelklicken, erzeugen Sie ein neues `xterm`. Der Code dafür ist ⟨action|hier|`(system "xterm &")`⟩.

Aus Sicherheitsgründen wird vom Nutzer normalerweise, wenn ausführbare Skripte eingeleitet werden sollen, eine Bestätigung verlangt. Das Sicherheitsniveau kann im Menü Bearbeiten → Einstellungen → Sicherheit eingestellt werden. Programmierer können auch bestimmte SCHEME-Routinen als „sicher” erklären. SCHEME-Programme, die nur sichere Routinen enthalten werden ohne Rückfrage ausgeführt.

⟨mutator|*content*|*script*⟩ (ein Konstrukt, das sich selbst modifizieren kann)

Der Inhalt, *content*, eines mutator-Konstrukts wird automatisch durch das SCHEME-Skript, *script*, erzeugt. Genauer, TEXMACS ermittelt periodisch, welche mutator-Konstrukte geöffnet sind. Für jeden setzt es den *Mutatorpfad* auf den Pfad zu dem Inhalt, *content*, und ruft dann das zugehörige SCHEME-Skript. Dieses Skript kann den Inhalt, *content*, modifizieren oder auch andere Stellen des Dokuments. Letzteres ist aber wenig effizient und wird deshalb nicht empfohlen. Um den *Mutatorpfad* aus SCHEME zu extrahieren, sollte man den SCHEME-Befehl `(get-mutator-path)`benutzen.

Mutatoren sind sehr nützlich, wenn TEXMACS mit externen Programmen kommuniziert. Die derzeitige Implementierung von Computeralgebra und anderen „Sessions“ benutzt Mutatoren, damit man an einem Dokument arbeiten kann, während extern Berechnungen durchgeführt werden. Da Mutatoren vom Editor automatisch gefunden werden, bleibt ihr Verhalten auch dann korrekt, wenn die Position des Mutators im Dokument geändert wird. Das geschieht beispielsweise, wenn man einen neuen Absatz am Beginn des Dokuments einfügt.

## 14.6. Graphik-Konstrukte

Die graphischen Konstrukte sind noch in voller Entwicklung und werden später erst dokumentiert.

⟨postscript|*url*|*width*|*height*|*clip-left*|*clip-bottom*|*clip-right*|*clip-top*⟩

Please document.

⟨superpose|*content-1*|···|*content-n*⟩

Please document.

⟨graphics|*gr-content-1*|···|*gr-content-n*⟩

Please document.

⟨text-at|*content*|*pos*|*hor-align*|*ver-align*⟩

Please document.

⟨point|*coord-1*|···|*coord-n*⟩

Please document.

⟨line|*point-1*|···|*point-n*⟩

Please document.

⟨cline|*point-1*|···|*point-n*⟩

Please document.

⟨spline|*point-1*|···|*point-n*⟩

Please document.

⟨spline*|*point-1*|···|*point-n*⟩

Please document.

⟨cspline|*point-1*|···|*point-n*⟩

Please document.

⟨fill|*unspecified*⟩

Please document.

## 14.7. Sonstige Konstrukte

⟨group|*content*⟩ (Atom)

Setze *content*, der Zeileninhalt sein muss, als Atom, d.h. als Einheit, die nicht getrennt wird und an deren Grenzen auch keine speziellen Operationen vorgenommen werden.

⟨float|*type*|*where*|*body*⟩ (bewegliche Einfügungen)

Bewegliche Einfügungen sind Seiten-Elemente, die keinen festen Ort haben. Sie bestehen aus zwei Boxen. Die Ankerbox markiert die Ausgangsposition im Text und die bewegliche Box enthält den Rumpfschriftsatz, *body*. Dieses Konstrukt wird für Fußnoten und bewegliche Blöcke benutzt.

Die beiden ersten Argument werden evaluiert. In den Beispielen werden aber zur Vereinfachung Zeichenketten verwendet. *body* kann Blockinhalt sein, selbst wenn das float-Konstrukt im Zeilen-kontext liegt.

⟨float|footnote||*body*⟩ fügt eine Fußnote ein. Dies sollte aber nur mit dem footnote-Makro benutzt werden und wird als Stil-Definition betrachtet. Die Fußnote wird an das Ende der Seite gesetzt, die die Ankerbox enthält.

⟨float|float|*where*|*body*⟩ erzeugt einen beweglichen Block, dies wird als normales Konstrukt angesehen. Die Position der beweglichen Box wird von Seitenumbruch-Algorithmus zugewiesen, der die durch die Beweglichkeit erzeugten Freiheitsgrade zur Minimierung der Seitenumbruchstrafe benützt.

*where* muss zu einer Zeichenkette evaluieren, die folgende Zeichen enthalten kann:

**t.** Gestatte eine Position der Box *ganz oben*.

**b.** Gestatte eine Position der Box *ganz unten*.

**h.** Gestatte eine Position der Box „*hier*”, inmitten der Seite nahe zur Ankerbox.

**f.** Erzwinge eine Position der Box auf der selben Seite wie die Ankerbox.

⟨specific|*medium*|*body*⟩ (medium-specific content)

Dieses Konstrukt sorgt dafür, dass *body* nur über ein bestimmtes „*medium*” ausgegeben werden kann. Die folgenden Werte von *medium* werden unterstützt:

**texmacs.** *body* wird als normaler Zeilen-Inhalte gesetzt.

**latex.** *body* muss eine Zeichenkette sein. Diese ist nicht sichtbar unter TeXmacs, wird aber „wörtlich“ übernommen, wenn das Dokument nach LaTeX exportiert wird.

**html.** Analog zu `latex`, aber für Export nach HTML.

**screen.** *body* wird nur auf dem Bildschirm sichtbar. Es kann während der Erstellung und Änderung von Dokumenten sehr nützlich sein, Kommentare anzubringen, die beim Druck verschwunden sind. Das Konstrukt flag kann ähnlich verwendet werden.

**printer.** Dies ist komplementär zu `screen`, *body* wird gedruckt, aber nicht auf dem Bildschirm angezeigt.

⟨raw-data|*data*⟩ (geschützte Daten)

In bestimmten Kontexten muss man Daten, meist Binärdaten, vor Veränderungen geschützt, einfügen. Das raw-data-Konstrukt verhindert Veränderungen im Editor.

# Kapitel 15

# Konstrukte für Stildefinitionen

## 15.1. Kontext-Konstrukte

Der aktuelle Kontext definiert alle Stil-Parameter, die den Prozess des Schriftsetzens beeinflussen, sowie alle zusätzlichen Anwender-Makros mit dem aktuellen Basis-Stil. Die Konstrukte in diesem Abschnitt dienen dazu, die Kontextvariablen zu ermitteln und zu verändern.

⟨assign|*var*|*val*⟩ (Variablen global setzen)

Dieses Konstrukt setzt die Kontextvariable *var* (Zeichenkette) auf den Wert *val*, der das Ergebnis eines evaluierten Ausdrucks sein kann. Er wird eingesetzt, um den Kontext zu ändern, wie z.B. zur In- oder Dekrementierung von Zählern.

Der Evaluierungs-Prozess kann über value, provides und Makro-Definitionen beeinflusst werden, desgleichen der Schriftsatz durch spezielle Schriftsatz-Variablen.

Beispiel 15.1. Seitenumbruch durch den Stil einschalten.

Die Variable `page-medium` wird genutzt, um den Seitenumbruch einzuschalten. Da nur der ursprüngliche Kontext-Wert benutzt wird, muss die Zuordnung in einer Stil-Definition erfolgen. Sie kann in einem Dokument nicht geändert werden.

```
⟨assign|page-medium|paper⟩
```

Beispiel 15.2. Den Kapitel-Zähler setzen.

Das folgende Codestück sorgt dafür, dass das folgende Kapitel die Nummer 3 bekommt. Das kann sinnvoll sein, um korrekte Nummerierung im Buch-Stil zu erreichen, wenn man in Projekten mit include arbeitet.

```
⟨assign|chapter-nr|2⟩
```

⟨with|*var-1*|*val-1*|⋯|*var-n*|*val-n*|*body*⟩ (Variablen lokal setzen)

Dieses Konstrukt setzt die Kontextvariablen *var-1* bis *var-n* (in dieser Reihenfolge) auf die evaluierten Werte *val-1* bis *val-n* und setzt *body* in dem modifizierten Kontext. Alle mit assign erfolgten Änderungen der *var-1* bis *var-n* in *body* werden beim Verlasen von with zurückgesetzt.

Dieses Konstrukt wird in großem Umfang in Stil-Definitionen eingesetzt, um den Kontext für den Schriftsatz zu ändern, beispielsweise um die Schriftart, den Absatz-Stil zu ändern und den Modus für Mathematik einzuschalten.

⟨value|*var*⟩ (Wert einer Variablen)

Dieses Konstrukt evaluiert zu dem aktuellen Wert der Variablen *var* (Zeichenkette). Das wird genutzt, um Zähler anzuzeigen und generell, um Kontext-abhängiges Verhalten zu implementieren.

Dieses Konstrukt wird häufig in Stil-Definitionen genutzt, um den Kontext zu verändern. Z.B., um lokal die Schriftart, den Absatz-Stil usw. zu ändern.

⟨provides|*var*⟩ (definiert?)

Dieses Konstrukt ist ein Prädikat, dass wahr, `true`, ergibt, wenn die Kontextvariable *var* (eine Zeichenkette) definiert ist und sonst falsch, `false`.

Das ist nützlich, um eine vernünftige Fehlerbehandlung zu erzeugen, wenn beispielsweise ein notwendiges Paket nicht vorhanden ist.

## 15.2. MAKRO-KONSTRUKTE

Makros können zur Definition von neuen Befehlen, Konstrukten, Tags benutzt werden und zur Konstruktion von abstrakten Prozeduren in Stil-Definitionen.

Ältere Versionen von TeXmacs machten einen Unterschied zwischen Makros, bei denen alle Kinder erreichbar waren und Funktionen, die Kinder nicht erreichen konnten. Das jetzige TeXmacs kennt nur Makros: die Erreichbarkeit von Kindern wird heuristisch ermittelt und kann durch drd-props gesteuert werden.

⟨macro|*var-1*|⋯|*var-n*|*body*⟩ (Makro mit festgelegter Argumentanzahl)

Dieses Konstrukt erzeugt ein Makro (das TeXmacs-Analogon eines $\lambda$-Ausdrucks) mit $n$ Argumenten mit den Namen der Zeichenkette *var-1* bis *var-n*.

Neue Tags werden definiert, indem die Makros im Kontext gespeichert werden. Meistens werden Makros global gespeichert mit assign, aber manchmal ist es besser, einem Tag lokal zu definieren mit with. Beispielsweise definieren nummerierte Kontexte die Variable item lokal.

BEISPIEL 15.3. Definition einer Abkürzung

⟨assign|*Abkürzung*|⟨macro|$x$|⟨group|$x$⟩⟩⟩

Wenn man ein Makro macro im Kontext speichert, wird ein Tag mit einer vorgegebenen Anzahl von Argumenten definiert.

⟨arg|*var*|*index-1*|⋯|*index-n*⟩ (Makro-Argumente ermitteln)

Dieses Konstrukt dient dazu, Werte von Variablen innerhalb eines Makro-Rumpfes zu ermitteln. Z.B. evaluiert ⟨arg|*var*⟩ zu dem aktuellen Wert von *var* (Zeichenkette). Natürlich muss dieses Argument davor in dem Makro, macro, definiert sein, das den arg-Konstrukt enthält.

Dieses Konstrukt ähnelt value, verhält sich aber in einigen wichtigen Punkten anders:

- Der Argument-Namensraum unterscheidet sich vom Kontext, daher liefern ⟨arg|*var*⟩ und ⟨value|*var*⟩ in der Regel unterschiedliche Werte. (Darauf sollte man sich aber nicht verlassen.)

- Der Wert von arg verbleibt an der Stelle des Makro-Arguments in dem Dokument-Baum. Deshalb können die Argumente eines Makro-Konstrukts im aktiven Zustand geändert werden.

Wenn mehr als Argument vorhanden ist, dann expandiert ⟨arg|*var*|*index-1*|⋯|*index-n*⟩ zu einem Unterbaus des Arguments *var*. Der Wert von *var* muss ein unbenanntes Makro, compound, sein, also keine Zeichenkette. Alle Argumente, *var* bis *index-n*, müssen zu positiven Ganzzahlen evaluieren und den Pfad zu einem Unterbaus des Makro-Arguments zeigen.

⟨xmacro|*var*|*body*⟩ (Makro mit beliebiger Argumentanzahl)

Dieses Konstrukt definiert ein Makro (das $\TeX_{\text{MACS}}$-Analogon eines $\lambda$-Ausdrucks), welches eine beliebige Anzahl von Argumenten annehmen kann. Die Argumente werden in der Makro-Variablen mit Namen *var* (einer Zeichenkette) während der Evaluierung des Rumpfes, *body*, gespeichert. Der Wert des $i$-ten Arguments kann dann mit ⟨arg|*var*|*i*⟩ erhalten werden.

⟨map-args|*foo*|*root*|*var*⟩
⟨map-args|*foo*|*root*|*var*|*first*⟩
⟨map-args|*foo*|*root*|*var*|*first*|*last*⟩ (ein Makro auf alle Kinder eines Baums anwenden)

Dieses Konstrukt evaluiert ein Baum, dessen Wurzel *root* ist und dessen Kinder durch Anwendung eines Makros *foo* auf die Kinder des Makro-Arguments mit Namen *var* entstehen.

Entsprechend der Vorgabe wird *foo* auf auf alle Kinder angewandt. Wenn *first* spezifiziert wurde, dann wird mit dem $i$-ten Kind begonnen, wenn $i$ das Resultat der Evaluierung von *first* ist. Mit *last* wird die Berechnung beim $j$-ten Kind von *var*, wobei das j-te Kind nicht einbezogen wird, wenn $j$ das Ergebnis der Evaluierung von *last* ist. Die Dimension des Baumes ist also $j-i$.

Anders ausgedrückt: map-args wendet *foo* auf alle Unter-Bäume oder auf ein Intervall von Unter-Bäumen an (falls *first* und *last* angegeben wurde) und sammelt das Ergebnis in einem Baum mit Namen *root*.

map-args ist das Analogon zur Scheme-Funktion `map`. Weil $\TeX_{\text{MACS}}$ aber Bäume mit Labeln verwendet, muss das Label für das Ergebnis mit übergeben werden.

Beispiel 15.4. Komma-separierte Listen.

Das comma-separated-Makro hat eine Dimension, auch wenn dies keinen Sinn macht bei der Dimension 0. Der Schriftsatz erfolgt so, dass seine Argumente durch Kommata getrennt werden.

```
⟨assign|comma-extra|⟨macro|x|, x⟩⟩

⟨assign|comma-separated|
    ⟨xmacro|args|
        ⟨concat|
            ⟨arg|args|0⟩|
            ⟨map-args|comma-extra|concat|args|1⟩⟩⟩⟩
```

⟨eval-args|*var*⟩ (Makro mit beliebiger Dimension)

Dieses Konstrukt evaluiert zu einem Baum mit dem dem gleichen Label wie die Expansion des Arguments *var*, dessen Unter-Bäume das Resultat der Evaluierung der Unter-Bäume von *var* sind.

⟨compound|*foo*|*arg-1*|⋯|*arg-n*⟩ (unbenanntes Makro)

Dieses Konstrukt ist besonders nützlich, wenn es darum geht, Makros zu expandieren, die selbst das Ergebnis einer Berechnung sind: Er wendet das Makro, welches das Resultat der Evaluierung von *foo* ist, auf die Argumente *arg-1* bis *arg-n* an. compound wird vor allem verwendet, wenn einem Makro ein anderes Makro als Argument übergeben wird, welches es dann beim Vorliegen bestimmter Bedingungen verwendet.

Allerdings kann in der derzeitigen Implementierung *foo* entweder zu einem Makro evaluieren oder zu einer Zeichenkette, die dann dem Makro ihren Namen gibt. Wir empfehlen Vorsicht bei der zweiten Variante.

BEISPIEL 15.5. Lambda-Programmierung mit Makros.

Im unten stehenden Code erwartet ⟨filter|*pred*|*t*⟩ ein Makro *pred* und ein Tupel *t* als Argumente und liefert ein Tupel zurück, das aus denjenigen Elemente von *t* besteht, für die *pred* den Wert `true` ergibt.

```
⟨assign|filter|
    ⟨macro|pred|t|
        ⟨if|
            ⟨equal|⟨length|t⟩|0⟩|
            ⟨tuple⟩|
            ⟨merge|
                ⟨if|
                    ⟨compound|pred|⟨look-up|t|0⟩⟩|
                    ⟨tuple|⟨look-up|t|0⟩⟩|
                    ⟨tuple⟩⟩|
                ⟨filter|pred|⟨range|t|1|⟨length|t⟩⟩⟩⟩⟩⟩⟩
```

Das lässt sich z.B. in einem Makro ⟨evens | *t*⟩ verwenden, das aus ein Tupel von Ganzzahlen *t* die geraden Zahlen extrahiert.

```
⟨assign|evens|⟨macro|t|⟨filter|⟨macro|x|⟨equal|⟨mod|x|2⟩|0⟩⟩|t⟩⟩⟩
```

⟨drd-props|*var*|*prop-1*|*val-1*|⋯|*prop-n*|*val-n*⟩ (setze D.R.D.-Eigenschaften)

Die Dimension und die Erreichbarkeit von Kindern wird normalerweise heuristisch ermittelt. Das drd-props-Konstrukt ändert die Voreinstellung für die Kontextvariable (normalerweise ein Makro) mit dem Namen *var*. Zur Zeit werden die folgenden Paarungen unterstützt:

**(arity, *n*)** — (Dimension, n) setzt die Dimension auf einen festen Wert $n$ (Ganzzahl).

**(accessible, all)** — (erreichbar, alle) Das Konstrukt kann dann im Editor nicht deaktiviert werden. Kinder, die nicht erreichbar sind, können nicht editiert werden.

**(accessible, none)** — Verhindert, dass der Cursor innerhalb eines Konstrukts positioniert wird, solange das Konstrukt aktiv ist. Kinder können also nur editiert werden, wenn das Konstrukt deaktiviert wird.

⟨get-label|*expression*⟩ (Label eines Ausdrucks)

Gibt den Label des Baumes zurück, wenn der Ausdruck, *expression*, evaluiert wird.

⟨get-arity|*expression*⟩ (Dimension eines Ausdrucks)

Gibt die Dimension des Baumes zurück, wenn der Ausdruck, *expression*, evaluiert wird.

## 15.3. Steuerung des logischen Ablaufs

⟨if|*condition*|*if-body*⟩
⟨if|*condition*|*if-body*|*else-body*⟩ (Bedingung)

Dieses Konstrukt setzt *if-body* nur dann, wenn die Bedingung *condition* erfüllt ist. Wenn ein optionales *else-body* spezifiziert wurde, dann wird diese nur gesetzt, wenn die Bedingung *condition* falsch ist.

Bemerkung 15.6. Man sollte beachten, das Verwendung von Bedingungen fehleranfällig ist. Das kommt davon, dass die Erreichbarkeit von Argumenten nicht vorab getestet werden kann. In dem Makro

```
⟨macro|x|⟨if|⟨visibility-flag⟩|x⟩⟩
```

ist das Makro-Argument *x* dann und nur dann erreichbar, wenn ⟨visibility-flag⟩ wahr ist. Das kann aber vorher nicht getestet werden. Bestimmte Editier-Operationen, wie Suchen oder Rechtschreibprüfung kann eine falsche Bestimmung der Erreichbarkeit dazu führen, dass der Cursor in nicht erreichbare Positionen gesetzt wird oder dass bestimmte Konstrukte ignoriert werden. Wir wollen dieses Verhalten des Editors verbessern. Bis dahin ist es besser, Bedingungen zu vermeiden.

Bemerkung 15.7. Die Bedingungs-Konstrukte sind nur für Zeileninhalt voll implementiert. Wenn man Bedingungen im Blockkontext braucht, dann muss man den Wenn-ja-Fall und den Wenn-nein-Fall getrennt in Makros definieren und mit einem unbenannten Makro die Bedingung einführen. Z.B.:

```
⟨assign|kalt|⟨macro|x|⟨with|color|blue|x⟩⟩⟩
⟨assign|heiss|⟨macro|x|⟨with|color|red|x⟩⟩⟩
⟨assign|adaptive|⟨macro|x|⟨compound|⟨if|⟨summer⟩|heiss|kalt⟩|x⟩⟩⟩
```

⟨case|*cond-1*|*body-1*|···|*cond-n*|*body-n*⟩
⟨case|*cond-1*|*body-1*|···|*cond-n*|*body-n*|*else-body*⟩ (Fallunterscheidungen)

Diese Befehle sind äquivalent zu

```
⟨if|cond-1|body-1|···⟨if|cond-n|body-n⟩⟩
⟨if|cond-1|body-1|···⟨if|cond-n|body-n|else-body⟩⟩
```

⟨while|*condition*|*body*⟩ (wiederholte Ausführung)

Solange die Bedingung *condition* erfüllt ist, wird *body* ausgeführt. Beispielsweise erzeugt der folgende das Makro

```
⟨assign|count|
    ⟨macro|from|to|
        ⟨with|i|from|
            ⟨concat|
                ⟨while|⟨less|i|to⟩|i, ⟨assign|i|⟨plus|i|1⟩⟩⟩|
                to⟩⟩⟩⟩
```

aus ⟨count|1|50⟩ dies:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50
```

## 15.4. Steuerung der Evaluierung

Dieser Abschnitt beschreibt verschiedene Konstrukte, die steuern, wie Ausdrücke in der Stil-Definitions-Sprache evaluiert werden. Die Konstrukte sind Analoga zu den SCHEME-Befehlen `eval`, `quote`, `quasiquote`, usw., wobei die TeXmacs-Konventionen sich etwas von normalen funktionellen Sprachen wie SCHEME unterscheiden.

⟨eval|*expr*⟩ (Evaluierung erzwingen)

Erzeuge Schriftsatz für das Resultat der Evaluierung von *expr*. Dieses Konstrukt wird meist kombiniert mit quote oder quasiquote, die die Evaluierung verzögern.

⟨quote|*expr*⟩ (verzögerte Evaluierung)

Evaluierung von ⟨quote|*expr*⟩ liefert *expr* zurück. Diese Art der verzögerten Evaluierung wir oft zusammen mit eval gebraucht.

⟨quasiquote|*expr*⟩ (Substitution mit verzögerter Evaluierung)

Dies ist eine Variante von quote, bei der alle Unter-Ausdrücke der Form ⟨unquote | *subexpr*⟩ durch die Evaluierungen von *subexpr* substituiert wurden. So definiert z.B.

