# ConTEXt

## reference manual

Hans Hagen, Taco Hoekwater

September 27, 2013

This document is typeset using LuaTEX.

svnversion: 329

# Content

# Preface

This manual is about ConTEXt, a system for typesetting documents. Central element in this name is the word TEX because the typographical programming language TEX is the base for ConTEXt.

People who are used to TEX will probably identify this manual as a TEX document. They recognise the use of \. One may also notice that the way pararaphs are broken into lines is often better than in the avarage typesetting system.

In this manual we will not discuss TEX in depth because highly recommended books on TEX already exist. We would like to mention:

1. the unsurpassed *The TEXBook* by Donald E. Knuth, the source of all knowledge and TEXnical inspiration,
2. the convenient *TEX by Topic* by Victor Eijkhout, the reference manual for TEX programmers, and
3. the recommended *The Beginners Book of TEX* by Silvio Levy and Raymond Seroul, the book that turns every beginner into an expert

For newcomers we advise (**3**), for the curious (**1**), and for the impatient (**2**). ConTEXt users will not necessarly need this literature, unless one wants to program in TEX, uses special characters, or has to typeset math. Again, we would advise (**3**).

You may ask yourself if TEX is not just one of the many typesetting systems to produce documents. That is not so. While many systems in eighties and nineties pretended to deliver perfect typographical output, TEX still does a pretty good job compared to others.

TEX is not easy to work with, but when one gets accustomed to it, we hope you will appreciate its features,

Hans Hagen, 1996–2002

# 1 Introduction

## 1.1 TEX

TEX was developed at Stanford University during the seventies. The designer, developer and spiritual father of TEX is Donald E. Knuth. Knuth developed TEX to typeset his own publications and to give an example of a systematically developed and annotated program.

The TEX project was supported by the American Mathematical Society and resulted in the programming language and program TEX, the programming language and program MetaFont, the Computer Modern typefaces and a number of tools and publications.

TEX is used worldwide, supports many languages, runs on almost every platform and is stable since 1982, which is rather unique in today's information technology.

TEX is a batch–oriented typesetting system. This means that the complete text is processed from beginning to end during which typesetting commands are interpreted. Because you tell your typesetting intentions to TEX, the system can also be qualified as an intentional typesetting system. In most documents one can stick to commands that define the structure and leave the typographic details to TEX. One can concentrate on the content, instead of on makeup; the author can concentrate on his reader and his intentions with the text. We prefer such an intentional system over a page–oriented system, especially in situations where you have to process bulky documents with regularly changing content. Furthermore an intentional typesetting system is rather flexible and makes it possible to change layout properties depending on its application. Thus, the same text source can be used to produce various results, for example, both on–line and printed output. It can also cooperate quite well with other text–processing programs and tools.

Developed in the early 1980s, LaTEX was intended to provide a high-level language that accesses the power of TEX. LaTEX essentially comprises a collection of TEX macros and a program to process LaTEX documents. Because the TEX formatting commands are very low-level, it is usually much simpler for end-users to use LaTEX. LaTEX is based on the idea that it is better to leave document design to document designers, and to let authors get on with writing documents.

ConTEXt is a document markup language and document preparation system also based on the TEX typesetting system. It was designed with the same general-purpose aims as LaTEX of providing an easy to use interface to the high quality typesetting engine provided by TEX. However, while LaTEX insulates the writer from typographical details, ConTEXt takes a complementary approach by providing structured interfaces for handling typography, including extensive support for colors, backgrounds, hyperlinks, presentations, figure-text integration, and conditional compilation. It gives the user extensive control over formatting while making it easy to create new layouts and styles without learning the TEX macro language. ConTEXt's unified design avoids the package clashes that can happen with LaTEX. ConTEXt also attempts to more closely respect the syntactical style of plain TEX.

## 1.2 ConTEXt

The development of ConTEXt started in 1990. A number of TEX based macro packages had been used to our satisfaction. However, the non–technical users at our company were not accustomed to rather complex and, in particular, non–Dutch interfaces. For this reason we initiated the development of ConTEXt with a parameter driven interface and commands that are easy to understand. Initially the user interface was only available in Dutch.

The functionality of ConTEXt was developed during the production of many complex educational materials, workplace manuals and handbooks. In 1994 the package was stable enough to warrant a Dutch user manual. Over the years many new features and a multi-lingual interface have been added (currently English, German, French, Italian, Romanian? . . . interfaces are supported). Though ConTEXt has matured and is as (un)stable as any other macro package, there are still a great number of wishes and development remains active. These will be implemented in the spirit of the existing ConTEXt commands.

**TODO:** Add some text about recent developments, especially the split between mkii and mkiv

## 1.3 Commands

A ConTEXt document is normally coded in utf or another plain text encoding like ISO Latin1. Any preferred text editor may be used. Inside such a file, the actual document text is interspersed with ConTEXt commands. These commands tell the system how the text should be typeset. A ConTEXt command begins with a backslash (\). An example of a command is `\italic`. Most of the time a command does something with the text that comes after the command. The text after the command `\italic` will be typeset *text in italic*.

When you use a command like `\italic` you are acting as a typesetter, and when you are writing paragraphs you are acting as an author. Typesetting and writing are conflicting activities; as an author you would probably rather spend as little time as possible typesetting. When you are actually writing text and you have to indicate that something special has to happen with the text, it is therefore best to use generic commands than specific typesetting commands. An example of such a generic command is `\em` (*emphasis*). By using `\em` instead of `\italic`, you enable the typesetter (who could also be you) to change the typeset result without him or her having to alter the text.

A TEX user normally speaks of macros instead of commands. A macro is a (normally small) program. Although this manual uses both 'command' and 'macro', we will try consistently use the word command for users and macro for programmers. A collection of macros is called a macro package.

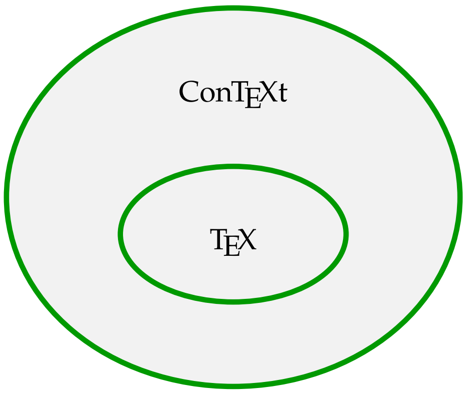


**Figure 1.1**

A command is often followed by setups and/or argument text. Setups are placed between brackets (`[]`), are optional and there may be more than one set. The scope or range of the command (the text acted upon) is placed between curly brackets (`{}`); there may be more than one of those as well. (Note that the word 'argument' in this manual is sometimes used both for setups and for the text acted upon.)

An example of a command with setups and an argument text is

```
\framed[width=3cm,height=1cm]{that's it}
```

When this input is processed by ConTEXt, the result will look like this:

that's it

Alternatively, using the default setups:

```
\framed{that's it}
```

that's it

Setups in ConTEXt come in two possible formats. First, there can be a list of comma-separated key–value pairs like we saw already

```
\setupsomething [variable=value, variable=value, ...]
```

Second, there can be a comma-separated list of just values

```
\setupsomething [option, option,...]
```

In both cases the setups are placed between `[]`. Spaces, tabs and even a newline between the command and the opening `[` or after any of the separation commas are ignored. But multiple newlines are disallowed, and whitespace before commas, around the equals sign and before the closing `]` is significant.

Some practical examples of correct command invocations are:

```
\setupwhitespace [big]
\setupitemize   [packed, columns]
\setuplayout    [backspace=4cm,
                 topspace=2.5cm]
```

Many typographical operations are performed on a text that is enclosed within a `start-stop` construction:

```
\startsomething
```

```
............................
\stopsomething
```

where `something` indicates a predefined keyword such as `narrower`. Often, keywords or key–value pairs can be passed, that inform ConTEXt of the user's wishes, like

```
\startnarrower[2*left,right]
............................
\stopnarrower
```

or

```
\startitemize[n,broad,packed]
\item ......................
\item ......................
\stopitemize
```

The simplest ConTEXt document is

```
\starttext
Hello World!
\stoptext
```

## 1.4 Running ConTEXt

For basic usage, running

```
context myfile
```

or

```
context myfile.tex
```

is sufficient to convert your ConTEXt source in `myfile.tex` to the pdf file `myfile.pdf`. Several command line options control the ConTEXt run. For a complete list of options, you can run

```
context --help
```

Here some examples:

| **option** | **meaning** |
|---|---|
| `--result=name` | rename the resulting output to the given name |
| `--mode=list` | enable given the modes (conditional processing in styles |
| `--once` | only run once (no multipass data file is produced) |
| `--nonstopmode` | run without stopping |
| `--version` | report installed context version |

**TODO:** Maybe some more explainations about `texexec` and `context` here, maybe from a text editor or environment like tex-works.

## 1.5 Advanced commands

There are also commands that are used to define new commands. For example:

```
\definesomething[name]
```

defines a `something` command called `\name`. For example,

```
\definehead[section]
```

defines a `head` command called `\section`. The name can be arbitrarily chosen by the user; but the type of command must be a predefined one.

Sometimes a definition inherits its characteristics from another (existing) one. In those situations a definition looks like:

```
\definesomething[clone][original]
```

In many cases one can also pass settings to these commands. In that case a definition looks like:

```
\definesomething[name][variable=value,...]
```

These setups can also be defined in a later stage with:

```
\setupsomething[name][variable=value,...]
```

An example of such a name coupled definition and setup is:

```
\definehead[section][chapter]
\setuphead[section][textstyle=bold]
```

This defines a `head` command called `\section`, which inherits characteristics from the existing `head` command called `\chapter`. It then defines setups for the new `\section` command.

The alternatives shown above are the most common appearances of the commands. But there are exceptions:

```
\defineenumeration[Question][location=inmargin]
\useexternalfigure[Logo][FIG-0001][width=4cm]
\definehead[Procedure][section]
\setuphead[Procedure][textstyle=slanted]
```

After the first command the newly defined command `\Question` is available which we can use for numbered questions and to place numbers in the margin. With the second command we define a picture named `Logo`, to be called later, that is scaled to a `width` of 4cm. After the third command a new command `\Procedure` is available that inherits its characteristics from the predefined command `\section`. The last command alters the characteristics of the newly defined head. Later we will discuss these commands in more detail.

We use `begin-end` constructions to mark textblocks. Marked textblocks can be typeset, hidden, replaced or called up at other locations in the document.

```
\beginsomething
.............................
\endsomething
```

These commands enable the author to type questions and answers in one location and place them at another location in the document. Answers could be placed at the end of a chapter with:

```
\defineblock[Answer]
\setupblock[Answer][bodyfont=small]
\hideblocks[Answer]
.............................
\chapter{........}
.............................
\beginAnswer
.............................
\endAnswer
.............................
```

In this case answers will be typeset in a smaller bodyfont size, but only when asked for. They are hidden by default, but stored in such a way, that they can later be typeset.

Commands come in many formats. Take for example:

```
\placefigure
  [left]
  [fig:logo]
  {This is an example of a logo.}
  {\externalfigure[Logo]}
```

This command places a picture at the left hand side of a text while the text flows around the picture. The picture has a reference `fig:logo`, i.e. a logical name. The third argument contains the title and the fourth calls the picture. In this case the picture is a figure defined earlier as `Logo`. **Figure 1.1** is typeset this way.

The last example has optional arguments between optional (`[]`). Many commands have optional arguments. In case these optional arguments are left out the default values become operative.

You may have noticed that you are allowed to have plenty of whitespace in your ascii text (within certain limits). In our opinion, this increases readability considerably, but you may of course decide to format your document otherwise. When the ConTEXt commands in this manual are discussed they are displayed in the following way:

```
\setupfootertexts [.1.] [.2.] [.3.]
                  OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

The command `\setupfootertexts`, which we will discuss in detail in a later chapter, has three (or more) setup arguments, of which the first is optional. Optional arguments are displayed with the word OPTIONAL below the brackets. Multiple alternative values are listed for each argument; the first argument can either `[text]`, `[margin]`, or `[edge]`. It defaults to `[text]`. Default values are underlined and placeholders (as opposed to literal keywords) are typeset in UPPERCASE. In this example, TEXT means that you can provide any footer text. ConTEXt is able to keep track of the status of information on the page, for instance the name of the current chapter. We call this kind of information MARK, so the command `\setupfootertexts` accepts references to marks, like those belonging to sectioning commands: `chapter`, `section`, etc. The argument `date` results in the current system date.

When the setup of some commands are displayed you will notice a ◀▶ in the right hand top corner of the frame. This indicates that this command has a special meaning in interactive or screen documents. Commands for the interactive mode only show solid arrows, commands with an additional functionality show gray arrows.

## 1.6 Programs

TEX does a lot of text manipulations during document processing. However, some manipulations are carried out by TEXutil. This program helps TEX to produce registers, lists, tables of contents, tables of formulas, pictures etc. This program is implemented as a Perl script.

Document processing can best be done with TEXexec. This Perl script enables the user to use different processing modes and to produce different output formats. It also keeps track of changes and processes the files as many times as needed to get the references and lists right.

## 1.7 Files

**TODO:** Where is a better place to write about TABLE? The old paragraph:
ConTEXt relies on plain TEX. Plain TEX, ConTEXt and a third package TABLE are brought together in a so called format file. TABLE is a powerful package for typesetting tables. A format file can be recognized by its suffix `fmt`. TEX can load format files rather fast and efficiently.

ConTEXt is used with utf-8 source files. utf-8 is a character encoding, that can represent every character in the Unicode character set, and that is backward-compatible with ascii. The utf-8 file with the prescribed extension `tex` is processed by ConTEXt. During this process ConTEXt produces a pdf (Portable Document Format) output file. pdf files are of high graphical quality and are also interactive (hyperlinked).

With the command

```
\enableregime[encoding]
```

ConTEXt supports also other character encodings such as ISO Latin1.

It is highly recommended, that all input files, i.e. the ConTEXt source and other included files such as image files, have only the letters a–z, digits and dashes in their names, that is in the names of their full paths, otherwise you can easily get into problems. Especially the characters *, ?, the space character and some more are known to cause problems, sometimes because of ConTEXt itself (internal pattern matching), sometimes because of issues with the underlying operating system. The dot '.' is somewhat special: it's used to separate the suffix from the rest of the file-name, but at other places in the path it can cause trouble just like '*' or '?'.

## 1.8 Texts

### 1.8.1 Characters

A traditional TEX source file contains only ascii characters. Higher ascii values to produce characters like ë, ô and ñ can also be used in this version of TEX (in preference to the traditional mechanism of escaped sequences such as `\"e`, `\^o`, etc.). However, some characters in TEX have special meanings (such as %, $, and of course {}). Each of these characters can be typeset by putting a \ in front of it. A % is obtained by typing `\%` (if one would type only a `%` the result would be undesirable because TEX interprets text after a % as comment that should not be processed), a $ is produced by \$ (a $ without a \ indicates the beginning of the mathematical mode), a # is displayed by \# (in TEX, # stands for arguments in macro definitions), and a pair of { and } without leading backslashes define the opening and closing of a group. Finally the backslash \ itself is entered by typing `\backslash`.

**TODO:** What about other 'reserved' symbols? Need to say something about `\enableregime[utf]` in mkii.

### 1.8.2 Paragraphs

TEX performs its operations mostly upon the text element *paragraph*. A paragraph is ended by `\par` or, preferably, by an empty line. Empty lines in an ascii text are preferred because of readability of the source.

### 1.8.3 Boxes

In this manual we will sometimes talk about boxes. Boxes are the building blocks of TEX. TEX builds a page in horizontal and vertical boxes. Every character is a box, a word is also a box built out of a number of boxes, a line is . . .

When TEX is processing a document many messages may occur on the screen. Some of these messages are warnings related to overfull or underful boxes. Horizontal and vertical boxes can be typeset by the TEX commands `\hbox` and `\vbox`. Displacements can be achieved by using `\hskip` and `\vskip`. It does not hurt to know a bit about the basics of TEX, because that way one can far more easily write his or her own alternatives to, for instance, chapter headers.

### 1.8.4 Fonts

TEX is one of the few typesetting systems that does mathematical typesetting right. To do so TEX needs a complete font–family. This means not only the characters and numbers but also full mathematical symbols. Complete fontfamilies are Computer Modern Roman and Lucida Bright. Both come in serif and sans serif characters and a monospaced character is also available. Other fontfamilies are available, more or less complete.

### 1.8.5 Dimensions

Characters have dimensions. Spacing between words and lines have dimensions. These dimensions are related to one of the units of **table 1.1**. For example the linespacing in this document is 13.8292pt.

| dimension | meaning | equivalent |
|---|---|---|
| `pt` | point | $72.27pt = 1in$ |
| `bp` | big point | $72bp = 1in$ |
| `pc` | pica | $1pc = 12pt$ |
| `in` | inch | $1in = 2.54cm$ |
| `cm` | centimeter | $2.54cm = 1in$ |
| `mm` | millimeter | $10mm = 1cm$ |
| `dd` | didot point | $1157dd = 1238pt$ |
| `cc` | cicero | $1cc = 12dd$ |
| `sp` | scaled point | $65536sp = 1pt$ |

**Table 1.1** Dimensions in TEX.

We will often specify layout dimensions in points or centimeters or milimeters. A point is about 0.3515 *mm* is an American publishing standard. The European Didot point, equivalent to $0.254 \cdot 1238/1157/72.27 = 0.376065\ mm$, is about 7% larger.

Next to the mentioned dimension TEX also uses `em` and `ex`. Both are font dependant. An `ex` has the height of an x, and an `em` the width of an M. In the Computer Modern Roman typefaces, numbers have a width of 1/2em, while a — (`---`) is one em.

### 1.8.6 Error messages

While processing a document, TEX generates status messages (what TEX is doing), warning messages (what TEX could do better) and error messages (what TEX considers wrong). An error message is always followed by a `halt` and processing will be stopped. A linenumber and a `?` will appear on screen. At the commandline you can type `H` for help and the available commands will be displayed.

Some fatal errors will lead to an `*` on the screen. TEX is expecting a filename and you have to quit processing. You can type `stop` or `exit` and if that doesn't work you can always try `ctrl-z` or `ctrl-c`.

## 1.9 Version numbers

TEX was frozen in 1982. This meant that no functionality would be added from that time on. However, exceptions were made for the processing of multi–language documents, the use of 8-bits ascii–values and composed characters. Additionally some bugs were corrected. At this moment TEX version 3.141592 is being used. The ultimate TEX version number will be $\pi$, while MetaFont will become the Euler number $e$.

ConTEXt can handle both $\varepsilon$-TEX and pdfTEX, which are extensions to TEX. Both are still under development so we suggest using the latest versions available. This manual was typeset using pdfeTEX, with $\varepsilon$-TEX version 2.2 and pdfTEX version 2000.

ConTEXt is still under development. Macros are continually improved in terms of functionality and processing speed. Improvements are also made within existing macros. For example the possibility to produce highly interactive pdf documents has altered some low–level functionality of ConTEXt but did not alter the interface. We hope that in due time ConTEXt will be a reasonable complete document processing system, and we hope that this manual shows much of its possibilities. This document was processed with ConTEXt version 2013.09.21 13:53.

## 1.10 Top ten

A novice user might be shooed away by the number of ConTEXt commands. Satisfying results can be obtained by only using the next ten groups of commands:

1. `\starttext`, `\stoptext`
2. `\chapter`, `\section`, `\title`, `\subject`, `\setuphead`, `\completecontent`
3. `\em`, `\bf`, `\cap`
4. `\startitemize`, `\stopitemize`, `\item`, `\head`
5. `\abbreviation`, `\infull`, `\completelistofabbreviations`
6. `\placefigure`, `\externalfigure`, `\useexternalfigures`
7. `\placetable`, `\starttable`, `\stoptable`
8. `\definedescription`, `\defineenumeration`
9. `\index`, `\completeindex`
10. `\setuplayout`, `\setupfootertexts`, `\setupheadertexts`

## 1.11 Warning

ConTEXt users have the possibility to define their own commands. These newly defined commands may come into conflict with plain TEX or ConTEXt commands. To avoid this, it is advisable to use capitalized characters in your own command definitions, such as:

```
\def\MyChapter#1%
  {\chapter{#1}\index{#1}}
```

This command starts a new chapter and defines an index entry with the same name.

# 2 Documents

## 2.1 Introduction

Why should one use TEX in the first place? Many people start using TEX because they want to typeset math. Others are charmed by the possibility of separating content and make–up. Yet another kind of user longs for a programmable system. And let us not forget those users that go for quality.

When using TEX one does not easily run into capacity problems. Typesetting large documents with hundreds of pages is typically a job for TEX. If possible, when coding a document one should look beyond the current document. These days we see documents that were originally typeset for paper being published in electronic format. And how about making a stripped version of a 700 page document? A strict separation between content and layout (make–up) on the one hand and an acceptable redundancy in structure on the other is often enough to guarantee multiple use of one document source.

A system like ConTEXt is meant to make life easier. When coding a document the feeling can surface that "this or that should be easier". This feeling often reflects the truth and the answer to the question can often be found in this manual, although sometimes obscured. It takes some time to learn to think in structure and content, certainly when one is accustomed to mouse driven word processors. In this chapter we focus on the structure of collections of documents.

## 2.2 Start and stop

In a self contained text we use the following commands to mark the begin and end of a text:

```
\starttext
\stoptext
```

The first command takes care of a number of initializations and the last command tells TEX that processing can stop. When this command is left out TEX will display a * (a star) on the command line at the end of the job. TEX will expect a command, for example `\end`.

It is advisable to type the document setups before the `\start`–command, the so called setup area of the document. In this way a clever word–processor can identify where the text starts, and therefore can include those setups when it partially processes the document, given of course that it supports partial processing of files.

In the example below a very simple layout is being used.

```
\starttext

\subject{Introduction}

\unknown\ America has always been a land set firmly not in the past, but
in the future. On a recent visit to England, I found dozens of wonderful
bookstores chock full of the past --- ancient history, rooms full of it,
and great literature in such monumental stacks as to be overwhelming. In
the usual American bookstore, history might occupy a few bookcases; great
literature has its honoured place, but this year's paperbacks dominate. The
```

```
past is not disregarded, but neither does it loom so large and run so deep
in our blood.

\blank

{\bf Greg Bear, introduction to Tangents (1989).}

\stoptext
```

The commands `\starttext...\stoptext` may be nested. Within a text a new text containing `\starttext` and `\stoptext` may be loaded.

## 2.3 Structure

In this section a structured approach of managing your documents is discussed. For very simple and self containing documents you can use the following approach:

```
\environment this
\environment that

\starttext
... some interesting text ...
\stoptext
```

When you have to typeset very bulky documents it is better to divide your document in logical components. ConTEXt allows you to setup a project structure to manage your texts. You have to know that:

- A group of texts that belong together have to be maintained as a whole. We call this a *project*.
- Layout characteristics and macros have to be defined at the highest level. For this, the term *environment* has been reserved.
- Texts that belong together in a project we call *products*.
- A product can be divided into components, these components can be shared with other products. Components can be processed individually.

Programmable word processors can be adapted to this structure.

A *project*, *environment* , *product* or *component* is started and stopped with one of the following commands:

```
\startproject .*. ... \stopproject

*  FILE NAME
```

```
\startproduct .*. ... \stopproduct

*  FILE NAME
```

```
\startenvironment .*. ... \stopenvironment

*  FILE NAME
```

```
\startcomponent  .*.  ... \stopcomponent

*   FILE NAME
```

Before a \start–\stop–pair commands can be added. When a file is not found on the directory ConTEXt looks for the files on higher level directories. This enables the user to use one or more environments for documents that are placed on several subdirectories.

| command | project | environment | product | component |
|---|---|---|---|---|
| \project <<name>> | | | (⋆) | (⋆) |
| \environment <<name>> | (⋆) | (⋆) | (⋆) | (⋆) |
| \product <<name>> | ⋆ | | | (⋆) |
| \component <<name>> | | | (⋆) | (⋆) |

**Table 2.1** The structure commands that can be used in the files that make up a project.

To treat products and components as individual documents, the commands in **table 2.1** are used. The commands marked with ⋆ are obligatory and the commands marked with (⋆) are optional. The content is typed before the \stop command.

```
\startproject documents

\environment layout

\product  teacher
\product  pupil
\product  curriculum

\stopproject
```

**Figure 2.1**

An example of a project file.

```
\startproduct teacher

\project   documents

\component teacher1
\component teacher2

\stopproduct
```

**Figure 2.2**

The product teacher.tex (a teacher manual) can be defined as shown on the opposite site.

```
\startcomponent teacher2

\project documents
\product teacher

... text ...

\stopcomponent
```

**Figure 2.3**

Here we see the component.

In most cases working with only \starttext and \stoptext in combination with \input or \enviroment is sufficient. A project structure has advantages when you have to manage a great number of texts. Although it is more obvious to process *products* as a whole, it also enables you to process *components* independently, given that the stucture is defined properly.

A project file contains only a list of products and environments, it cannot be typeset. If you have only one product, you don't really need a project file. This manual for example has a product/component structure without a project file. Every chapter is a component and the product file loads some environments.

Schematically the coherence between files could be displayed as illustrated in **figures 2.4, 2.5 and 2.6**.

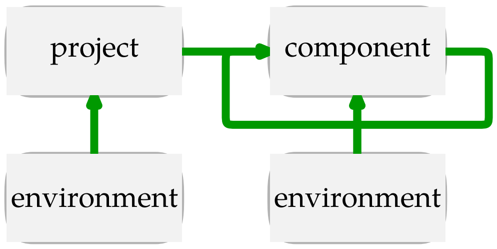


**Figure 2.4** An example of project structure.

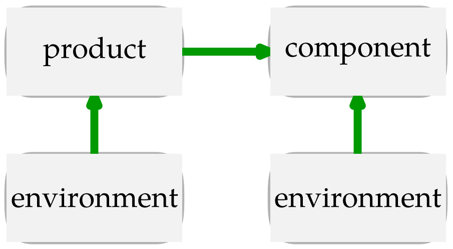


**Figure 2.5** An example with only products.

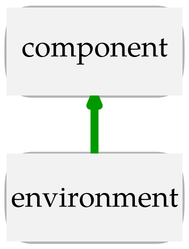


**Figure 2.6** An example with only one component.

It is good practice to put all setups in one environment. In case a component or product has a different layout you could define *localenvironments*:

```
\startlocalenvironment[<<names>>]
... setups ...
\stoplocalenvironment
```

A local environment can be typed in an environment file or is a separate file itself. When a separate file is used the local environment is loaded with:

```
\localenvironment <<name>>
```

Below you will find an example of a project structure.

```
\startproject demos

\environment environ
\product     example

\stopproject
```

**Figure 2.7**

file: `demos.tex`

This file is used to define the products and environments.

```
\startenvironment environ

\setupwhitespace[big]

\setupfootertexts[part][chapter]

\stopenvironment
```

**Figure 2.8**

file: `environ.tex`

In the environment we type the setups that relate to all the different products. More than one environment or local environments per product can be used.

```
\startproduct example

\project demos

\startfrontmatter
  \completecontent
\stopfrontmatter

\startbodymatter
  \component first
  \component second
\stopbodymatter

\startbackmatter
  \completeindex
\stopbackmatter

\stopproduct
```

**Figure 2.9**

file: `example.tex`

The product file contains the structure of the product. Because indexes and registers can be evoked quite easily we do not use a separate file.

```
\startcomponent first

\environment environ
\product example

\part{One}

\completecontent

\chapter{First}

..... text .....

\chapter{Second}

..... text .....

\completeindex

\stopcomponent
```

**Figure 2.10**

file: `first.tex`

In the components of a product we place the textual content, figures etc. It is also possible to request the tables of content and registers per product.

```
\startcomponent second

\environment environ
\product example

\part{Two}

\completecontent

\chapter{Alfa}

..... text .....

\chapter{Beta}

..... text .....

\completeindex

\stopcomponent
```

file: `second.tex`

The product contains more than one component. We could have defined a product for each part and a component for each chapter.

**Figure 2.11**

The files `first.tex`, `second.tex` and `example.tex` can be processed separately. If you process an environment there will be no pages of output.

## 2.4 Directories

Many TEX implementations look for a file in all directories and subdirectories when a requested file is not in the current directory. This is not only time–consuming but may lead to errors when the wrong file (a file with the same name) is loaded.

For this reason ConTEXt works somewhat differently. A file that is not available on the working directory is searched for on the parent directories. This means that environments can be placed in directories that are parents to the products that use them. For example:

```
/texfiles/course/layout.tex
/texfiles/course/teacher/manual.tex
/texfiles/course/student/learnmat.tex
/texfiles/course/otherdoc/sheets.tex
```

The last three files (in different subdirectories) all use the same environment `layout.tex`. So, instead of putting all files into one directory, one can organize them in subdirectories. When a project is properly set up, that is, as long as the project file and specific environments can be found, one can process components and products independently.

## 2.5 Versions

During the process of document production it is useful to generate a provisional version. This version shows the references and the typesetting failures. The provisional version is produced when you type:

```
\version [.*.]

*   final concept temporary
```

By default the definitive version is produced. In case a preliminary version is produced the word *concept* is placed at the bottom of each page. The keyword `temporary` shows some information on for instance overfull lines, references, figure placement, and index entries. Most messages are placed in the margin. In some cases these messages refer to the next pages because TEX is processing in advance.

## 2.6 Modes

TEX can directly produce dvi or pdf. A document can be designed for paper and screen, where the last category often has additional functionality. From one document we can generate different alternatives, both in size and in design. So, from one source several alternatives can be generated.

Processing a file in practice comes down to launching TEX with the name of the file to be processed. Imagine that by default we generate dvi output. Switching to pdf is possible by enabling another output format in the file itself or a configuration file, but both are far from comfortable.

```
\setupoutput[pdftex]
```

for direct pdf output, or for pdf produced from PostScript:

```
\setupoutput[dvips,acrobat]
```

The key to the solution of this problem is TEXexec. This Perl script provides ConTEXt with a command–line–interface. When we want pdf instead of dvi, we can launch TEXexec with:

```
texexec --pdf filename
```

There are more options, like making A5–booklets; more on these features can be found in the manual that comes with TEXexec. However, one option deserves more time: modes.

```
texexec --pdf --mode=screen filename
```

The idea behind modes is that within a style definition, at each moment one can ask for in what mode the document is processed. An example of a mode dependant definition is:

```
\startmode[screen]
  \setupinteraction[state=start]
  \setupcolors[state=start]
\stopmode
```

if needed, accompanied by:

```
\startnotmode[screen]
  \setupcolors[state=start,conversion=always]
\stopnotmode
```

One can also pass more than one mode, separated by comma's. There are also some low level mode dependant commands. Given that we are dealing with a screen mode, we can say:

```
\doifmodeelse {screen} {do this} {and not that}
\doifmode     {screen} {do something}
\doifnotmode  {screen} {do something else}
```

A mode can be activated by saying:

```
\enablemode[screen]
\disablemode[screen]
```

Again, we can pass more modes:

```
\enablemode[paper,A4]
```

One strength of TEXexec is that one is not forced to enable modes in a file: one can simply pass a command line switch. Just as with choosing the output format: the less we spoil the document source with output and mode settings, the more flexible we are.

To enable users to develop a style that adapts itself to certain circumstances, ConTEXt provides system modes. For the moment there are:

| | |
|---|---|
| `*list` | the list one called for is placed indeed |
| `*register` | the register one called for is placed indeed |
| `*interaction` | interaction (hyperlinks etc) are turned on |
| `*sectionblock` | the named sectionblock is entered |

System modes are prefixed by a `*`, so they will not conflict with user modes. An example of a sectionblock mode is `*frontmatter`. One can use these modes like:

```
\startmode[*interaction]
  \setuppapersize[S6][S6]
\stopmode
```

## 2.7 Modes Manual

**TODO:** Merge with previous section

Every user will at one moment run into modes. Modes are used for conditional processing. You enable or disable modes:

```
\enablemode[screen]
\disablemode[proof]
```

as well as prevent modes being set:

```
\preventmode[doublesided]
```

Later on you can act upon this mode using:

```
\startmode[screen]
  \setupinteraction[state=start]
\stopmode
```

The counterpart of this command is:

```
\startnotmode[screen]
  \setupinteraction[state=start]
\stopnotmode
```

You can set modes in your document or in styles, but you can also do that at runtime:

```
texexec --pdf --mode=screen --result=myfile-s myfile
texexec --pdf --mode=A4     --result=myfile-a myfile
texexec --pdf --mode=letter --result=myfile-l myfile
```

You can test for more modes at the same time:

```
\startmode[color,colour]
  \setupcolors[state=start]
\stopmode
```

If you want to satisfy a combination of modes, you use:

```
\startmode[final]
  \setuplayout[markings=on]
\stopmode
\startallmodes[final,color]
  \setuplayout[markings=color]
\stopallmodes
```

The counterpart is

```
\startnotallmodes[print,proof]
  \setuplayout[markings=off]
\stopnotallmodes
```

Instead of the `start–stop` variants, you can use the `\doif` alternatives. These have the advantage that they can be nested.

```
\doifmodeelse     {modes} {action} {alternative}
\doifmode         {modes} {action}
\doifnotmode      {modes} {action}
\doifallmodeselse {modes} {action} {alternative}
\doifallmodes     {modes} {action}
\doifnotallmodes  {modes} {action}
```

Mode can be combined with variables:

```
\setupvariables[document][alternative=print]

\enablemode[document:\getvariable{document}{alternative}]

\startmode[document:print]
  ...
\stopmode

\startmode[document:screen]
  ...
\stopmode
```

An alternative for such an selective approach is to use setups:

```
\setupvariables[document][alternative=print]

\startsetups[document:print]
  ...
\stopsetups

\startsetups[document:screen]
  ...
\stopsetups
```

```
\setups[document:\getvariable{document}{alternative}]
```

The difference is that mode blocks are processed in the order that the document (or style) is loaded, while setups are stored and recalled later.

In addition to your own modes, ConTEXt provides a couple of system modes. These are preceded by a `*`, as in:

```
\startmode[*first]
  % this is the first run
\stopmode
```

The following system modes are available (more will implemented):

`color-c,color-m,color-y,color-k` These are rather special modes related to color separation. They are only set when channels are split off.

`figure` This mode is set when a graphic is found. You can use this mode in for instance figure postprocessing actions.

`text, project, product, component, environment` These modes are set when one enters one of the associated structuring environments. Nesting is supported.

`list` After using `\determinelistcharacteristics` this mode reflects if list entries were found.

`pairedbox` This mode is enabled when a paired box (legenda and such) is constructed.

`combination` This mode is enabled when a combination (often used for graphics) is constructed.

`interaction` When interaction is enabled, this mode is true. You can for instance use this mode to add different content to for instance screen and paper versions.

`register` After using `\determineregistercharacteristics` this mode reflects if register entries were found.

`sectionnumber` This mode is enabled when a section head is numbered. You can access the mode while building the section head, which is true when you have your own commands hooked into the head mechanism.

`frontpart, bodypart, backpart, appendix` The state of main sections in a document as well as user defined ones, are reflected in system modes.

`suffix-\jobfilesuffix` You can use this mode to differentiate between input file types. We use this for instance to distinguish between different XML content variants when pretty-printing (given that they can be recognized on their suffix).

`first` Often multiple runs are needed to get a document right. Think of cross references, object references, tables of contents, indices, etc. You can use this mode to determine if the first run is taking place. For instance, when you do real time graphic conversions, it makes sense to do that only once.

| | |
|---|---|
| `last` | This mode is set if the last run in a session is taking place. Normally this is not known in advance, unless one has asked for an additional imposition pass. |
| `background` | This mode is set when there is a (new) background defined. |
| `postponing` | While postponing some content using the postpone mechanism this mode is enabled. |
| `grid` | When you are typesetting on a grid, special care has to be taken not spoil grid snapping. You can use this mode to test if you are in grid typesetting mode. |
| `header` | This mode is enabled when there is a page header, i.e. the header has non-zero dimensions. |
| `footer` | This mode is enabled when there is a page footer, i.e. the header has non-zero dimensions. |
| `makeup` | The makeup mechanisms are used to build single pages like title pages. This mode is set during construction. |
| `pdf, dvi` | One of these modes is set, which one depends on the output driver that is loaded. |
| `*language-id, language-id` | When a language is chosen, its id is set as mode. For example, when the main language is English, and the current language Dutch, we can test for the modes `**en` and `*nl` (watch the extra `*`). |
| `marking` | This flag is set when a marking (e.g. in a header or footer) is being typeset (processed). |

## 2.8 Regimes

When you key in an english document, a normal QWERTY keyboard combined with the standard ascii character set will do. However, in many countries dedicated keyboards and corresponding input encodings are used. This means that certain keystrokes correspond to non–standard ascii characters and these need to be mapped onto the characters present in the font. Unless the input encoding matches the output (font) encoding, intermediate steps are needed to take care of the right mapping. For instance, input code 145 can become command `\eacute` which can result in character 123 of a certain font.

Although all kind of intermediate, direct or indirect, mappings are possible, in ConTEXt the preferred method is to go by named glyphs. The advantage of this method is that we can rather comfortably convert the input stream into different output streams as needed for typesetting text (the normal TEX process) and embedding information in the file (like annotations or font vectors needed for searching documents).

The conversion from input characters into named glyphs is handled by regimes. While further mapping is done automatically and is triggered by internal processes, regimes need to be chosen explicitly. This is because only the user knows what he has input.

Most encodings (like `il2`) have an associated regime. You can get some insight in what a regime involves by showing it:

```
\showregime[il2]
```

In addition there are a couple of platform dependent ones:

| `regime` | **platform** |
|---|---|
| `ibm` | the old standard msdos code page |
| `win` | the western europe MS Windows code page |

If you want to know what regimes are available, you can take a look at the `regi-*.tex` files. A regime that becomes more and more popular is the utf-8 regime. If you want some insight in what vectors provide, you can use commands like:

```
\showunicodevector[001]
```

and

```
\showunicodetable[001]
```

where the last one produces a rather large table.

# 3 Page design

## 3.1 Introduction

While processing a text TEX makes use of the actual `\hsize` (width) and `\vsize` (height). As soon as `\vsize` is exceeded TEX's output routine is launched. The output routine deals with the typeset part — most of the time this will be a page. It takes care of typesetting the headers and footers, the page number, the backgrounds and footnotes, tables and figures. This rather complex process makes it obvious that the output routine actually makes use of more dimensions than `\hsize` and `\vsize`.

## 3.2 Paper dimensions

With the command `\setuppapersize` the dimensions of the paper being used are defined. There is a difference between the dimensions for typesetting and printing.