```
⟨assign | hello | ⟨quasiquote | ⟨macro | name | ⟨unquote | ⟨localize | Hello⟩⟩
name.⟩⟩⟩
```

ein Makro hello, dessen Wert in der aktuellen Sprache ausgegeben wird. In einem französischen Dokument würde dies äquivalent zu

```
⟨assign|hello|⟨macro|name|Bonjour name.⟩⟩
```

sein.

Beachten Sie bitte, dass es für solche Anwendungsfälle besser ist, quasiquote nicht zu verwenden, denn wenn einfach

```
⟨assign|hello|⟨macro|name|⟨localize|Hello⟩ name.⟩⟩
```

verwendet wird, passt sich der Schriftsatz von ⟨hello|Name⟩ automatisch der Sprache an, während in obigen Version immer die Sprache verwendet werden, die bei der Makrodefinition aktuell war. Dennoch hat die Verwendung von quasiquote den Vorteil, dass die Sprachanpassung von „Hello” nur einmal erfolgen muss. Daher kann quasiquote manchmal zur Steigerung der Effizienz eingesetzt werden.

⟨unquote|*subexpr*⟩ (Evaluierung zulassen)

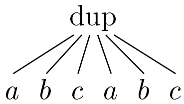
Dieses Konstrukt wird meist zusammen mit quasiquote und quasi verwendet, um Ausdrücke zu markieren, die evaluiert werden sollen.

⟨unquote*|*subexprs*⟩ (Evaluierung von Listen zulassen)

Dies ähnelt dem unquote, nur dass das Argument *subexprs* zu einer Liste von Unterausdrücken evaluiert, die in die Argumente des Elternknotens substituiert werden. So liefert der Schriftsatz von

```
⟨assign|fun|
    ⟨xmacro|x|
        ⟨quasi|
            ⟨tree|dup|⟨unquote*|⟨quote-arg|x⟩⟩|⟨unquote*|⟨quote-arg|
            x⟩⟩⟩⟩⟩
```

mit ⟨fun|a|b|c⟩ folgendes




⟨quasi|*expr*⟩ (Substitution)

Dies ist eine Abkürzung von ⟨eval|⟨quasiquote|*expr*⟩⟩. Dieses Konstrukt wird oft in TeXmacs-Stil-Definitionen verwendet, Makros zu schreiben, die eine Liste von Makros definieren. So können mit dem Makro

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

Theorem-ähnliche Kontexte erzeugt werden.

⟨quote-value|*var*⟩ (nicht evaluierter Wert)

Normalerweise, wenn man den Wert einer Kontextvariablen *var* erfahren möchte, ist man an daran interessiert, wie sie im Schriftsatz erscheint und wie ihn ⟨value|*var*⟩ erzeugt. Manchmal möchte man den wahren, nicht evaluierten Wert haben, was mit ⟨quote-value|*var*⟩ geschehen kann.

⟨quote-arg|*var*|*index-1*|···|*index-n*⟩ (nicht evaluiertes Argument)

Normalerweise, wenn man den Wert eines Unter-Ausdrucks eines Makro-Arguments *var* erfahren möchte, ist man an daran interessiert, wie diese im Schriftsatz erscheinen, in eben der Form, die ⟨arg|*var*|*index-1*|···|*index-n*⟩ erzeugt. Manchmal möchte man die wahren, nicht evaluierten Werte haben, was mit⟨quote-arg|*var*|*index-1*|···|*index-n*⟩ geschehen kann.

## 15.5. FUNKTIONELLE OPERATOREN

Funktionelle Operatoren werden zu Berechnungen während des Schriftsatzes eingesetzt, um beispielsweise Zähler zu inkrementieren, Sprachanpassungen vorzunehmen usw.. Es existiert ein Satz von eingebauten fundamentalen Konstrukten für diese Aufgaben. Neue funktionelle Operatoren können mit dem extern-Konstrukt leicht hinzugefügt werden. Fünf Haupt-Typen von Argumenten haben diese Operatoren: Zeichenketten, Zahlen, Längen, boolesche Variablen und Tupel. Einige Operatoren sind „überladen" arbeiten also mit mehreren Typen.

### 15.5.1. Text-Operatoren

⟨length|*expr*⟩ (Länge einer Zeichenkette)

Wenn *expr* eine Zeichenkette ist wird deren Länge zurückgegeben. ⟨length | Hello⟩ evaluiert zu 5.

⟨range|*expr*|*start*|*end*⟩ (Extrahiere eine Teilkette)

Gebe den Teil der Zeichenkette *expr* zurück, der an der Position *start* beginnt und an *end* aufhört. Die Endposition ist nicht dabei. ⟨range|hottentottenttententententoonstelling|9|15⟩ evaluiert zu tenten.

⟨merge|*expr-1*|···|*expr-n*⟩ (Verkette Zeichenketten)

Damit werden mehrer Zeichenketten *expr-1* bis *expr-n* zu einer neuen Zeichenkette zusammengefügt. ⟨merge|Hello|World⟩ gibt HelloWorld.

⟨number|*number*|*render-as*⟩ (unterschiedliche Darstellungen von Zahlen)

Stelle eine Zahl *number* in einer bestimmten Weise dar. Mögliche Werte für *render-as* sind

**roman.** kleine römische Zahlen: ⟨number|18|roman⟩ ⟶ xviii.

**Roman.** Große römische Zahlen: ⟨number|18|Roman⟩ ⟶ XVIII.

**alpha.** kleine Buchstaben: ⟨number|18|alpha⟩ ⟶ r.

**Alpha.** Großbuchstaben: ⟨number|18|Alpha⟩ ⟶ R.

⟨date⟩
⟨date|*format*⟩

⟨date|*format*|*language*⟩ (aktuelles Datum)

gibt das aktuelle Datum in einer bestimmten landesspezifischen Darstellung *format* zurück, wenn dieses Feld leer gelassen wird, wird die landesspezifische Voreinstellung benutzt, und, wenn angegeben, in einer bestimmten Sprache *language* ansonsten Englisch. Das Format ähnelt der Ausgabe des UNIX `date`-Befehls. ⟨date⟩ evaluiert zu „27 April 2005", ⟨date||french⟩ zu „27 avril 2005" und ⟨date|%d %B om %k:%M|dutch⟩ zu „27 April om 12:53".

⟨translate|*what*|*from*|*into*⟩ (Übersetzung)

Gibt die Übersetzung der Zeichenkette *what* aus der Sprache *from* in die Sprache *into* zurück. Dabei wird das TeXmacs-Wörterbuch verwendet. Die Sprachen sind in kleinen Buchstaben in englischer Sprache anzugeben. ⟨translate | File | english | french⟩ liefert „Fichier".

Eine Liste der verfügbaren Sprachen findet sich im Menü Dokument → Sprache. Die englische Schreibweise kann man leicht ermitteln, indem man vorübergehend die Sprach unter Bearbeiten → Einstellungen → Sprache auf Englisch einstellt. Die eingebauten TeXmacs-Wörterbücher findet man in

```
$TEXMACS_PATH/languages/natural/dic
```

Wenn man versucht ein nicht vorhandenes Wörterbuch zu benutzen, kann das Programm abstürzen. Meistens ist es einfacher das Makro localize zu benutzen, das eine Zeichenkette aus dem Englischen in die aktuelle Sprache umsetzt.

### 15.5.2. Arithmetische Operatoren

⟨plus|*expr-1*|*expr-2*⟩
⟨minus|*expr-1*|*expr-2*⟩ (Addition und Subtraktion)

Addiere oder subtrahiere zwei Zahlen oder Längen. Wenn *expr-1* und *expr-2* Längen sind, werden die Einheiten in die interne Längeneinheit umgerechnet. ⟨plus|1|2.3⟩ ergibt also 3.3 und ⟨plus|1cm|5mm⟩ produziert ⟨tmlen|90708.6⟩.

⟨times|*expr-1*|*expr-2*⟩ (Multiplikation)

Multipliziere zwei Zahlen *expr-1* und *expr-2* oder multipliziere eine Zahl mit einer Länge. ⟨times|3|3⟩ evaluiert zu 9 und ⟨times|3|2cm⟩ zu ⟨tmlen|362835⟩.

⟨over|*expr-1*|*expr-2*⟩ (Division)

Dividiere zwei Zahlen *expr-1* und *expr-2*, dividiere eine Länge durch eine Zahl oder dividiere zwei Längen. ⟨over|1|3⟩ evaluiert zu 0.333333, ⟨over|3cm|7⟩ zu ⟨tmlen|25916.7⟩ und ⟨over|1cm|1pt⟩ zu 28.4528.

⟨div|*expr-1*|*expr-2*⟩
⟨mod|*expr-1*|*expr-2*⟩ (Ganzzahlendivision)

⟨div|*expr-1*|*expr-2*⟩ gibt das Ergebnis einer Ganzzahlendivision, das Modul ⟨mod|*expr-1*|*expr-2*⟩ den Rest. ⟨plus|⟨times|⟨div|*expr-1*|*expr-2*⟩|*expr-2*⟩|⟨mod|*expr-1*|*expr-2*⟩⟩ evaluiert zu *expr-1*, ⟨div|18|7⟩ zu 2 und ⟨mod|18|7⟩ zu 4.

⟨equal|*expr-1*|*expr-2*⟩

⟨unequal|*expr-1*|*expr-2*⟩
⟨less|*expr-1*|*expr-2*⟩
⟨lesseq|*expr-1*|*expr-2*⟩
⟨greater|*expr-1*|*expr-2*⟩
⟨greatereq|*expr-1*|*expr-2*⟩ (Zahlen oder Längen vergleichen)

Gibt wahr (true) oder falsch (false) zurück für die Tests gleich?, ungleich?, kleiner?, kleiner-oder-gleich?, größer und größer-oder-gleich?. Z.B. ⟨less|123|45⟩ liefert false und ⟨less|123mm|45cm⟩ true.

### 15.5.3. Boolesche Operatoren

⟨or|*expr-1*|···|*expr-n*⟩
⟨and|*expr-1*|···|*expr-n*⟩

Gibt des Ergebnis einer Oder- bzw. Und-Verknüpfung, or/and, der Ausdrücke *expr-1* bis *expr-n* zurück. Z.B. ergibt ⟨or|false|⟨equal|1|1⟩|false⟩ true.

⟨xor|*expr-1*|*expr-2*⟩

Exklusives Oder zweier Ausdrücke *expr-1* und *expr-2*. ⟨xor|true|true⟩ ergibt false.

⟨not|*expr*⟩

Negation von *expr*.

### 15.5.4. Operatoren für Tupel

⟨tuple|*expr-1*|···|*expr-n*⟩ (Konstruktion eines Tupels)

Macht einen Tupel aus den Ausdrücken *expr-1* bis *expr-n*.

⟨is-tuple|*expr*⟩ (Ein Tuple?)

Testet, ob *expr* zu einem Tuple evaluiert.

⟨length|*expr*⟩ (Dimension eines Tupels)

Wenn *expr* ein Tuple ist, wird seinen Dimension ausgegeben. ⟨length|⟨tuple|hop|hola⟩⟩ evaluiert zu 2.

⟨look-up|*tuple*|*which*⟩ (Element eines Tupels extrahieren)

Gibt das Element an der Position *which* (Ganzzahl mit 0) in dem Tupel *tuple* zurück. Die Zählung der Elemente beginnt mir 0. ⟨look-up|⟨tuple|a|b|c⟩|1⟩ gibt b.

⟨range|*expr*|*start*|*end*⟩ (Extrahiere ein Untertupel)

Gibt ein Untertupel von *expr* zurück, welches an der Position *start* beginnt und an der Position *end* endet. Das Element an der Position *end* ist nicht mehr dabei. ⟨range|⟨tuple|a|hola|hop|b|c⟩|2|4⟩ evaluiert zu ⟨tuple|hop|b⟩.

⟨merge|*expr-1*|···|*expr-n*⟩ (Tupel verketten)

Mehrere Tupel *expr-1* bis *expr-n* werden unter Erhalt der Reihenfolge zu einem Tupel zusammengefasst. ⟨merge|⟨tuple|1|2⟩|⟨tuple|3|4|5⟩⟩ ergibt ⟨tuple|1|2|3|4|5⟩.

## 15.6. Darstellung und Aktivität von Stil-Konstrukten

Die Befehle, die in diesem Abschnitt beschrieben werden, dienen zur Steuerung der Darstellung von Stildefinitionen und Stil-Elementen. Sie enthalten sowohl Befehle zur Darstellung wie als auch zur Aktivierung und Desaktivierung von Code.

⟨active|*content*⟩
⟨active*|*content*⟩
⟨inactive|*content*⟩
⟨inactive*|*content*⟩ (Aktivierung/Desaktivierung von Inhalten)

Diese Befehle (Tags) dienen dazu, die Aktivität von Inhalten *content* vorübergehend oder permanent zu ändern. In gewöhnlichen Dokumenten sind Befehle aktiv gemäß Voreinstellung. In Stildefinitionen dagegen ist die Voreinstellung inaktiv. Inaktiv wird ⟨frac|1|2⟩ als ⟨frac|1|2⟩ dargestellt, aktiv $\frac{1}{2}$.

active und inactive aktivieren oder deaktivieren nur die Wurzel des Inhalts *content*. Gewöhnlich kann ein Tag, der verborgene Informationen enthält wie z.B. hlink dadurch deaktiviert werden, dass der Cursor direkt dahinter positioniert wird und dann die `backspace`-Taste gedrückt wird. Damit wird der Hyperlink in einen inaktiven Tag der Form ⟨inactive|⟨hlink|*body*|*ref*⟩⟩ transformiert.

Die Varianten active* und inactive* dienen zum Aktivieren bzw. Deaktivieren des gesamten Inhalts *content*, außer, wenn sich weitere Aktivierung- bzw. Deaktivierung-Tags innerhalb von *content* befinden. inactive* wird häufig in der vorliegenden Dokumentation benutzt, um die inaktive Darstellung von TeXmacs-Code zu zeigen. Manchmal ist es notwendig, einen bestimmten Unter-Baum innerhalb von inaktivem Inhalt zu aktivieren, was man mit active* machen kann. Z.B. enthält das folgende Stück mit inactive* inaktivierten Codes

⟨assign|*love*|⟨macro|*from*|♡♡♡ from *from*.⟩⟩

den mit active* reaktivierten Ausdruck ♡♡♡.

⟨inline-tag|*name*|*arg-1*|⋯|*arg-n*⟩ (Darstellung von Zeilen-Befehlen)

Dieser Befehl dient zur Vorgabe der Darstellung eines inaktiven Tags mit dem Namen *name* und den Argumenten *arg-1* bis *arg-n*. ⟨inline-tag|foo|x|y⟩ erzeugt z.B. ⟨foo|x|y⟩. Der Darstellungsstil kann im Menü Dokument → Ansicht → Codeformatierung oder durch Modifizierung der Kontext-Variablen *src-style*, *src-special*, *src-compact* und *src-close* angepasst werden.

⟨open-tag|*name*|*arg-1*|⋯|*arg-n*⟩
⟨middle-tag|*name*|*arg-1*|⋯|*arg-n*⟩
⟨close-tag|*name*|*arg-1*|⋯|*arg-n*⟩ (Darstellung mehrzeiliger Konstrukte)

Diese Befehle haben ähnliche Aufgaben wie inline-tag allerdings für den Fall, das sich die Argumente über mehrere Zeilen erstrecken. Typische HTML-ähnliche Tags entsprechen ⟨open-tag|*name*⟩ und ⟨close-tag|*name*⟩. Da aber TeXmacs-Makros mehr als ein Argument haben können, gibt es den middle-tag-Befehl für die Darstellung von Argumente. Außerdem können diese Befehle zusätzliche Zeilen-Argumente annehmen. Beispielsweise wird der Code

⟨open-tag|theorem⟩
⟨indent|Das Wetter sollte heute schön werden.⟩
⟨close-tag|theorem⟩

so dargestellt

⟨theorem|

Das Wetter sollte heute schön werden.

⟩

Die Darstellung kann analog zu inline-tag gesteuert werden.

⟨style-with|*var-1*|*val-1*|⋯|*var-n*|*val-n*|*body*⟩
⟨style-with*|*var-1*|*val-1*|⋯|*var-n*|*val-n*|*body*⟩ (Änderung der Darstellung von Stildefinitionen)

Dieser Befehl dient zur zeitweiligen Änderung von inaktiven Befehlen, indem lokal innerhalb des Schriftsatzes des Rumpfes, *body*, die Variablen *var-i* auf *val-i* gesetzt werden. Wenn eine Stil-Definition importiert wird, wird jeder style-with/style-with*-Befehl durch seinen Rumpf. *body*, ersetzt. Bei style-with ist die modifizierte Darstellung auf die Wurzel des Rumpfes beschränkt. Im Fall von style-with* erstreckt die Wirkung auf den ganzen Rumpf *body*.

⟨style-only|⟨foo|*content*⟩⟩
⟨style-only*|⟨foo|*content*⟩⟩ (Inhalt für Stildefinitionen)

Die Befehl dient dazu, einen inaktiven Befehl so darzustellen, als ob das Makro foo darauf angewendet worden sei. Wenn eine Stil-Definition importiert wird, wird jeder style-only/style-only*-Befehl durch seinen *content* ersetzt. Bei style-only ist die modifizierte Darstellung auf die Wurzel von *content* beschränkt. Im Fall von style-only* erstreckt die Wirkung auf den ganzen *content*.

⟨symbol|*symbol*⟩
⟨latex|*cmd*⟩
⟨hybrid|*cmd*⟩
⟨hybrid|*cmd*|*arg*⟩ (Hilfsbefehle zur Eingabe von speziellem Inhalt)

Die Befehle sind werden nur während der Eingabe von speziellen Inhalten verwendet.

Wenn `C-q` gedrückt wird, wird ein symbol-Befehl erzeugt. Nachdem der Name eines Symbols oder sein ASCII-Code eingegeben wurde und die `Rücklauf`-Taste gedrückt wurde, wird der Symbol-Befehl durch das entsprechende Symbol ersetzt, das ist normalerweise eine Zeichenkette in spitzen Klammern <>.

Wenn man `\` eingibt, wird ein hybrid-Befehl erzeugt. Nachdem eine Zeichenkette eingegeben und die `Rücklauf`-Taste gedrückt wurde, wird festgestellt, ob es sich bei der Zeichenkette um einen LaTeX-Befehl, einen Makro-Befehl oder eine Kontext-Variable handelt (in dieser Reihenfolge). Ist dies der Fall, so wird der hybrid-Befehl durch den zutreffenden Inhalt ersetzt. Wenn man `\` eingibt, während eine Auswahl aktiv ist, wird die Auswahl automatisch das Argument des hybrid-Befehls oder der Befehl selbst, wenn er erkannt wird.

Der latex-Befehl arbeitet ähnlich wie der hybrid-Befehl, nur dass der Befehl ausschließlich LaTeX-Befehle erkennt.

Die Darstellungs-Makros für Quell-Code sind fest in $\mathrm{T_EX_{MACS}}$ eingebaut. Aber sie sollten eigentlich nicht als fundamentale Konstrukte betrachtet werden. Sie sind aber kein Teil irgendeiner Stil-Definition.

⟨indent|*body*⟩ (Einzüge)

Setze *body* mit Einzug.

⟨rightflush⟩ (Blockkontext bis zum rechten Rand verbreitern)

Das Makro sorgt dafür, dass ein Block-Kontext, nicht notwendigerweise der sichtbare Inhalt, sich über die ganze verfügbare Breite erstreckt. Das gibt ein besseres Layout für die Informationboxen im Editor und hilft beim Positionieren des Cursors.

⟨src-macro|*macro-name*⟩
⟨src-var|*variable-name*⟩
⟨src-arg|*argument-name*⟩
⟨src-tt|*verbatim-content*⟩
⟨src-integer|*interger*⟩
⟨src-length|*length*⟩
⟨src-error|*message*⟩ (Syntaktische Hervorhebungen)

Diese Makros dienen zur syntaktischen Hervorhebung von Code. Sie bestimmen, wie Unter-Bäume, die Makro-Namen, Variablen-Namen, Argument-Namen, wörtlichem Inhalt, Ganzzahlen, Längen und Fehlermeldungen dargestellt werden sollen.

⟨src-title|*title*⟩
⟨src-style-file|*name*|*version*⟩
⟨src-package|*name*|*version*⟩
⟨src-package-dtd|*name*|*version*|*dtd*|*dtd-version*⟩ (administrative Stil-Funktionen)

Diese Makros dienen zur Identifikation von Stil-Definitionen, -Paketen und der zugehörigen D.T.D.s. src-title ist ein Container für src-style-file, src-package, src-package-dtd, src-license und src-copyright.

src-style-file spezifiziert den Namen *name* und Version *version* einer Stil-Definition und setzt die Kontext-Variable *name-style* auf *version*. src-package-dtd spezifiziert den Namen *name* und die Version *version* eines Pakets sowie die zugehörige D.T.D. *dtd* und seine Version *dtd-version*. Es setzt die Kontext-Variable *name-package* auf *version* und *dtd-dtd* auf *dtd-version*. Der src-package -Befehl ist eine Kurzversion von src-package-dtd für alle Fälle, in denen der D.T.D.-Name mit dem Namen des Pakets übereinstimmt.

## 15.7. Sonstige Stil-Konstrukte

⟨extern|*scheme-foo*|*arg-1*|···|*arg-n*⟩ (Scheme-Makros verwenden)

Dieses Konstrukt dient zur Implementierung von Makros, die in Scheme geschrieben sind. Die Scheme-Funktion *scheme-foo* bzw. das Scheme-Makro *scheme-foo* wird auf die Argumente *arg-1* bis *arg-n* angewandt. Der Code ⟨extern | `(lambda (name) ‘(concat "Hallo " ,name))`|Emil⟩ liefert beispielsweise „Hallo Emil".

Die Argumente *arg-1* bis *arg-n* werden evaluiert und dann als Bäume an *scheme-foo* übergeben. Wenn man ein Makro schreibt, dass externen Scheme-Code benutzt, sollte man also die Argumente unter Benutzung des quote-arg Konstrukts übergeben:

```
⟨assign|inc-div|
    ⟨macro|x|y|
        ⟨extern|
            (lambda (x y) ‘(frac ,x (concat "1+" ,y)))|
            ⟨quote-arg|x⟩|
            ⟨quote-arg|y⟩⟩⟩⟩
```

Es war an sich vorgesehen, dass die Erreichbarkeit von Makro-Argumenten auch hier erhalten bleibt. Da aber $\text{T}_\text{E}\text{X}_\text{MACS}$ SCHEME-Code nicht heuristisch untersucht, muss man die D.R.D.-Eigenschaften mit drd-props von Hand setzen.

Man beachte ferner, dass die SCHEME-Funktion *scheme-foo* nur sichere SCHEME-Funktionen verwendet werden und nicht etwa solche, die Ihre Festplatte löschen. SCHEME-Funktionen in Plugins, die ein Anwender implementiert hat, können mit der Option `:secure` als sicher definiert werden. Es wird dann auf eine Rückfrage verzichtet. Alternativ kann man natürlich auch alle SCHEME-Funktionen im Menü Bearbeiten → Einstellungen → Sicherheit → Alle Scripts akzeptieren als sicher akzeptieren.

⟨write|*aux*|*content*⟩ (Zusätzlich Informationen zu Quellcode)

Diese verborgenen Zusatzinformationen werden nur im Quell-Modus dargestellt.

⟨flag|*content*|*color*⟩
⟨flag|*content*|*color*|*var*⟩ (informatorische Flags)

Diese Konstrukte dienen dazu, dem Anwender sichtbare Informationen zu geben, die nicht ausgedruckt werden sollen. $\text{T}_\text{E}\text{X}_\text{MACS}$ zeigt solche informatorischen Flags für Label, Formatierbefehle, wie Seitenumbrüche usw.. Im Menü Dokument → Ansicht → Informative Flags kann der Anwender einstellen, wie solche Flags dargestellt werden sollen.

Die Variante mit zwei Argumenten gibt ein informatives Flag mit einem spezifischen Inhalt *content* und Farbe *color*. Der *content* wird nur gezeigt, wenn die Darstellungsweise von informativen Flags auf mit Info eingestellt ist, z.B. im Menü Dokument → Ansicht → Informative Flags → Mit Info. Beispielsweise wird ⟨flag|warning|red⟩ im Text in der Voreinstellung als

und mit Einstellung „mit Info“ so  dargestellt. Das optionale Argument *var* dient zur Steuerung der Sichtbarkeit. Wenn *var* zu einem erreichbaren Dokumentteil gehört, dann wird es dargestellt, sonst nicht. $\text{T}_\text{E}\text{X}_\text{MACS}$ generiert automatisch Abschnitts-Labels, damit sie in das Inhaltsverzeichnis aufgenommen werden können. Es ist aber unschön, informatorische Flags in diesen Fällen zu zeigen.

## 15.8. INTERNE KONSTRUKTE

Die hier präsentierten Konstrukte sind nur für den internen Gebrauch von $\text{T}_\text{E}\text{X}_\text{MACS}$ gedacht. Sie werden überhaupt nur der Vollständigkeit wegen erwähnt. Sie sollten sie nur gebrauchen, wenn Sie wirklich wissen, was sie tun und auch dann nur mit großer Sorgfalt.

⟨unknown⟩ (Unbekannter Inhalt oder uninitialisierte Daten)

Dieses Konstrukt dient dazu, uninitialisierte Kontextvariablen zu kennzeichnen.

⟨error|*message*⟩ (Fehlermeldung)

Dies sollte in Dokumenten nie erscheinen. Es dient dazu, falsche Fehler in Konstrukten aufzufinden. Es wird während der Evaluierung von Konstrukten generiert, denen unzulässige Operanden übergeben werden.

⟨collection|*binding-1*|⋯|*binding-n*⟩
⟨associate|*key*|*value*⟩ (Hash-Tabellen)

Das collection-Konstrukt dient zur Definition von Hash-Tabellen mit den Elementen *binding-1* bis *binding-n*. Jedes Element hat die Form ⟨associate|*key*|*value*⟩ mit einem Schlüssel *key* und dem dazugehörigen Wert *value*.

⟨attr|*key-1*|*val-1*|⋯|*key-n*|*val-n*⟩ (XML-artige Attribute)

Dieses Konstrukt wurde eingefügt, um in der Zukunft Kompatibilität mit XML zu haben. Es dient zur Codierung von XML-Stil-Attributen durch TeXmacs-Bäume. Das XML-Fragment

```
<blah color="blue" emotion="verbose">
  Some XML stuff
</blah>
```

würde beispielsweise durch