```
\setuppapersize [...,...] [...,...]
                 1          2
                            OPTIONAL
1   A3 A4 A5 A6 letter ... CD IDENTIFIER landscape mirrored rotated 90 180 270
2   negative inherits from \setuppapersize
```

The dimensions of DIN formats are given in **table 3.1**.

| format | size in mm | format | size in mm |
|---|---|---|---|
| A0 | 841 × 1189 | A5 | 148 × 210 |
| A1 | 594 × 841 | A6 | 105 × 148 |
| A2 | 420 × 594 | A7 | 74 × 105 |
| A3 | 297 × 420 | A8 | 52 × 74 |
| A4 | 210 × 297 | A9 | 37 × 52 |

**Table 3.1** Default paper dimensions

There are a great number of standardized formats like `B0–B9` and `C0–C9`. These formats are predefined inConTEXt as well. You can also use: `letter`, `legal`, `folio` and `executive`, `envelope 9–14`, `monarch`, `check`, `DL` and `CD`. Another series of predefined formats comprise the `RA` and `SRA` types of paper sizes.

A new format can be defined by:

```
\definepapersize [.1.] [..,..=..,..]
                           2
1   IDENTIFIER
2   width   =  DIMENSION
    height  =  DIMENSION
    offset  =  DIMENSION
    scale   =  NUMBER
```

For example `CD` was defined as:

```
\definepapersize[CD][width=12cm,height=12cm]
```

After defining `CD` you can type:

```
\setuppapersize[CD][A4]
```

This means that for typesetting ConTEXt will use the newly defined size `CD`. The resulting, rather small page, is positioned on an `A4` paper size. This second argument is explained in detail later.

ConTEXt can also be used to produce screen documents. For that purpose a number of screen formats are available that relate to the screen dimensions. You can use: `S3`–`S6`. These generate screens with widths varying from 300 to 600 pt and a height of 3/4 of the width.

When one chooses another paper format than `A4`, the default settings are scaled to fit the new size.

All defined paper sizes can be used either in portrait or landscape orientation. You can tell ConTEXt the orientation of the paper in the `\setupapersize` command:

```
\setuppapersize[CD][A4,landscape]
```

## 3.3 Page texts

Page texts are texts that are placed in the headers, footers, margins and edges of the so called pagebody. This sentence is for instance typeset in the bodyfont in the running text. The fonts of the page texts are set up by means of different commands. The values of the parameters may be something like `style=bold` but `style=\ss\bf` is also allowed. Setups like `style=\ssbf` are less obvious because commands like `\cap` will not behave the way you expect.

Switching to a new font style (`\ss`) will cost some time. Usually this is no problem but in interactive documents where we may use interactive menus with dozens of items and related font switches the effect can be considerable. In that case a more efficient font switching is:

```
\setuplayout[style=\ss]
```

Border texts are setup by its command and the related key. For example footers may be set up with the key `letter`:

```
\setupfooter[style=bold]
```

## 3.4 Page composition

In page composition we distinguish the main text area, headers and footers, and the margins (top, bottom, right and left). The main text flows inside the main text area. When defining a layout, one should realize that the header, text and footer areas are treated as a whole. Their position on the page is determined by the topspace and backspace dimensions (see **picture 3.1**).

The header is located on top and the footer below of the main text area. Normally, in the header and footer page numbers and running titles are placed. The left and/or right margins are often used for structural components like marginal notes and/or chapter and section numbers. The margins are located in the backspace (along the spine) and in the white space to the right/left
**left** of the main text area. Their width has *no* influence on the location of the typesetting area on **right**
the page.

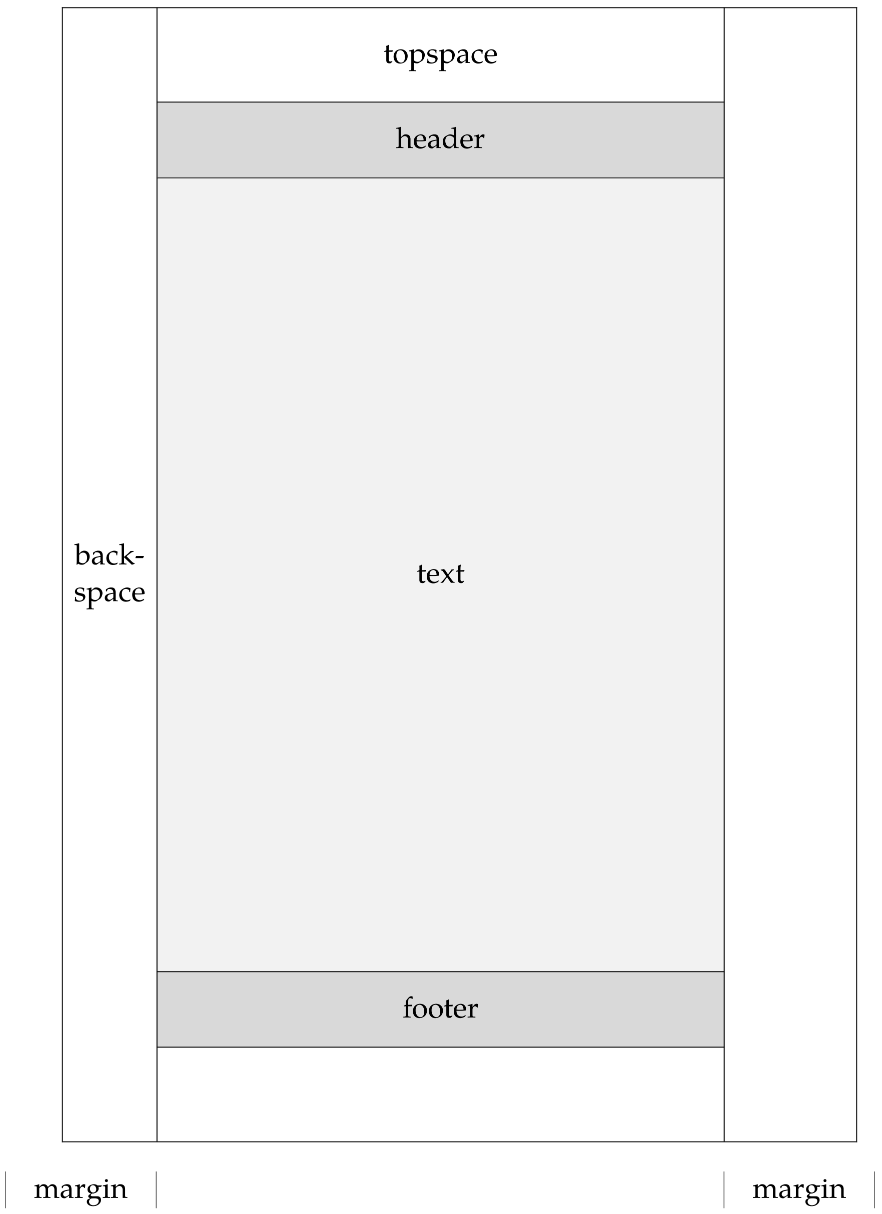


**Figure 3.1** The A4 typesetting area and margins (height = header + text + footer).

On the contrary, the height of the header and footer influences the height of the text area. When talking about the height, we think of the sum of the header, text and footer areas. This approach enables you to occasionally hide the header and/or footer, without introducing inconsistency in the layout.

The dimensions and location of all those areas are set up with `\setuplayout`.

Setting up the left or right margin has no influence on the typesetting area. In paper documents this parameter is only of use when keywords or other text are placed in the margin (hyphenation).

For paper documents it is sufficient to set up the height, header, footer, top space and back space. For electronic and screen documents however we need some extra space for navigational tools

```
\setuplayout [..,.*=.,..]

*   width                 =  DIMENSION fit middle
    height                =  DIMENSION fit middle
    backspace             =  DIMENSION
    topspace              =  DIMENSION
    margin                =  DIMENSION
    leftmargin            =  DIMENSION
    rightmargin           =  DIMENSION
    header                =  DIMENSION
    footer                =  DIMENSION
    top                   =  DIMENSION
    bottom                =  DIMENSION
    leftedge              =  DIMENSION
    rightedge             =  DIMENSION
    headerdistance        =  DIMENSION
    footerdistance        =  DIMENSION
    topdistance           =  DIMENSION
    bottomdistance        =  DIMENSION
    leftmargindistance    =  DIMENSION
    rightmargindistance   =  DIMENSION
    leftedgedistance      =  DIMENSION
    rightedgedistance     =  DIMENSION
    horoffset             =  DIMENSION
    veroffset             =  DIMENSION
    style                 =  normal bold slanted boldslanted type cap small... COMMAND
    color                 =  IDENTIFIER
    marking               =  on off color screen TEXT
    location              =  left middle right bottom top singlesided doublesided
    scale                 =  DIMENSION
    nx                    =  NUMBER
    ny                    =  NUMBER
    dx                    =  DIMENSION
    dy                    =  DIMENSION
    lines                 =  NUMBER
    columns               =  NUMBER
    columndistance        =  DIMENSION
    grid                  =  yes no
    bottomspace           =  DIMENSION
    cutspace              =  DIMENSION
    textdistance          =  DIMENSION
    textwidth             =  NUMBER
    textmargin            =  DIMENSION
    clipoffset            =  DIMENSION
    page                  =  IDENTIFIER
    paper                 =  IDENTIFIER
```

(see chapter ??). In screen documents it is common practice to use backgrounds. Therefore it is also possible to set up the space between the text area and the header and footer on a page, and thereby visually separating those areas.

| **Parameter** | **Value** | Comment |
|---|---|---|
| width | dimension | Determines the width of the typesetting area. `Middle` sets the white space right to the typesetting area to the value of the backspace. typeFit takes values set |

| | | |
|---|---|---|
| | | for margins, edges and margin and edge distances into account. |
| height | dimension | The `height` is the sum of the text height, header, footer, headerdistance, footerdistance. Middle sets the bottom white space to the value of the topspace. `Fit` calculates the text height based on the other vertical height-elements. |
| backspace | dimension | `Backspace` determines the left boundary of the typesetting area. |
| topspace | dimension | `Topspace` determines the top boundary of the typesetting area. Together `backspace` and `topspace` determine the left top corner of the typesetting area. |
| margin | dimension | Setting this parameters makes left and right margin equally large. |
| leftmargin | dimension | For documents with different size of the left and right margin, the left margin size is determined. |
| rightmargin | dimension | For documents with different size of the left and right margin, the right margin size is determined. |
| header | dimension | Determines the height of a running header. The header height is part of the `height` parameter. |
| footer | dimension | Determines the height of the footer. The footer height is part of the `height` parameter. |
| top | dimension | Makes space available in the `topspace` area. This parameter is not part of the text height. |
| bottom | dimension | Makes space available underneath the typesetting area. This parameter is not part of the text height. |
| leftedge | dimension | This space located left to the left margin is for screen documents only. |
| rightedge | dimension | This space located right to the right margin is for screen documents only. |
| headerdistance | dimension | All parameters ending on ...`distance` create white space between adjacent elements. |
| footerdistance | dimension | |
| leftmargindistance | dimension | |
| rightmargindistance | dimension | |
| leftedgedistance | dimension | |
| rightedgedistance | dimension | |

| | | |
|---|---|---|
| topdistance | dimension | |
| bottomdistance | dimension | |
| horoffset | dimension | A horizontal offset moves the complete layout horizontally, starting from the place indicated by the parameter `location`. |
| veroffset | dimension | A vertical offset moves the complete layout vertically, starting from the place indicated by the parameter `location`. |
| style | normal bold slanted boldslanted type cap small... COMMAND | With the style parameter one can setup the general style of the font(s) used in the document. |
| marking | on off color screen TEXT | When this parameter is set to `on`, then crop marks are placed around the page. `Color` displays a color bar, whereas `screen` shows a gray-values bar. |
| location | left middle right bottom top singlesided doublesided duplex | `location` determines where the page is placed on the paper. It allows to typeset single and double sided documents and documents for duplex printing (**see: 3.6**). |
| scale | number | With `scale` it is possible to scale a page before placing it on the defined paper. |
| nx | number | In case that a given text should be placed multiple times on a defined paper, `nx` gives the number of pages on the x-axis and `ny` the number of pages on the y-axis. |
| ny | number | |
| dx | dimension | With `dx` and `dy` the distances of the pages indicated in `nx` and `ny` can be manipulated. |
| dy | dimension | |
| lines | number | Determines the textheight in terms of the number of lines-heights. |
| columns | number | |
| columndistance | dimension | |
| grid | yes no | Typesetting on the grid is activated with grid=yes. |
| bottomspace | dimension | `Bottomspace` increases the white space at the bottom of the page without altering the page-layout. |
| cutspace | dimension | `Cutspace` increases the white space at the right side of the page without altering the page-layout. |
| textdistance | dimension | |

| | |
|---|---|
| textwidth | dimension |
| textmargin | dimension |
| clipoffset | dimension |
| page | identifier |
| paper | identifier |

In order to get information on the current settings the following commands can be issued:

```
\showframe [.*.]
           OPTIONAL
*   TEXT margin edge
```

The dimensions can be displayed by:

```
\showsetups
```

A multi–page combination of both is generated with:

```
\showlayout
```

The width of a text is available as `\hsize` and the height as `\vsize`. To be on the safe side one can better use ConTEXt's `\dimen`–registers `\textwidth` and `\textheight`, `\makeupwidth` and `\makeupheight`.

When we are typesetting in one column of text `\textwidth` and `\makeupwidth` are identical. In case of a two columned text the `\textwidth` is somewhat less than half the `makeupwidth`. The `\textheight` is the `\makeupheight` minus the height of the header and footer.

| **variable** | **meaning** |
|---|---|
| `\makeupwidth` | width of a text |
| `\makeupheight` | height of a text |
| `\textwidth` | width of a column |
| `\textheight` | height – header – footer |

**Table 3.2** Some `\dimen` variables

There are also other dimensions available like `\leftmarginwidth` and `\footerheight`, but be aware of the fact that you can only use these variables, you can not set them up. The width of a figure could for instance be specified as `width=.9\leftmarginwidth`.

Basically documents are typeset automatically. However, in some cases the output would become much better if a line would be moved to another page. For these situations you can adjust the layout temporarily (just for that page) by typing:

```
\adaptlayout [...,...] [..,..=..,..]
                  1          2
              OPTIONAL
1   NUMBER

2   height  =  DIMENSION max
    lines   =  NUMBER
```

The use of this command should be avoided inside a text, because after altering your document the adjustment could possibly not be necessary anymore. So, if you use this command, use it at the top of your document. For example:

```
\adaptlayout[21,38][height=+.5cm]
```

The layout of page 21 and 38 will temporarily be 0.5 cm higher though the footer will be maintained at the same height. The numbers to be specified are the page numbers in the output file.

If the layout is disturbed you can reset the layout by:

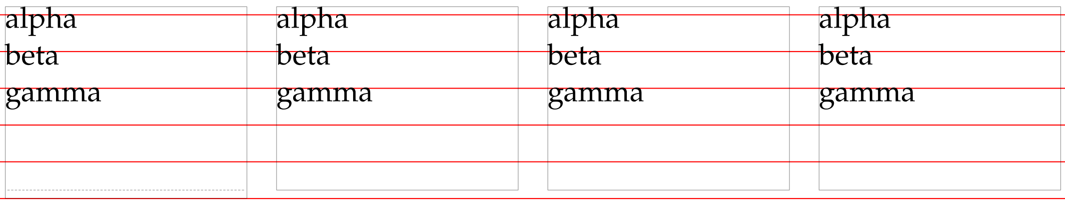
```
\setuplayout[reset]
```

In some commands you can set up the parameters `width` and `height` with the value `fit`. In that case the width and height are calculated automatically.

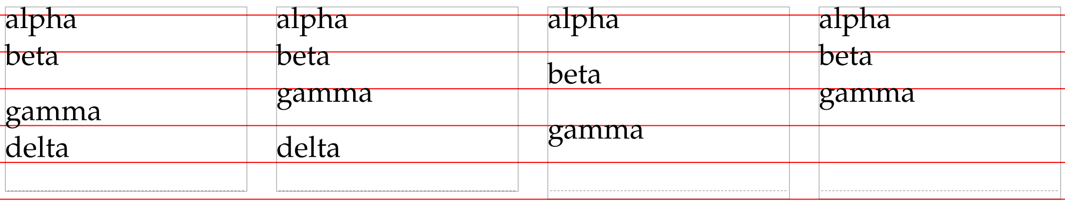
On the next pages we will show a number of A5 page layouts centered on an A4. The default setups (dimensions) are adequate for standard documents like manuals and papers. The setup adjusts automatically to the paper size. Note the use of `middle` while setting up the parameters width and height.

## 3.5 Grids

There are many ways to align text on a page. Look at the example below and notice the vertical alignment of the words and the white space between the words on the mini pages.



The first three alternatives result in an undesired output. The fourth alternative will lead to pages with unequal length. So we rather make the white space between the lines a little stretchable.[1]



A stretchable line spacing has the disadvantage that lines of two pages or two columns that are displayed close to each other, will seldom align. This is very disturbing for a reader.[2] In those situations we prefer to typeset on a grid. The means to do this in TEX are very limited but ConTEXt has some features to support grid typesetting.[3]

Grid typesetting is part of the page layout. To enable it globally, the parameter `grid` needs to be set to `yes` (or `normal`), which starts snapping with the default behavior. If this does not lead to the desired results, the value `strict` will adjust the grid more rigidly and `tolerant` more loosely.

```
\setuplayout[grid=yes]
```

During typesetting on a grid the heads, figures, formulas and the running text are set on a fixed line spacing. If a typographical component for any reason is not placed on the grid one can snap this component to the grid with:

```
\placeongrid{\framed{This is like a snapshot.}}
```

This will result in:

1 Hey, watch this. A footnote!
2 Here! Another footnote.
3 Finally, the last footnote!

This is like a snapshot.

This mechanism can be influenced with an argument:

```
\placeongrid[bottom]{\framed{Do you like the snapshot?}}
```

Now an empty line will appear below the framed text. Other parameters are: `top` and `both`. The last parameter divides the linespace between over and below the framed text.

Now the snapshot looks better.

These examples don't show pretty typesetting. The reason is that `\framed` has no depth because TEX handles spacing before and after a line in a different way than text. ConTEXt has a solution to this:

```
\startlinecorrection
\framed{This is something for hotshots.}
\stoplinecorrection
```

The command `\startlinecorrection` tries to typeset the lines as good as possible and takes the use of grid in account.

This is something for hotshots.

Because line correction takes care of the grid we have to use yet another command to stretch the framed text:

```
\startlinecorrection
\framed{Anyhow it is good to know how this works.}
\stoplinecorrection
```

As you can see this results in somewhat more space:

Anyhow it is good to know how this works.

For test purposes one can display the grid with the command `\showgrid`. So grid related commands are:

```
\placeongrid [.1.] {.2.}
               OPTIONAL
1  inherits from \moveongrid
2  CONTENT
```

```
\showgrid [...,¹...] {.².}
              OPTIONAL
1   reset top bottom none all lines frame nonumber right left
2   CONTENT
```

## 3.6 Printing

In an earlier section we used page and paper dimensions. In this section we will discuss how these two can be manipulated to yield a good output on paper.

In figure 3.3 and 3.4 we see some alternatives to manipulate the page composition by means of `\setuppapersize` and`\setuplayout`. So it is possible to put a page in a corner or in the middle of the paper, to copy a page and to use cutting marks.

When the parameter paper size is set to `landscape` width and height are interchanged. This is not the same as rotation! Rotation is done by typing `90`, `180` and `270` in the first argument of `\setuppapersize`.