```
⟨blah|⟨attr|color|blue|emotion|verbose⟩|Some XML stuff⟩
```

in TeXmacs codiert.

⟨tag|*content*|*annotation*⟩
⟨meaning|*content*|*annotation*⟩ (Einen Inhalt mit einer Bedeutung versehen)

Einem Inhalt *content* eine bestimmte Bedeutung hinzufügen. Zur Zeit werden diese Konstrukte praktisch nicht benutzt.

⟨backup|*save*|*stack*⟩ (Werte auf dem Stack sichern)

Dient zur zeitweiligen Sicherung von Werten auf dem Stack.

⟨dbox⟩ (Markierung für Dekorationen)

Dieses Konstrukt ist für den ausschließlichen internen Gebrauch durch die Konstrukte datoms, dlines und dpages gedacht.

⟨rewrite-inactive|*t*|*var*⟩ (Internes Konstrukt zur Darstellung von inaktiven Befehlen)

Dieses interne Konstrukt schreibt inaktive Bäume in neue Bäume um, deren Darstellung dem inaktiven Bäumen entspricht.

⟨new-dpage⟩
⟨new-dpage*⟩ (Neue Doppelseite)

Konstrukt zur Erzeugung einer neuen Doppelseite. Muss erst implementiert werden.

⟨identity|*markup*⟩ (Identitäts-Makro)

Das Identitäts-Makro ist ein Teil von TeXmacs. Es sollte aber eigentlich nicht als fundamentales Konstrukt verstanden werden, obwohl es kein Teil von einer Stil-Definition ist.

Außer diesen Konstrukten gibt es noch weitere, die veraltet sind und nicht mehr von TeXmacs benutzt werden. Man sollte aber vermeiden, ihre Namen bei der Erstellung eigener Makros zu benutzen. Es sind dies: format, line-sep, with-limits, split, old-matrix, old-table, old-mosaic, old-mosaic-item, set, reset, expand, expand*, hide-expand, apply, begin, end, func, env, authorize.

# Kapitel 16

# Die Standard-TeXmacs-Stile

Für jedes Dokument kann man einen Stil für ein Dokument im Menü Dokument → Stil auswählen. Dieser Stil soll die wichtigsten Merkmale eines Dokuments bestimmen und heißt deshalb Basis-Stil. Er entspricht im Allgemeinen der Art des Dokuments, das erstellt werden soll: ein Brief, ein Buch usw. oder einem bestimmten Layout, wie einem ein Artikel für ein bestimmtes Journal. Zusätzlich zu einem Basis-Stil kann der Anwender ein oder mehrere Pakete im Menü Dokument → Paket zufügen. Solche Pakete können den Basis-Stil modifizieren, zusätzliche Hervorhebungen einführen, oder beides.

In diesem Abschnitt werden wir einen Überblick über die Standard-Dokument-Stile geben, die von TeXmacs bereitgestellt werden. Die meisten Basis-Stile und Pakete haben eine abstrakte Schnittstelle, die D.T.D. (data domain definition), die festlegt, welche Makros von dem Stil oder dem Paket exportiert werden und wie sie zu benutzen sind. Unterschiedliche Stile oder Pakete wie z.B. Kopfzeile-Artikel, `header-article`, und Kopfzeile-Buch, `header-book`, können dieselbe D.T.D. benutzen aber in der Darstellung völlig verschieden sein. Deshalb werden wir uns hier vorwiegend mit der Beschreibung der Standard-D.T.D.s beschäftigen, außer wenn wir uns auf die Darstellung konzentrieren. Man kann die Standard-Stile modifizieren, wenn man neue definiert, die der abstrakten Schnittstelle entsprechen (siehe auch den Abschnitt über TeXmacs Stil-Definitionen).

## 16.1. Standard-TeXmacs-Basis-Stile

### 16.1.1. TeXmacs-Standard-Basis-Stile

Die wichtigsten Stile von TeXmacs sind folgende:

`generic`**. = allgemein:** Dieser Basis-Stil ist die Vorgabe, wenn Sie ein neues Dokument erzeugen. Der Sinn dieses Stils ist es, einfach simple Dokumente zu schreiben. Darum gibt es keine Nummerierung von Abschnitten. Die Absätze werden durch einen zusätzlichen vertikalen Leerraum gekennzeichnet, nicht durch Einrückungen.

`article`**. = Artikel:** Dieser Basis-Stil dient zum Schreiben kurzer wissenschaftlicher Veröffentlichungen, die in Abschnitte gegliedert sind. Die Nummerierung von nummerierten Kontexten, wie Satz, Bemerkung, usw. ist fortlaufend im ganzen Dokument. Wenn man das Paket `number-long-article` (= Zahl-lang-Artikel) benutzt, dann erhält die fortlaufende Nummer als Präfix die Abschnitts-Nummer.

`book`**. = Buch:** Dies ist der Basis-Stil zum Schreiben von Büchern. Bücher sind gegliedert in Kapitel. Der Nummerierung in den einzelnen Kapiteln wird die Kapitel-Nummer als Präfix vorangestellt. Im allgemeinen ist es besser, jedes Kapitel in einer eigenen Datei abzuspeichern. Die Editierung wird so viel effizienter. Die damit zusammenhängenden Fragen werden im Abschnitt Bücher, aus mehreren Dateien bestehende Dokumente ausführlicher behandelt.

**seminar.** = **Folien:** Dokumente, die auf diesem Stil basieren, dazu gedacht, auf Folien ausgedruckt und mit dem Overhead-Projektor präsentiert zu werden. Es kann auch sein, dass Sie sie direkt vom Laptop aus, mit dem Beamer projizieren möchten, indem Sie Ansicht → Präsentationsmodus wählen. Beachten Sie aber, dass Folien realen Seiten entsprechen, während man im Präsentationsmodus eher *Schalter* verwenden sollte.

**source.** = **Quellcode:** Das ist der spezielle Stil zum Editieren von Stil-Definitionen und -Paketen. Er schaltet auf den Quellmodus um, damit der Quellcode in einer Weise dargestellt wird, der die Struktur erkennbar macht. Genaueres findet man im Abschnitt Darstellung von Basis-Stil-Dateien und Paketen.

Der Basis-Stil article (Artikel) besitzt verschiedene Varianten, um der unterschiedlichen Layout-Politik verschiedener Journale Rechnung zu tragen. Bisher sind die folgenden Analoga von LaTeX -Stilen implementiert worden.

**amsart.** Basis-Stil der American Mathematical Society. Weitere Informationen unter: http://www.ams.org/tex/author-info.html

**acmconf.** Basis-Stil der Association for Computing Machinery. Weitere Informationen unter: http://www.ams.org/tex/author-info.htmlhttp://www.portal.acm.org

**jsc.** Basis-Stil des Journal of Symbolic Computation. Weitere Informationen unter: http://www.elsevier.com

Wir wollen außerdem Basis-Stile tmarticle und tmbook als Alternativen zu article und book entwickeln.

Zusätzlich zu den Varianten der Basis-Stile article und book liefert $\mathrm{T_EX_{MACS}}$ noch einige andere Stil-Varianten:

**letter.** = **Brief:** Dieser Stil basiert auf generic, besitzt einige zusätzliche Makros hauptsächlich für Briefkopf und -Schluß.

**exam.** = **Examensarbeit:** Dieser Basis-Stil basiert auch auf generic, hat aber zusätzliche Makros für den Kopf und für die Darstellung von Übungen.

**tmdoc.** = **tmdoc:** Dieser Stil dient zum Schreiben der $\mathrm{T_EX_{MACS}}$-Dokumentation und liefert eine Anzahl spezieller Makros. Einige Aspekte sind noch voll in der Entwicklung.

### 16.1.2. $\mathrm{T_EX_{MACS}}$-Standard-Pakete

$\mathrm{T_EX_{MACS}}$ stellt eine Anzahl von Paketen bereit, mit denen sich das Verhalten der Standard-Stile ändern lässt:

**number-long-article.** = **Zahl-lang-Artikel:** Dieses Paket sorgt dafür, dass in allen nummerierten Kontexten (Satz, Bemerkung, Gleichungen, Abbildung, usw.) die Nummern mit der aktuellen Abschnitts-Nummer als Präfix versehen werden. Dieses Paket wird meist mit dem Basis-Stil Artikel, article, oder Buch, book, benutzt.

**number-europe.** = **Zahl-europäisch:** Normalerweise benutzt TeXmacs den Amerikanischen Nummerierung-Stil, d.h., dass ein und derselbe Zähler für alle ähnlichen Zähler wie z.B. Satz oder Proposition verwendet werden. In anderen Worten eine Bemerkung, die auf „Satz 3" folgt, hat die Nummerierung „Bemerkung 4". Wenn Sie für jeden dieser Fälle einen eigenen Zähler haben wollen, müssen Sie das Paket Zahl-europäisch, number-europe, wählen.

**number-us.** = **Zahl-US Stil:** Dieses Paket kann benutzt werden, um zum amerikanischen Stil der Nummerierung zurückzukehren, wenn ein von Dritten stammendes Stil-Paket europäische Nummerierung erzwingt.

**structured-list.** = **structured-Liste:** Das ist ein noch ein Experiment. Normalerweise haben unnummerierte Listen keine Argumente und Punkte in Beschreibungen ein Argument. Wenn man das Paket structured-list verwendet, dann sie können ein weiteres optionales Argument erhalten.

**structured-section.** = **structured-Abschnitt:** Das ist ein noch ein Experiment. Normalerweise haben Abschnitte nur den Titel als Argument. Wenn man structured-section verwendet, können sie ein weiteres Argument annehmen. Außerdem kann das Konstrukt rsection rekursiv verwendet werden.

**varsession.** = **varsession:** Dieses Paket dient dazu, interaktive Sitzungen, bei denen TeXmacs als Schnittstelle und Oberfläche für andere Programme dient, anders darzustellen. Die Darstellung ist für interaktive Sitzungen geeignet, aber weniger gut für den Druck.

Zusätzlich zu den genannten Paketen und den vielen Paketen für den internen TeXmacs-Gebrauch, gibt es in TeXmacs ein paar persönliche Beispiel-Pakete: allouche, bpr und vdh sowie verschiedene Pakete Stil-Pakete zur Benutzung mit externen Plug-Ins (axiom, giaca, macaulay2, usw.).

## 16.2. Die gemeinsame Basis der meisten Stile

Die std D.T.D. enhält die Konstrukte, die praktisch allen Stilen zu Grunde liegen. Sie ist in die folgenden Teile gegliedert:

### 16.2.1. Standard-Kontexte

In std-markup wird eine größere Anzahl von unterschiedlichen Kontexten definiert. Die folgenden Text-Befehle haben alle ein Argument. Die meisten können im Menü Text → Inhaltliche Auszeichnung gefunden werden.

⟨strong|*content*⟩ = **stark**

Geeignet zur Hervorhebung von **wichtigen** Textstellen. Sie können diesen Tag mit Text → Inhaltliche Auszeichnung → Stark einfügen.

⟨em|*content*⟩ = **em**

Hebt Text mit der Schriftart *Italic* hervor. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Hervorheben.

⟨dfn|*content*⟩ = **Definition**

Für *Definitionen*. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Definition.

⟨samp|*content*⟩ = **besonderes Textstück**

Eine Sequenz von Zeichen wie z.B. ae ligature æ. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Beispiel.

⟨name|*content*⟩ = **Name**

Der Name, die Bezeichnung von etwas (nicht einer Person), z.B. Linux. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Name.

⟨person|*content*⟩= **Person**

Der Name einer person, z.B. Joris. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Person.

⟨cite*|*content*⟩= **Zitat***

Ein bibliographisches Zitat im Text (Hinweis auf ein Buch oder Zeitschrift). Z.B. Melville's *Moby Dick*. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Zitat. **Achtung!**, dies sollte nicht mit cite verwechselt werden. Letzteres ist dient zum Referenzieren von Literaturzitaten, die sich in einer besonderen Datei befinden.

⟨abbr|*content*⟩ = **abbr**

Eine Abkürzung, z.B. Ich arbeite am C.N.R.S. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Abkürzung, Kann auch mit `A-a` eingefügt werden.

⟨acronym|*content*⟩ = **Akronym**

Ein Akronym ist eine Abkürzung, die aus den ersten Buchstaben jedes Wortes eines Namens oder eines Satzes besteht, z.B. HTML oder IBM. Die Buchstaben werden nicht durch Punkte getrennt. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Akronym.

⟨verbatim|*content*⟩ = **wörtlich**

Wörtlicher Text, wie z.B. die Ausgabe eines anderen Computer-Programm: Das Programm sagte: `Hallo`. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Wörtlich. Der Befehl kann auch als Kontext für Text benutzt werden, der aus mehreren Absätzen besteht.

⟨kbd|*content*⟩ = **kbd**

Text, der über die Tastatur eingegeben werden soll. Bitte geben Sie „`end`“ ein. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Tastatur.

⟨code*|*content*⟩ = **Quellcode***

Quellcode von Programmiersprachen, z.B. „`cout << 1+1;` gibt 2”. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Quellcode. Für mehrzeiligen Quellcode sollte code benutzt werden.

⟨var|*content*⟩ = **var**

Variablen in einem Computerprogramm, wie z.B. in „`cp` ***`src-datei ziel-datei`***“. Entspricht dem Menü-Eintrag Text → Inhaltliche Auszeichnung → Variable.

⟨math|*content*⟩ = **Math**

Dieser Befehl wird in Zukunft für Mathematik innerhalb von normalem Text Verwendung finden. Beispiel: Die Formel $\sin^2 x + \cos^2 x = 1$ ist gut bekannt.

⟨op|*content*⟩ = **op**

**op** dient dazu Operatoren innerhalb von mathematischen Texten benutzt werden, um hervorzuheben, dass dieser Operator allein, ohne Argumente steht wie z.B. $\mathbb{R}$ in: „Die + Operation ist eine Funktion von $\mathbb{R}^2$ in $\mathbb{R}$“. Dieser Kontext wird möglicherweise überflüssig.

⟨tt|*content*⟩ = **tt**

Hiermit wird die Schrift auf `Schreibmaschine` umgestellt. Es wird zur Kompatibilität mit HTML gebraucht, wir raten von der Verwendung ab. Kann über das Menü Formate → Variante → Schreibmaschine eingestellt werden.

Standard-Kontexte:

⟨verbatim|*body*⟩ = **wörtlich**

Bereits oben beschrieben.

⟨code|*body*⟩ = **Quellcode**

Ähnlich code*, aber für mehrzeiligen Quellcode.

⟨quote-env|*body*⟩ = **quote-env**

Formatierung von Zitaten, die aus einem einzigen Absatz bestehen.

⟨quotation|*body*⟩ = **Zitat**

Formatierung von Zitaten, die aus mehreren Absätzen bestehen.

⟨verse|*body*⟩ = **Vers**

Formatierung von Poesie.

⟨center|*body*⟩ = **zentriert**

Damit kann eine oder auch mehrere Zeilen zentriert dargestellt werden. Es dient der Kompatibilität mit HTML, wir raten von der Verwendung ab. Kann über das Menü Formate → Ausrichtung → Zentriert eingestellt werden.

Einige Standard Tabellen-Formate:

Links ausgerichtete (normale) Tabelle ohne sichtbare Zellränder (Gitter). Tabellen können mit dem Menü Einfügen → Tabelle → Normaler Tabulatormodus so formatiert werden.

⟨tabular*|*table*⟩ = **tabular***

Zentrierte Tabelle ohne sichtbare Zellränder (Gitter). Tabellen können mit dem Menü Einfügen → Tabelle → Zentrierter Tabulatormodus so formatiert werden.

⟨block|*table*⟩ = **block**

Links ausgerichtete Tabelle mit sichtbaren Standard-Zellrändern (Gitter, `1ln`). Tabellen können mit dem Menü Einfügen → Tabelle → Normaler Block so formatiert werden.

⟨block*|*table*⟩ = **block***

Zentrierte Tabelle mit sichtbaren Standard-Zell-Rändern (Gitter, `1ln`). Tabellen können mit dem Menü Einfügen → Tabelle → Zentrierter Block so formatiert werden.

Folgende Tags haben keine Argumente:

⟨TeXmacs⟩

Das TeXmacs-Logo.

⟨TeXmacs-version⟩

Diese Version von TeXmacs, (z.B.: 1.0.5).

⟨TeX⟩

Das TeX-Logo.

⟨LaTeX⟩

Das LaTeX-Logo.

⟨hflush⟩, ⟨left-flush⟩, ⟨right-flush⟩

Wird von Entwicklern gebraucht, um bei der Kontext-Entwicklung Randmarkierungen/Hilfslinien anzupassen.

⟨hrule⟩

Ein horizontaler Strich auf der Basislinie wie der da unten:

---

Die folgenden Tags haben ein oder mehrere Argumente:

⟨overline|*content*⟩ = **überstrichen**

Versieht den Inhalt mit einem Überstrich, der mehrere Zeilen überstreichen kann. Kann mit Text → Schriftform → Überstrichen eingefügt werden.

⟨underline|*content*⟩ = **unterstrichen**

Versieht den Inhalt mit einem Unterstrich, der mehrere Zeilen überstreichen kann. Kann mit Text → Schriftform → Unterstrichen eingefügt werden.

⟨fold|*summary*|*body*⟩ = **zusammenfalten**

*summary* wird auf dem Bildschirm dargestellt und *body* verborgen. Mit Einfügen → Switches → Verborgenen Text zeigen kann der versteckte Text *body* sichtbar gemacht werden. Mit der Tastenkombination `M-A-Bild_nachunten` kann das gleiche erreicht werden. Ganz entsprechend verbirgt `M-A-Bild_nachunten` den Text. *summary* ist meist eine kurze Überschrift oder Feststellung, *body* eine kurzer Text oder eine Erläuterung. Wird meist in Präsentationen benutzt.

⟨unfold|*summary*|*body*⟩ = **verborgenen Text zeigen**

Die Darstellung von ⟨fold|*summary*|*body*⟩, in der der gesamte Text (*summary* und *body*) gezeigt wird. Das zweite Argument *body* kann mit Einfügen → Switches → Zusammenfalten unsichtbar gemacht werden. Genaueres siehe oben.

⟨switch|*current*|*alternatives*⟩ = **Ebene einfügen**

Mehrere Textebenen (Text-Alternativen), von denen jeweils nur eine gezeigt wird. *current* ist die sichtbare Ebene. *alternatives* hat die Form: ⟨tuple|⟨tmarker⟩|⟨*document*|`alt2`⟩|⟨*document*|`alt3`⟩⟩, wobei angenommen wurde, dass *current* die erste Alternative ist. Die Tastenkombination `M-A-nachoben` schaltet zur ersten Ebene, `M-A-nachunten` zur letzten, `M-A-links` um eine nach vorne und `M-A-rechts` um eine nach hinten. Es können auch die Menübefehle Einfügen → Switches → Zur vorherigen Ebene bis Einfügen → Switches → Zur letzten Ebene benutzt werden.

Zur Erzeugung einer ersten Ebene dient Einfügen → Switches → Ebene einfügen, die dann mit Text versehen werden kann. Wenn man in der Ebene ist, lassen sich weitere mit Einfügen → Switches → Ebene davor einfügen und Einfügen → Switches → Ebene danach einfügen oder mit Einfügen → Switches → Entferne diese Ebene entfernen.

⟨superpose|*text1*|*text2*⟩ = **überlagern**

*text1* wird von *text2* überlagert, z.B. „⟨superpose| 000 |+++++⟩“ gibt „+000+“.

⟨phantom|*content*⟩ = **phantom**

Dieses Makro setzt anstelle des Inhaltes *content* einen gleichlangen Leerraum - der Inhalt wird also nicht gezeigt oder gedruckt, ergibt „⟨phantom | phantom⟩“ das: „        “.

⟨set-header|*header-text*⟩ = **setze-Kopfzeile**

Ein Makro, mit dem die Kopfzeile permanent geändert werden kann. Bestimmte Konstrukte in Basis-Stil-Definitionen haben Vorrang diesem Befehl.

⟨set-footer|*footer-text*⟩ = **setze-Fußzeile**

Ein Makro, mit dem die Fuß-Zeile permanent geändert werden kann. Bestimmte Konstrukte in Basis-Stil-Definitionen haben Vorrang diesem Befehl.

### 16.2.2. Standard-Symbole

Die `std-symbol` D.T.D. definiert die speziellen Symbole ¢, ¤, ¥, ©, ↄ⃝, ®, °, ², ³, ¹, µ, ¶, ¼, ½, ¾, € und ™. Wenn die Unterstützung von Schriftarten gut genug geworden ist, wird diese D.T.D. überflüssig werden.

### 16.2.3. Standard-Mathematik

`std-math` definiert die mathematischen Standard-Formate:

⟨binom|*among*|*nr*⟩

Binomialkoeffizienten, z.B. mit *among* = n und *nr* = m: $\binom{n}{m}$.

⟨choose|*among*|*nr*⟩

Alternative für binom, aber überholt.

⟨shrink-inline|*among*|*nr*⟩

Ein Makro, das auf die Größe von Index-Stilen schrumpft, wenn es nicht in eigenständigen Formeln verwendet wird. Dieses Makro wird hauptsächlich von Entwicklern benutzt. Z.B. setzt das Makro binom dieses Makro ein.

Außerdem werden die folgenden tabellarischen Standard-Formate definiert:

⟨matrix|*table*⟩

Für Matrizen wie z.B. $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

⟨det|*table*⟩

Für Determinanten wie z.B. $\Delta = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$.

⟨choice|*table*⟩

Für mehrfache Bedingungen o.ä. $|x| = \begin{cases} -x, & \text{if } x \leqslant 0 \\ x, & \text{if } x \geqslant 0 \end{cases}$.

## 16.2.4. Standard-Listen

### 16.2.4.1. Listenkontext benutzen

Die Standard-TeXmacs-Listen werden in `std-list` definiert.

Es gibt die folgenden unnummerierten Listen:

⟨itemize|*body*⟩

Die Marke vor jedem Punkt hängt von der Tiefe der Gliederung ab (LaTeX-Stil).

⟨itemize-minus|*body*⟩

Benutzt $-$ .

⟨itemize-dot|*body*⟩

Benutzt $\bullet$ .

⟨itemize-arrow|*body*⟩

Benutzt $\rightarrow$ .

Nummerierte Listen:

⟨enumerate|*body*⟩

Die Art der Nummerierung hängt von der Tiefe der Gliederung ab (LaTeX-Stil).

⟨enumerate-numeric|*body*⟩

Nummeriere so: 1, 2, 3 usw.

⟨enumerate-roman|*body*⟩

Nummeriere so: i, ii, iii usw.

⟨enumerate-Roman|*body*⟩

Nummeriere so: I, II, III usw.

⟨enumerate-alpha|*body*⟩

„Nummeriere" so: a), b), c) usw.

⟨enumerate-Alpha|*body*⟩

„Nummeriere” so: A), B), C) usw.

Beschreibende Auflistungen:

⟨description|*body*⟩

Das Konstrukt für die Vorgabe beschreibender Aufzählungen (meist description-compact).

⟨description-compact|*body*⟩

Stelle die Punkte der Liste linksbündig dar, dann das Trennzeichen **.** und positioniere den Text anschließend einem kurzem Abstand.

⟨description-dash|*body*⟩

Ähnlich description-compact, mit Trennzeichen — .

⟨description-align|*body*⟩

Das Trennzeichen **.** wird an einer festen Stelle gesetzt. Rechts davon der Text. Links vom Trennzeichen wird rechtsbündig zum Trennzeichen die Beschreibung gesetzt.

⟨description-long|*body*⟩

Setzt die Beschreibung und den eigentlichen Text auf zwei verschiedene Zeilen.

Neue Punkte in einer Liste werden mit item oder im Fall von beschreibenden Auflistungen mit item* eingefügt. item hat keine Argumente, der Text folgt anschließend an den Befehl, item* wie item, hat ein Argument die Beschreibung. Wenn das experimentelle `structured-list` Paket benutzt wird können diese Befehle ein weiteres optionales Rumpfargument erhalten. In Zukunft sollen alle „item“-Tags so sein.

Als Vorgabe werden alle Unterlisten so nummeriert wie in den normalen Listen. Jeder Listen-Kontext *list* besitzt aber die Variante *list**, deren Punkte als Präfix den zugehörigen Punkt in der Hauptliste erhalten.

#### 16.2.4.2. Listenkontexte anpassen

`std-list` stellt die folgenden redefinierbaren Makros zur Darstellung von Listen und Punkten in den Listen bereit:

⟨render-list|*body*⟩

Dieses Block-Konstrukt dient zur Darstellung des Rumpfes *body* einer Liste. Normalerweise wird die Listen eingerückt und vertikaler Leerraum vor und dahinter eingefügt.

⟨aligned-item|*item-text*⟩

Dieses Zeilen-Konstrukt dient zur rechtsbündigen Darstellung von *item-text*, der Trennmarke bzw. der Beschreibung. Der eigentliche Text erscheint dahinter linksbündig.

⟨compact-item|*item-text*⟩

Dieses Zeilen-Makro dient zur Darstellung von linksbündigem *item-text* der Trennmarke bzw. der Beschreibung. Der eigentliche Text wird daher um die Breite von *item-text* eingerückt. Es sei denn, dieser befindet sich in einem neuen Absatz.

### 16.2.5. Automatisch erzeugte Standard-Verzeichnisse

Die `std-automatic` D.T.D. enthält Makros zur automatischen Erzeugung von Inhalten sowie zu deren Darstellung. Es gibt im Wesentlichen vier Arten von solchem Inhalt in TeXmacs: *Literaturverzeichnisse* (Quellen-Verzeichnisse), *Inhaltsverzeichnisse*, *Indexe* (Stichwortverzeichnisse) und *Glossare* (Wortlisten mit Erklärungen oder internen Verweisen). Andere Typen von automatische generierten Inhalten leiten sich von den vier oben genannten ab, z.B. für das Abbildungs-Verzeichnis wurde der Typ *Glossar* verwendet. Die Darstellung der Abschnitte als Ganzen, die die Literaturverzeichnisse, Inhaltsverzeichnisse, Indexe und Glossare enthalten, werden mit der `section-base` D.T.D. definiert.

#### 16.2.5.1. Literaturverzeichnisse

Die folgenden Makros dienen als Marken im Haupt-Text, die auf ein Zitat-Eintrag in einer bibliographischen „Datenbank" verweisen:

⟨cite|*ref-1*|···|*ref-n*⟩

Jedes Argument *ref-i* ist ein Zitat, das zu einem Eintrag in einer BiB-TeX-Datei gehört. Die Zitate werden genauso dargestellt, wie die zugehörigen Einträge in der Datei stehen. Außerdem stellen sie Hyperlinks zu den Referenzen dar. Wenn der Referenz-Eintrag fehlt, dann erscheint anstelle des Zitats ein Fragezeichen. Kann mit Einfügen → Verknüpfung → Zitat → Sichtbar eingefügt werden.

⟨nocite|*ref-1*|···|*ref-n*⟩

Ähnlich zu cite, aber die Zitate erscheinen nicht im Haupt-Text. Kann mit Einfügen → Verknüpfung → Zitat → Unsichtbar eingefügt werden.

⟨cite-detail|*ref*|*info*⟩

Eine ähnliche bibliographische Referenz wie oben. Zusätzlich zur Marke *ref* können weitere Informationen *info* gegeben werden, z.B. Kapitel- oder Seiten-Nummer. Kann mit Einfügen → Verknüpfung → Zitat → Mit Info eingefügt werden.

Die folgenden Makros können redefiniert werden, wenn Sie die Darstellung von Zitaten im Text oder in dem erzeugten Verzeichnis ändern wollen:

⟨render-cite|*ref*⟩

Ein Makro zur Darstellung des Zitates *ref* am Ort, wo mit cite zitiert wurde. Der Inhalt kann ein einfaches Zitat wie „TM98" sein, oder ein Liste von Referenzen wie z.B. „Euler1, Gauss2".

⟨render-cite-detail|*ref*|*info*⟩

Ähnlich render-cite, aber für ausführliche Zitate mit Zusatzinformation, die mit cite-detail erstellt wurden.

⟨render-bibitem|*content*⟩

Momentan werden Bibliographien mit BibTeX gemacht und nach TeXmacs importiert. Die so erzeugten Verzeichnisse sind Listen von bibliographischen Punkten, die auf speziellen LaTeX-spezifischen Makros (bibitem, block, protect, usw.) basieren. Diese Makros sind intern in TeXmacs definiert und rufen im Endeffekt render-bibitem auf, welches ähnlich wie item* arbeitet und welches vom Anwender umdefiniert werden kann.

#### 16.2.5.2. Inhaltsverzeichnisse

Die folgenden Makros dienen im Haupt-Text zur Einfügung von Einträgen in das Inhaltsverzeichnis. Diese werden im allgemeinen von den Abschnitts-Makros automatisch erzeugt. Manchmal ist es aber erwünscht, zusätzlich eigene Einträge manuell vorzunehmen.

⟨toc-main-1|*entry*⟩
⟨toc-main-2|*entry*⟩

Erzeuge einen Eintrag *entry* der primären Gliederungsebene in dem Inhaltsverzeichnis. Das Makro toc-main-1 ist für besonders hervorzuhebende Gliederungsstufen zu verwenden, wie z.B. Buchteile oder Bücher einer Folge von Büchern. Es muss in der Regel manuell eingefügt werden. Das Makro toc-main-2 wird üblicherweise als oberste Gliederungsstufe, wie z.B. für Kapitel bei Büchern oder Abschnitte in Artikeln benutzt. Meist wird es **fett** gesetzt.

⟨toc-normal-1|*entry*⟩
⟨toc-normal-2|*entry*⟩
⟨toc-normal-3|*entry*⟩

Diese sind für Einträge *entry* der Gliederungsstufen unter der primären Gliederungsstufe gedacht. toc-normal-1 entspricht in Büchern den Abschnitten, toc-normal-2 Unter-Abschnitten und toc-normal-3 Unter-Unter-Abschnitten.

⟨toc-small-1|*entry*⟩
⟨toc-small-2|*entry*⟩

Trägt Einträge *entry* weiterer tieferer Gliederungsstufen ein, z.B. Absatz. toc-small-1 und toc-small-2 sind oft in den Basis-Stilen nicht vorgesehen und werden einfach ignoriert.

Die folgenden Makros können redefiniert werden, um die Darstellung anzupassen:

⟨toc-strong-1|*content*|*where*⟩
⟨toc-strong-2|*content*|*where*⟩

Diese Makros steuern die Darstellung von toc-main-1 bzw. toc-main-2 an der(n) Seitenzahl(en) *where*.

⟨toc-1|*content*|*where*⟩
⟨toc-2|*content*|*where*⟩
⟨toc-3|*content*|*where*⟩
⟨toc-4|*content*|*where*⟩
⟨toc-5|*content*|*where*⟩

Diese Makros steuern die Darstellung von toc-normal-1, toc-normal-2, toc-normal-3, toc-small-1 bzw. toc-small-2 an der(n) Seitenzahl(en) *where*.

⟨toc-dots⟩

Die Trennzeichen zwischen einem Eintrag und der(den) Seitenzahl(en). Die Vorgabe sind horizontale Punkte.

#### 16.2.5.3. Stichwortverzeichnisse

Die folgenden Makros dienen im Haupt-Text zur Einfügung von Stichwort-Einträgen:

⟨index|*primär*⟩

Trage *primär* als den primären Eintrag in das Stichwortverzeichnis ein.

⟨subindex|*primär*|*sekundär*⟩

Trage *sekundär* als den sekundären (Unter-)Eintrag von *primär* das Stichwortverzeichnis ein.

⟨subsubindex|*primär*|*sekundär*|*tertiär*⟩

Ähnlich subindex aber für den Unter-Unter-Eintrag *tertiär*.

⟨index-complex|*key*|*how*|*range*|*entry*⟩

Trage einen komplexen Eintrag in das Stichwort-Register ein. Dies wird im Abschnitt Stichwortverzeichnisse erzeugen genauer erklärt.

⟨index-line|*key*|*entry*⟩

Fügt den Eintrag *entry* hinzu, sortiert nach *key*.

Die folgenden Makros können redefiniert werden, um die Darstellung anzupassen:

⟨index-1|*entry*|*where*⟩
⟨index-2|*entry*|*where*⟩
⟨index-3|*entry*|*where*⟩
⟨index-4|*entry*|*where*⟩
⟨index-5|*entry*|*where*⟩

Makro zur Darstellung eines Eintrags *entry* auf der(n) Seite(n) *where*. index-1 gehört zu Primär-Einträgen, index-2 zu Sekundär-Einträgen usw..

⟨index-1*|*entry*⟩
⟨index-2*|*entry*⟩
⟨index-3*|*entry*⟩
⟨index-4*|*entry*⟩
⟨index-5*|*entry*⟩

Ähnlich index-1 bis index-5, aber ohne Seitenzahl(en).

⟨index-dots⟩

Makro zur Erzeugung der Punkte zwischen Glossar-Eintrag und der entsprechenden Seitenzahl(en).

#### 16.2.5.4. Glossare

Die folgenden Makros dienen im Haupt-Text zur Einfügung von Glossar-Einträgen:

⟨glossary|*entry*⟩

Einen Eintrag *entry* in das Glossar einfügen.

⟨glossary-dup|*entry*⟩

dient dazu eine zusätzliche Seiten-Nummer für einen Eintrag *entry* hinzuzufügen, der bereits eingetragen wurde.

⟨glossary-explain|*entry*|*explanation*⟩

Ein Glossar-Eintrag *entry* mit einer Erklärung *explanation*.

⟨glossary-line|*entry*⟩

Glossar-Eintrag *entry* ohne Seitenzahl.

Die folgenden Makros können redefiniert werden, um die Darstellung anzupassen:

⟨glossary-1|*entry*|*where*⟩

Makro zur Darstellung eines Glossar-Eintrags und der entsprechenden Seitenzahl(en) *where*.

⟨glossary-2|*entry*|*explanation*|*where*⟩

Makro zur Darstellung eines Glossar-Eintrags, seiner Erklärung und der entsprechenden Seitenzahl(en) *where*.

⟨glossary-dots⟩

Makro zur Erzeugung der Punkte zwischen Glossar-Eintrag und der entsprechenden Seitenzahl(en).

### 16.2.6. Zähler und Zählergruppen

TeXmacs benutzt zum automatischen Nummerieren von Abschnitten, Sätzen usw. Zähler „counter". Solche Zähler können individuelle Zähler sein wie z.B. *equation-nr* oder zu einer Gruppe ähnlicher Zähler gehören wie etwa *theorem-nr*. TeXmacs erlaubt die Anpassung von Zählern auf individueller oder auf Gruppenbasis. Man kann die Darstellung des Zählers ändern, diesen beispielsweise mit römischen Zahlen ausgeben, oder man kann spezielle Aktionen durchführen, dem Zähler z.B. erhöhen oder einen Unterzähler neu initialisieren.

Neue individuelle Zähler können mit dem folgenden Makro definiert werden:

⟨new-counter|*x*⟩

definiert einen neuen Zähler mit dem Namen *x*. Der Zähler wird in dem numerischen Kontext *x-nr* gespeichert. Gleichzeitig werden die folgenden Makros erzeugt:

⟨the-*x*⟩

Greife auf den Zähler zu, so wie er auf dem Bildschirm erscheint.

⟨reset-*x*⟩

Setze den Zähler auf 0.

⟨inc-*x*⟩

Inkrementiere den Zähler um 1. Dieses Makro kann so geändert werden, dass es auch andere Zähler inkrementiert, auch wenn das so in den Standard-Stil-Definitionen nicht gemacht wird.

⟨next-*x*⟩

Inkrementiere den Zähler um 1, zeige den Zähler auf dem Bildschirm und setze das aktuelle Label.

Um Anpassungen zu erleichtern, definiert new-counter zusätzlich die folgenden beiden Makros:

⟨display-*x*|*nr*⟩

Dieses Makro transformiert den numerischen Wert des Zählers in denjenigen, der auf dem Bildschirm gezeigt wird.

⟨counter-*x*|*x*⟩

Dieses interne Makro gibt den Namen desjenigen Kontexts zurück, dem der Zähler angehört. Normalerweise ist dies „nr-x”, es kann aber auch umdefiniert worden sein, wenn der Zähler zu einer Gruppe gehört.

Wie bereits gesagt benutzt TeXmacs *Zählergruppen*, um ähnliche Zähler auf ähnliche Weise zu behandeln. Z.B. gehören zu Zählergruppe `theorem-env` die Zähler `theorem`, `proposition`, `lemma`, usw.. Man definiert neue Zählergruppen werden mit:

⟨new-counter-group|*g*⟩

erzeugt eine neue Zählergruppe mit dem Namen *g* und gleichzeitig folgende Makros:

⟨display-in-*g*|*x*|*nr*⟩
⟨counter-in-*g*|*x*⟩

Diese Makros verhalten sich analog zu den oben beschriebenen Makros display-*x* und counter-*x* jedoch für Zähler der Gruppe *g*. Sie übernehmen als Argument den Namen *x* des Zählers.

Neue Zähler können zur Gruppe zugefügt werden mit:

⟨add-to-counter-group|*x*|*g*⟩

Damit wird ein neuer Zähler *x* definiert und zur Gruppe *g* hinzugefügt. Die Rolle der Makros display-*x* und counter-*x* wird bei Gruppen von den Makros display-in-*g* und counter-in-*g* übernommen. Zusätzlich werden aber zwei weitere Makros definiert ind-display-*x* und ind-counter-*x*, die die Rolle von display-*x* und counter-*x* in denjenigen Fällen übernimmt, in denen die Gruppe aus individuellen Zählern besteht.

Jederzeit kann man entscheiden, ob die Zähler einer Gruppe einen gemeinsamen Gruppenzähler oder sie individuelle Zähler benutzt. Dies wird z.B. gebraucht, wenn man zwischen dem Amerikanischen Nummerierungsstil und dem europäischen wechselt:

⟨group-common-counter|*g*⟩

Benutze einen gemeinsamen Zähler für die Zählergruppe. Dieser wird in der Kontextvariablen *g-nr* gespeichert.

⟨group-individual-counters|*g*⟩

Benutze einen individuellen Zähler für jedes Gruppenmitglied. Das ist die Vorgabe.

Es sei darauf hingewiesen, dass Gruppenzähler rekursiv zu Obergruppen gehören können. Das zeigen z.B. die folgenden Ausdrücke aus `env-base.ts`:

```
⟨new-counter-group|std-env⟩
⟨new-counter-group|theorem-env⟩
⟨add-to-counter-group|theorem-env|std-env⟩
⟨group-common-counter|theorem-env⟩
```

### 16.2.7. Standard-Konstrukte für Programme

Die `program` D.T.D. dient zur Darstellung von Computer-Programmen. Diese Makros sollten aber als sehr unzuverlässig angesehen werden, denn wir planen, sie durch detailliertere zu ersetzen:

⟨algorithm|*name*|*body*⟩

Der Name *name* des Algorithmus und sein Rumpf *body* mit möglichen zusätzlichen spezifizierenden Inhalten.

⟨body|*body*⟩

Der wahre Rumpf des Algorithmus.

⟨indent|*contenta*⟩

Einzüge für Teile eines Algorithmus.

### 16.2.8. Standard-Konstrukte für die Schnittstelle zu Fremdprogrammen

Die `session` D.T.D. erzeugt Kontexte für die Benutzung von $\mathrm{T_EX_{MACS}}$ als interaktive Oberfläche für andere Programme, Computer Algebra (CAS): Sitzung, session.

⟨session|*body*⟩

Erzeugt den Kontext für eine Sitzung.

Alle folgenden Makros sind nur innerhalb eines Sitzungs-Kontextes zulässig:

⟨input|*prompt*|*body*⟩

Ein Eingabefeld mit einer Eingabeaufforderung *prompt* und der wirklichen Eingabe *body*.

⟨output|*body*⟩

Ein Ausgabefeld.

⟨textput|*body*⟩

Feld mit normalem Text, als Kommentar oder Erläuterung geeignet.

⟨errput|*body*⟩

Dieses Makro wird innerhalb eines Ausgabefeldes benutzt, um Fehlermeldungen auszugeben.

Diese Konstrukte basieren auf *lan*-session, *lan*-input, *lan*-output, *lan*-textput und *lan*-errput für jede Programm-Sprache *lan*.

Wenn sprach-spezifische Konstrukte nicht existieren, werden die $\mathrm{T_EX_{MACS}}$-Vorgeben generic-session, generic-input, generic-output, generic-textput und generic-errput an ihrer Stelle benutzt. Wir empfehlen, sprach-spezifische Konstrukte auf Basis dieser Konstrukte zu implementieren. Sie können je nach Stil (`varsession` Paket in Dokument → Paket zufügen → Sitzung) unterschiedlich implementiert sein. Dafür dient das Konstrukt generic-output*, das generic-output entspricht, bei dem aber die Ränder aber unverändert erhalten bleiben.

## 16.3. Standard-Kontexte

Die `env` D.T.D. enthält die Standard-Kontexte, die in den meisten Basis-Stilen angeboten werden. Sie besteht aus den folgenden vier Teilen:

### 16.3.1. Definition neuer Kontexte

Die `env-base` D.T.D. enthält Konstrukte, die man benutzen kann, um neue nummerierte Kontexte, wie z.B. Satz, Bemerkung, Aufgabe und Abbildung:

⟨new-theorem|*env-name*|*display-name*⟩

> Dieses Meta-Makro wird zur Definition neuer nummerierter (Theorem-ähnlicher) Kontexte benutzt. Das erste Argument *env-name* spezifiziert den Namen des Kontextes z.B. „Experimente" und *display-name* den dazugehörigen Text z.B. „Versuch". Wenn ein Theorem-ähnlicher Kontext definiert wird, z.B. Experimente, wird gleichzeitig eine nicht nummerierte Variante Experimente* automatisch erzeugt.

⟨new-remark|*env-name*|*display-name*⟩

> Ähnlich new-theorem, aber für Bemerkung.

⟨new-exercise|*env-name*|*display-name*⟩

> Ähnlich new-theorem, aber für Aufgabe.

⟨new-figure|*env-name*|*display-name*⟩

> Ähnlich new-theorem, aber für Abbildung. Wenn man einen neuen Abbildungs-Typ definiert, z.B. „Gemälde", dann erzeugt das new-figure-Makro gleichzeitig einen Zeilen-Kontext small-picture und einen Block-Kontext big-picture, sowie die nicht nummerierten Varianten small-picture* und big-picture*.

Die nummerierten (Theorem-ähnlichen) Kontexte gehören alle zu der Haupt-Gruppe `theorem-env`. Voreinstellung ist die amerikanische Art der Nummerierung: ein gemeinsamer Zähler für alle Kontexte. Wenn man aber das Paket `number-europe`, Zahl-europäisch, wählen, hat jeder Kontext einen eigenen Zähler. Jede Aufgabe und jede Abbildung benutzt eine eigene Zählergruppe.

Allgemein gilt, dass die `std-env` Zählergruppe die Zähler für alle TeXmacs-Standard-Kontexte umgruppiert. Typischerweise werden alle Gruppen auf ähnliche Weise mit einem Präfix, z.B. der Kapitel-Nummer, versehen. Abbildung 16.1 zeigt die hierarchische Struktur dieser Zählergruppe.

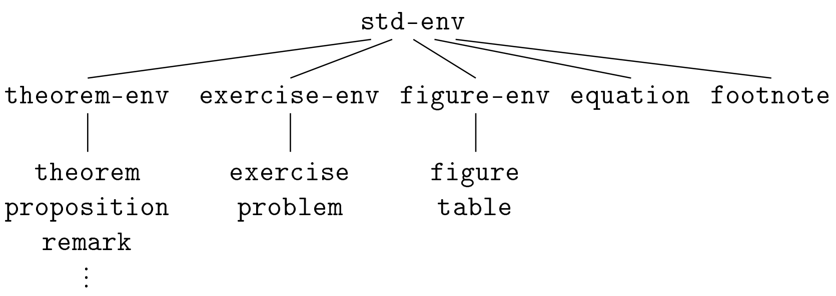



**Abbildung 16.1.** Organisation der Zähler von TeXmacs-Standard-Kontexten.

Zusätzlich zu den Standard-Typen, Satz-ähnliche, Bemerkung-ähnliche, Aufgabe-ähnliche und Abbildung-ähnliche Kontexte, können weitere nummerierte Text-Kontexte mit dem new-env-Makro erzeugt werden. Diese können auf beliebigen Zählergruppen basieren:

⟨new-env|*group*|*env*|*env-name*|*display-name*⟩

Das erste Argument ist der Name der Zählergruppe *group*, zu der der neue Kontext gehören soll. Das zweite Argument *env* ist der Name eines binären Makros zur Darstellung des Kontexts. Die Argumente des darstellenden Makros sind: ein Name (z.B. „Theorem 3.14") und sein Rumpf. Die verbleibenden Argumente entsprechen denjenigen von new-theorem. Beispielsweise wird in den Standard-Basis-Stilen new-theorem so

```
⟨assign|new-theorem|⟨macro|env|name|⟨new-env|env|name|theorem-env|render-theorem⟩⟩⟩
```

definiert.

Es sei daran erinnert, dass man neue Zähler und Zählergruppen zu `theorem-env` mit Hilfe von new-counter-group und add-to-counter-group, wie das in dem Abschnitt über Zähler erklärt wurde.

### 16.3.2. Mathematische Kontexte

Die `env-math` D.T.D. spezifiziert, welche mathematischen Kontexte im Text-Modus benutzt werden können. Mit anderen Worten, diese Kontexte sollen innerhalb von Text eingesetzt werden, ihre Rümpfe enthalten aber mathematische Formeln oder Tabellen von mathematischen Formeln.

⟨equation|*body*⟩

Eine nummerierte Gleichung.

⟨equation*|*body*⟩

Eine unnummerierte Gleichung.

⟨eqnarray|*table*⟩

Ein Array von nummerierten Gleichungen (noch nicht implementiert).

⟨eqnarray*|*table*⟩

Ein Array von unnummerierten Gleichungen.

Innerhalb von eqnarray* kann man eq-number benutzen, um die Gleichung zu nummerieren.

Warnung 16.1. Die Nummerierung von Gleichungen innerhalb von Tabellen funktioniert noch nicht so, wie sie eigentlich sollte. Zur Zeit sind eqnarray und eqnarray* dasselbe. Später soll aber eqnarray korrekt implementiert werden. Dann wird es einen Befehl no-number geben, um die Nummerierung zu unterdrücken sowie ein Stil-Paket, um linke oder rechte Nummerierung zu einstellen zu können.

Warnung 16.2. Es gibt zur Zeit keine Option zur Platzierung der Nummerierung an der linken Seite. Man kann dafür aber manuell leq-number benutzen. Es existiert auch der Befehl next-number, der die nächste Zahl auf dem Bildschirm zeigt und den Zähler um 1 erhöht.

WARNUNG 16.3. Wir raten vom Gebrauch der AMS-TEX-Befehle `align`, `gather` und `split` ab. Trotzdem sind sie unter den Namen align, gather, eqsplit mit den Varianten align*, gather* und eqsplit* vorhanden. Für die Zukunft sind mächtigere Konstrukte geplant.

## 16.3.3. Nummerierte Kontexte

### 16.3.3.1. Nummerierte Kontexte nutzen

Die `env-theorem` D.T.D. enthält die nummerierten (Theorem-ähnlichen) Standard-Kontexte und andere Standard-Text-Kontexte, die über das Menü Text → Umgebung erreicht werden können. Sie können in drei Haupt-Kategorien unterteilt werden:

**Theorem-(Satz)-Varianten.** Die Rümpfe dieser Kontexte werden normalerweise in besonderer Weise hervorgehoben. TEXMACS stellt folgende Kontexte bereit: theorem (Satz), proposition (Proposition), lemma (Lemma), corollary (Folgerung), axiom (Axiom), definition (Definition), notation (Notierung), conjecture (conjecture), die über das Menü Text → Umgebung zugänglich sind.

**Bemerkung-Varianten.** Im MenüText → Umgebung lassen sich folgende erzeugen: remark (Bemerkung), example (Beispiel), note (Anmerkung), warning (Warnung), convention (Konvention).

**Aufgabe-Varianten.** Zwei solche Kontexte besitzt TEXMACS, die über das Menü Text → Umgebung erzeugt werden können: exercise (Aufgabe) und problem (Problem).

Alle diese Kontexte gibt es auch in einer unnummerierten Variante theorem*, proposition*, usw.. Man kann den Kurzbefehl `A-*` benutzen, um zwischen der nummerierten und der unnummerierten Variante zu wechseln. Außerdem gibt es noch:

⟨proof|*body*⟩

Für Beweise (von Sätzen).

⟨dueto|*who*⟩

Ein Makro, das dazu dient die Quelle des Satzes zu kennzeichnen. Es sollte innerhalb eines Satz-Kontexts verwendet werden, z.B.

SATZ. (PYTHAGORAS) $a^2+b^2=c^2$.

### 16.3.3.2. Nummerierte Kontexte anpassen

Die folgenden Makros dienen zur Darstellung von Text-Kontexten. Sie können alle umdefiniert werden, um die Darstellung speziellen Bedürfnissen anzupassen.

⟨render-theorem|*name*|*body*⟩

Dieses Makro dient zur Darstellung von nummerierten, Theorem-artigen Kontexten. Das erste Argument Name *name* gibt den Namen des „Theorem“ (den kennzeichnenden Text), z.B. „Theorem 1.2” und das zweite Argument enthält den Rumpf. Dieser Kontext wird in Konstrukten gebraucht, die mit new-theorem definiert wurden.

⟨render-remark|*name*|*body*⟩

Ähnlich render-theorem, aber für Bemerkungs-artige Kontexte.

⟨render-exercise|*name*|*body*⟩

Ähnlich render-theorem, aber für Aufgabe-artige Kontexte.

⟨render-proof|*name*|*body*⟩

Ähnlich render-theorem, aber für Beweis. Dies wird hauptsächlich dazu benutzt, den Namen von eines Beweises anzupassen, z.B. wie in „Ende des Beweise von Satz 1.2".

Beachten Sie, dass Sie diese Makros dazu benutzen können, einen Kontext zu erzeugen, der nur sich im Namen unterscheidet, z.B. anstelle von Satz Korollar.

Die folgenden Befehle geben weitere Möglichkeiten zur Darstellungs-Anpassung:

⟨theorem-name|*name*⟩

Diese Makro kontrolliert die Darstellung von Namen in Theorem-artigen und Bemerkungs-artigen Kontexten. Die meisten Basis-Stile benutzen **fett** oder Kapitälchen.

⟨exercise-name|*name*⟩

Ähnlich theorem-name, aber für Aufgabe.

⟨theorem-sep⟩

Das Trennzeichen zwischen dem Namen in einem Theorem-artigem und Bemerkungs-artigem Kontext und dem Rumpf. Die Voreinstellung ist ein Punkt mit anschließendem Leerzeichen.

⟨exercise-sep⟩

Ähnlich theorem-sep, aber für Aufgabe.

### 16.3.4. Kontexte für bewegliche Objekte

#### 16.3.4.1. Kontexte für bewegliche Objekte nutzen

Die `env-float` D.T.D. erzeugt die folgenden Kontexte für bewegliche Objekte:

⟨small-figure|*body*|*caption*⟩

Dieses Makro erzeugt ein Zeilen-Objekt mit Abbildung *body* und der Beschriftung *caption*. Zeilen-Abbildungen können z.B. benutzt werden, um mehrere kleinere Bilder nebeneinander innerhalb eines beweglichen Objekts zu setzen.

⟨big-figure|*body*|*caption*⟩

Dieses Makro erzeugt eine große Abbildung, die sich über die ganze Breite eines Absatzes erstrecken kann, mit der eigentlichen Abbildung *body* und der Beschriftung *caption*.

⟨small-table|*body*|*caption*⟩

Ähnlich small-figure, aber für *kleine Tabellen*.

⟨big-table|*body*|*caption*⟩

Ähnlich big-figure, aber für *große Tabellen*.

⟨footnote|*body*⟩

Erzeugt eine Fußnote.

Die Abbildungs-ähnlichen Kontexte haben auch unnummerierte Varianten small-figure*, big-figure*, usw., zu denen man mit dem Kurzbefehl `A-*` wechseln kann und umgekehrt.

#### 16.3.4.2. Kontexte für bewegliche Objekte anpassen

Die folgenden Makros dienen zur Anpassung der Darstellung von Abbildungs-artigen Kontexten:

⟨render-small-figure|*aux*|*name*|*body*|*caption*⟩

Dieses Makro steuert die Darstellung von kleinen Abbildungen (small-figure). Das erste Argument *aux* spezifiziert einen „*auxiliary channel*" (wie z.B. „figure" oder „table"), der dazu benutzt wird, die Beschriftung in die Liste der Abbildungen aufzunehmen. Das zweite Argument spezifizierte den Namen *name*, den kennzeichnenden Textes, wie z.B. „Abbildung 2.3" oder „Tabelle 5". Die letzten Argumente *body* und *caption* sind die Abbildung selbst und die Beschriftung.

⟨render-big-figure|*aux*|*name*|*body*|*caption*⟩

Ähnlich render-small-figure, aber für große Abbildungen.

Die folgenden Makros werden benutzt, um den Text um Abbildungen, Tabellen und Fußnoten anzupassen:

⟨figure-name|*name*⟩

Dieses Makro steuert das Aussehen des Textes „Abbildung". Die Voreinstellung ist **fett**.

⟨figure-sep⟩

Dieses Makro definiert das Trennzeichen zwischen der Nummerierung und der Beschriftung. Die Voreinstellung ist ein Punkt mit einem nachfolgenden Leerzeichen.

⟨footnote-sep⟩

Dieses Makro definiert das Trennzeichen zwischen der Nummerierung und der Beschriftung. Die Voreinstellung ist ein Punkt mit einem nachfolgenden Leerzeichen.

## 16.4. Standard Titel und Kopfzeilen

### 16.4.1. Standard Kopfzeilen

Die `header` D.T.D. stellt Befehle für Kopfzeilen bereit. Die Optionen zur Anpassung basieren auf der Idee, dass man einen *Seitentext* für jede einzelne Seite spezifizieren können möchte. Dieser *Seitentext* kann ein laufender Titel sein oder der Name des jeweiligen Kapitels. Der *Seitentext* kann für gerade und ungerade Seiten verschieden sein und kann auch in einer anderen Aufmachung bei speziellen Seiten, wie z.B. dem Beginn neuer Kapitel erscheinen. Die folgenden Befehle steuern die Darstellung des Bildschirm- und Druckbildes verschiedener Typen von Seiten:

⟨start-page|*page-text*⟩

Dieses Makro spezifiziert das Layout der ersten Seite eines neuen Kapitels oder eines Abschnitts.

⟨odd-page-page|*page-text*⟩

Ähnlich start-page, aber für ungerade normale Seiten.

⟨even-page-page|*page-text*⟩

Ähnlich start-page, aber für gerade normale Seiten.

Die folgenden Befehle steuern die logischen Kopfzeilen-Aktionen die zur Darstellung von Titel, Autor gebraucht werden, oder die benötigt werden, um einen neuen Abschnitt zu beginnen:

⟨header-title|*title*⟩

Dieses Makro dient zur Spezifizierung des Titels *title* eines Dokuments in der Kopfzeile.

⟨header-author|*author*⟩

Dieses Makro dient zur Spezifizierung des/der Autors/Autoren *author* eines Dokuments in der Kopfzeile.

⟨header-primary|*section-title*⟩

Dieses Makro dient zum Start eines neuen Haupt-Abschnitts, z.B. Kapitel chapter im Buch-Basis-Stil oder Abschnitt section in dem Artikel-Stil.

⟨header-secondary|*section-title*⟩

Dieses Makro dient zum Start eines neuen Sekundär-Abschnitts, z.B. Abschnitt chapter im Buch-Basis-Stil oder Unter-Abschnitt subsection in dem Artikel-Stil.

## 16.4.2. Standard Titel

### 16.4.2.1. Titel und Zusammenfassungen einfügen

Die `header-title` D.T.D. stellt Befehle für Titel bereit. Die folgenden Befehle können nur innerhalb eines make-title-Kontexts benutzt werden:

⟨title|*title*⟩

Spezifiziert den *title* eines Dokuments.

⟨author|*author*⟩

Spezifiziert einen oder mehrere Autoren.

⟨author|*address*⟩

Spezifiziert die Adresse des Autors.

⟨author-block|*address*⟩

Spezifiziert die Adresse(n) eines Autors mit mehreren Adressen.

⟨title-email|*email*⟩

Spezifiziert die Email-Adresse eines Autors.

⟨title-date|*email*⟩

Spezifiziert das Erstellungsdatum des Dokuments, oft heute ⟨date⟩.

title und author benutzen die Befehle header-title und header-author, um die fortlaufenden Titel und Kopfzeilen zu spezifizieren. Sie können das ändern, indem Sie header-title bzw. header-author umdefinieren.

Die `header-title` D.T.D. definiert auch den Befehl abstract für Zusammenfassungen. Innerhalb von Zusammenfassungen (abstracts) können Sie den Befehl keywords benutzen, um Schlüsselworte für Ihre Veröffentlichung zu definieren sowie den Befehl AMS-class, um eine „A.M.S.-subject-classification" durchzuführen.

#### 16.4.2.2. Die globale Darstellung von Titeln anpassen

Abhängig von der Art der Attribute haben komplexe Titel oft gleichzeitig mehrere verschiedene Darstellungs-Stile. Genauer gesagt, ein Titel besteht normalerweise aus folgenden Teilen:

- Einem gut sichtbaren besonders hervorgehobenen Teil ganz oben auf der Seite.
- Zusätzliche Anmerkungen, die in der Fuß-Zeile erscheinen sollen.
- Einen Teil der möglicherweise nicht sichtbar sein soll, wie laufende Titel oder Autoren.
- Ein zurückgestellter Teil, der nur in der Zusammenfassung (abstract) dargestellt werden soll.

Sind mehrere Autoren vorhanden, dann kann der jeder individuelle auch einen Hauptteil haben, de im eigentlichen Titel dargestellt werden soll und zusätzliche Anmerkungen, die als Fußnote erscheinen sollen. Außerdem ändern sich oft das Layout mit dem Autor.

Der Mechanismus, der in $\text{T}_\text{E}\text{X}_\text{MACS}$ Titel zur Darstellung bringt, besitzt daher eine Anzahl von Makros die die notwendigen Informationen für die einzelnen Teile heraus filtert. Dieser Prozess kann außerdem Sortiervorgänge enthalten, wie z.B. den Autor vor das Datum stellen oder umgekehrt. In einer zweien Stufe wird dann die herausgefilterte Information an Darstellungs-Makros weitergereicht.

Die folgenden Makros dienen zur Extraktion von Titel-Informationen:

⟨doc-data-main|*data-1*|⋯|*data-n*⟩
⟨doc-data-main*|*data-1*|⋯|*data-n*⟩

Dieses Makro sammelt und sortiert Daten, die im eigentlichen Titel erscheinen sollen. Die doc-data-main* Variante wird benötigt, wenn mehr als ein Autor vorhanden ist.

⟨doc-data-note|*data-1*|⋯|*data-n*⟩

Dieses Makro sammelt und sortiert Daten, die in der Fußnote erscheinen sollen.

⟨doc-data-abstract|*data-1*|⋯|*data-n*⟩

Dieses Makro sammelt und sortiert Daten, die in der Zusammenfassung erscheinen sollen.

⟨doc-data-hidden|*data-1*|⋯|*data-n*⟩

Dieses Makro sammelt und sortiert Daten, die möglicherweise nicht sichtbar sein sollen oder die keinesfalls sichtbar erscheinen sollen.

In ähnlicher Weise extrahieren die folgenden Makros Autor-Informationen:

⟨doc-author-main|⟨doc-author-data|*data-1*|⋯|*data-n*⟩⟩

Dieses Makro sammelt und sortiert Daten, die in der Fußnote erscheinen sollen.

⟨doc-author-note|*data-1*|⋯|*data-n*⟩

Dieses Makro sammelt und sortiert Daten, die in der Fußnote erscheinen sollen.

Jedes der oben genannten Makros liefert ein document-Konstrukt zurück mit den gesammelten Daten als Kinder. Z.B.