```
\setuppapersize[A5,landscape][A4]
```

These examples don't show that we can correct for duplex printing. For example when we type:

```
\setuppapersize[A5][A4]
\setuplayout[location=middle,marking=on]
```

the front and back side will be placed in the middle of the paper. The markings enable you to cut the paper at the correct size. If we only want to cut twice, we type:

```
\setupppapersize[A5][A4]
\setuplayout[location=duplex]
```

This has the same meaning as `{duplex,left}`. At this setup ConTEXt will automatically move front and back side to the correct corner. In figure 3.2 we show both alternatives.

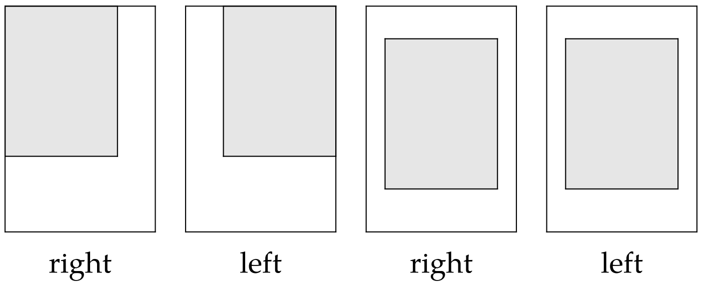
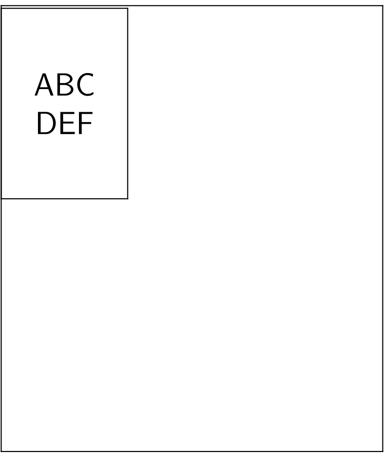


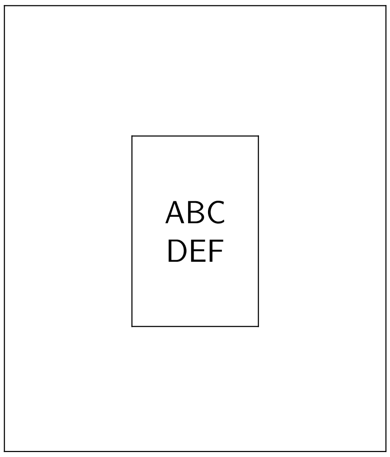
**Figure 3.2** Positioning the page on paper for cutting.

Rotating, mirroring, scaling, duplicating and placing pages on paper are independent operations. By combining these operations the desired effects can be reached. Rotating and mirroring and page and paper size are set up at the same time. The other operations are set up with `\setuplayout`.

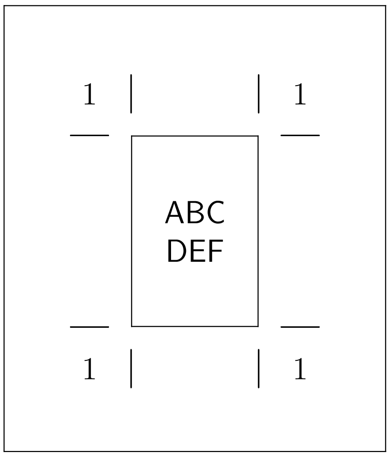
## 3.7 Arranging pages

Simplified we can say that TEX typesets pages. If the typeset material should become a book, then there are two options. Firstly the book will be produced on multiple sheets carrying only one page either on one or on both sides of the sheet. Second option is to produce arrangements of multiple pages per sheet of paper which will be folded into sections, using imposition schemes.

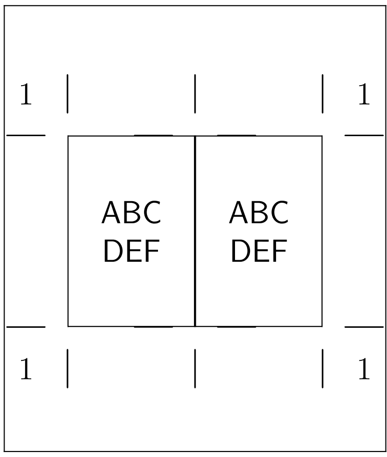




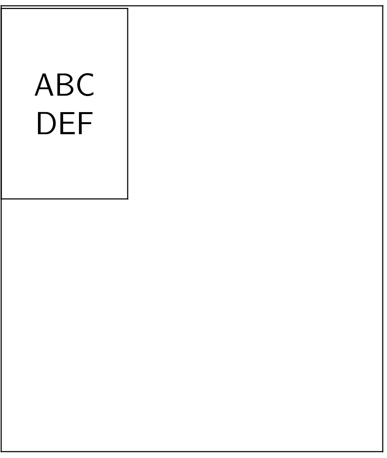
location=middle



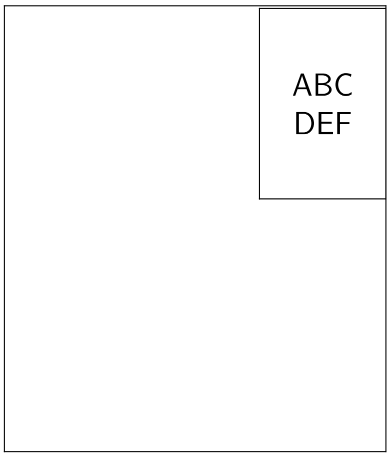
marking=on
location=middle



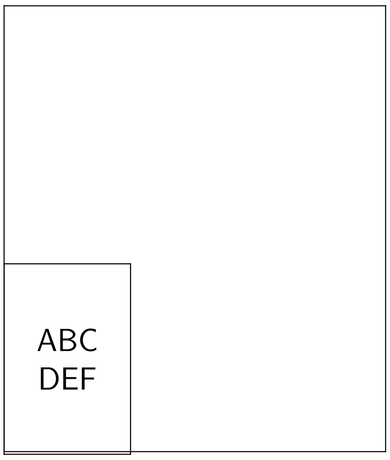
marking=on
location=middle
nx=2



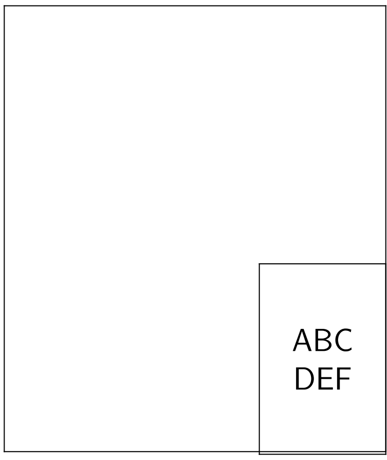
location=left



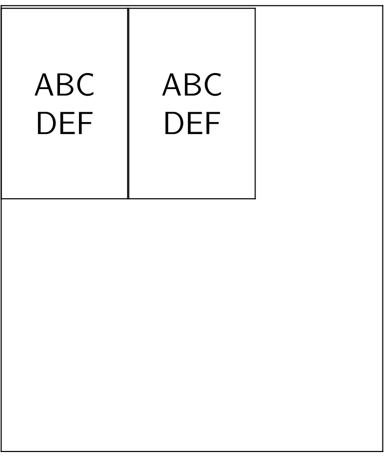
location=right



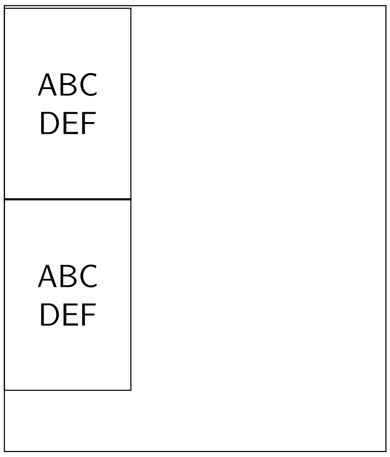
location=left,bottom



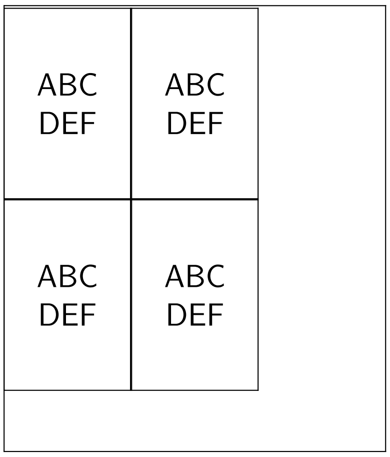
location=right,bottom



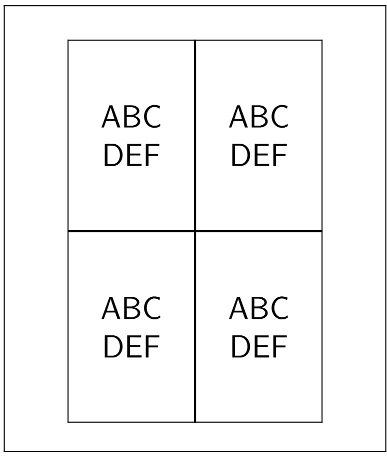
nx=2,ny=1



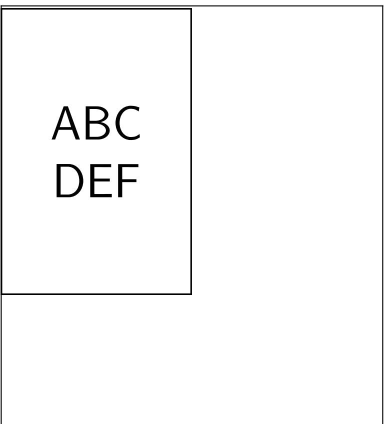
nx=1,ny=2

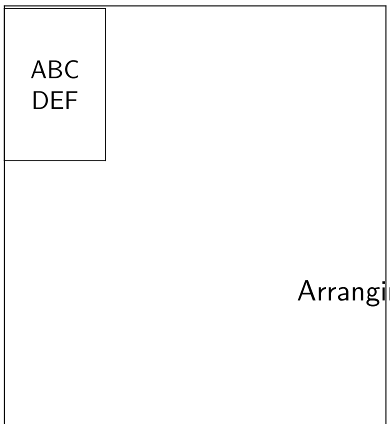


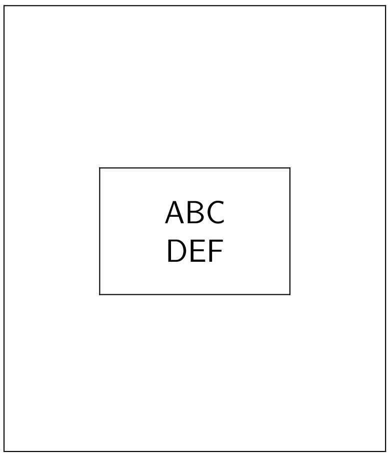
nx=2,ny=2



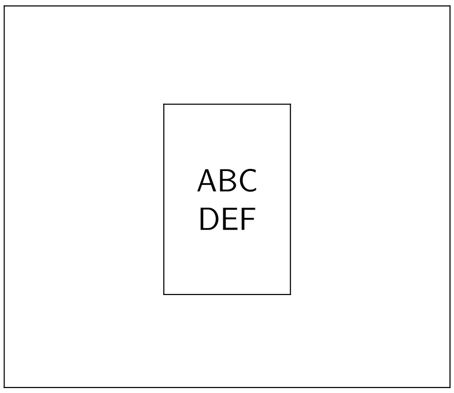
nx=2,ny=2
location=middle



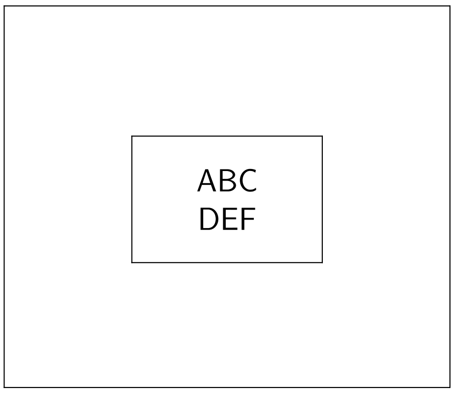




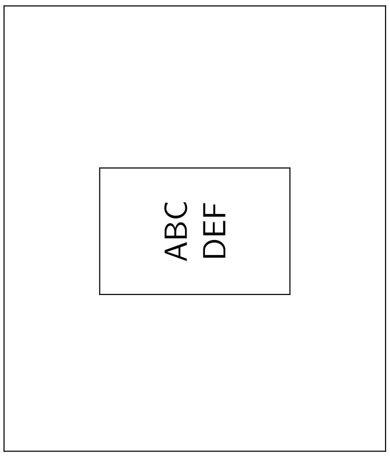
landscape



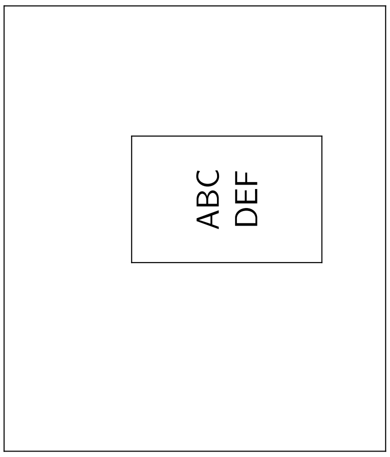
landscape



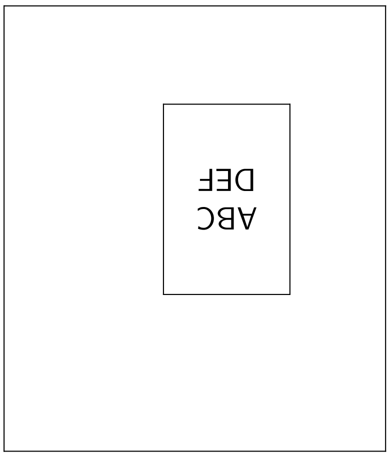
landscape
landscape



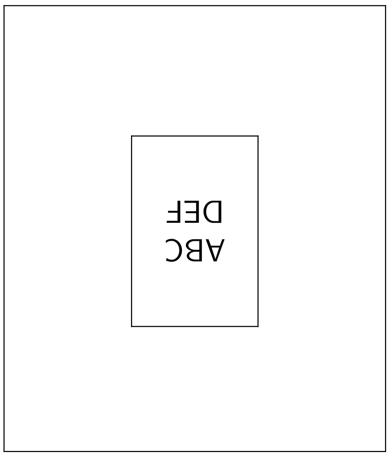
90



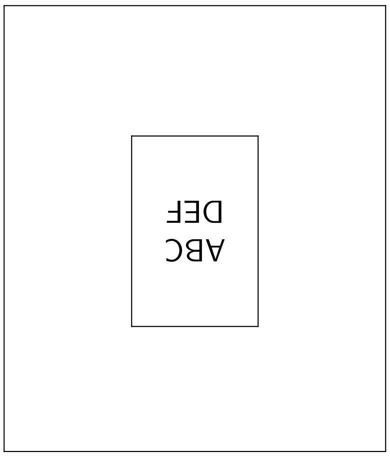
90



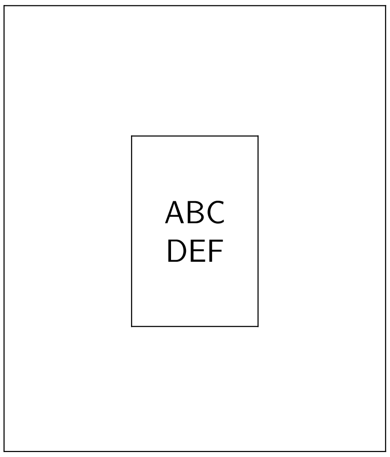
90
90



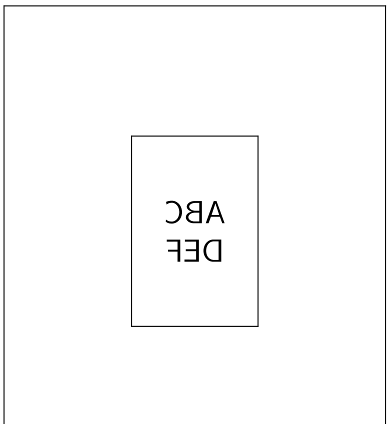
180

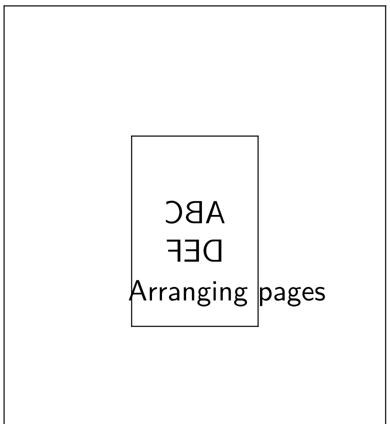


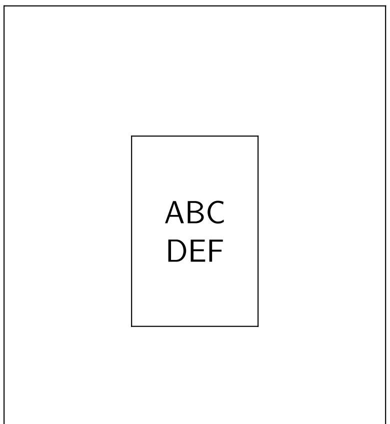
180



180
180







ConTEXt offers tools to achieve both options.

In the following table an overview is given about all currently available arranging schemes.

| Key for `\setuparranging` | Meaning |
|---|---|
| `[2SIDE]` | 2 pages next to each other single sided only! |
| `[2TOP]` | 2 pages above each other, single sided only! |
| `[1*8]` | 1 sheet 1 x 8 pages = 8 pages single sided! |
| `[1*4]` | 1 sheet 1 x 4 pages = 4 pages single sided! |
| `[1*2*Conference]` | 2 pages on top of each other, 1 page rotated |
| `[1*4*Conference]` | 2 odd pages next to each other, even page rotated on top |
| `[XY]` | Arrangement in nx columns and ny rows, uses the setup `\setuppaper[dx=,dy=,n` |
| `[2UP]` | 2 pages next to each other, n sheets arranged for a single booklet! |
| `[2DOWN]` | 2 pages above each other, n sheets arranged for a single booklet! |
| `[2TOPSIDE]` | 2 odd pages on one side, 2 even pages verso, above each other |
| `[2*16]` | Section: one sheet 2 x 16 pages = 32 pages |
| `[2*8]` | Section: one sheet 2 x 8 pages = 16 pages |
| `[2*8*Z]` | Section: one sheet 2 x 8 pages = 16 pages, special folding: zig-zag |
| `[2*6*Z]` | Section: one sheet 2 x 6 pages = 12 pages, special folding: zig-zag |
| `[2*4]` | Section: one sheet 2 x 4 pages = 8 pages |
| `[2*2]` | Section: one sheet 2 x 2 pages = 4 pages |
| `[2**2]` | Section: one sheet 2 x 2 pages = 4 pages |
| `[2*4*2]` | Section of 16 pages: 2 sheets, 4 pages front and backside |
| `[2*2*4]` | Section of 16 pages: 4 sheets, 2 pages front and backside |
| `[3SIDE]` | 3 odd pages recto, 3 even pages verso |
| `[2*2*2]` | Section: two sheets 2 x 2 pages = 8 pages |
| `[2*2*3]` | Section: three sheets 2 x 2 pages = 12 pages |
| `[TRYPTICHON]` | Leaflet: one sheet 2 x 3 pages = 6 pages |
| `[DOUBLEWINDOW]` | Leaflet: one sheet 2 x 4 pages = 8 pages |
| `[ZFLYER-8]` | Leaflet: one sheet 2 x 4 pages = 8 pages |
| `[ZFLYER-10]` | Leaflet: one sheet 2 x 5 pages = 10 pages |
| `[ZFLYER-12]` | Leaflet: one sheet 2 x 6 pages = 12 pages |
| `[MAPFLYER-12]` | Leaflet: one sheet 2 x 6 pages = 12 pages |

When talking about book-printing the industry produces different kinds of sections, consisting commonly out of 32 or 16 pages. Consider, that sections of 32 pages may be quite thick. At binding if the sections are sewn and the spine is rounded the fore edge can become stepped. This is aesthetically less satisfying. Best results are normally obtained with sections of 16 pages.

For special purposes or in case of special papers also less than 16 pages per section are arranged.

The command to arrange pages with ConTEXt is

For (standard) sections the following list of schemes is available:

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[2*16]` | section: one sheet 2 × 16 pages | = | 32 pages |
| `\setuparranging[2*8]` | section: one sheet 2 × 8 pages | = | 16 pages |
| `\setuparranging[2*4]` | section: one sheet 2 × 4 pages | = | 8 pages |
| `\setuparranging[2*2]` | section: one sheet 2 × 2 pages | = | 4 pages |
| `\setuparranging[2**2]` | section: one sheet 2 × 2 pages | = | 4 pages |

| | | | |
|---|---|---|---|
| `\setuparranging[2*8*Z]` | section: one sheet 2 × 8 pages | = | 16 pages, special folding: zig-zag |
| `\setuparranging[2*6*Z]` | section: one sheet 2 × 6 pages | = | 12 pages, special folding: zig-zag |
| `\setuparranging[2*4*2]` | section: 2 sheets, 4 pages front and backside | = | 16 pages |
| `\setuparranging[2*2*4]` | section: 4 sheets, 2 pages front and backside | = | 16 pages |
| `\setuparranging[2*2*2]` | section: 2 sheets 2 × 2 pages | = | 8 pages |
| `\setuparranging[2*2*3]` | section: 3 sheets 2 × 2 pages | = | 12 pages |

On the following pages we show pictures of arranged pages for the mentioned imposition schemes.

The above mentioned imposition schemes are meant for the professional printing industry.

But also with an office printer one can produce sections. Sections with less than 16 pages can be produced with the following folding schemes:

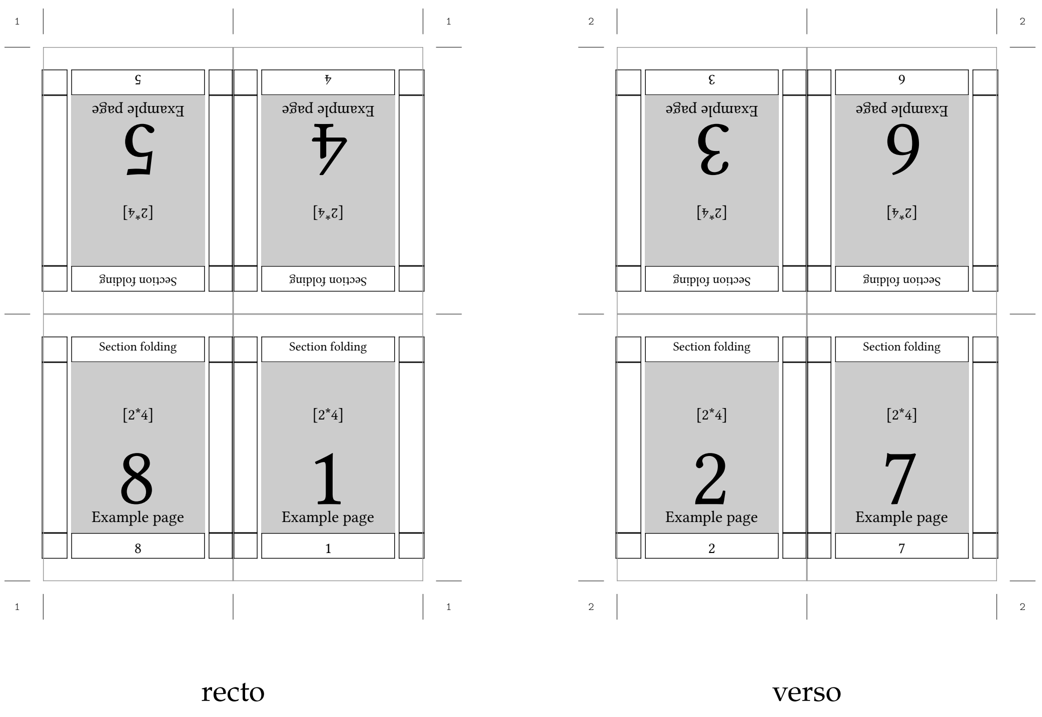
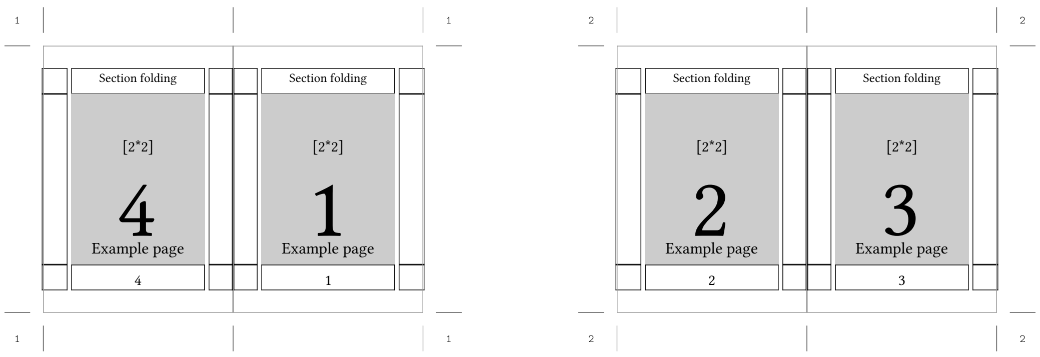


**Figure 3.5** 8 pages

The last two examples (Figure 3.6 and 3.7) differ only in the fact, that the verso side carries the two pages in reversed order.

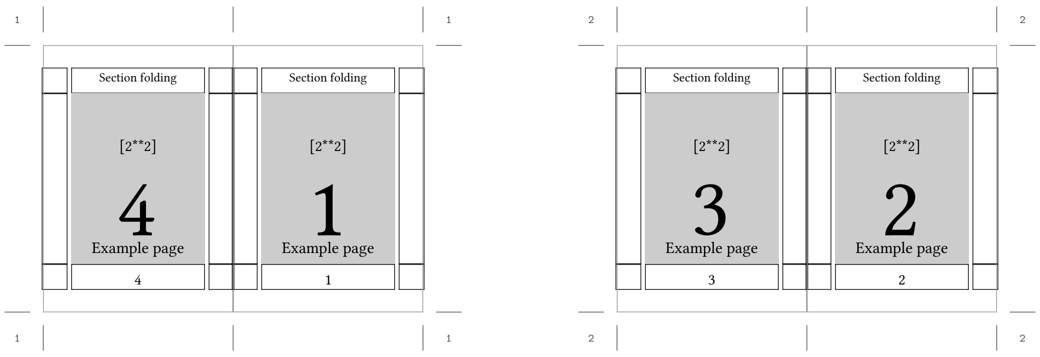
The simplest version of a section is booklet-printing. In this case all pages are arranged in such a way, that with a single fold a booklet is formed.

| **Arrangement** | **Result** | **Number of pages** |
|---|---|---|
| `\setuparranging[2UP]` | 2 pages next to each other, n sheets arranged for a single booklet | |
| `\setuparranging[2DOWN]` | 2 pages above each other, n sheets arranged for a single booklet | |



recto verso

**Figure 3.6** 4 pages



recto verso

**Figure 3.7** 4 pages

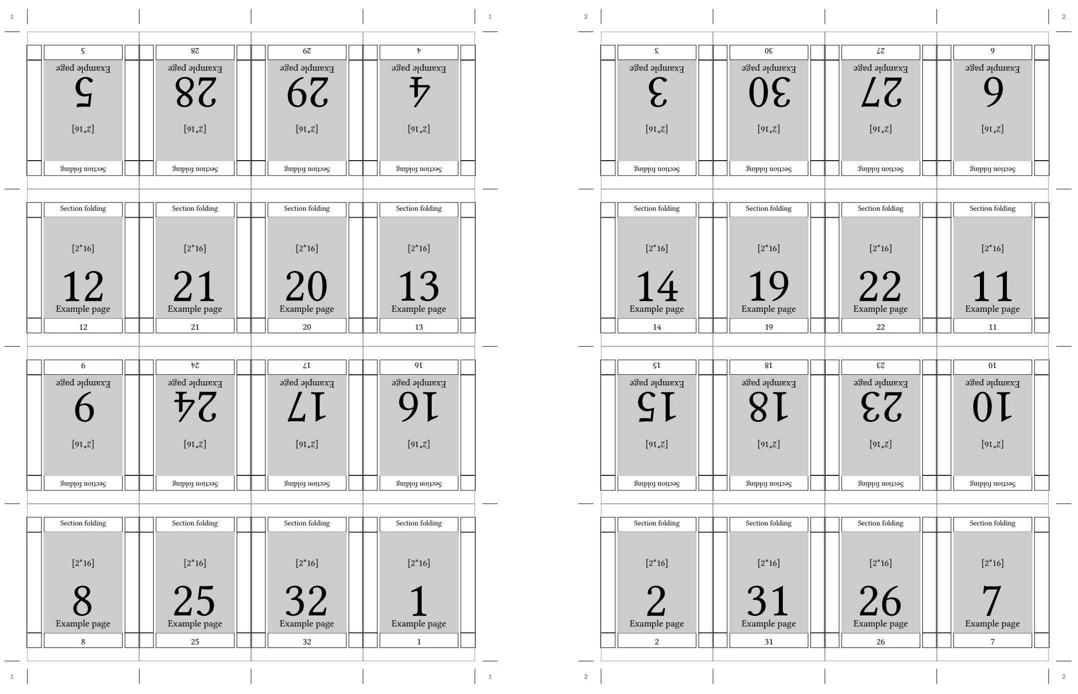


**Figure 3.8** 32 pages

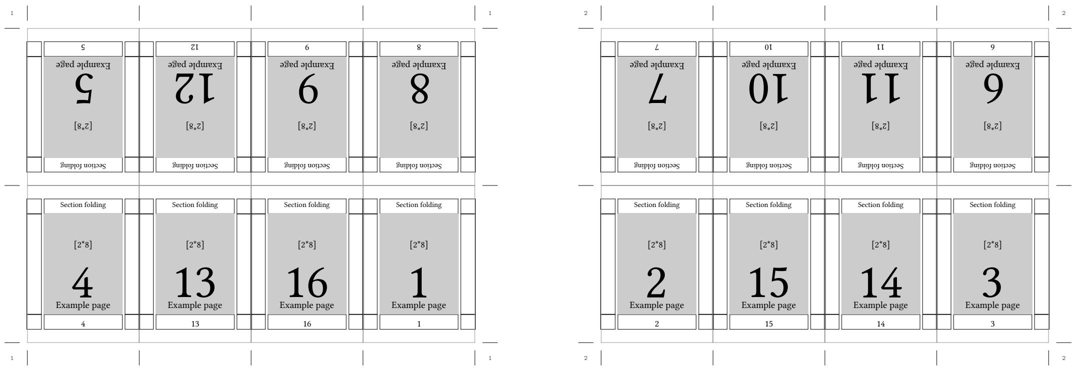


**Figure 3.9** 16 pages

‘2UP’ results in a booklet with the fold on the long egde of the page. ‘2DOWN’ gives a booklet with a short-edge binding of the pages.

For those who want to print their own book with sections on the office printer ConTEXt offers four schemes which use 2, 3 and 4 sheets of paper respectively to form a section.

| **Arrangement** | **Result** | **Number of pages** |
|---|---|---|
| `\setuparranging[2*4*2]` | section: 2 sheets, 4 pages front and backside = | 16 pages |
| `\setuparranging[2*2*4]` | section: 4 sheets, 2 pages front and backside = | 16 pages |

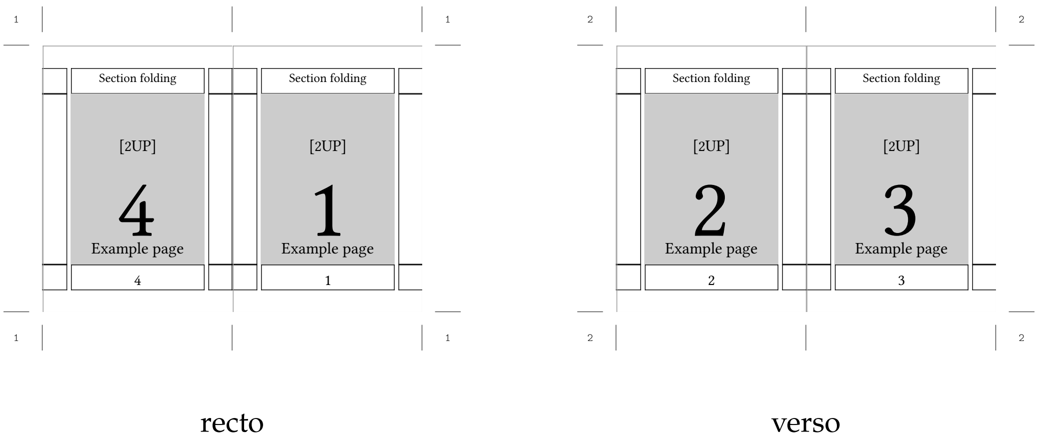


**Figure 3.10** 2 UP booklet: long edge binding

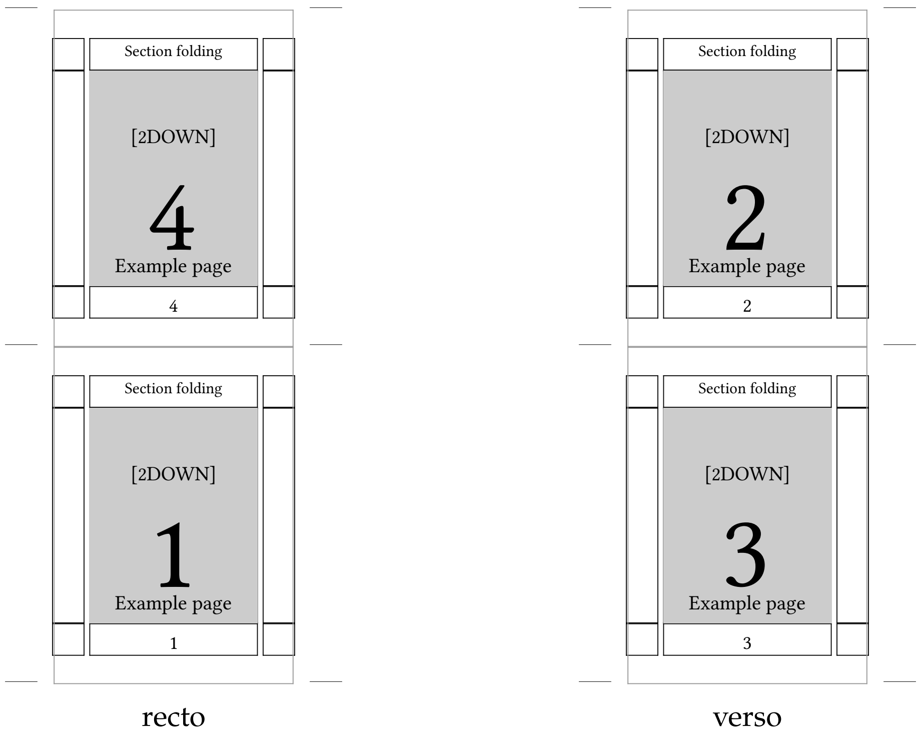


**Figure 3.11** 2 DOWN booklet: short edge binding

| | | | |
|---|---|---|---|
| `\setuparranging[2*2*2]` | section: 2 sheets 2 × 2 pages | = | 8 pages |
| `\setuparranging[2*2*3]` | section: 3 sheets 2 × 2 pages | = | 12 pages |

Yet another way to print sections is to use z-folding, which is a zig-zag folding combined with a single fold in the spine. ConTEXt comes with two types of sections, one with 12 pages and one with 16 pages.

Next to the imposition schemes involving folding ConTEXt offers possibilities to arrange pages in such a way, that after cutting the pile of sheets book blocks can be assembled. The resulting

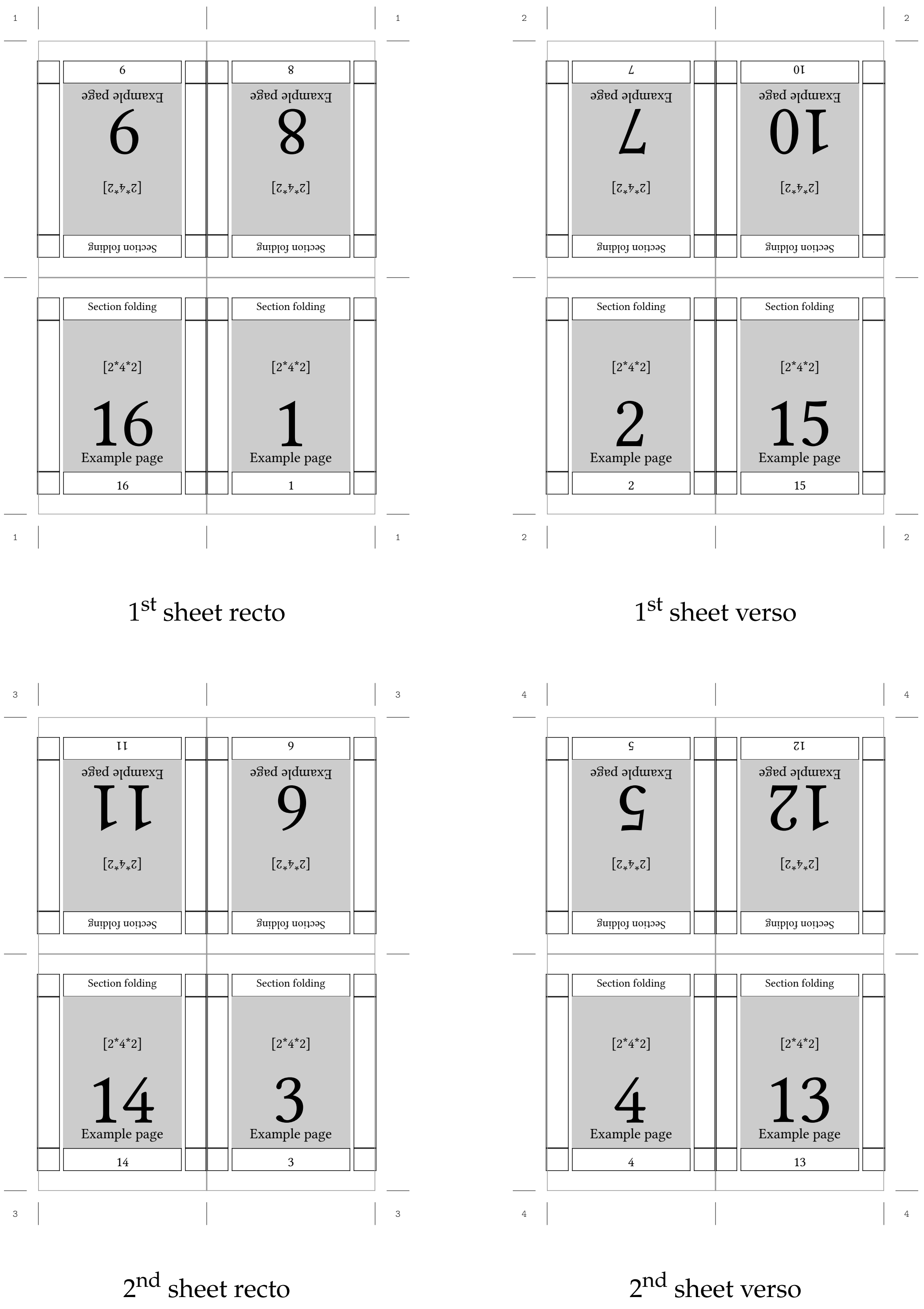
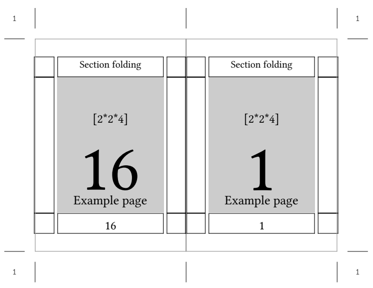
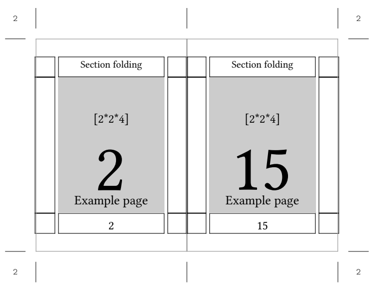


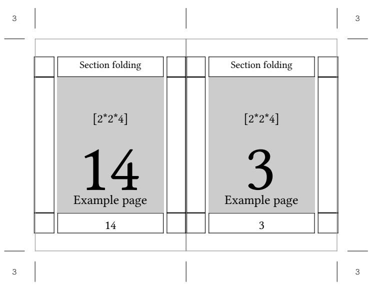
**Figure 3.12** 16 pages, 2 sheets



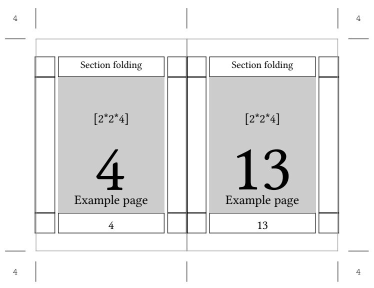
1st sheet recto



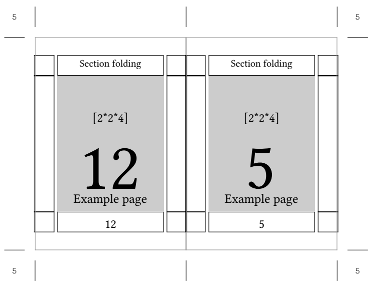
1st sheet verso



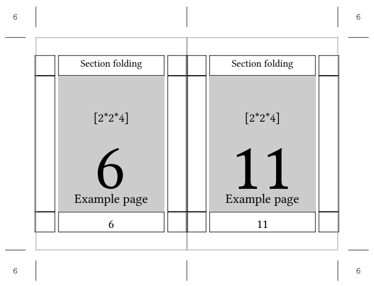
2nd sheet recto