```
⟨doc-author-main|
    ⟨author-address|Somewhere in Africa⟩|
    ⟨author-name|The big GNU⟩|
    ⟨author-note|Very hairy indeed!⟩⟩
```

gibt typischerweise

```
⟨document|
    ⟨author-address|Somewhere in Africa⟩|
    ⟨author-name|The big GNU⟩⟩
```

zurück. Die einzige Ausnahmen ist doc-data-hidden, welches ein concat-Konstrukt zurückgibt.

#### 16.4.2.3. Titel anpassen

Die Titel-Konstrukte benutzen die folgenden Befehle zur Darstellung auf dem Bildschirm bzw. beim Druck:

⟨title*|*title*⟩

Dieses Makro stellt den Dokument-Titel *title* dar.

⟨author*|*author*⟩

Dieses Makro dient zur Darstellung des/der Autoren *author* des Dokuments.

⟨address*|*address*⟩

Dieses Makro dient zur Darstellung der Adresse *address* eines Autors.

⟨title-email*|*email*⟩

Dieses Makro dient zur Darstellung der Email-Adresse *email* eines Autors.

⟨title-date*|*email*⟩

Dieses Makro dient zur Darstellung des Erstellungsdatum *date* des Dokuments.

## 16.5. Abschnitte, LaTeX-artig

### 16.5.1. Abschnitt-Kontexte nutzen

Die section-base D.T.D. stellt die Standard-Befehle für Abschnitte zur Verfügung, und zwar die gleichen wie in LaTeX. Die meisten Abschnitts-Befehle haben ein Argument: den Namen des Abschnitts.

Mit den folgenden Befehlen werden nummerierte Abschnitte erzeugt:

⟨chapter|*title*⟩
⟨section|*title*⟩
⟨subsection|*title*⟩
⟨subsubsection|*title*⟩
⟨paragraph|*title*⟩
⟨subparagraph|*title*⟩
⟨appendix|*title*⟩

Diese Makros erzeugen nummerierte Titel für Kapitel usw.. Die Nummerierung ist nicht notwendig, sie ist aber eine wichtige Option. Absätze, paragraph, und Unter-Absätze, subparagraph, werden normalerweise nicht nummeriert, können es aber sein. Manche Basis-Stile, z.B. allgemein kennen überhaupt keine Nummerierung.

Befehle chapter*, section*, subsection*, subsubsection*, paragraph*, subparagraph* und appendix* dienen zur Erzeugung der unnummerierten Varianten.

Als Vorgabe produzieren die Abschnitts-Konstrukte nur die Abschnitts-Titel. Wenn man aber das experimentelle Paket structured-section (structured-Abschnitt) benutzt, dann übernehmen alle Abschnitts-Befehle einen Abschnitts-Rumpf als weiteres Argument. Dazu gibt es den zusätzlichen Befehl rsection, um rekursiv eingebettete Abschnitte zu erzeugen. So verhält sich beispielsweise eine rsection innerhalb eines Abschnitts section wie ein Unter-Abschnitt subsection. Wir planen alle Abschnitte *strukturiert* zu gestalten.

Die section-base D.T.D. liefert außerdem die folgenden Abschnitt-artigen Kontexte für automatisch erzeugte Verzeichnisse:

⟨bibliography|*aux*|*style*|*file-name*|*body*⟩

Dieses Makro dient zur Erstellung von Literaturverzeichnissen. Das erste Argument *aux* spezifiziert einen *auxiliary channel* mit den Daten zur Erzeugung der Bibliographie (`bib` als Vorgabe). Die Argumente *style* und *file-name* sind der Verzeichnis-Stil und die Datei mit der bibliographischen Datenbasis. Der Rumpf *body* entspricht dem automatisch erzeugten Verzeichnis.

⟨table-of-contents|*aux*|*body*⟩

Dieses Makro erzeugt Inhaltsverzeichnisse. Das erste Argument *aux* spezifiziert einen *auxiliary channel* mit den Daten zur Erzeugung der Bibliographie (`toc` als Vorgabe). Der Rumpf *body* entspricht dem automatisch erzeugten Verzeichnis.

⟨the-index|*aux*|*body*⟩

Ähnlich table-of-contents aber für Stichwortverzeichnisse mit dem Vorgabe-*auxiliary channel* `idx`.

⟨the-glossary|*aux*|*body*⟩

Ähnlich table-of-contents aber für Glossare mit dem Vorgabe-*auxiliary channel* `gly`.

Die vorstehenden Befehle haben die Varianten bibliography*, table-of-contents*, the-index* und the-glossary* mit dem zusätzlichen Argument *name* vor dem Argument *body*. *name* spezifiziert den Namen des Abschnitts. Der the-glossary*-Befehl wird z.B. zur Erzeugung der Liste der Abbildungen und der Liste der Tabellen benutzt.

### 16.5.2. Abschnitt-Kontexte anpassen

Die `section-base` D.T.D. enthält außerdem Befehle zur Darstellung von Abschnitten und zur Steuerung des Verhaltens von Abschnitten.

Die folgenden beiden Befehle wirken auf alle Abschnitte:

⟨sectional-sep⟩

Ein Makro zur Festlegung des Trennzeichens zwischen der Nummer eines Abschnitts und seinem Titel. Als Vorgabe verwenden wir zwei Leerzeichen.

⟨sectional-short-style⟩

Ein Prädikat, das abfragt, ob beabsichtigt ist, ein als kurzes oder langes Dokument zu schreiben. Wenn das Prädikat sectional-short-style wahr, `true`, zurückgibt, dann werden Anhänge, Bibliographien usw. als spezielle Arten von Abschnitten behandelt. Andernfalls sind sie spezielle Kapitel.

Für jeden Abschnitts-Befehl *x*, gibt es die folgenden Befehle zur Anpassung:

⟨*x*-title|*title*⟩

Ein Makro zur Darstellung des unnummerierten Abschnitts-Titels.

⟨*x*-numbered-title|*title*⟩

Ein Makro zur Darstellung des nummerierten Abschnitts-Titels.

⟨*x*-display-numbers⟩

Ein Prädikat welches spezifiziert, ob Nummern wirklich auf dem Bildschirm oder im Druck dargestellt werden sollen. Im Fall eines Absatzes, paragraph, evaluiert das Makro zu „false“ und, obwohl *x*-numbered-title den nummerierten Titel in der Tat *darstellt*, werden die Absatz-Titel unnummeriert dargestellt, denn das Haupt-Makro *x* ruft in diesem Fall *x*-title und nicht *x*-numbered-title auf.

⟨*x*-sep⟩

Ein Makro zur Festlegung des Trennzeichens zwischen der Nummer eines Abschnitts und seinem Titel. Als Vorgabe wird sectional-sep benutzt.

⟨*x*-clean⟩

Alle Unterzähler in dem Abschnitt werden zurückgesetzt.

⟨*x*-header|*name*⟩

Ändern den Seiten-Kopf.

⟨*x*-toc|*name*⟩

Erzeugt einen Eintrag in das Inhaltsverzeichnis.

Schließlich hat die `section-base` D.T.D. noch Makros zur Darstellung von automatisch erstellten Verzeichnissen render-table-of-contents, render-bibliography, render-index und render-glossary, die jeweils zwei Argumente haben, den Namen und den Rumpf.

### 16.5.3. Hilfsmakros für die Darstellung von Abschnitts-Titeln

Die `section-base` D.T.D. stellt eine Reihe von Hilfsmakros bereit, die benutzt werden sollten, wenn die Darstellung von Abschnitts-Titeln angepasst wird:

⟨sectional-short|*body*⟩
⟨sectional-short-italic|*body*⟩
⟨sectional-short-bold|*body*⟩

> Diese Makros sollten zur Erzeugung von *kurzen Abschnitts-Titeln* benutzt werden, bei denen der Rumpf unmittelbar rechts vom Titel beginnt. Absätze und Unter-Absätze werden meist mit solchen Titeln versehen, während andere Abschnitts-Titel sich gewöhnlich über die ganze Text-Breite erstrecken.

⟨sectional-normal|*body*⟩
⟨sectional-normal-italic|*body*⟩
⟨sectional-short-bold|*body*⟩

> Diese Makros sollten für normale linksbündige Titel verwendet werden. Sie gehen über die ganze Absatz-Breite.

⟨sectional-centered|*body*⟩
⟨sectional-centered-italic|*body*⟩
⟨sectional-centered-bold|*body*⟩

> Diese Makros sollten für normale zentrierte Titel verwendet werden. Sie gehen über die ganze Absatz-Breite.

# Kapitel 17

# Kompatibilität mit anderen Formaten

TeXmacs ist voll kompatibel zu PostScript (und PDF), das den Druck von Dokumenten benutzt wird. TeXmacs hat besitzt Konvertierer von und nach LaTeX und ein Filter für den Export von TeXmacs-Dokumenten in Html.

## 17.1. Kompatibilität mit LaTeX

Obwohl der Entwurf von TeXmacs keine volle Kompatibilität mit LaTeX vorsieht, ist es möglich, TeXmacs-Dokumente nach LaTeX zu konvertieren und umgekehrt. Die Resultate nicht immer perfekt. Dabei liefern die Konversionen von TeXmacs nach LaTeX im Allgemeinen die besseren Resultate als die umgekehrte Konvertierung. TeXmacs kann einigermaßen gut zur Erstellung von Publikationen benutzt werden, die zum Einreichen beim Verlag nach LaTeX konvertiert werden müssen. In diesem Kapitel beschreiben wir, was zu beachten ist, damit das Ergebnis so gut wie möglich ausfällt.

### 17.1.1. Konversion von TeXmacs nach LaTeX

Die häufigste Aufgabe ist es, einen Artikel von TeXmacs nach LaTeX zu konvertieren, um ihn an eine Zeitschrift zur Veröffentlichung zu schicken. Aus einer TeXmacs-Datei, die sie (in den Puffer) geladen haben, können Sie mit dem Menü-Befehl Datei → Exportieren → Latex eine TeX-Datei machen. Ihnen wird Ihnen ein Name vorgeschlagen, den Sie belassen oder ändern können. Am besten wenden Sie dann versuchsweise LaTeX auf die neue Datei an, um zu sehen, ob Sie ein ansprechendes Resultat erhalten. Wenn das gelingt, dann sollten Sie zusammen mit der TeX-Datei auch den die Stil-Datei `TeXmacs.sty` weitergeben. Sie finden diese Stil-Datei in einem Unterverzeichnis ihres TeXmacs-Verzeichnisses (`misc/latex`).

Oft hat die Zeitschrift, bei der Sie einreichen wollen, ihren eigenen LaTeX-Stil, z.B. `journal.sty`. In diesem Fall sollten Sie die Datei

```
styles/article.ts
```

, die sich in Unterverzeichnis ihres TeXmacs-Verzeichnisses befindet nach

```
~/.TeXmacs/styles/journal.ts
```

kopieren und als Ihren Basis-Stil im Menü Dokument → Stil → Anders definieren. Dann können Sie `journal.ts` so anpassen, dass das Layout so gut wie möglich dem vorgeschriebenen Layout entspricht. In manchen Fällen muss man außerdem auch `TeXmacs.sty` kopieren und die Kopie anpassen, um Kompatibilität mit dem Stil `journal.sty` zu erreichen.

Wenn Ihr erster Versuch der Konvertierung nicht zu einem Ergebnis kam, werden Sie oft finden, dass nur ein kleiner Teil des Dokuments falsch konvertiert wurde. Das kann mehrere Gründe haben, wichtigsten sind die folgenden:

- Ihr Text benutzte einige TeXmacs-spezifische Optionen, die in LaTeX nicht existieren.
- Sie benutzten TeXmacs-Optionen, die noch nicht in dem Konversions-Algorithmus implementiert sind.
- Sie fanden einen Bug im Konversions-Algorithmus.

Dies wird im folgenden Abschnitt etwas genauer behandelt. Zunächst jedoch soll ein Weg aufgezeigt werden, um geringfügige Darstellungs-Probleme zu korrigieren:

Eine naive Strategie zur Problem-Korrektur besteht darin, die LaTeX-Datei zu korrigieren und dann die korrigierte Datei an die Zeitschrift zu senden. Das hat aber den Nachteil, das man man diese Korrekturen jedes mal machen muss, wenn man nach Änderungen erneut exportiert. Eine bessere Strategie besteht darin die Menübefehle Einfügen → Spezifisch → Latex und Einfügen → Spezifisch → Texmacs zu benutzen, um Text zuschreiben, der nur in jeweils einer Version (entweder TeXmacs oder LaTeX) sichtbar ist.

Nehmen wie einmal beispielhaft an, dass das Wort „frühkonstantinopolitanisch„ in TeXmacs korrekt getrennt wird aber nicht in der LaTeX-Version. Dann können sie so vorgehen:

1. „frühkonstantinopolitanisch„ auswählen.
2. Auf Einfügen → Spezifisch → Texmacs klicken, um es TeXmacs-spezifisch zu machen.
3. Auf Einfügen → Spezifisch → Latex klicken.
4. Dann `früh\-kon\-stan\-ti\-no\-po\-li\-ta\-nisch` mit der korrekten Trennung ein.
5. Drücken Sie `return`, um den LaTeX-spezifischen Text zu aktivieren.

In ähnlicher Weise können Sie LaTeX-spezifische Zeilen- und Seitenumbrüche, vertikalen Leerraum, Stil-Parameter usw. einfügen.

## 17.1.2. Konvertier-Probleme

### 17.1.2.1. TeXmacs-spezifische Besonderheiten

Einige TeXmacs-Konstrukte haben keine Analogien in LaTeX. Der Konversions-Algorithmus liefert dann nur Leerraum. TeXmacs-spezifisch sind vor allem die folgenden:

- Linke obere Indexe.
- Große Separatoren innerhalb großer Klammern.
- Mosaiken.
- Bäume.
- Komplexe Benutzer-Makros.

- Vertikale Abstände „vor" und „hinter".
- Einzugsmarken „vor" und „hinter".

Sie sollten diese TeXmacs-spezifischen Optionen vermeiden, wenn Sie Ihr Dokument nach LaTeX konvertieren müssen. In ferner Zukunft könnte das Konversionsprogramm in diesen Fällen als Voreinstellung *encapsulated PostScript* erzeugen.

#### 17.1.2.2. Noch nicht implementierte Konversionen

Obwohl wir versuchen, den Konversions-Algorithmus so komplett wie möglich zu halten, gibt es Dinge, die noch nicht implementiert sind. Einige Beispiele dafür sind:

- Schriftarten, die nicht zum Standard gehören.
- Konversion von Tabellen.
- Kontext-Parameter.

Alle Anregungen zu wünschenswerten Erweiterungen sollten an

`contact@texmacs.org`

gerichtet werden. Wir werden versuchen, sie sobald wie möglich zu implementieren. Allerdings wird es einige Zeit dauern, bis korrekte Kontext-Parameter implementiert sein werden, denn sie sind bei TeXmacs und LaTeX verschieden. Auch können Layout-Unterschiede zwischen TeXmacs und LaTeX nicht völlig eliminiert werden.

#### 17.1.2.3. Fehler im Konversions-Algorithmus

Am frustierendsten ist es, wenn man einen Haufen Fehler beim „LaTeXen" der konvertierten Datei erhält oder wenn das Resultat nichts mit dem Original gemein hat. In diesem Fall haben Sie wahrscheinlich einen Fehler im Konversions-Algorithmus entdeckt oder in Ihrer LaTeX-Installation. Versuchen Sie bitte die Ursache herauszufinden und schicken Sie eine Email an

`TeXmacs@math.u-psud.fr`

#### 17.1.2.4. Dennoch konvertieren

Das Design von TeXmacs hat nicht das Ziel, mit LaTeX voll kompatibel zu sein. Unser Ziel ist es vor allem, *Hilfe* zu geben, damit bereits existierende Dokumente aus LaTeX nach TeXmacs übertragen werden können. Solange diese Dokumente keine ungewöhnlichen Kontexte und ungewöhnliche Befehle enthalten, sollte es Ihnen möglich sein, einigermaßen vernünftige Konvertierungen zu erhalten. Ist das nicht der Fall, schlagen wir vor, ihre bestehenden Dokumente so zu modifizieren, dass das gelingt, um dann TeXmacs-Dokumente zu korrigieren.

### 17.1.3. Konversion von LaTeX nach TeXmacs

Das der Konvertierung von LaTeX nach TeXmacs, ist es, *Ihnen zu helfen*, alte alte LaTeX-Dokumente nach TeXmacs zu übertragen, damit sie für Ihre Arbeit nicht vollständig verloren sind. Es ist nicht das Ziel, eine vollständige Übertragung mit identischen Layout zu erreichen.

Im allgemeinen ist die Umwandlung von LATEX nach TEXMACS schwieriger als umgekehrt. Wenn Sie sich aber auf die gebräuchlichsten LATEX Kommandos beschränken, sollten Sie ihre Dokumente einigermaßen ordentlich übertragen können. Beispielsweise wurden alle TEXMACS-Hilfe-Dateien in LATEX geschrieben und dann nach TEXMACS konvertiert, um das Konversions-Programm zu testen.

Sie können ein LATEX-Dokument`name.tex` mit dem Befehl Datei → Importieren → Latex nach TEXMACS importieren und unter dem Namen `name.tm` sichern. Wenn Ihr LATEX-Dokument gut genug geschrieben wurde, dann sollte das Resultat der Konvertierung einigermaßen akzeptabel sein, von einigen unbekannten Befehlen abgesehen, die rot markiert sind. Ein gute Lösung dafür, besteht darin, eine eigene Stil-Definition basierend auf dem Original Stil zu schreiben, in der die unbekannten Befehle definiert werden.

Dennoch gibt es Fälle, in denen das ganze Dokument ein großes unlesbares Durcheinander ist. Das kommt meist davon, dass TEX und LATEX es gestatten, dass der Parser während seiner Ausführung dynamisch verändert wird, indem Sie z.B. den `\catcode` Befehl verwenden. In solchen Fällen kann das Konvertierungs-Programm manchmal den Intentionen nicht folgen und macht unzutreffende Annahmen. Dann wird z.B. Text in mathematische Formeln umgewandelt, mathematischer Code in wörtlichen Text usw.. Meist lassen sich die problematischen Befehle in `name.tex` aber leicht herausfinden, wenn man LATEX mit der TEXMACS-Version vergleicht. Dann kann man meist den problematischen LATEX-Code ersetzen und so eine einigermaßen akzeptable Konvertierung erreichen.

Wir planen auch eine Konverter für TEX-Stil-Dateien nach TEXMACS sowie einige andere Erweiterungen, die die Konversion von Anwender-Befehlen zu erleichtern, die in einem anderen Dokument definiert sind, als dem, das Sie gerade konvertieren wollen.

## 17.2. DOKUMENTE ZWISCHEN TEXMACS UND HTML PORTIEREN

Wir haben damit begonnen die Konversion zwischen HTML und TEXMACS zu implementieren. Im Moment gelingt nur der Import von HTML-Dokumenten mit dem Menü-Befehl Datei → Importieren → Html. Das meiste HTML 2.0 und Teile von HTML 3.0 werden derzeit unterstützt. Browser-Fähigkeiten wurden noch nicht implementiert. Für die Zukunft planen wir den Import von Math-ML.

Wenn HTML-Dokumente importiert werden, deren Namen mit `http:` oder `ftp:` beginnen, werden diese über das Netzwerk mit `wget` heruntergeladen. Wenn Sie TEXMACS selbst kompiliert haben, können Sie `wget` von

```
ftp://ftp.gnu.org/pub/gnu/wget/
```

herunterladen. In den Binär-Distributionen ist `wget` enthalten.

## 17.3. KONVERTIERER UND NEUE DATENFORMATE ERSTELLEN

Mir der GUILE/SCHEME-Sprache kann man, neue Datenformate und Konvertierer zu TEXMACS als Module hinzufügen. Normalerweise werden die zusätzlichen Formate und Konvertierer in ihrer persönlichen Initialisierungs-Datei `~/.TeXmacs/progs/my-init-texmacs.scm` definiert oder in einem speziellen Plugin. Einige Beispiele finden Sie in einem Unterverzeichnis `progs/convert` ihres TEXMACS-Verzeichnisses, wie z.B. init-html.scm.

**neue formate definieren.**

Ein neues Format kann mit dem Befehl

```
(define-format format
  (:name format-name)
  options)
```

erzeugt werden. *format* ist ein Symbol, dass für das Format steht und *format-name* eine Zeichenkette, die in Menüs benutzt werden kann. Tatsächlich gibt es ein Datenformat meist in mehreren Varianten: ein Format *format*`-file` für Dateien, ein Format *format*`-document` für ganze Dokumente und ein Format *format*`-snippet` für kurze Zeichenketten, wie z.B. Auswahlen (selections), und schließlich *format*`-object` für die bevorzugte interne Scheme-Repräsentation bei Konversionen (dies ist Parser-Variante des Formats). Konvertierer von *format*`-file` nach *format*`-document` und umgekehrt werden automatisch erzeugt.

Der Anwender kann zusätzliche Optionen zur automatischen Erkennung von Formaten mit Hilfe von Datei-Suffixen oder -Inhalten spezifizieren. Z.B. können die erlaubten Suffixe eines Daten-Formats mit der Voreinstellung als erstem durch

```
(:suffix default-suffix other-suffix-1 ... other-suffix-n)
```

angegeben werden. Eine (heuristische) Routine, um festzustellen, ob ein Dokument zu einem bestimmten Format gehört, kann mit

```
(:recognize predicate)
(:must-recognize predicate)
```

erreicht werden. Im ersten Fall hat die Suffix-Erkennung Vorrang vor der heuristischen Erkennung. Im zweiten Fall ist nur die heuristische Erkennung durch das Prädikat, predicate, maßgeblich.

**neue konvertierer erzeugen.**

Neue Konvertierer können mit

```
(converter from to
  options)
```

erzeugt werden. Der eigentliche Konvertierer wird durch eine der folgenden Optionen spezifiziert:

```
(:function converter)
(:function-with-options converter-with-options)
(:shell prog prog-pre-args from progs-infix-args to prog-
post-args)
```

Im ersten Fall ist der Konvertierer *converter* eine Routine die ein Objekt im Daten-Format *from* übernimmt und ein Objekt im Daten-Format *to* zurückgibt. Im zweiten Fall übernimmt der *converter* eine assoziative Liste als zweites Argument und Optionen für den Konvertierer. Im letzten Fall wird ein *shell*-Befehl angegeben, der die Konvertierung zwischen den beiden Datei-Formaten durchführt. Der Konvertierer wird dann und nur dann aktiviert, wenn das Programm *prog* gefunden wird. Hilfsdateien können automatisch erzeugt und wieder gelöscht werden.

TeXmacs berechnet automatisch die transitive „Closure” aller Konvertierer, indem einen „kürzesten Pfad Algorithmus” benutzt. Mit anderen Worten, wenn es einen Konvertierer für $x$ nach $y$ gibt und einen Konvertierer von $y$ nach $z$, dann hat man auch einen von $x$ nach $z$. Es für jede Konvertierung kann eine „Strafe/Abstand zwischen den Formaten“ mit

```
(:penalty floating-point-distance)
```

angeben werden, um so Hinweise zum automatischen Finden eines optimalen Weges für die Konvertierung zu geben.

Weitere Optionen für Konvertierer sind:

```
(:require cond)
(:option option default-value)
```

Die erste Option spezifiziert eine Bedingung, die erfüllt sein muss, damit der Konvertierer benutzt werden kann. Diese Option sollte als erste oder zweite Option spezifiziert werden und immer nach der `:penalty` Option. Die `:option` Option spezifiziert eine Option für den Konvertierer, mit einer Voreinstellung. Diese Option wird automatisch an alle Konvertierer mit Optionen weitergegeben.

# Anhang A
# Configuration of TeXmacs

## A.1. Introduction

Before you start using TeXmacs, it may be wise to configure the program first in Bearbeiten → Einstellungen, so that it will fit your needs best. Most importantly, you should choose a "look and feel" in Bearbeiten → Einstellungen → Aussehen und Verhalten. This will enable you for instance to let the keyboard shortcuts used by TeXmacs be similar to what you are used to in other applications.

Also, TeXmacs comes with a powerful keyboard shortcut system, which attempts to optimize the use of the modifier keys like `shift` and `control` on your keyboard. However, on certain systems these modifier keys are not well configured, so that you may wish to redo this yourself.

## A.2. Configuration of the modifier keys

TeXmacs uses five major keyboard modifiers: `shift`, `control`, `alternate`, `meta` and `hyper`, which are abbreviated as `S-`, `C-`, `A-`, `M-` and `H-`. The `shift` and `control` keys are present on virtually all keyboards and the `alternate` key on almost all. Most keyboards for PC's nowadays also have a `windows` key, which is usually equivalent to `meta` for TeXmacs.

Before reconfiguring your keyboard, you should first check that this is indeed necessary. If you have keys which correspond to `shift`, `control`, `alternate` and `meta` in a suitable way, then you probably do not want to do anything. A possible exception is when you want to use a simple key like `caps-lock` for typing mathematical symbols. In that case, you should map `caps-lock` to `hyper`.

In order to reconfigure the keyboard, you simply select the logicial modifier that you want to correspond to a given physical key in Bearbeiten → Einstellungen → Tastatur. For instance, selecting Windows-Taste → Dem Modifikator M zuordnen, the `windows` key will correspond to the `meta` modifier. Similarly, when selecting Caps-lock key → Dem Modifikator H zuordnen, the `caps-lock` key will correspond to the `hyper` modifier.

Unfortunately, X Window only allows system-wide reconfiguration. Consequently, if you reconfigure the `caps-lock` key inside TeXmacs, then the new behaviour of `caps-lock` will affect all other applications too. It is therefore important to reconfigure only those keys which you do not use for something else in other applications. For instance, the `windows` key is not used by many applications, so it generally does not do any harm to reconfigure it. You may also prefer to perform an appropriate system-wide configuration. This can be done using the `xmodmap` command; see the corresponding manual page for more information.

In certain cases, you already have keys on your keyboard which correspond to `alternate`, `meta` and `hyper`, but not in the way you want. This can be done by remapping the `A-`, `M-` and `H-` prefixes to other logical modifiers in the first group of submenus of Bearbeiten → Einstellungen → Tastatur.

For instance, for Emacs compatability, you might want to permute the `meta` or `windows` key with `alternate` without making any system-wide changes. This can be done by finding out which modifiers correspond to these keys; usually this will be `Mod1` for `alternate` and `Mod4` for `meta` or `windows`. We next perform the necessary permutation in Bearbeiten → Einstellungen → Tastatur, by selecting A Modifikator → Entspricht Mod4 and M Modifikator → Entspricht Mod1.

## A.3. Notes for Russian and Ukranian users

In order to type Russian (and similarly for Ukranian) text, you may several options:

- Select Russian as your default language in Bearbeiten → Einstellungen → Sprache → Russisch. If TEXMACS starts with Russian menus, then this is done automatically if the Russian locale is set.
- Select Russian for an entire document using Dokument → Sprache → Russisch.
- Select Russian for a portion of text in another document using Formate → Sprache → Russisch.

If your X server uses the xkb extension, and is instructed to switch between the Latin and Russian keyboard modes, you need not do anything special. Just switch your keyboard to the Russian mode, and go ahead. All the software needed for this is included in modern Linux distributions, and the xkb extension is enabled by default in `XF86Config`. With the xkb extension, keysyms are 2-byte, and Russian letters are at 0x6??. The keyboard is configured by `setxkbmap`. When X starts, it issues this command with the system-wide `Xkbmap` file (usually living in `/etc/X11/xinit`), if it exists; and then with the user's `~/.Xkbmap`, if it exists. A typical `~/.Xkbmap` may look like

```
ru basic grp:shift_toggle
```

This means that the keyboard mode is toggled by `l-shift r-shift`. Other popular choices are `control shift` or `control alternate`, see `/usr/X11R6/lib/X11/xkb/` for more details. This is the preferred keyboard setup for modern Linux systems, if you plan to use Russian often.

In older Linux systems, the xkb extension is often disabled. Keysyms are 1-byte, and are configured by `xmodmap`. When X starts, it issues this command with the system-wide `Xmodmap` (usually living in `/etc/X11/xinit`), if it exists; and then with the user's `~/.Xmodmap`, if it exists. You can configure the mode toggling key combination, and use a 1-byte Russian encoding (such as koi8-r) in the Russian mode. It is easier to download the package `xruskb`, and just run

```
xrus jcuken-koi8
```

at the beginning of your X session. This sets the layout jcuken (see below) and the encoding koi8-r for your keyboard in the Russian mode. If you use such keyboard setup, you should select Options → international keyboard → russian → koi8-r.

It is also possible to use the Windows cp1251 encoding instead of koi8-r, though this is rarely done in UNIX. If you do use `xrus jcuken-cp1251`, select cp1251 instead of koi8-r.

All the methods described above require some special actions to "russify" the keyboard. This is not difficult, see the Cyrillic-HOWTO or, better, its updated version `http://www.inp.nsk.su/~baldin/Cyrillic-HOWTO-russian/Cyrillic-HOWTO-russian.html`

Also, all of the above methods globally affect all X applications: text editors (emacs, nedit, kedit...), xterms, TeXmacs etc.

If you need to type Russian only once, or very rarely, a proper keyboard setup may be more trouble than it's worth. For the benefit of such occasional users, TeXmacs has methods of Russian input which require no preliminary work. Naturally, such methods affect only TeXmacs, and no other application.

The simplest way to type some Russian on the standard US-style keyboard with no software setup is to select Bearbeiten → Einstellungen → Tastatur → Kyrillische Eingabemethode → translit. Then, typing a Latin letter will produce "the most similar" Russian one. In order to get some Russian letters, you have to type 2- or 3-letter combinations:

| Shorthand | for | Shorthand(s) | for |
|---|---|---|---|
| `A-" e` | ё | `A-" E` | Ё |
| `y o` | ё | `Y o` `Y O` | Ё |
| `z h` | ж | `Z h` `Z H` | Ж |
| `j Tab` | ж | `J Tab` | Ж |
| `c h` | ч | `C h` `C H` | Ч |
| `s h` | ш | `S h` `S H` | Ш |
| `s c h` | щ | `S c h` `S C H` | Щ |
| `e Tab` | э | `E Tab` | Э |
| `y u` | ю | `Y u` `Y U` | Ю |
| `y a` | я | `Y a` `Y A` | Я |

**Tabelle A.1.** Typing Cyrillic text on a Roman keyboard.

If you want to get, e.g., "сх", and not "ш", you have to type `s / h`. Of course, the choice of "optimal" mapping of Latin letters to Russian ones in not unique. You can investigate the mapping supplied with TeXmacs and, if you don't like something, override it in your `~/.TeXmacs/progs/my-init-texmacs.scm`.

If you select jcuken instead of translit, you get the "official" Russian typewriter layout. It is so called because the keys "qwerty" produce "йцукен". This input method is most useful when you have a Russian-made keyboard, which has additional Russian letters written on the key caps in red, in the jcuken layout (a similar effect can be achieved by attaching transparent stickers with red Russian letters to caps of a US-style keyboard). It is also useful if you are an experienced Russian typist, and your fingers remember this layout.

Those who have no Russian letters indicated at the key caps often prefer the yawerty layout, where the keys "qwerty" produce "яверты". Each Latin letter is mapped into a "similar" Russian one; some additional Russian letters are produced by `shift`-digits. TeXmacs comes with a slightly modified yawerty layout, because it does not redefine the keys `$`, `£`, `\`, which are important for TeXmacs, are not redefined. The corresponding Russian letters are produced by some `shift`-digit combinations instead.

# Anhang B

# About GNU TeXmacs-1.0.5

## B.1. Summary

| GNU TeXmacs | |
|---|---|
| Installed version | 1.0.5 |
| Supported systems | Most GNU/Linux systems |
| Copyright | © 1998–2002 by Joris van der Hoeven |
| License | GNU General Public License |
| Web sites | `http://www.texmacs.org`<br>`http://www.gnu.org/software/texmacs` |
| Contact | `contact@texmacs.org` |
| Regular mail | Dr. Joris van der Hoeven<br>Dépt. de Mathématiques (Bât. 425)<br>Université Paris-Sud<br>91405 Orsay Cedex<br>France |

**Tabelle B.1.** Summary of the principal information about GNU TeXmacs.

## B.2. The philosophy behind TeXmacs

### B.2.1. A short description of GNU TeXmacs

GNU TeXmacs is a free scientific text editor, which was both inspired by TeX and GNU Emacs. The editor allows you to write structured documents via a wysiwyg (what-you-see-is-what-you-get) and user friendly interface. New styles may be created by the user. The program implements high-quality typesetting algorithms and TeX fonts, which help you to produce professionally looking documents.

The high typesetting quality still goes through for automatically generated formulas, which makes TeXmacs suitable as an interface for computer algebra systems. TeXmacs also supports the Guile/Scheme extension language, so that you may customize the interface and write your own extensions to the editor.

Converters exist for TeX/LaTeX and they are under development for Html/MathML/Xml. In the future, TeXmacs is planned to evolve towards a complete scientific office suite, with spreadsheet capacities, a technical drawing editor and a presentation mode.

GNU TeXmacs is hosted by the Centre de Ressources Informatiques de Haute Savoie, Archamps, France.

### B.2.2. Why freedom is important for scientists

One major objective of TeXmacs is to promote the development of free software for and by scientists, by significantly reducing the cost of producing high quality user interfaces. If you plan to write an interface between TeXmacs and other software, then please contact us.

As a mathematician, I am deeply convinced that only free programs are acceptable from a scientific point of view. I see two main reasons for this:

- A result computed by a "mathematical" system, whose source code is not public, can not be accepted as part of a mathematical proof.

- Just as a mathematician should be able to build theorems on top of other theorems, it should be possible to freely modify and release algorithms of mathematical software.

However, it is strange, and a shame, that the main mathematical programs which are currently being used are proprietary. The main reason for this is that mathematicians often do not consider programming as a full scientific activity. Consequently, the development of useful software is delegated to "engineers" and the resulting programs are used as black boxes.

This subdivision of scientific activity is very artificial: it is often very important from a scientific point of view to know what there is in the black box. Inversely, deep scientific understanding usually leads to the production of better software. Consequently, I think that scientists should advocate the development of software as a full scientific activity, comparable to writing articles. Then it is clear too that such software should be diffused in a way which is compatible with the requirements of science: public availability, reproducibility and free usability.

## B.3. The authors of TeXmacs

The GNU TeXmacs system, which is part of the GNU project, was designed and written by Joris van der Hoeven. The system was inspired both by the TeX system, written by D. Knuth, and by Emacs, written by R. Stallman. Special thanks goes to them, as well as to the C.N.R.S. (the French national institute for scientific research), which employs me and authorized me to freely distribute this program. Further thanks go to the contributors below.

### B.3.1. Developers of TeXmacs

- Dan Martens made the Windows port.

- Andrey Grozin has constantly helped us with many issues: interfaces to several computer algebra systems, support for Cyrillic, tools for the manipulation of dictionaries, etc.

- David Allouche replaced the gencc preprocessor by the more standard C++ template system. He also made many other patches, bug reports and he did a lot of the administration of TeXmacs.

- Henri Lesourd is working on native technical drawing support in TeXmacs. He also fixed a bug in the presentation mode.
- Andreas Seidl has been helping with documentation, a Cygwin package and several other things.
- Dan Grayson helped me to implement communications with computer algebra systems via pipes. He also provided some money support for TeXmacs, and he made many useful comments and suggestions.
- Karim Belabas designed and developed with me the first protocol for interfacing TeXmacs with scientific computation or computer algebra systems. He also implemented the interface with the Pari system.
- Felix Breuer helped with the support of Unicode and other character encodings. He also made a donation to the project.
- Josef Weidendorfer made several patches for improving the performance of TeXmacs.
- Stéphane Payrard made an important bugfix for destroying windows.
- Josef Weidendorfer for two patches to improve the speed of TeXmacs.
- Johann Dréo for the new TeXmacs icon and many other graphics.
- Bill Page and David Mentré for the support of the free version of Axiom.
- Chu-Ching Huang for writing CAS documentation and making a Knoppix CD for TeXmacs.
- Mickael Floc'hlay and Arnaud Ébalard for their work on searching for help.
- Gwenael Gabard for some fixes in the LaTeX to TeXmacs converter.
- Igor V. Kovalenko and Teemu Ikonen for their help on debugging TeXmacs and a few patches.
- Gareth McCaughan made several patches and comments.
- Immanuel Normann is working on an OpenMath converter.
- Jonas Lööf for a precise installation procedure on Cygwin.
- Rob Clark made a patch which improves the system time support.
- Stanislav Brabec for several patches so as to increase portability.

### B.3.2. Authors and maintainers of plugins for TeXmacs

**Axiom** — Andrey Grozin, Bill Page, David Mentré and Tim Daly.

**DraTeX** — Nicolas Ratier.

**Eikleides** — Mark Arrasmith.

**Giac** — Bernard Parisse.

**GNUplot** — Stephan Mucha.

**Graphviz** — Jorik Blaas.

**GTybalt** — Stefan Weinzierl.

**Macaulay 2** — Dan Grayson.

**Maple** — Christian Even.

**Mathemagix** — Joris van der Hoeven.

**Maxima** — Andrey Grozin and James Amundson.

**Mupad** — Christopher Creutzig and Andrey Grozin.

**Octave** — Michael Graffam.

**Pari** — Karim Belabas.

**Python** — Ero Carrera.

**Qcl** — Andrey Grozin.

**R** — Michael Lachmann.

**Reduce** — Andrey Grozin.

**Scilab** — Serge Steer and Claude Gomez.

**Shell** — Joris van der Hoeven.

**XYpic** — Nicolas Ratier.

**Yacas** — Ayal Pinkus.

### B.3.3. Administration of TEXMACS and material support

- Rennes Métropôle and the C.N.R.S. for financially supporting the development of TEXMACS.
- Christoph Benzmueller and his team for financially supporting the development of TEXMACS.
- Springer-Verlag for their financial support for making a better Windows version.
- Jean-Claude Fernandez, Fabien Salvi and the other persons from the CRI host and administrate the TEXMACS website.
- Álvaro Tejero Cantero maintains up the TEXMACS Wiki.
- Loic Dachary made TEXMACS accessible on Savannah.

### B.3.4. Porting TeXmacs to other platforms

- Dan Martens is working on a the experimental Windows port.
- Marciano Siniscalchi ported TeXmacs to Cygwin. His work was further perfected by Loïc Pottier. Andreas Seidl made a the standard Cygwin package.
- Martin Costabel ported TeXmacs to MacOSX.
- Ralf Treinen and others has been ensuring the portability of TeXmacs to all architectures supported by Debian Gnu/Linux.
- Bruno Haible and Gregory Wright helped with porting TeXmacs to the SUN system and maintaining it.
- Philipp Tomsich and Chuck Sites for their help with the IRIX port.

### B.3.5. Contributors to TeXmacs packages

- Ralf Treinen maintains the Debian package for TeXmacs.
- Christophe Merlet and Bo Forslund helped with making a portable RPM package.
- Lenny Cartier maintains the TeXmacs RPM for Mandrake Cooker.
- Jean Pierre Demailly and Yves Potin made TeXmacs part of the CNDP project to support free software.

### B.3.6. Internationalization of TeXmacs

**Czech.** David Rezac.

**Danish.** Magnus Marius Rohde.

**Dutch.** Joris van der Hoeven.

**Finnish.** Teemu Ikonen.

**French.** Michèle Garoche, Joris van der Hoeven.

**German.** Dietmar Jung, Hans Dembinski, Jan Ulrich Hasecke, Christoph Strobel, Joris van der Hoeven, Thomas Langen, Ralf Treinen.

**Hungarian.** András Kadinger.

**Italian.** Andrea Centomo, Lucia Gecchelin, Xav and Daniele Pighin, Gian Luigi Gragnani.

**Polish.** Robert Janusz, Emil Nowak, Jan Alboszta.

**Portuguese.** Ramiro Brito Willmersdorf, Márcio Laurini, Alexandre Taschetto de Castro.

**Romanian.** Dan Ignat.

**Russian.** Andrey Grozin.

**Slovene.** Ziga Kranjec.

**Spanish.** Álvaro Cantero Tejero, Pablo Ruiz Múzquiz, David Moriano Garcia, Offray Vladimir Luna Cárdenas.

**Swedish.** Harald Ellmann.

**Ukrainian.** Volodymyr Lisivka.

### B.3.7. Other contributors

Final thanks go to all others who have contributed to $\mathrm{T_EX_{MACS}}$, for instance by sending bug reports or by giving suggestions for future releases: Alexandre Abbes, Alessio Abogani, Aaron Acton, Till Adam, Murali Agastya, Guillaume Allègre, Larry D'Anna, Eizo Akiyama, Javed Alam, Doublet Alban, Michele Alessandrin, Andreas Almroth, Tom Alsberg, James Amundson, Piero D'Ancona, Daniel Andor, Ayal Anis, Javier Arantegui Jimenez, André Arnold, Uwe Assmann, Philippe Audebaud, Daniel Augot, Olaf Bachmann, Franky Backeljauw, Nick Bailey, Adrian Soto Banuelos, Pierre Barbier de Reuille, Marc Barisch, Giovanni Maniscalco Basile, Claude Baudouin, Marten Bauer, Luc Béhar, Roman Belenov, Odile Bénassy, Paul Benham, Roy C. Bentley, Attila Bergou, Christophe Bernard, Konrad Bernloehr, Karl Berry, Matthias Berth, Matteo Bertini, Cédric Bertolini, Matthew Bettencourt, Raktim Bhattacharya, Anne-Laure Biolley, Benedikt Birkenbach, Giovanni Biczó, Jim Blandy, Sören Blom, Christof Boeckler, François Bochatay, Christof Boeckler, Anton Bolfing, Robert Borys, Didier Le Botlan, Mohsen Bouaissa, Thierry Bouche, Adrien Bourdet, Michel Brabants, Didier Bretin, Jean-Yves Briend, Henrik Brink, Simon Britnell, Alexander M. Budge, Daniel Bump, Yoel Callev, José Cano, Charles James Leonardo Quarra Cappiello, Patrick Cardona, Niclas Carlsson, Dominique Caron, António Carvalho, Michel Castagner, Topher Cawlfield, Carlo Cecati, Beni Cherniavsky, Kuo-Ping Chiao, Teddy Fen-Chong, Henri Cohen, Johann Cohen-Tanugi, Vincenzo Colosimo, Dominique Colnet, Claire M. Connelly, Christoph Conrad, Riccardo Corradini, Paulo Correia, Olivier Cortes, Robert J. Cristel, Maxime Curioni, Allan Curtis, Jason Dagit, Stefano Dal Pra, François Dausseur, Thierry Dalon, Jon Davidson, Mike Davidson, Thomas Delzant, Jean-Pierre Demailly, Peter Denisevich, Alessio Dessi, Benno Dielmann, Lucas Dixon, Mikael Djurfeldt, Gabriel Dos Reis, Alban Doublet, Steingrim Dovland, Michael John Downes, Benjamin Drieu, Jose Duato, Amit Dubey, Daniel Duparc, Guillaume Duval, Tim Ebringer, Dirk Eddelbuettel, Magnus Ekdahl, Ulf Ekström, Sreedhar Ellisetty, Luis A. Escobar, Thomas Esser, Stephan Fabel, Robin Fairbairns, Tony Falcone, Vladimir Fedonov, Hilaire Fernandes, Ken Feyl, Juan Flynn, Jens Finke, Thomas Fischbacher, Cedric Foellmi, Enrico Forestieri, Ted Forringer, Christian Forster, Charlie Fortner, Stefan Freinatis, Michael P Friedlander, Nils Frohberg, Rudi Gaelzer, Maciej Gajewski, Lionel Garnier, Philippe Gogol, Björn Gohla, Patrick Gonzalez, Nirmal Govind, Albert Graef, Michael Graffam, Klaus Graichen, Ian Grant, Frédéric Grasset, Guido Grazioli, Wilco Greven, Cyril Grunspan, Laurent Guillon, Yves Guillou, Tae-Won Ha, Harri Haataja, Sébastien Hache, Irwan Hadi, James W. Haefner, Sam Halliday, Ola Hamfors, Aaron Hammack, Guillaume Hanrot, Alexander K. Hansen, Peter I. Hansen, Zaid Harchaoui, Jesper Harder, Philipp Hartmann, P. L. Hayes, Karl M. Hegbloom, Jochen Heinloth, Gunnar Hellmund, Ralf Hemmecke, Roy Henk, John Hernlund, Alain Herreman, Alexander Heuer, Johannes Hirn, Santiago Hirschfeld, Andreas Horn, Peter Horn, Chu-Ching Huang, Sylvain Huet, Ed Hurst, Karl Jarrod Hyder, Richard Ibbotson, Benjamin T. Ingram, Alexander Isacson, Michael Ivanov, Vladimir G. Ivanovic, Maik Jablonski, Frederic de Jaeger, Pierre Jarillon, Neil Jerram, Paul E. Johnson, Pierre-Henri Jondot, Peter Jung, Antoun

Kanawati, Tim Kaulmann, Mukund S. Kalisi, Antoun Kanawati, Yarden Katz, Bernhard Keil, Samuel Kemp, Jeremy Kephart, Michael Kettner, Salman Khilji, Iwao Kimura, Simon Kirkby, Ronny Klein, Peter Koepke, Matthias Koeppe, John Kollar, Denis Kovacs, Jeff Kowalczyk, Dmitri Kozionov, Ralph Krause, Neel Krishnaswami, Anthony Lander, Friedrich Laher, Winter Laite, Anthony Lander, Russell Lang, David Latreyte, Christopher Lee, Milan Lehocky, Torsten Leidig, Patrick Lenz, Kalle Lertola, Tristan Ley, Joerg Lippmann, Marc Longo, Pierre Lorenzon, Ralph Lõvi, V. S. Lugovsky, Gregory Lussiana, Bud Maddock, Duraid Madina, Camm Maguire, Yael Maguire, Paul Magwene, Jeremiah Mahler, Vincent Maillot, Giacomo Mallucci, Lionel Elie Mamane, Sourav K. Mandal, Andy P. Manners, Yun Mao, Chris Marcellin, Sylvain Marchand, Bernd Markgraf, Eric Marsden, Chris Marston, Evan Martin, Carlos Dehesa Martínez, Paulo Jorge de Oliveira Cantante de Matos, Tom McArdell, Bob McElrath, Alisdair McDiarmid, Robert Medeiros, Phil Mendelsohn, Sébastien de Menten, Jean-Michel Mermet, Jon Merriman, Herve le Meur, Ingolf Meyer, Amir Michail, Franck Michel, Jan David Mol, Klaus-Dieter Möller, Juan Fresneda Montano, André Moreau, Vijayendra Munikoti, Arkadiusz Miskiewicz, Sasha Mitelman, Dirk Moebius, Jack Moffitt, Harvey Monder, Guillaume Morin, Julian Morrison, Bernard Mourrain, Stephan Mucha, Toby Muhlhofer, Nathan Myers, Norbert Nemec, Thomas Neumann, Thien-Thi Nguyen, Han-Wen Nienhuys, Nix N. Nix, Eduardo Nogueira, Immanuel Normann, Jean-Baptiste Note, Ralf Nuetzel, Kostas Oikonomou, Ondrej Pacovsky, Bill Page, Santtu Pajukanta, Pierre Pansu, Ilya Papiashvili, Bernard Parisse, Frédéric Parrenin, André Pascual, Fernández Pascual, Frédéric Parrenin, Yannick Patois, Alen L. Peacock, François Pellegrini, Antonio Costa Pereira, Enrique Perez-Terron, Jacob Perkins, Bernard Perrot, Jan Peters, Jean Peyratout, Jacques Peyriere, Valery Pipin, Dimitri Pissarenko, Yves Pocchiola, Martin Pollet, Benjamin Poussin, Benjamin Podszun, Isaías V. Prestes, Rui Prior, Julien Puydt, Nguyen-Dai Quy, Manoj Rajagopalan, Ramakrishnan, Adrien Ramparison, Nicolas Ratier, Olivier Ravard, Leo Razoumov, Kenneth Reinhardt, Cesar A. Rendon, Diego Restrepo, Christian Requena, Chris Retford, Robert Ribnitz, Thomas CLive Richards, Staffan Ringbom, Eric Ringeisen, Christian Ritter, William G. Ritter, Will Robinson, Pascal Romon, Juan Pablo Romero, Juergen Rose, Mike Rosellini, Mike Rosing, Bernard Rousseau, Eyal Rozenberg, Olivier Ruatta, Filippo Rusconi, Gaetan Ryckeboer, Philippe Sam-Long, John Sandeman, Duncan Sands, Breton Saunders, Claire Sausset, David Sauzin, Gilles Schaeffer, Guido Schimmels, Rainer Schöpf, David Schweikert, Stefan Schwertheim, Rui Miguel Seabra, Chung-Tsun Shieh, Sami Sieranoja, Vasco Alexandre da Silva Costa, Marciano Siniscalchi, Daniel Skarda, Murray Smigel, Vaclav Smilauer, Dale P. Smith, Luke Snow, René Snyders, Pekka Sorjonen, Kasper Souren, Rodney Sparapani, Bas Spitters, Bas Spitters, Ivan Stanisavljevic, Starseeker, Harvey J. Stein, Peter Sties, Bernard Stloup, Peter Stoehr, Thierry Stoehr, James Su, Przemyslaw Sulek, Ben Sussman, Roman Svetlov, Milan Svoboda, Dan Synek, Pan Tadeusz, Luca Tagliacozzo, Sam Tannous, John Tapsell, Dung TaQuang, Gerald Teschl, Laurent Thery, Eric Thiébaut, Nicolas Thiery, Helfer Thomas, Reuben Thomas, Dylan Thurston, Kurt Ting, Janus N. Tøndering, Philippe Trébuchet, Marco Trevisani, Boris Tschirschwitz, Elias Tsigaridas, Michael M. Tung, Andreas Umbach, Miguel A. Valle, Rémi Vanicat, Harro Verkouter, Jacques Vernin, Sawan Vithlani, Philip A. Viton, Marius Vollmer, Guy Wallet, Adam Warner, Thomas Wawrzinek, Maarten Wegewijs, Duke Whang, Lars Willert, Grayson Williams, Barton Willis, Claus-Peter Wirth, Ben Wise, Wiebe van der Worp, Pengcheng Wu, Damien Wyart, Wang Yin, Lukas Zapletal, Volker Zell, Oleg Zhirov, Vadim V. Zhytnikov, Richard Zidlicky, Sascha Ziemann, Reinhard Zierke, Paul Zimmermann.

### B.3.8. Contacting us

You can either contact us by email at

```
contact@texmacs.org
```

or by regular mail at

```
Joris van der Hoeven
Dépt. de Mathématiques (Bât. 425)
Université Paris-Sud
91405 Orsay Cedex
France
```

There are also several TeXmacs mailing lists:

```
texmacs-users@texmacs.org
texmacs-info@texmacs.org
texmacs-dev@gnu.org
```

## B.4. Important changes in TeXmacs

Below, we briefly describe the most important changes which have occurred in TeXmacs since version 0.3.3.15. We also maintain a more detailed change log.

In general, when upgrading to a new version, we recommend you to make backups of your old TeXmacs files before opening them with the newer version of TeXmacs. In the unlikely case when your old file does not open in the correct way, please send a bug report to

```
bugs@texmacs.org
```

and send your old document as an attached file. Do not forget to mention your version of TeXmacs and the system you are using.

### B.4.1. Improved titles (1.0.4.1)

From now on, titles of documents are more structured. This makes it easier to render the same title information in the appropriate ways for different styles. Old-style titles are automatically upgraded, but the result is only expected to be correct for documents with a single author. For documents with multiple authors, you may have to re-enter the title using our new interface.

### B.4.2. Improved style sheets and source editing mode (1.0.3.5)

We are making it easier for users to edit style sheets. This improvement made it necessary to simplify many of the standard TeXmacs styles and packages, so that it will be easier to customize them. However, if you already designed some style files, then this may break some of their features. We mainly redesigned the list environments, the section environments and automatic numbering. Please report any problems to us.

### B.4.3. Renaming of tags and environment variables (1.0.2.7 – 1.0.2.8)

Most environment variables and some tags have been renamed, so that these names no longer contain whitespace and only dashes (and no underscores) as separators.

### B.4.4. Macro expansion (1.0.2.3 – 1.0.2.7)

An important internal change concerning the data format has been made: macro expansions and function applications like

```
(expand tag arg-1 ... arg-n)
```

```
(apply tag arg-1 ... arg-n)
```

are now replaced by hard-coded tags

```
(tag arg-1 ... arg-n)
```

Moreover, functions have systematically been replaced by macros. The few built-in functions which may take an arbitrary number of arguments have been rewritten using the new xmacro construct. If you ever wrote such a function yourself, then you will need to rewrite it too.

The new approach favorites a uniform treatment of macros and functions and makes the internal representation match with the corresponding Scheme representation. More and more information about tags will gradually be stored in the D.R.D. (Data Relation Definition). This information is mostly determined automatically using heuristics.

Notice that some perverse errors might arise because of the above changes. Please keep copies of your old files and report any suspicious behaviour to us.

### B.4.5. Formatting tags (1.0.2 – 1.0.2.1)

All formattings constructs without arguments (like line breaks, indentation directives, etc.) have been replaced by tags of arity zero. This makes most new documents badly unreadable for older versions of TeXmacs and subtle errors might occasionnaly occur when saving or loading, or during other editing operations.

### B.4.6. Keyboard (1.0.0.11 – 1.0.1)

The TeXmacs keybindings have been rationalized. Here follows a list of the major changes:

- The `E-` prefix has been renamed to `M-`.
- `escape` is equivalent to `M-` and `escape`-`escape` to `A-`.
- Mode dependent commands are now prefixed by `A-`. In particular, accents are typed using `A-` instead of `E-`.
- Variants are now obtained using `tab` instead of `*` and you can circle back using `shift-tab`.
- Greek characters are now typed using `A-C-`, `F5`, or the hyper modifier, which can be configured in Edit → Preferences. You may also obtain Greek characters as variants of Latin characters. For instance, `p tab` yields $\pi$.
- The signification of the cursor keys in combination with control, alt and meta has changed.

You may choose between several "look and feels" for the keyboard behaviour in Edit → Preferences → Look and feel. The default is Emacs, but you may choose Old style if you want to keep the behaviour to which you may be used now.

### B.4.7. Menus (1.0.0.7 – 1.0.1)

Several changes have been made in the menus. Here follows a list of the major changes:

- Buffer has been renamed as Go.
- Several items from File have been moved to View.
- The Edit → Import and Edit → Export items have been moved to Tools → Selections.
- The Insert menu has been split up into the menus Insert, Text and Mathematics.
- The Text and Paragraph menus have been merged together in one Format menu.
- Options has been spread out across Document, View, Tools and Edit → Preferences.

### B.4.8. Style files (1.0.0.4)

Many changes have been made in the organization of the TeXmacs style files. Personal style files which depend on intermediate TeXmacs packages may require some slight adaptations.

We are working towards a stabilization of the standard style files and packages. At the end of this process, it should be easy to adapt existing LaTeX style files for journals to TeXmacs by customizing these standard style files and packages. As soon as we have time, we plan to provide online documentation on how to do this at Help → Online documentation.

### B.4.9. Tabular material (0.3.5)

The way tabular material is treated has completely changed. It has become much easier to edit tables, matrices, equation arrays, etc. Also, many new features have been implemented, such as background color, border, padding, hyphenation, subtables, etc. However, the upgrading of old tabular material might sometimes be erroneous, in which case we invite you to submit a bug report.

### B.4.10. Document format (0.3.4)

The TeXmacs document format has profoundly changed in order to make TeXmacs compatible with XML in the future. Most importantly, the old style environments like

```
<assign|env|<environment|open|close>>,
```

which are applied via matching pairs <begin|env>text<end|env>, have been replaced by macros

```
<assign|env|<macro|body|open<body>close>>,
```

which are applied via single macro expansions <expand|env|text>. Similarly, matching pairs <set|var|val>text<reset|var> of environment variable changes are replaced by a <with|var|val|text> construct (close to XML attributes). From a technical point of view, these changes lead to several complications if the text body consists of several paragraphs. As a consequence, badly structured documents may sometimes display differently in the new version (although I only noticed one minor change in my own documents). Furthermore, in order to maintain the higher level of structure in the document, the behaviour of the editor in relation to multiparagraph environments has slightly changed.

# Anhang C

# Contributing to GNU $\TeX_{\text{MACS}}$

## C.1. Use $\TeX_{\text{MACS}}$

One of the best ways to contribute to GNU $\TeX_{\text{MACS}}$ is by using it a lot, talk about it to friends and collegues, and to report me about bugs or other unnatural behaviour. Please mention the fact that you wrote articles using $\TeX_{\text{MACS}}$ when submitting them. You can do this by putting the made-by-TeXmacs tag somewhere inside your title using Text → Titel → TeXmacs Notiz.

Besides these general (but very important) ways to contribute, your help on the more specific subjects below would be appreciated. Don't hesitate to contact us if you want to contribute to these or any other issues. In the Hilfe menu you can find documentation about the source code of $\TeX_{\text{MACS}}$, its document format, how to write interfaces with other formats, and so on.

## C.2. Making donations to the $\TeX_{\text{MACS}}$ project

**Making donations to TeXmacs through the SPI organization.**

One very important way to support $\TeX_{\text{MACS}}$ is by donating money to the project. $\TeX_{\text{MACS}}$ is currently one of the projets of SPI (Software in the Public Interest; see `http://www.spi-inc.org`). You may make donations of money to TeXmacs via this organization, by noting on your check or e-mail for wire transfers that your money should go to the TeXmacs project. You may also make donations of equipment or services or donations through vendors. See the SPI website for more information. We will maintain a webpage with a list of donors soon (if you agree to be on the list).

**Details on how to donate money.**

To make a donation, write a check or money order to:

> *Software in the Public Interest, Inc.*

and mail it to the following address:

> Software in the Public Interest, Inc.
> P.O. Box 502761
> Indianapolis, IN 46250-7761
> United States

To make an electronic transfer (this will work for non-US too), you need to give your bank the routing number and account number as follows:

> The SPI bank account is at American Express Centurion Bank.
> `Routing Number: 124071889`
> `Account Number: 1296789`

Don't forget to note on your check or e-mail for wire transfers that the money should be spent on the TeXmacs projet. In addition you may specify a more specific purpose on which you would like us to spend the money. You may also contact us for a more detailed discussion on this issue.

**Important notes.**

Let the SPI Treasurer (`treasurer@spi-inc.org`) know if you have problems. When you have completed the electronic wire, please send a copy of the receipt to the above address so there is a copy of your donation. The copy you send to the treasurer is important. You may also want to contact the TeXmacs team in order to make sure that the money arrived on the TeXmacs account.

***Note:*** *The SPI address and account numbers may change from time to time. Please do not copy the address and account numbers, but rather point to the page* `http://www.spi-inc.org/donations` *to ensure that donors will always see the most current information.*

*Donations in Europe* can be done through our partner in Germany, ffis e.V. If you are interested in using their bank account (to save international money transfer costs), please check the instructions on `http://www.ffis.de/Verein/spi-en.html`.

## C.3. CONTRIBUTE TO THE GNU $\mathrm{T_EX_{MACS}}$ DOCUMENTATION

There is a high need for good documentation on $\mathrm{T_EX_{MACS}}$ as well as people who are willing to translate the existing documentation into other languages. The aim of this site is to provide high quality documentation. Therefore, you should carefully read the guide-lines on how to write such documentation.

### C.3.1. Introduction on how to contribute

High quality documentation is both a matter of content and structure. The content itself has to be as pedagogic as possible for the targeted group of readers. In order to achieve this, you should not hesitate to provide enough examples and illustrative screen shots whenever adequate. Although the documentation is not necessarily meant to be complete, we do aim at providing relatively stable documentation. In particular, you should have checked your text against spelling errors. The more experimental documentation should be put in the `incoming` directory or on the $\mathrm{T_EX_{MACS}}$ Wiki.

It is also important that you give your documentation as much structure as possible, using special markup from the `tmdoc` style file. This structure can be used in order to automatically compile printable books from your documentation, to make it suitable for different ways of viewing, or to make it possible to efficiently search a certain type of information in the documentation. In particular, you should always provide copyright and license information, as well as indications on how to traverse your documentation, if it contains many files.

WARNUNG C.1. Don't forget to select Dokument → Sprache → Ihre Sprache for each translated file. This will cause some content to be translated automatically, like the menus or some names of keys. Also, we recommend to run the TEXMACS spell checker on each translated document; this also requires the prior selection of the right document language.

### C.3.2. Using CVS

The present TEXMACS documentation is currently maintained on `texmacs.org` using CVS (Concurrent Version System). In order to contribute, you should first create an account as explained on

http://www.texmacs.org/tmweb/download/cvs.en.html

In fact, the CVS system is not ideal for our documentation purpose, because it is not very dynamic. In the future, we plan to create a dedicated publication website, which will allow you to save documents directly to the web. It should also allow the automatic conversion of the documentation to other formats, the compilation of books, etc.

### C.3.3. Conventions for the names of files

Most documentation should be organized as a function of the topic in a directory tree. The subdirectories of the top directory are the following:

**devel.** Documentation for developers.

**examples.** Examples of TEXMACS documents.

**incoming.** Incoming documentation, which is still a bit experimental.

**main.** The main documentation.

**meta.** How to write documentation and the compilation of documentation.

Please try to keep the number of entries per directory reasonably small.

File names in the main directory should be of the form `type-name.language.tm`. In the other directories, they are of the form `name.language.tm`. Here `type` is a major indication for the type of documentation; it should be one of the following:

**adv.** Documentation for advanced users.

**man.** For inclusion in the TEXMACS manual.

**tut.** For inclusion in the TEXMACS tutorial.

You should try to keep the documentation on the same topic together, regardless of the type. Indeed, this allows you to find more easily all existing documentation on a particular topic. Also, it may happen that you want to include some documentation which was initially meant for the tutorial in the manual. The `language` in which is the documentation has been written should be a two letter code like `en`, `fr`, etc. The main `name` of your file should be the same for the translations in other languages. For instance, `man-keyboard.en.tm` should not be translated as `man-clavier.fr.tm`.

### C.3.4. Copyright information & the Free Documentation License

All documentation on the `texmacs-doc` site falls under the GNU Free Documentation License. If you write documentation for TeXmacs on this site, then you have to agree that it will be distributed under this license too. The copyright notice

```
Permission is granted to copy, distribute and/or modify this
document under the terms of the GNU Free Documentation License,
Version 1.1 or any later version published by the Free Software
Foundation; with no Invariant Sections, with no Front-Cover
Texts, and with no Back-Cover Texts. A copy of the license
is included in the section entitled "GNU Free Documentation
License".
```

should be specified at the end of *each* file. This should be done inside the tmdoc-license macro, in a similar way as at the end of the present document. When automatically generating a printed book from several documentation files, this will enable us to include the license only once.

You keep (part of) the copyright of all documentation that you will write for TeXmacs on the official `texmacs-doc` site. When you or others make additions to (or modifications in, or translations of) the document, then you should add your own name (at an appropriate place, usually at the end) to the existing copyright information. The copyright notice should be specified using the tmdoc-copyright function just before the license information at the end of the document. The first argument of this function contains a year or a period. Each remaining argument indicates one of the copyright holders. When combining (pieces of) several documents into another one, you should merge the copyright holders. For cover information (on a printed book for instance), you are allowed to list only the principal authors, but a complete list should be given at a clearly indicated place.

### C.3.5. Traversing the TeXmacs documentation

As a general rule, you should avoid the use of sectioning commands inside the TeXmacs documentation and try to write small help pages on well identified topics. At a second stage, you should write recursive "meta help files" which indicate how to traverse the documentation in an automatic way. This allows the reuse of a help page for different purposes (a printed manual, a web-oriented tutorial, etc.).

The `tmdoc` style provides three markup macros for indicating how to traverse documentation. The traverse macro is used to encapsulate regions with traversal information. The branch macro indicates a help page which should be considered as a subsection and the continue macro indicates a follow-up page. Both the branch and the continue macro take two arguments. The first argument describes the link and the second argument gives the physical relative address of the linked file.

Typically, at the end of a meta help file you will find several branch or continue macros, inside one traverse macro. At the top of the document, you should also specify a title for your document using the tmdoc-title macro. When generating a printed manual from the documentation, a chapter-section-subsection structure will automatically be generated from this information and the document titles. Alternatively, one might automatically generate additional buttons for navigating inside the documentation using a browser.

## C.3.6. Using the tmdoc style

Besides the copyright information macros and traversal macros, which have been documented before, the tmdoc style comes with a certain number of other macros and functions, which you should use whenever appropriate:

**key.** This macro is used to indicate keyboard input like `C-x C-s`. The specialized macros kbd-gen, kbd-text, kbd-math, kbd-symb, kbd-big, kbd-large, kbd-ia, kbd-exec and kbd-table are used for keyboard input corresponding to a specific type of action or mode. For instance, kbd-math corresponds to keyboard shortcuts for mathematical operations, such as `A-f`, which starts a fraction.

**menu.** This function with an arbitrary number of arguments indicates a menu like Datei or Dokument → Sprache. Menu entries are automatically translated by this function.

**markup.** This macro is used in order to indicate a macro or a function like section.

**tmstyle.** This macro indicates the name of a TeXmacs style file or package like `article`.

**tmpackage.** This macro indicates the name of a TeXmacs package like `std-markup`.

**tmdtd.** This macro indicates the name of a TeXmacs d.t.d. like `number-env`.

Notice that the contents of none of the above tags should be translated into foreign languages. Indeed, for menu tags, the translations are done automatically, so as to keep the translations synchronized with the translations of the actual TeXmacs menus. In the cases of markup, styles, packages and d.t.d.s, it is important to keep the original name, because it often corresponds to a file name.

The following macros and functions are used for linking and indexing purposes, although they should be improved in the future:

**simple-link.** This macro takes an URL $x$ as argument and is a hyperlink with name and destination $x$.

**hyper-link.** This macro is a usual hyperlink.

**concept-link.** This macro takes a concept as argument. Later on an appropriate hyperlink might be created automatically from this and the other documentation.

**only-index.** Index a simple string.

**def-index.** Definition of a new concept; the text is printed in italic and indexed.

**re-index.** Reappearance of an already defined concept; the text is printed in roman and put in the index.

The following tags are also frequently used:

**icon.** Link to an icon in a central directory like `$TEXMACS_PATH/doc/images/pixmaps`.

**screenshot.** Link to a screenshot. The actual screenshots are stored in a central directory like `$TEXMACS_PATH/doc/images/screenshots`.

**scheme.** The SCHEME language.

**cpp.** The C++ language.

**framed-fragment.** For displaying a piece of code in a nice frame.

**scheme-fragment.** For multi-paragraph SCHEME code.

**cpp-fragment.** For multi-paragraph C++ code.

**tm-fragment.** For a piece of TEXMACS markup code in SCHEME format.

**scheme-code.** For a short piece of SCHEME code.

**cpp-code.** For a short piece of C++ code.

**descriptive-table.** For descriptive tables; such tables can be used to document lists of keyboard shortcuts, different types of markup, etc.

The `tmdoc` style inherits from the `generic` style and you should use macros like em, verbatim, itemize, etc. from this style whenever appropriate.

## C.4. INTERNATIONALIZATION

The support of a maximal number of foreign languages is another major challenge in which your help would be appreciated. Making the translations to support a new language usually requires several days of work. We therefore recommend you to find some friends or collegues who are willing to help you.

The procedure for adding a new language is as follows

- You copy the file `english-new.scm` to `english-`*`yourlanguage`*`.dic` in `langs/natural/dic` and fill out the corresponding translations. You may want to use Andrey Grozin's dictionary tool at

  `http://www.texmacs.org/Data/dictool.py.gz`

  In order to use it, may sure that Python is installed on your system, download the file, gunzip it, make it executable and run it.

- You tell me about any special typographical rules in your language and handy keystrokes for producing special characters.

- I take care of the hyphenation and typographical issues, but you test them.

- If you have enough time, you may also consider the translation of (part of) the existing documentation.

Of course, the support for languages get out of date each time that new features are added to TEXMACS. For this reason, we also maintain a file `miss-english-`*`yourlanguage`*`.dic` with all missing translation for your language, once that it has been added. Please do not hesitate to send inclomplete versions of `english-`*`yourlanguage`*`.dic` or `miss-english-`*`yourlanguage`*`.dic`; someone else may be willing to complete them.

## C.5. Writing data converters

If you are familiar with TeX, LaTeX, Html, Xml, Sgml, Mathml, Pdf, Rtf, or any other frequently used data format, please consider contributing to writing good converters for one or more of these formats. In Hilfe → Quellcode → Datenformat you will find details about the TeXmacs data format and in Hilfe → Quellcode → Datenkonvertierung we give some suggestions which might be helpful for these projects.

## C.6. Porting TeXmacs to other platforms

Currently, TeXmacs is supported on most major Unix/X-Window platforms and a Windows port should be ready soon. Nevertheless, your help is appreciated in order to keep the existing ports working. Some remaining challenges for porting TeXmacs are:

- A native port for MacOS-X.
- Ports to PDAs, first of all those which run Linux. It should be noticed that, with the current support for Freetype, TeXmacs no longer depends on TeX/LaTeX for its fonts. We expect it to be possible to obtain a reasonable ports for TeXmacs on PDAs with 32Mb and at least 100MHz clock-speed. Of course, one also needs to customize the menus and/or icon bars, but this should not be hard.

  TeXmacs ports to PDAs would be particularly interesting in combination with the available plug-ins for doing scientific computations.

## C.7. Interfacing TeXmacs with other systems

It is quite easy to write interfaces between TeXmacs and computer algebra systems or other scientific programs with structured output. Please consider writing interfaces between TeXmacs and your favorite system(s). TeXmacs has already been interfaced with several other free systems, like Giac, Macaulay 2, Maxima, GNU Octave, Pari, Qcl, gTybalt, Yacas. Detailed documentation on how to add new interfaces is available in the Hilfe → Schnittstellen menu.

## C.8. TeXmacs over the network and over the web

With the current technology of mutator tags, it should be quite easy to write a plug-in for TeXmacs for doing instant messenging or live-conferencing. We are very interested in people who would like to help with this. The same techniques might be used for collaborative authoring and educational purposes.

Besides live conferencing, we are also interested by people who are willing to program better integration of TeXmacs with the web. As a first step, this would require an internal C++ plug-in based on Wget or Curl for accessing web-pages, which supports cookies, security, etc. At a second stage, these features should be exploited by the Html converters. At the last stage, one might develop more general web-based services.

## C.9. Become a TeXmacs developer

Apart from the kind of contributions which have been described in more detail above, there are many more issues where your help would be appreciated. Please take a look at our plans for the future for more details. Of course, you should feel free to come up with your own ideas and share them with us on the `texmacs-dev@gnu.org` mailing list!

# Anhang D
# Interfacing TeXmacs with other programs

## D.1. Introduction

In this chapter we describe how to interface TeXmacs with an extern application. Such interfaces should be distributed in the form of plugins. The plug-in may either contain the extern application, or provide the “glue” between TeXmacs and the application. Usually, interfaces are used interactively in shell sessions (see Text → Sitzung). But they may also be designed for background tasks, such as spell checking or typesetting.

The communication between TeXmacs and the application takes place using a customizable input format and the special *TeXmacs meta-format* for output from the plug-in. The meta-format enables you to send structured output to TeXmacs, using any common format like `verbatim`, LaTeX, PostScript, HTML, or TeXmacs itself. This is useful when adding a TeXmacs interface to an existing system, since LaTeX or PostScript output routines are often already implemented. It will then suffice to put the appropriate markers in order to make a first interface with TeXmacs.

As soon as basic communication between your application and TeXmacs is working, you may improve the interface in many ways. Inside shell sessions, there is support for prompts, default inputs, tab-completion, mathematical and multi-line input, etc. In general, your application may take control of TeXmacs and modify the user interface (menus, keyboard, etc.) or add new Scheme routines to TeXmacs. Your application may even extend the typesetter.

In the directory `$TEXMACS_PATH/examples/plugins`, you can find many examples of simple plug-ins. In the next sections, we will give a more detailed explanation of the interfacing features of TeXmacs on the hand of these examples. In order to try one of these examples, we recall that you just have to copy it to either one of the directories

`$TEXMACS_PATH/plugins`

`$TEXMACS_HOME_PATH/plugins`

and run the `Makefile` (if there is one).

## D.2. Basic input/output using pipes

The configuration and the compilation of the `minimal` plug-in is described in the chapter about plug-ins. We will now study the source file `minimal/src/minimal.cpp`. Essentially, the `main` routine is given by

```
int
main () {
  display-startup-banner
  while (true) {
    read-input
    display-output
  }
  return 0;
}
```

By default, TeXmacs just send a '\n'-terminated string to the application as the input. Consequently, the code for ***read-input*** is given by

```
char buffer[100];
cin.getline (buffer, 100, '\n');
```

The output part is more complicated, since TeXmacs needs to have a secure way for knowing whether the output has finished. This is accomplished by encapsulating each piece of output (in our case both the display banner and the interactive output) inside a block of the form

```
DATA_BEGIN format:message DATA_END
```

Here `DATA_BEGIN` and `DATA_END` stand for special control characters:

```
#define DATA_BEGIN   ((char) 2)
#define DATA_END     ((char) 5)
#define DATA_ESCAPE  ((char) 27)
```

The `DATA_ESCAPE` is used for producing the `DATA_BEGIN` and `DATA_END` characters in the ***message*** using the rewriting rules

```
DATA_ESCAPE DATA_BEGIN   ⟶  DATA_BEGIN
DATA_ESCAPE DATA_END     ⟶  DATA_END
DATA_ESCAPE DATA_ESCAPE  ⟶  DATA_ESCAPE
```

The ***format*** specifies the format of the ***message***. For instance, in our example, the code of ***display-startup-banner*** is given by

```
cout << DATA_BEGIN << "verbatim:";
cout << "Hi there!";
cout << DATA_END;
fflush (stdout);
```

Similarly, the code of ***display-output*** is given by

```
cout << DATA_BEGIN << "verbatim:";
cout << "You typed " << buffer;
cout << DATA_END;
fflush (stdout);
```

BEMERKUNG D.1. For synchronization purposes, TeXmacs will assume that the output is finished as soon as it encounters the `DATA_END` which closes the initial `DATA_BEGIN`. So all output has to be inside one *single* outer `DATA_BEGIN`-`DATA_END` block: if you send more blocks, then TeXmacs will retake control before reading all your output. It *is* possible to nest `DATA_BEGIN`-`DATA_END` blocks though, as we will see below.

BEMERKUNG D.2. In our example, the C++ code for the application is included in the plug-in. In the case when you are writing a TeXmacs interface for an existing application ***myapp***, the convention is to create a `--texmacs` option for this program. Then it is no longer necessary to have ***myapp***`/src` and ***myapp***`/bin` directories for your plug-in and it suffices to configure the plug-in by putting something like the following in ***myapp***`/progs/init-`***myapp***`.scm`:

```
(plugin-configure myapp
  (:require (url-exists-in-path? "myapp"))
  (:launch "myapp --texmacs")
  (:session "Myapp"))
```

In the case when you do not have the possibility to modify the source code of ***myapp***, you typically have to write an input/output filter `tm_myapp` for performing the appropriate rewritings. By looking at the standard plug-ins distributed with TeXmacs in

`$TEXMACS_PATH/plugins`

you can find several examples of how this can be done.

## D.3. Formatted and structured output

In the previous section, we have seen that output from applications is encapsulated in blocks of the form

```
DATA_BEGIN format:message DATA_END
```

In fact, the ***message*** may recursively contain blocks of the same form. Currently implemented formats include `verbatim`, `latex`, `html`, `ps`, `scheme`. The `scheme` format is used for sending TeXmacs trees in the form of Scheme expressions.

### The `formula` plug-in.

The `formula` plug-in demonstrates the use of LaTeX as the output format. It consists of the files

`formula/Makefile`

`formula/progs/init-formula.scm`

`formula/src/formula.cpp`

The body of the main loop of `formula.cpp` is given by

```
int i, nr;
cin >> nr;
cout << DATA_BEGIN << "latex:";
cout << "$";
for (i=1; i<nr; i++)
  cout << "x_{" << i << "}+";
cout << "x_{" << i << "}$";
cout << DATA_END;
fflush (stdout);
```

Similarly, the use of nested output blocks is demonstrated by the `nested` plug-in; see in particular the source file `nested/src/nested.cpp`.

Bemerkung D.3. At the moment, we only implemented LaTeX as a standard transmission format for mathematical formulas, because this is the format which is most widely used. In the future, we intend to implement more semantically secure formats, and we recommend you to keep in mind the possibility of sending your output in tree format.

Nevertheless, we enriched standard LaTeX with the `\*` and `\bignone` commands for multiplication and closing big operators. This allows us to distinguish between

```
a \* (b + c)
```

(i.e. $a$ multiplied by $b+c$) and

```
f(x + y)
```

(i.e. $f$ applied to $x+y$). Similarly, in

```
\sum_{i=1}^m a_i \bignone + \sum_{j=1}^n b_j \bignone
```

the `\bignone` command is used in order to specify the scopes of the `\sum` operators.

It turns out that the systematic use of the `\*` and `\bignone` commands, in combination with clean LaTeX output for the remaining constructs, makes it *a priori* possible to associate an appropriate meaning to your output. In particular, this usually makes it possible to write additional routines for copying and pasting formulae between different systems.

### The `markup` plug-in.

It is important to remind that structured output can be combined with the power of TeXmacs as a structured editor. For instance, the `markup` plug-in demonstrates the definition of an additional tag foo, which is used as an additional primitive in the output of the application. More precisely, the `markup` plug-in consists of the following files:

markup/Makefile

markup/packages/session/markup.ts

markup/progs/init-markup.scm

markup/src/markup.cpp

The style package `markup.ts` contains the following definition for foo:

⟨with|*mode*|math|⟨assign|*foo*|⟨macro|*x*|⟨frac|1|1+*x*⟩⟩⟩⟩

The foo tag is used in the following way in the body of the main loop of `markup.cpp`:

```
char buffer[100];
cin.getline (buffer, 100, '\n');
cout << DATA_BEGIN << "latex:";
cout << "$\\foo{" << buffer << "}$";
cout << DATA_END;
fflush (stdout);
```

Notice that the style package `markup.ts` also defines the markup-output environment:

⟨assign|*markup-output*|⟨macro|*body*|⟨generic-output|⟨with|*par-mode*|center|*body*⟩⟩⟩⟩

This has the effect of centering the output in sessions started using Text → Sitzung → Markup.

## D.4. Output channels, prompts and default input

Besides blocks of the form

```
DATA_BEGIN format:message DATA_END
```

the $\mathrm{T_EX_{MACS}}$ meta-format also allows you to use blocks of the form

```
DATA_BEGIN channel#message DATA_END
```

Here ***channel*** specifies an "output channel" to which the body ***message*** has to be sent. The default output channel is `output`, but we also provide channels `prompt` and `input` for specifying the prompt and a default input for the next input in a session. Default inputs may be useful for instance be useful for demo modes of computer algebra systems. In the future, we also plan to support `error` and `status` channels.

### The `prompt` plug-in.

The `prompt` plug-in shows how to use prompts. It consists of the files

`prompt/Makefile`

`prompt/progs/init-prompt.scm`

`prompt/src/prompt.cpp`

The routine for displaying the next prompt is given by

```
void
next_input () {
  counter++;
  cout << DATA_BEGIN << "prompt#";
  cout << "Input " << counter << "] ";
  cout << DATA_END;
}
```

This routine is both used for displaying the startup banner

```
cout << DATA_BEGIN << "verbatim:";
cout << "A LaTeX -> TeXmacs converter";
next_input ();
cout << DATA_END;
fflush (stdout);
```

and in the body of the main loop

```
char buffer[100];
cin.getline (buffer, 100, '\n');
cout << DATA_BEGIN << "verbatim:";
cout << DATA_BEGIN;
cout << "latex:$" << buffer << "$";
cout << DATA_END;
next_input ();
cout << DATA_END;
fflush (stdout);
```

## D.5. SENDING COMMANDS TO TEXMACS

The application may use `command` as a very particular output format in order to send SCHEME commands to TEXMACS. In other words, the block

```
DATA_BEGIN command:cmd DATA_END
```

will send the command ***cmd*** to TEXMACS. Such commands are executed immediately after reception of `DATA_END`. We also recall that such command blocks may be incorporated recursively in larger `DATA_BEGIN`-`DATA_END` blocks.

### The `menus` plug-in.

The `nested` plug-in shows how an application can modify the TEXMACS menus in an interactive way. The plug-in consists of the files

menus/Makefile

menus/progs/init-menus.scm

menus/src/menus.cpp

The body of the main loop of `menus.cpp` simply contains

```
char buffer[100];
cin.getline (buffer, 100, '\n');
cout << DATA_BEGIN << "verbatim:";
cout << DATA_BEGIN << "command:(menus-add \""
     << buffer << "\")" << DATA_END;
cout << "Added " << buffer << " to menu";
cout << DATA_END;
fflush (stdout);
```

The SCHEME macro `menus-add` is defined in `init-menus.scm`:

```
(menu-bind menus-menu
  ("Hi" (insert "Hello world")))

(menu-extend texmacs-extra-menu
  (if (equal? (get-env "prog language") "menus")
      (=> "Menus" (link menus-menu))))

(define-macro (menus-add s)
  `(menu-extend menus-menu
     (,s (insert ,s))))
```

The configuration of `menus` proceeds as usual:

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

# D.6. Background evaluations

Until now, we have always considered interfaces between TeXmacs and applications which are intended to be used interactively in shell sessions. But there also exists a Scheme command

```
(plugin-eval plugin session expression)
```

for evaluating an expression using the application. Here ***plugin*** is the name of the plug-in, ***session*** the name of the session and ***expression*** a Scheme expression which represents a TeXmacs tree.

### The `substitute` plug-in.

Background evaluations may for instance be used in order to provide a feature which allows the user to select an expression and replace it by its evaluation. For instance, the `substitute` plug-in converts mathematical LaTeX expressions into TeXmacs, and it provides the `C-F12` keyboard shortcut for replacing a selected text by its conversion. The plug-in consists of the following files

`substitute/Makefile`

`substitute/progs/init-substitute.scm`

`substitute/src/substitute.cpp`

The main evaluation loop of `substitute.cpp` simply consists of

```
char buffer[100];
cin.getline (buffer, 100, '\n');
cout << DATA_BEGIN;
cout << "latex:$" << buffer << "$";
cout << DATA_END;
fflush (stdout);
```

Moreover, the configuration file `init-substitute.scm` contains the following code for replacing a selected region by its evaluation

```
(define (substitute-substitute)
  (import-from (texmacs plugin plugin-cmd))
  (if (selection-active-any?)
      (let* ((t (tree->stree (the-selection)))
             (u (plugin-eval "substitute" "default" t)))
        (clipboard-cut "primary")
        (insert (stree->tree u)))))
```

as well as the keyboard shortcut for `C-F12`:

```
(kbd-map
  ("C-F12" (substitute-substitute)))
```

Notice that these routines should really be defined in a separate module for larger plug-ins.

### The `secure` plug-in.

Another example of using an interface in the background is the `secure` plug-in which consists of the files

- `secure/Makefile`
- `secure/packages/secure.ts`
- `secure/progs/init-secure.scm`
- `secure/progs/secure-secure.scm`
- `secure/src/secure.cpp`

Just as `substitute.cpp` above, the main program `secure.cpp` just converts mathematical LATEX expressions to TEXMACS. The `secure-secure.scm` module contains the *secure* SCHEME routine `latexer`:

```
(tm-define (latexer s)
  (:type (tree -> object))
  (:synopsis "convert LaTeX string to TeXmacs tree using
plugin")
  (:secure #t)
  (plugin-eval "secure" "default" (tree->string s)))
```

It is important to define `latexer` as being secure, so that it can be used in order to define additional markup using the extern primitive. This is done in the style file `secure.ts`:

See a LaTeX math command as a TeXmacs expression via plug-in
⟨assign|*latexer*|⟨macro|*x*|⟨extern|`latexer`|*x*⟩⟩⟩

After compilation, installation, relaunching TEXMACS and selecting Dokument → Paket benutzen → secure, you will now be able to use latexer as a new primitive. The primitive takes a mathematical LATEX expression as its argument and displays its TEXMACS conversion.

## D.7. MUTATOR TAGS

A mutator tag is of the form ⟨mutator|*body*|*cmd*⟩, where *body* is the visible body of the tag and *cmd* a secure SCHEME script which is called periodically and which is allowed to modify the *body* of the mutator (or even any other part of the document). During the execution of *cmd* the function `mutator-path` yields the position of the *body* in the document tree.

Mutator tags are a particularly interesting feature of TEXMACS for producing highly interactive documents. For instance, inside computer algebra sessions, the output is retrieved inside a mutator tag (which automatically removes itself when the output is complete). In the future, mutators might be used for live conferencing or in interfaces with proof systems.

BEMERKUNG D.4. Mutator tags only work properly when the explicitly occur in the document; they will not work if they merely occur in the body of a macro, even if the body of the mutator remains accessible.

Indeed, the current implementation of TEXMACS searches for mutator tags in all documents after a small period of inactivity. Documents which are known not to contain mutator tags are ignored during this research. Of course, this implementation is both quite efficient and incompatible with the macro system. So there is room for future improvements.

BEMERKUNG D.5. For efficiency reasons, it is recommended that mutator tags mainly modify there own bodies and not other parts of the document, except at really exceptional occasions.

**The `mutator` plug-in.**

A very simple example with two types of mutators is provided in the **`mutator`** plug-in. It provides the user with two keyboard shortcuts `C-F11` and `C-F12`, which respectively insert the current time and some blinking text. The plug-in consists of the single file

`mutator/progs/init-mutator.scm`

The `C-F11` key simply inserts ⟨mutator|text|(mutate-date)⟩ into the main text:

```
(kbd-map ("C-F11" (insert '(mutator "" "(mutate-date)"))))
```

The secure SCHEME routine **`mutate-date`** is defined as follows:

```
(tm-define (mutate-date)
  (:secure #t)
  (let* ((p (the-mutator-path))
         (date (var-eval-system "date +\"%H:%M:%S\"")))
    (tm-assign-diff p date)))
```

The **`tm-assign-diff`** command is convenient, because it only modifies the document if a real change occurred.

The insertion of blinking content is slightly more complex, since it also takes into account the current content of the mutator tag. The `C-F12` key inserts ⟨mutator|text|(mutate-blink)⟩ into the main text and puts the cursor after the text in the body of the mutator:

```
(kbd-map ("C-F12" (insert-go-to '(mutator "text" "(mutate-
blink)")
                                          '(0 4))))
```

The secure SCHEME routine **`mutate-blink`** is defined as follows:

```
(tm-define (mutate-blink)
  (:secure #t)
  (let* ((mod (lambda (x y) (* y (- (/ x y) (floor (/ x
y))))))
         (p (the-mutator-path))
         (t (tm-subtree p))
         (s (string->number (var-eval-system "date +\"%S\"")))
         (e (mod s 4)))
    (if (and (<= e 1) (not (match? t '(strong :1))))
        (tm-ins-unary p 'strong))
    (if (and (>= e 2) (match? t '(strong :1)))
        (tm-rem-unary p))))
```

BEMERKUNG D.6. Notice that the above examples are only meant to illustrate the use of mutators. Ideally speaking, dates and blinking content should not make use of mutators, since mutators continuously *modify* the document (think about undoing changes, for instance). In the future, we plan to add primitives like animations and movies to $\text{T}_\text{E}\text{X}_{\text{MACS}}$, for which the content remains fixed, but whose presentation changes over time.

## D.8. Mathematical and customized input

The TeXmacs meta-format allows application output to contain structured text like mathematical formulas. In a similar way, you may use general TeXmacs content as the input for your application. By default, only the text part of such content is kept and sent to the application as a string. Moreover, all characters in the range 0–31 are ignored, except for `'\t'` and `'\n'` which are transformed into spaces. There are two methods to customize the way input is sent to your application. First of all, the configuration option

```
(:serializer ,routine)
```

specifies a scheme function for converting TeXmacs trees to string input for your application, thereby overriding the default method. This method allows you for instance to treat multi-line input in a particular way or the perform transformations on the TeXmacs tree.

The `:serialize` option is a very powerful, but also a very abstract way to customize input: it forces you to write a complete input transformation function. In many circumstances, the user really wants to rewrite two dimensional mathematical input to a more standard form, like rewriting $\frac{a}{b}$ to `((a)/(b))`. Therefore, a second way for customizing the input is to use the command

```
(plugin-input-converters myplugin
  rules)
```

This command specifies input conversion rules for ***`myplugin`*** for "mathematical input" and reasonable defaults are provided by TeXmacs. Each rule is of one of the following two forms:

**Leaf transformation rules.**

Given two strings ***`symbol`*** and ***`conversion`***, the rule

```
(symbol conversion)
```

specifies that the TeXmacs symbol ***`symbol`*** should be converted to ***`conversion`***.

**Tag transformation rules.**

Given a symbol ***`tag`*** and a SCHEME function ***`routine`***, the rule

```
(tag routine)
```

specifies that ***`routine`*** will be used as the conversion routine for ***`tag`***. This routine should just write a string to the standard output. The SCHEME function `plugin-input` may be used for the recursive transformation of the arguments of the tag.

### The `input` plug-in.

The `input` plug-in demonstrates the use of customized mathematical input. It consists of the files

`input/Makefile`

`input/packages/session/input.ts`

`input/progs/init-input.scm`

`input/progs/input-input.scm`

input/src/input.cpp

The SCHEME configuration code in `init-input.scm` is given by

```
(plugin-configure input
  (:require (url-exists-in-path? "input.bin"))
  (:initialize (input-initialize))
  (:launch "input.bin")
  (:session "Input"))
```

Here `input-initialize` is an initialization routine which adds the new input conversion rules in a lazy way:

```
(define (input-initialize)
  (import-from (texmacs plugin plugin-convert))
  (lazy-input-converter (input-input) input))
```

In other words, the module `input-input.scm` will only be loaded when we explicitly request to make a conversion. The conversion rules in `input-input.scm` are given by

```
(plugin-input-converters input
  (frac input-input-frac)
  (special input-input-special)
  ("<vee>" "||")
  ("<wedge>" "&&"))
```

This will cause $\vee$ and $\wedge$ to be rewritten as `||` and `&&` respectively. Fractions $\frac{a}{b}$ are rewritten as `((a):(b))` using the routine

```
(define (input-input-frac t)
  (display "((")
  (plugin-input (car t))
  (display "):(")
  (plugin-input (cadr t))
  (display "))"))
```

In the additional style file `input.ts` we also defined some additional markup special:

```
⟨assign|special|
  ⟨macro|body|
    ⟨block|
      ⟨tformat|
        ⟨cwith|1|1|1|1|cell-background|pastel green⟩|
        ⟨table|
          ⟨row|⟨cell|body⟩⟩⟩⟩⟩⟩⟩
```

This tag is rewritten using the special conversion rule

```
(define (input-input-special t)
  (display "[[[SPECIAL:")
  (plugin-input (car t))
  (display "]]]"))
```

As to the C++ code in `input.cpp`, the startup banner automatically puts the shell session in mathematical input mode:

```
cout << DATA_BEGIN << "verbatim:";
cout << DATA_BEGIN << "command:(session-use-math-input #t)"
     << DATA_END;
cout << "Convert mathematical input into plain text";
cout << DATA_END;
fflush (stdout);
```

In the main loop, we content ourselves the reproduce the input as output:

```
char buffer[100];
cin.getline (buffer, 100, '\n');
cout << DATA_BEGIN << "verbatim:";
cout << buffer;
cout << DATA_END;
fflush (stdout);
```

## D.9. TAB-COMPLETION

By default, TEXMACS looks into your document for possible tab-completions. Inside sessions for your application, you might wish to customize this behaviour, so as to complete built-in commands. In order to do this, you have to specify the configuration option

```
(:tab-completion #t)
```

in your `init-`*`myplugin`*`.scm` file, so that TEXMACS will send special tab-completion requests to your application whenever you press `Tab` inside a session. These commands are of the form

```
DATA_COMMAND (complete input-string cursor-position) return
```

Here `DATA_COMMAND` stands for the special character `'\20'` (ASCII 16). The ***`input-string`*** is the complete string in which the `Tab` occurred and the ***`cursor-position`*** is an integer which specifies the position of the cursor when you pressed `Tab`. TEXMACS expects your application to return a tuple with all possible tab-completions of the form

```
DATA_BEGIN scheme:(tuple root completion-1 ··· completion-n) DATA_END
```

Here ***`root`*** corresponds to a substring before the cursor for which completions could be found. The strings ***`completion-1`*** until ***`completion-n`*** are the list of completions as they might be inserted at the current cursor position. If no completions could be found, then you may also return the empty string.

BEMERKUNG D.7. In principle, the tab-completion mechanism should still work in mathematical input mode. In that case, the ***`input-string`*** will correspond to the serialization of the TEXMACS input.

Bemerkung D.8. The way $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ sends commands to your application can be customized in a similar way as for the input: we provide a `:commander` configuration option for this, which works in a similar way as the `:serializer` option.

**The `complete` plug-in.**

A very rudimentary example of how the tab-completion mechanism works is given by the `complete` plug-in, which consists of the following files:

complete/Makefile

complete/progs/init-complete.scm

complete/src/complete.cpp

The startup banner in `complete.cpp` takes care of part of the configuration:

```
cout << DATA_BEGIN << "verbatim:";
format_plugin ();
cout << "We know how to complete 'h'";
cout << DATA_END;
fflush (stdout);
```

Here `format_plugin` is given by

```
void
format_plugin () {
  // The configuration of a plugin can be completed at startup
time.
  // This may be interesting for adding tab-completion a
posteriori.
  cout << DATA_BEGIN << "command:";
  cout << "(plugin-configure complete (:tab-completion #t))";
  cout << DATA_END;
}
```

In the main loop, we first deal with regular input:

```
char buffer[100];
cin.getline (buffer, 100, '\n');
if (buffer[0] != DATA_COMMAND) {
  cout << DATA_BEGIN << "verbatim:";
  cout << "You typed " << buffer;
  cout << DATA_END;
}
```

We next treat the case when a tab-completion command is sent to the application:

```
else {
  cout << DATA_BEGIN << "scheme:";
  cout << "(tuple \"h\" \"ello\" \"i there\" \"ola\"
\"opsakee\")";
  cout << DATA_END;
}
fflush (stdout);
```

As you notice, the actual command is ignored, so our example is really very rudimentary.

## D.10. Dynamic libraries

Instead of connecting your system to TeXmacs using a pipe, it is also possible to connect it as a dynamically linked library. Although communication through pipes is usually easier to implement, more robust and compatible with gradual output, the second option is faster.

In order to dynamically link your application to TeXmacs, you should follow the TeXmacs communication protocol, which is specified in the following header file:

`$TEXMACS_PATH/include/TeXmacs.h`

In this file it is specified that your application should export a data structure

```
typedef struct package_exports_1 {
  char* version_protocol; /* "TeXmacs communication protocol
1" */
  char* version_package;
  char* (*install) (TeXmacs_exports_1* TeXmacs,
                    char* options, char** errors);
  char* (*evaluate) (char* what, char* session, char**
errors);
} package_exports_1;
```

which contains an installation routine for your application, as well as an evaluation routine for further input (for more information, see the header file). TeXmacs will on its turn export a structure

```
typedef struct TeXmacs_exports_1 {
  char* version_protocol; /* "TeXmacs communication protocol
1" */
  char* version_TeXmacs;
} TeXmacs_exports_1;
```

It is assumed that each application takes care of its own memory management. Hence, strings created by TeXmacs will be destroyed by TeXmacs and strings created by the application need to be destroyed by the application.

The string `version_protocol` should contain `"TeXmacs communication protocol 1"` and the string `version_package` the version of your package. The routine `install` will be called once by TeXmacs in order to initialize your system with options `options`. It communicates the routines exported by TeXmacs to your system in the form of a pointer to a structure of type `TeXmacs_exports_1`. The routine should return a status message like

`"yourcas-version successfully linked to TeXmacs"`

If installation failed, then you should return `NULL` and `*errors` should contain an error message.

The routine `evaluate` is used to evaluate the expression `what` inside a TeXmacs-session with name `session`. It should return the evaluation of `what` or `NULL` if an error occurred. `*errors` either contains one or more warning messages or an error message, if the evaluation failed. The formats being used obey the same rules as in the case of communication by pipes.

Finally, the configuration file of your plug-in should contain something as follows:

```
(plugin-configure myplugin
  (:require (url-exists? (url "$LD_LIBRARY_PATH"
"libmyplugin.so")))
  (:link "libmyplugin.so" "myplugin_exports" "")
  further-configuration)
```

Here *myplugin*`_exports` is a pointer to a structure of the type `package_exports_1`.

BEMERKUNG D.9. It is possible that the communication protocol changes in the future. In that case, the data structures `TeXmacs_exports_1` and `package_exports_1` will be replaced by data structures `TeXmacs_exports_n` and `package_exports_n`, where `n` is the version of the protocol. These structures will always have the abstract data structures `TeXmacs_exports` and `package_exports` in common, with information about the versions of the protocol, TeXmacs and your package.

### The `dynlink` plug-in.

The `dynlink` plug-in gives an example of how to write dynamically linked libraries. It consists of the following files:

`dynlink/Makefile`

`dynlink/progs/init-dynlink.scm`

`dynlink/src/dynlink.cpp`

The `Makefile` contains

```
tmsrc = /home/vdhoeven/texmacs/src/TeXmacs
CXX = g++
LD  = g++

lib/libtmdynlink.so: src/dynlink.cpp
        $(CXX) -I$(tmsrc)/include -c src/dynlink.cpp -o
src/dynlink.o
        $(LD) -shared -o lib/libtmdynlink.so src/dynlink.o
```

so that running it will create a dynamic library `dynlink/lib/libdynlink.so` from `dynlink.cpp`. The `tmsrc` variable should contain `$TEXMACS_PATH`, so as to find the include file `TeXmacs.h`. The configuration file `init-dynlink.scm` simply contains

```
(plugin-configure dynlink
  (:require (url-exists? (url "$LD_LIBRARY_PATH"
                              "libtmdynlink.so")))
  (:link "libtmdynlink.so" "dynlink_exports" "")
  (:session "Dynlink"))
```

As to the C++ file `dynlink.cpp`, it contains a string

```
static char* output= NULL;
```

with the last output, the initialization routine

```
char*
dynlink_install (TeXmacs_exports_1* TM, char* opts, char**
errs) {
  output= (char*) malloc (50);
  strcpy (output, "\2verbatim:Started dynamic link\5");
  return output;
}
```

the evaluation routine

```
char*
dynlink_eval (char* what, char* session, char** errors) {
  free (output);
  output= (char*) malloc (50 + strlen (what));
  strcpy (output, "\2verbatim:You typed ");
  strcat (output, what);
  strcat (output, "\5");
  return output;
}
```

and the data structure with the public exports:

```
package_exports_1 dynlink_exports= {
  "TeXmacs communication protocol 1",
  "Dynlink 1",
  dynlink_install,
  dynlink_eval
};
```

Notice that the application takes care of the memory allocation and deallocation of `output`.

## D.11. Miscellaneous features

Several other features are supported in order to write interfaces between TeXmacs and extern applications. Some of these are very hairy or quite specific. Let us briefly describe a few miscellaneous features:

**Interrupts.**

The "stop" icon can be used in order to interrupt the evaluation of some input. When pressing this button, TeXmacs will just send a `SIGINT` signal to your application. It expects your application to finish the output as usual. In particular, you should close all open `DATA_BEGIN`-blocks.

**Testing whether the input is complete.**

Some systems start a multiline input mode as soon as you start to define a function or when you enter an opening bracket without a matching closing bracket. TeXmacs allows your application to implement a special predicate for testing whether the input is complete. First of all, this requires you to specify the configuration option

```
(:test-input-done #t)
```

As soon as you will press `return` in your input, TeXmacs will then send the command

```
DATA_COMMAND(input-done? input-string)return
```

Your application should reply with a message of the form

```
DATA_BEGINscheme:doneDATA_END
```

where ***done*** is either `#t` or `#f`. The `multiline` plug-in provides an example of this mechanism (see in particular the file `multiline/src/multiline.cpp`).

## D.12. Plans for the future

There are many improvements to be made in the TeXmacs interface to computer algebra systems. First of all, the computer algebra sessions have to be improved (better hyphenation, folding, more dynamic subexpressions, etc.).

As to interfaces with computer algebra systems, out main plans consist of providing tools for semantically safe communication between several system. This probably will be implemented in the form of a set of plug-ins which will provide conversion services.

# Index