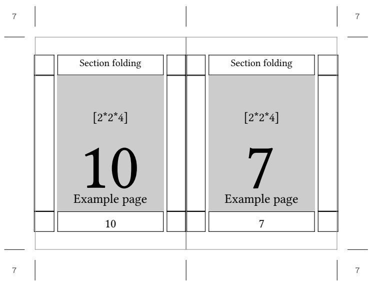
2nd sheet verso



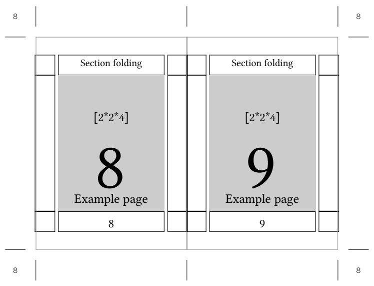
3rd sheet recto



3rd sheet verso



4th sheet recto



4th sheet verso

**Figure 3.13** 16 pages, 4 sheets

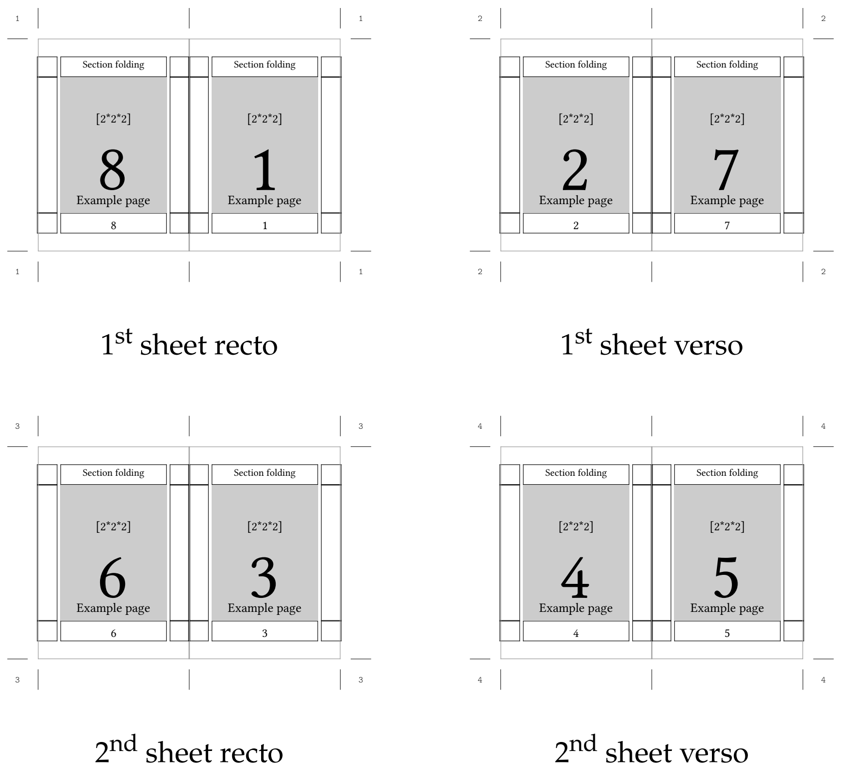


**Figure 3.14** 8 pages, 2 sheets

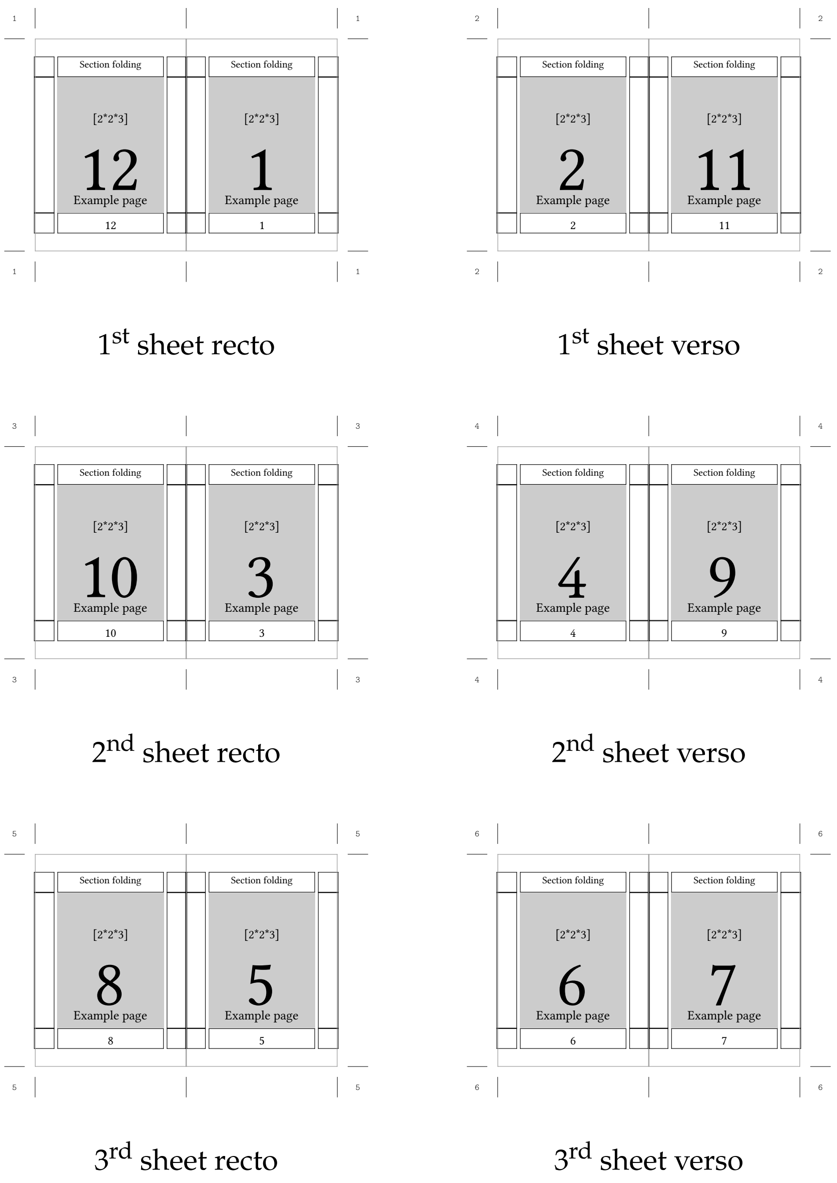


**Figure 3.15** 12 pages, 3 sheets

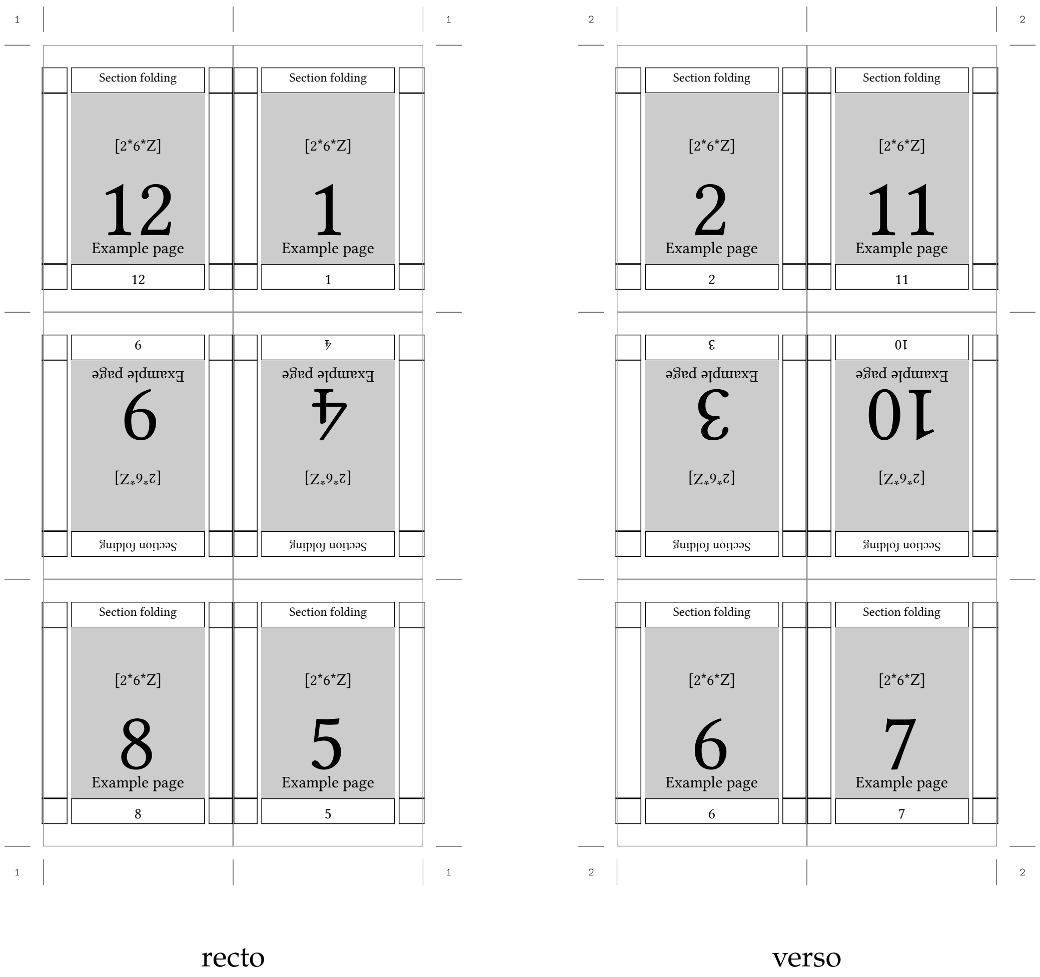
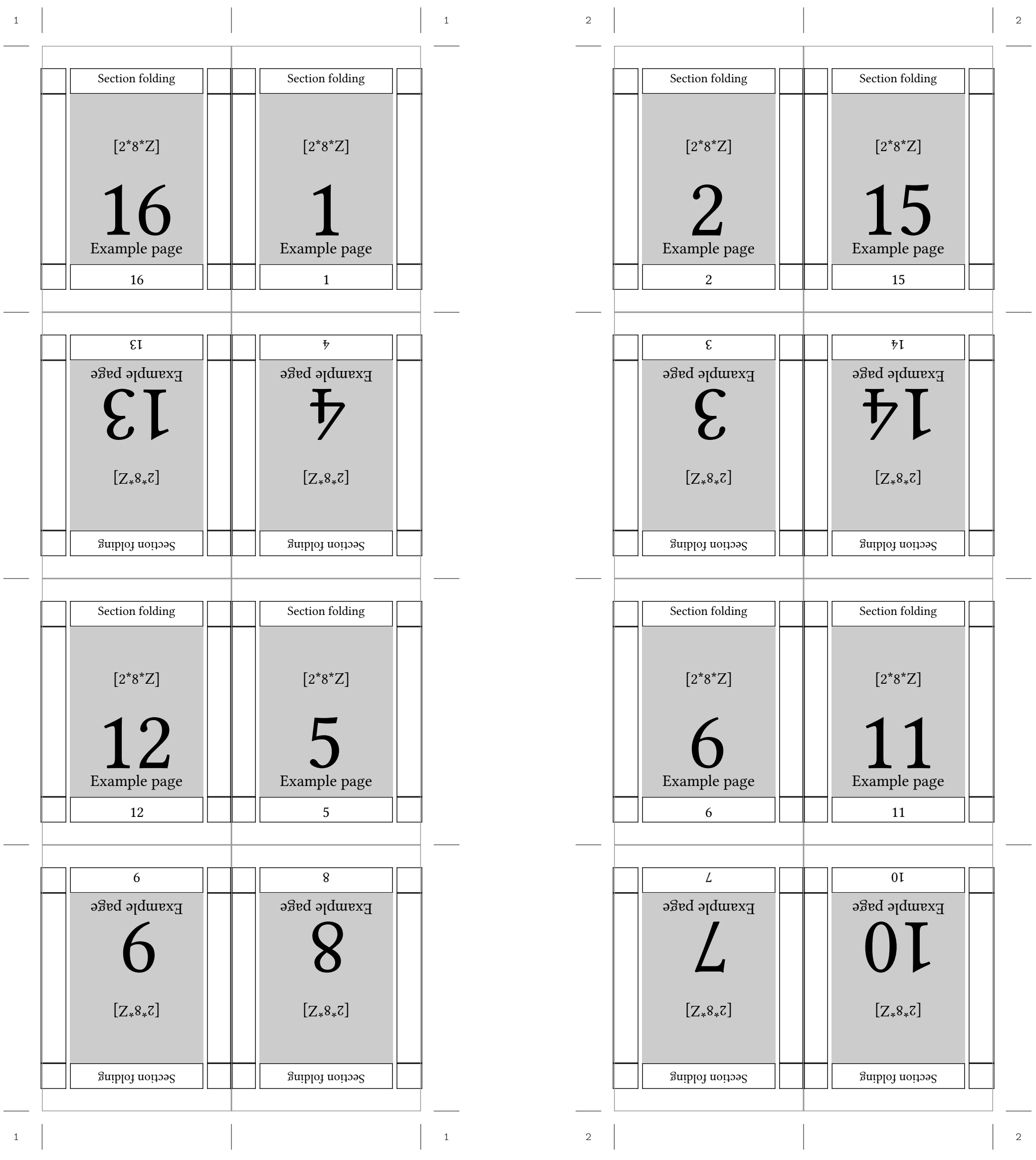


**Figure 3.16** 12 pages z-folding



recto verso

**Figure 3.17** 16 pages z-folding

book block consists of loose sheets of paper and will be glued along the spine to prepare e.g. a paperback.

ConTEXt has an arranging scheme for two odd pages above each other and two even pages on the backside of the sheet. In order to build the book block the sheets need to be cut and the the two piles must be merged.

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[2TOPSIDE]` | recto 2 odd pages, verso 2 even pages per sheet | = | 4 pages |

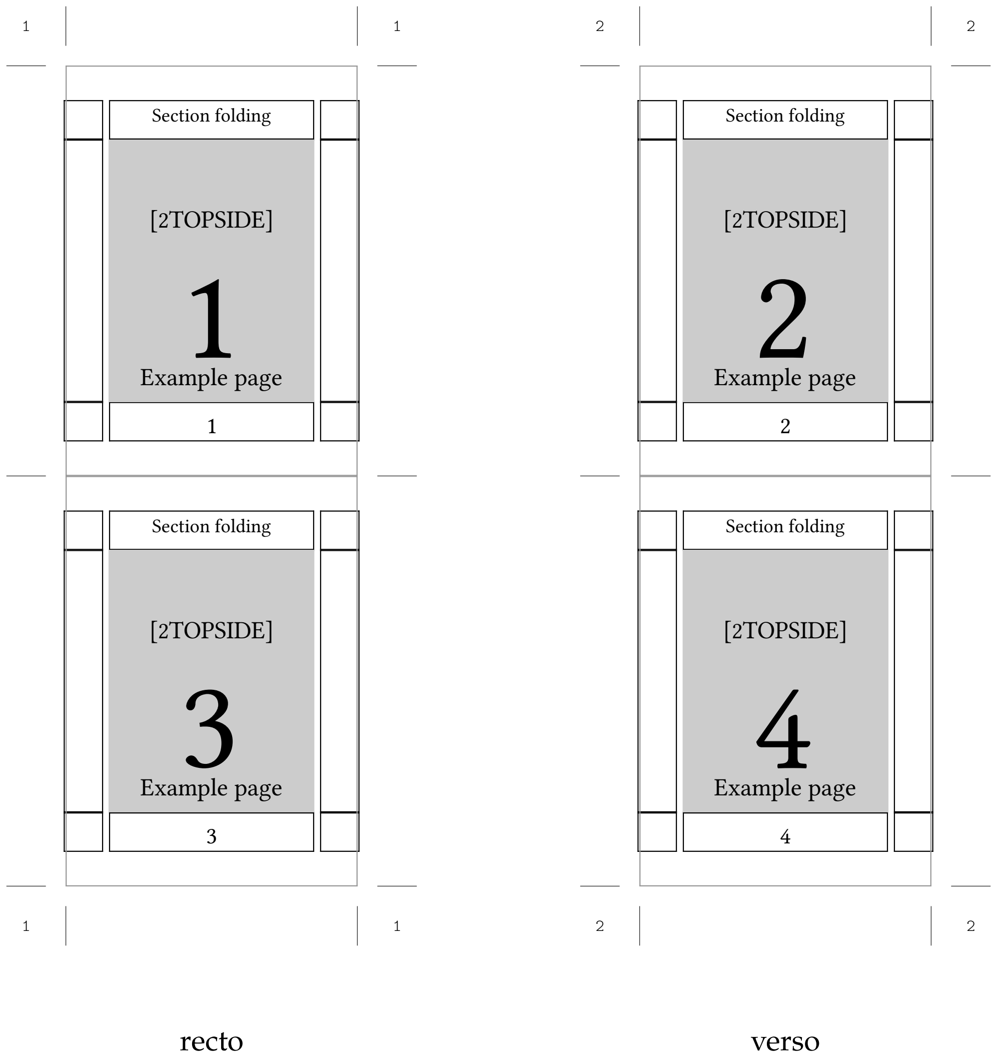


**Figure 3.18** 4 pages, 1 sheet

The following schemes can be used for the preparation of handouts from presentations. They also can be used to assemble book blocks after cutting and merging the piles.

The first scheme arranges 4 pages on the front side of the sheet.

The second scheme puts two pages on the front side of a sheet next to each other.

The third scheme works like the previous one but instead of putting the pages next to each other the pages are placed on top of each other.

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[1*4]` | one sheet recto 4 pages | = | 4 pages |
| `\setuparranging[2SIDE]` | one sheet recto 2 pages | = | 2 pages |
| `\setuparranging[2TOP]` | one sheet recto 2 pages | = | 2 pages |

There are a couple of arranging schemes for special purposes. The first one places 8 pages on the recto side of the paper. It is intentioned for single sided prints only. The arrangement is

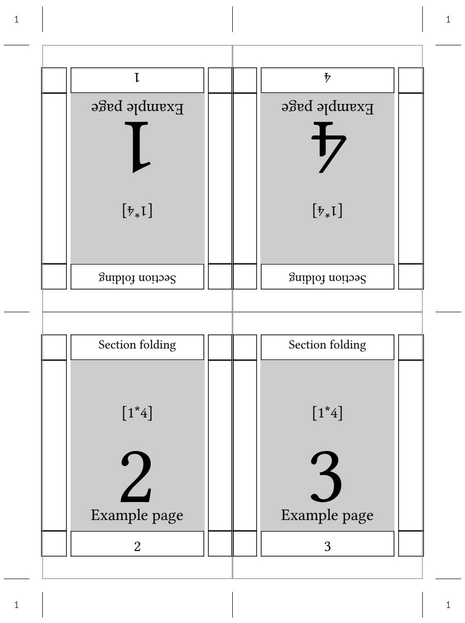
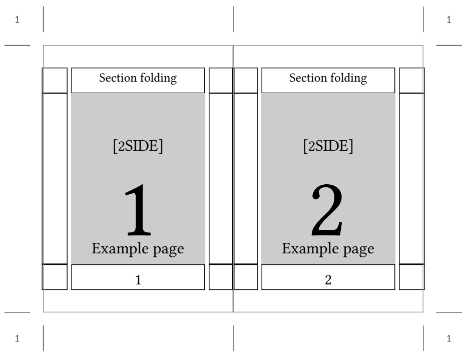
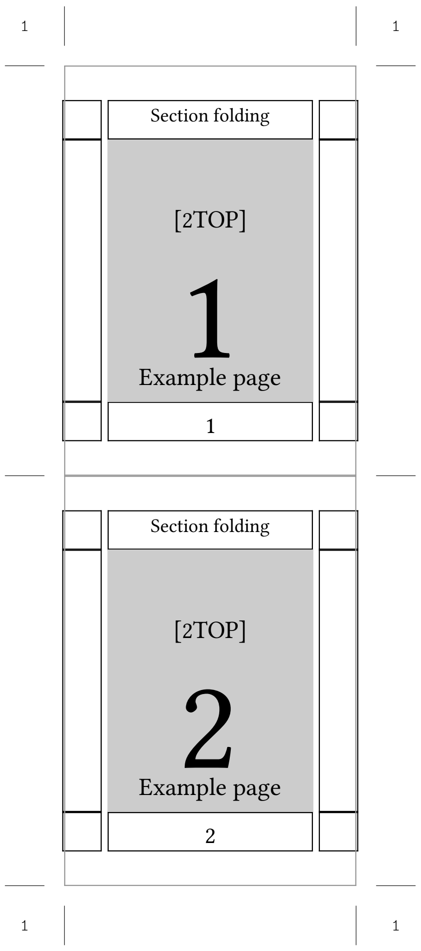


**Figure 3.19** 4 pages, singlesided, 1 sheet



`\setuparranging[2SIDE]`



`\setuparranging[2TOP]`

**Figure 3.20** 2 pages, single sided, 1 sheet

made in such a way, that it is possible to fold the paper into a booklet, where while turning the pages now empty pages are shown.

| **Arrangement** | **Result** | **Number of pages** |
|---|---|---|
| `\setuparranging[1*8]` | "section": one sheet 1 × 8 pages = | 8 pages |

For those who will have to produce name-card displays for e.g. conferences or for the preparation of menue-displays in a restaurant the following schemes might be of use.

| **Arrangement** | **Result** |
|---|---|
| `\setuparranging[1*2*Conference]` | one sheet 2 pages on top of each other, 1 page rotated |
| `\setuparranging[1*4*Conference]` | one sheet 2 odd pages next to each other, even page rotated on top |

There are diary systems, where three pages are place next to each other. The following scheme provides this arranging scheme:

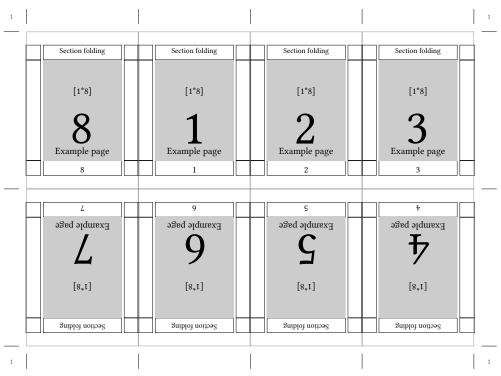
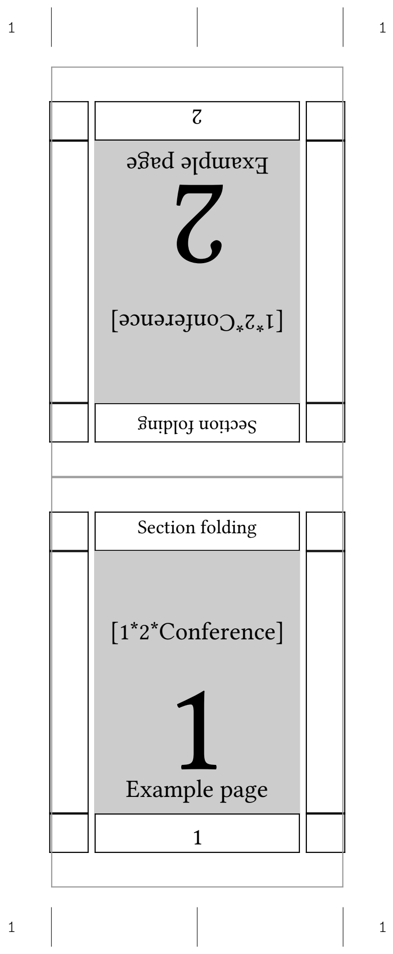
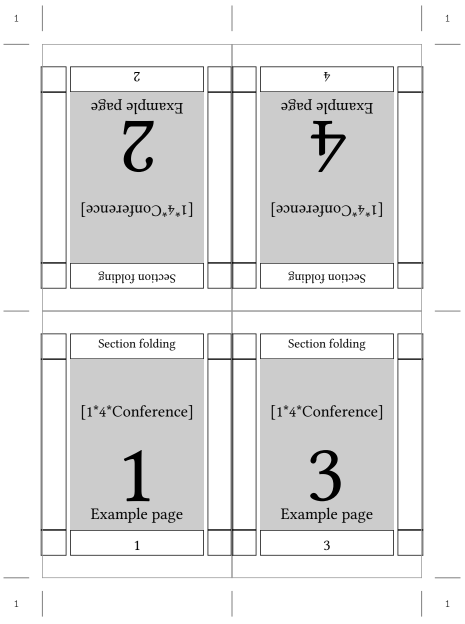


**Figure 3.21** 8 pages, single sided, 1 sheet





1 card with 2 pages

1 card with 4 pages

**Figure 3.22** Display cards

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[3SIDE]` | 3 odd pages recto, 3 even pages verso | = | 6 pages |

ConTEXt can also arrange pages for the production of flyers. There is a great variety of such flyers. ConTEXt supports flyers with 6, 8, 10 and 12 pages. It is also possible to make a flyer with 12 pages which is folded like a map.

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[TRYPTICHON]` | Leaflet: one sheet 2 × 3 pages | = | 6 pages |
| `\setuparranging[DOUBLEWINDOW]` | Leaflet: one sheet 2 × 4 pages | = | 8 pages |
| `\setuparranging[ZFLYER-8]` | Leaflet: one sheet 2 × 4 pages | = | 8 pages |
| `\setuparranging[ZFLYER-10]` | Leaflet: one sheet 2 × 5 pages | = | 10 pages |
| `\setuparranging[ZFLYER-12]` | Leaflet: one sheet 2 × 6 pages | = | 12 pages |
| `\setuparranging[MAPFLYER-12]` | Leaflet: one sheet 2 × 6 pages | = | 12 pages |

As a representative of the Z-folded flyers the flyer with 8 pages is shown.

Last but not least is the X-Y-arrangement of pages. This scheme is intended for the placement of a number of pages in sequence on a single sided sheet of paper e.g. on sheets carrying labels or for the placement of other information which must return several times on a sheet.

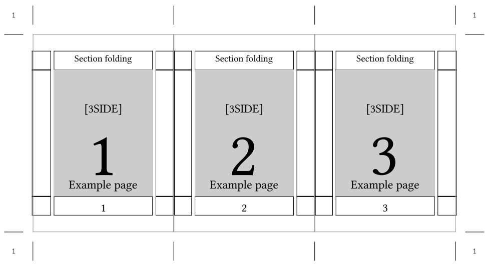
Before issuing the command `\setuparranging[XY]` the xy-arrangement must be setup. For this purpose the command `\setuppaper[...]` is used.

```
\setuppaper [..,.=.,..]
                  *
*   paper      =  IDENTIFIER
    page       =  IDENTIFIER
    nx         =  NUMBER
    ny         =  NUMBER
    width      =  DIMENSION
    height     =  DIMENSION
    topspace   =  DIMENSION
    backspace  =  DIMENSION
    option     =  max fit
```

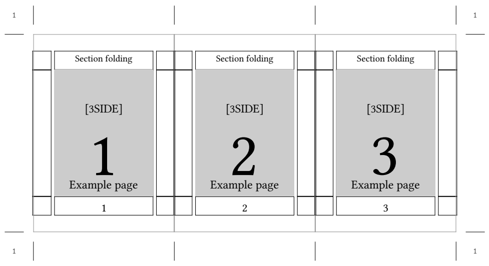
‘nx’ denominates the number of pages in the x-direction and ‘ny’ determines the number of pages in the y-direction. With ‘dx’ and ‘dy’ the whitespace between the pages in x and y direction can be set.

| **Arrangement** | **Result** | | **Number of pages** |
|---|---|---|---|
| `\setuparranging[XY]` + `\setuppaper[dx=,dy=,nx=,ny=]` | *nx* × *ny* pages, single sided | = | n × m pages |

There is culprit in arranging pages. If multiple layers of paper are folded, the outermost paper will require more width because it has to turn around the inner paper layers. This effect occurs as well in the spine folds as also in the head folds. How much width is required depends on the number of folds and the thickness of the paper. In professional book printing this effect is accounted for by displacing the pages depending on their position in horizontal and vertical direction. The result is that there will be a perfect look-through registering of all pages. There are no simple rules to indicate the required amount of displacement. Mostly it is a matter of experience to set up the page shift information.

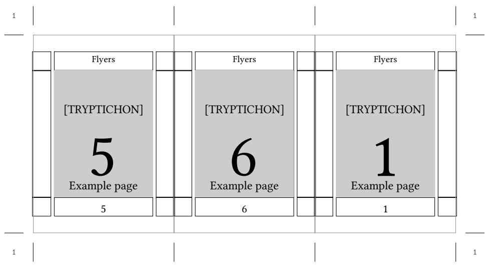


3 pages recto

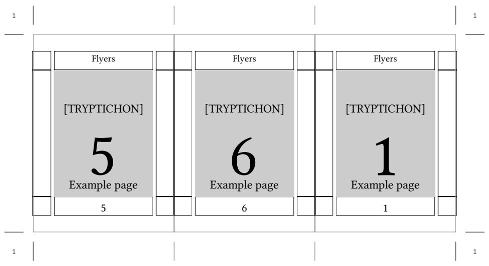


3 pages verso

**Figure 3.23** 3 pages per side

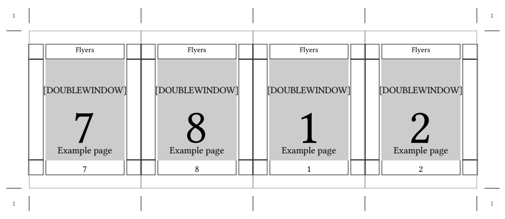


3 pages recto

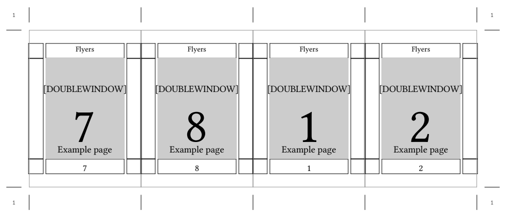


3 pages verso

**Figure 3.24** Tryptichon type of flyer

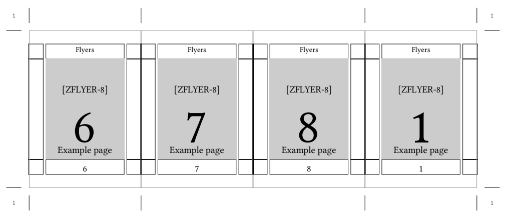


4 pages recto

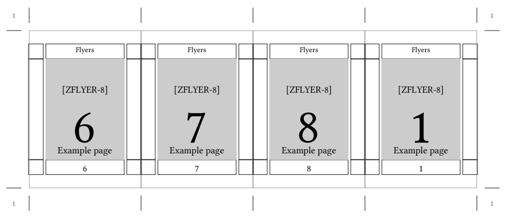


4 pages verso

**Figure 3.25** Double window type of flyer

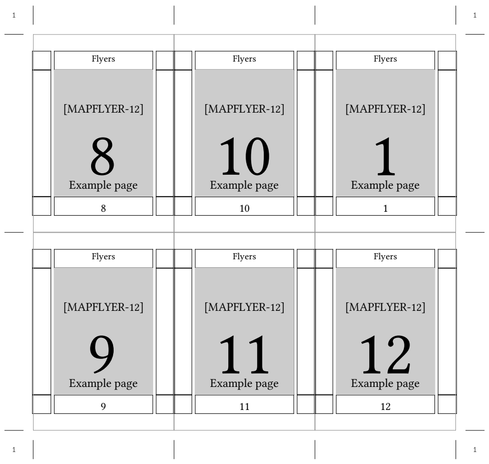


4 pages recto

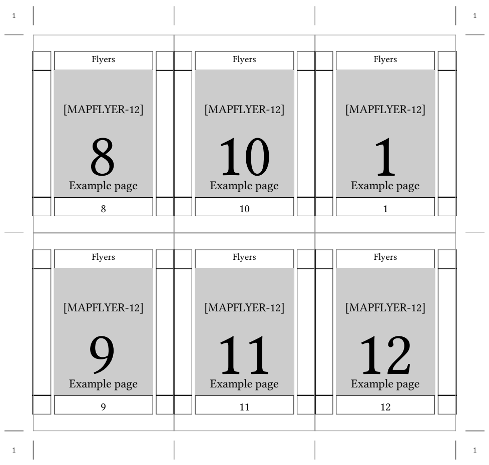


4 pages verso

**Figure 3.26** Z-folded type of flyer



6 pages recto



6 pages verso

**Figure 3.27** Map type of flyer

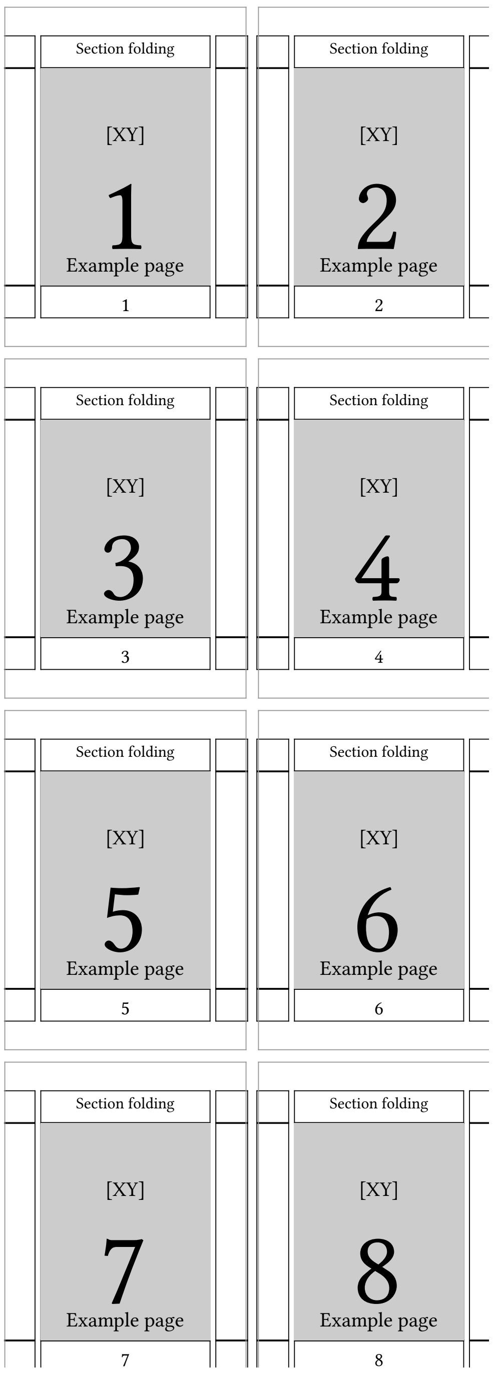


**Figure 3.28** 8 pages, singlesided, 1 sheet, XY-arrangement

ConTEXt is equipped with a mechanism, which allows to move pages on a sheet apart from each other in horizontal as well as in vertical direction. The mechanism is build on two shift-lists, one for horizontal and one for vertical page shifting. The mechanism works through cycling over the lists which contain a shift amount for each page in a section. For filling in such a shift-list knowledge and understanding the position of a page on the printed sheet is necessary.

In order to use a horizontal shift list this list must be defined and setup.

For a section of 16 pages a horizontal shift list is filled in where for each page the amount of displacement is given. Such a list could look as follows:

```
\definepageshift[Hor][horizontal]
   [0.25mm, %1
   -0.25mm, %2
   0.15mm,  %3
   -0.15mm, %4
   0.05mm,  %5
   -0.05mm, %6
   0mm,     %7
   0mm,     %8
   0mm,     %9
   0mm,     %10
   0.05mm,  %11
   -0.05mm, %12
   0.15mm,  %13
   -0.15mm, %14
   0.25mm,  %15
   -0.25mm] %16
```

For illustration purposes the following list for horizontal page-shift with exaggerated values is used in a Z-folding with 12 pages.

**FIXME:** Why do we need an explicit page break here?

```
\definepageshift[Hor][horizontal]
    [1mm,    %1
    -1mm,    %2
    0.5mm,   %3
    -0.5mm,  %4
    0mm,     %5
    0mm,     %6
    0mm,     %7
    0mm,     %8
    0.5mm,   %9
    -0.5mm,  %10
    1mm,     %11
    -1mm]    %12
```

In a similar fashion also vertical shift lists can be defined.

```
\definepageshift[Vert][vertical]
    [1.5mm,  %1
    1.25mm,  %2
    0.75mm,  %3
    1.0mm,   %4
    1.0mm,   %5
    0.75mm,  %6
    1.25mm,  %7
    1.5mm,   %8
    1.5mm,   %9
    1.25mm,  %10
    0.75mm,  %11
    1.0mm,   %12
    1.0mm,   %13
    0.75mm,  %14
    1.25mm,  %15
    1.5mm]   %16
```

For each page in a section the shift amount must be indicated. The above presented list has exaggerated values just for making clear what happens:

While arranging these lists can be used in the following way:

Only one list is used:

```
\setuppageshift[paper][Hor]
```

or

```
\setuppageshift[paper][Vert]
```

Both lists are used:

```
\setuppageshift[paper][Hor][Vert]
```

The next examples show the cooperation of the commands `\setuppapersize`, `\setuplayout` and `\setuparranging`.

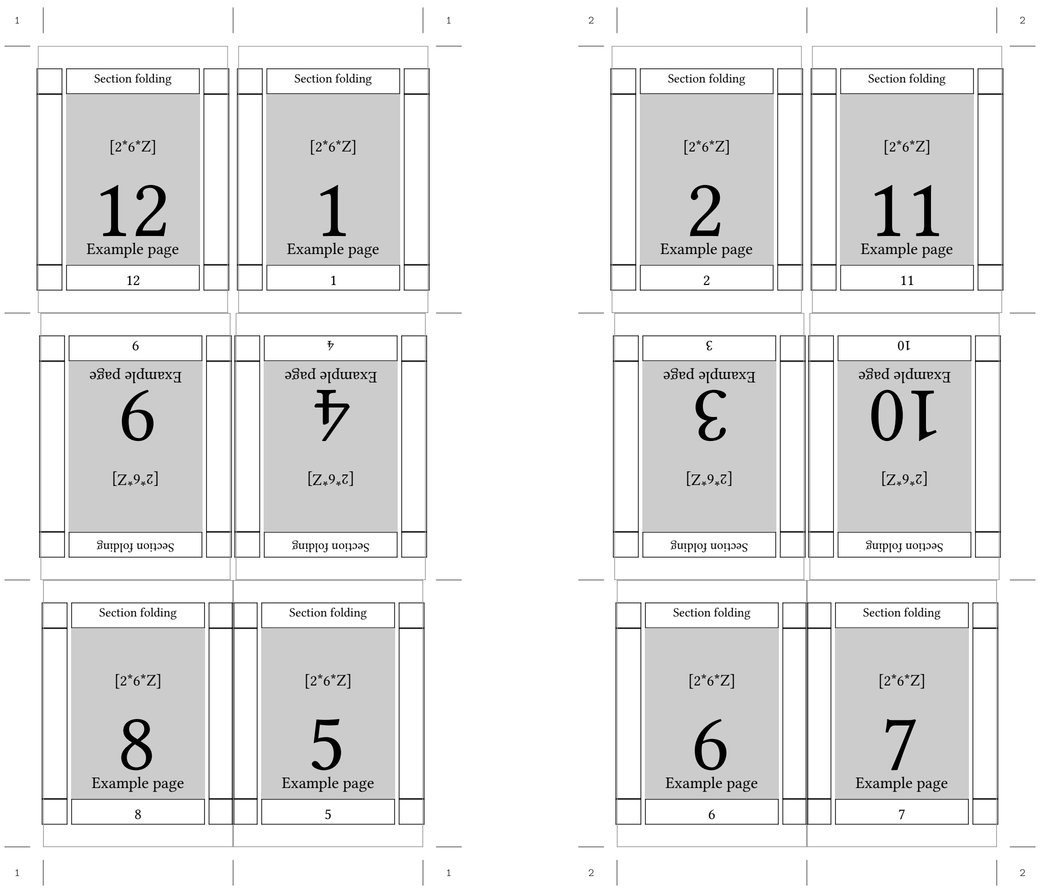


**Figure 3.29** Horizontal page-shift

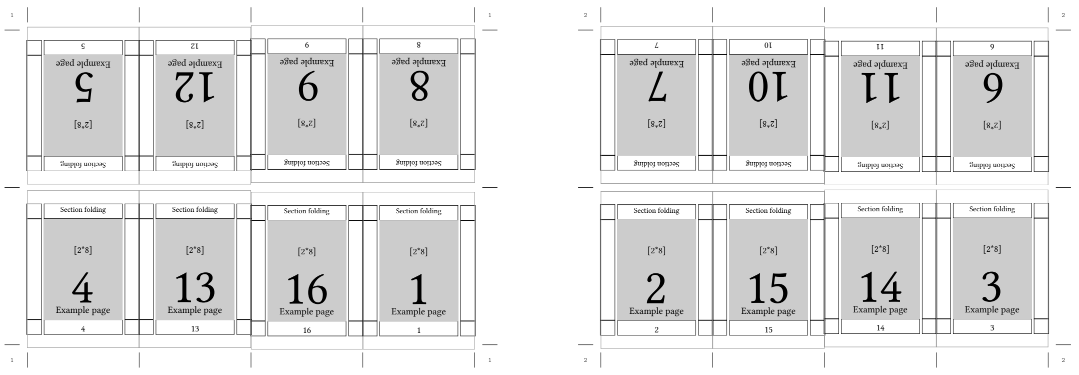


**Figure 3.30** Vertical page-shift

```
\setuppapersize    [A7][A3,mirrored] %negative creates an out of memory
error in Acrobat 8.2.2. on the MAc OSX 10.6.3
```

```
\setuparranging     [2*8,rotated,doublesided]
\setuppagenumbering [alternative=doublesided]
```

With the above shown preamble you get sections of 16 pages of the size of A7, where both sides of the A3 paper carry 8 pages [2*8]. For two reasons the A7 pages must be rotated on the paper. First in this imposition scheme there will be 4 A7 pages next to each other so they need to be aligned along the long edge of the A3. Secondly and this is important for book-printing, the grain direction of the paper must be in the direction of the spine i.e. in the height of the A7. Since A3 has its grain direction normally along the short edge it is correct to rotate the A7 pages. Further more there is the 'doublesided' directive in the `\setuparranging` command. This is to rotate the whole content of the verso side of the A3 paper by 180°in order to enable automatic double sided printing on the printing machine. `\setuppagenumbering` tells ConTEXt to use a doublesided lay-out, resulting in left and right pages.

Yet there is inside the `\setuppapersize` command the directive 'mirrored'. Using this directive, the content of the A3 paper is mirrored along the long edge of the paper, this results in mirrored typeset text.

```
\setuppapersize     [A5][A3]
\setuparranging     [2UP,rotated,doublesided]
\setuppagenumbering [alternative=doublesided]
```

What this does is placing two A5 pages side by side on a A3 sheet of paper. Both the page and the paper are in portrait orientation. Because A5 fits better on a A3 when the page is rotated the `\setuparranging` command carries the 'rotated' directive. The resulting sheet of paper will be printed on an automatic double-sided printing machine. Often these machines require, that the verso side of the paper is printed reversed, this is achieved with 'doublesided' in the `\setuparranging` command.

Instead of using the 'rotated' directive in `\setuparanging` you can also say:

```
\setuppapersize     [A7][A3,landscape]
\setuparranging     [2*8,doublesided]
\setuppagenumbering [alternative=doublesided]
```

You rotate the A3 paper by means of the 'rotated' directive in `\setuppapersize`.

There is one thing which should be kept in mind when using `\setuparranging`: TEX compilations with ConTEXt are most of the time multi-pass runs. If there is a table of content or other lists, this information is stored in auxiliary files or tables in LuaTEX. In order not to loose the content of those lists it is important to run the file first without the `\setuparranging` command enabled. If all went well, run the file a single time with the `\setuparranging` command enabled.

## 3.8 Logo types

Logos were removed in mkiv.

# 4 Layout

**TODO:** Split this chapter, it is much too large even in it's current incomplete state

## 4.1 Introduction

The look of a publication is determined by the page design, the chosen fonts and other aspects like vertical spacing. In this chapter we will explore the latter. Sometimes we will go into detail but a novice user can skip such parts. In normal applications, the default setups are most adequate, because they will adapt to the different situations. For the impatient reader we will just mention a few setups. Spacing between paragraphs is defined by:

```
\setupwhitespace[big]
```

In your source file you can best use an empty line between paragraphs. This increases readability and it makes the typing of `\par` at the end of each paragraph obsolete. Indentation at every new paragraph is obtained by:

```
\setupindenting[medium]
```

A doublesided publication is generated when you type:

```
\setuppagenumbering[alternative=doublesided]
```

As you might expect this might generate page numbering on the right and left hand side of a paper and the margins will be mirrored automatically.

As we have said before only the curious have to read on.

## 4.2 Paragraphs

The most important unit in TEX is paragraph. A new paragraph is forced by:

1. an empty line
2. the TEX–command`\par` or `\endgraf`
3. the ConTEXt–command `\paragraph`

The first alternative is the most obvious. You will obtain a readable input file (ascii file) and errors are minimized. The second alternative is chosen when it is mandatory to the used command. For example in definitions (see 13.2).

## 4.3 Indentation

When a text has little whitespacing, for example in a novel, it is a custom to indent each new paragraph. Indentation is setup with:

```
\setupindenting [...,...]
                     *
*   never none not no yes always first next small medium big normal odd even DIMENSION
```

By default there is 'no' indentation. When indentation is turned on, when possible the commands will determine whether indentation is necessary. For example, it doesn't look good to indent after a vertical whitespace. In a number of cases it is even undesirable to indent. Think for example of headers and itemizations.

This manual is typeset without indentation. The great quantity of short sentences and examples would result in a very messy page layout.

When indentation is used, we may have to tell TEX in some cases *not* to indent. This is done by:

```
\noindenting
```

We can set up indenting by:

```
\indenting [...,...]
                *
*   never none not no yes always first next small medium big normal odd even DIMENSION
```

The meaning of the setups is described in **table 4.1**. Next to the commands described above we could use the TEX–commands `\indent` and `\noindent`.

| **setup** | **result** |
|---|---|
| `no / not` | don't indent the next paragraph |
| `yes / always` | turn on indentation |
| `never` | turn off indentation |
| `first` | indent first paragraphs too |
| `next` | don't indent first paragraphs |

**Table 4.1** The way of indenting.

The settings `first` and `next` determine if paragraphs following whitespace should be indented or not. It is a sort of custom not to indent these.

A text may be typeset smaller than the default textwidth. In that case the complete text will be indented on both sides.

```
\startnarrower [...,...] ... \stopnarrower
                   *
               OPTIONAL
*   left middle right
```

For example:

```
\startnarrower[3*left,2*right]
The relatively small revolution in in Russia in 1917 had big consequences
for
```

```
this country as well as the rest of the world. It is interesting to see
that
some 80~years later a just as small revolution was needed to undo the 1917
one. In both cases, the main reason for the revolutions was to prevent
democracy from arising.
\stopnarrower
```

Will become:

> The relatively small revolution in in Russia in 1917 had big consequences for this country as well as the rest of the world. It is interesting to see that some 80 years later a just as small revolution was needed to undo the 1917 one. In both cases, the main reason for the revolutions was to prevent democracy from arising.

Next to using `left`, `right` and `middle` also combinations and manifolds are possible. Indentation in the example above could have obtained by typing `2*middle,left`. So, `middle` is equivalent to `left,right`.

The value of indentation is set up by:

```
\setupnarrower [..,.*=.,..]

*   left    =  DIMENSION
    right   =  DIMENSION
    middle  =  DIMENSION
```

## 4.4 Vertical spacing (whitespacing)

Vertical spacing between paragraphs is set up by:

```
\setupwhitespace [.*.]
                 OPTIONAL
*   none small medium big line fixed fix DIMENSION
```

Instead of a random value it is better to use one of the pre defined dimension. Default there is no vertical spacing. Without any set up values the vertical spacing is related to the actual fontsize.

Vertical spacing can be forced by either:

```
\whitespace
```

```
\nowhitespace
```

These commands have only effect when vertical spacing is set up. In fact these commands will not be necessary for ConTEXt takes care of most situations.

TEX handles vertical spacing around lines quite different from that around text. In case these problematic situations occur one can use the following commands. Spacing around figures and tables is dealt with by ConTEXt, so only use these commands when the typeset text looks really bad.

```
\startlinecorrection ... \stoplinecorrection
```

For example:

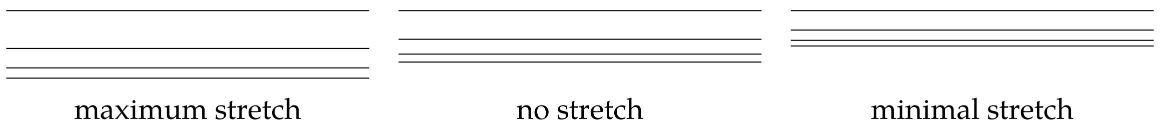
```
\startlinecorrection
\framed{To boxit or not, that's a delicate question!}
\stoplinecorrection
```

One can add vertical spacing with the TEX command `\vskip`, but please don't. We advise you to use:

```
\blank [...,...]
            *
        OPTIONAL
*   small medium big nowhite back white disable force reset line halfline FORMULA fixed
    flexible none always outer joinedup
```

We can use a value of one of the keywords `small`, `medium` or `big`. A big jump is twice a medium jump which is four times a small jump. A value however can be left out (`\blank`) when the default vertical space is desired. It is advisable to set up the vertical spacing only once in the setup area of your document. Local alterations throughout your document will result in a badly–spaced document.

Normally there is some stretch in the vertical spacing. This enables TEX to fill out a page optimally. In the next example we see what happens when we add stretch to whitespace. Each sample shows from top to bottom three `\blank`'s of `big`, `medium` and `small`. The left and right sample show the range of the stretch. The rightmost sample shows that adding stretch can result in shrink.



The last vertical space can be undone by typing `\blank[back]` and the next blank can be blocked by `disable`. With `reset` a `disable` is ignored.

The command `\blank` is one of the more advanced commands. The next call is allowed:

```
\blank[2*big,medium,disable]
```

Since `medium` is half the amount of `big`, this results in adding a vertical spaces of 2.5 times `big`. The previous vertical space will be undone automatically and the `disable` suppressed the next `\blank`.

A lasting vertical space can be sustained by `force`. For example, if you want some extra spacing at the top of a page you will have to type `force`.

The default vertical spaces are set up with:

```
\setupblank [.*.]
                OPTIONAL
*   normal default standard line halfline DIMENSION big medium small fixed flexible global
    unknown
```

An example of such a definition is:

```
\setupblank[big]
```

The vertical spaces will be automatically adapted to the fontsize and they are flexible. Changing the default set up locally is therefore not advisable. Without an argument `\setupblank` adapts to the actual fontsize!

The keywords `fixed` and `flexible` are used to end or reinstate this adaptive characteristic. In columns it is recommended to use the setup `[fixed,line]` or the opposite setup `[flexible,standard]`.

This text is typeset a bodyfont of 11 pt and is downscaled by a few percent. The setup that is used in this document is shown in **table 4.2**. We see some stretch in the vertical spacing. The stretching enables TEX to fill out a page satisfactorily. Default the maximal vertical space is 75% of the line space and the stretch maximal of 25%.

| **setup** | **value** |
|---|---|
| `small` | 2.59297pt plus 0.86432pt minus 0.86432pt |
| `medium` | 5.18594pt plus 1.72864pt minus 1.72864pt |
| `big` | 10.37189pt plus 3.45729pt minus 3.45729pt |
| `line` | 13.8292pt |

**Table 4.2** The whitespace values to a 11 pt bodyfont.

In paragraph ?? it was said that the vertical spacing can be set up with the command `\setupwhitespace`. Default there is no whitespace between paragraphs. The setup of vertical spacing and line spacing are related to each other.

Instead of direct setup you can use an indirect way. This has the advantage that you can change the layout more easily. In that case we use:

```
\defineblank [.1.] [.2.]

1   IDENTIFIER

2   inherits from \setupblank
```

If we type for example:

```
\defineblank[aroundverbatim][medium]
```

than `aroundverbatim` is equal to `medium`, which can be used, for example around verbatim, as in:

```
\setuptyping
  [before={\blank[aroundverbatim]},
        after={\blank[aroundverbatim]}]
```

If we want some more whitespacing we only have to change the definition of `aroundverbatim`:

```
    \defineblank[aroundverbatim][big]
```

The vertical spacing between two lines can be suppressed with the command:

```
\packed
```

Vertical spacing between more than one line is suppressed by:

```
\startpacked [.*.] ... \stoppacked
              OPTIONAL
*   blank
```

The spacing around 'packed' text is automatically corrected. Opposed to this command is:

```
\startunpacked ... \stopunpacked
```

Skipping more than one vertical space is done with:

```
\godown [.*.]

*   DIMENSION
```

One of the most important lessons to be learned is to avoid using `\vskip` in running text. This can interfere with some hidden mechanisms of ConTEXt.

Sometimes TEX is not able to sort out spacing on its own. In such situations one can insert the next command at the troublesome location.

```
\correctwhitespace {.*.}

*   CONTENT
```

Normally one will not need this command, although sometimes when writing macros, it can be added to make sure that the spacing is okay. Use this kind of tweaking with care!

## 4.5 Word spacing

Default a space is placed after a period that ends a sentence. In some countries it is custom to stretch the space after a period. Especially documents typeset in small columns will look better that way. Because this is a language specific feature. the default depends on the language. One can however (temporarily) change this spacing.

```
\setupspacing [.*.]

*   broad packed
```

In many cases we combine words and numbers that should not be separated at linebreaking, for example number 12. These combinations can be connected by a tight space: `number~12`. Word and number will never be separated at linebreaking on that spot. A space can be made visible by:

```
\space
```

Undesired spaces can be suppressed by:

```
\nospace
```

When you want to align a row of numbers you can use tight spaces with the width of a number. Tight spaces are activated by:

```
\fixedspaces
```

After this command the ~ (tilde) generates a tight space with the width of a number.

## 4.6 Struts

A strut is a little invisible block without width but with the maximal height and depth of a character or line. If you want to force these maximal dimensions, for example when you are using boxes in your own commands, than you can use the command `\strut`:

```
\hbox{\strut test}
```

If we leave out the strut in this example the box has no depth. The characters in the word test don't reach under the baseline. Compare for example test (with strut) with test.

Many commands use struts automatically. If for some reason you don't want struts you can try to suppress them by `\setnostrut`. However take care that this command works only locally. A strut can be set by `\setstrut`.

The struts that are used by ConTEXt can be made visible with the command:

```
\showstruts
```

## 4.7 Text in the margin

**FIXME:** The syntax of the margin commands has changed in mark IV.

Texts can be placed in the margins with:

```
\inmargin [.1..] [.2..] {.3..}
           OPTIONAL OPTIONAL
1   + - low
2   REFERENCE
3   CONTENT
```

A new line in a margin text is forced with \\. An example of a margin text is:

```
\inmargin{the marginal\\influence of\\advertisement}It would be great
if the recent reduction in washing powder needed to get your wash
perfectly clean had resulted in an equal reduction of time needed to
advertise this kind of products.
```

or:

**the marginal influence of adver- tisement** It would be great if the recent reduction in washing powder needed to get your wash perfectly clean had resulted in an equal reduction of time needed to advertise this kind of products.

**over here** When this command is used in the middle of a paragraph the margin text will appear on the same line in the margin. The command `\inmargin` puts the text in the left or right margin. The location where the text will show up depends on the character of the document: single–sided or double–sided. You can also force the text into a specific margin, using:

```
\inleft [.1..] [.2..] {.3..}
         OPTIONAL OPTIONAL
1   + - low
2   REFERENCE
3   CONTENT
```

```
\inright [.1..] [.2..] {.3..}
          OPTIONAL OPTIONAL
1   + - low
2   REFERENCE
3   CONTENT
```

There is also:

```
\inothermargin [.1..] [.2..] {.3..}
                OPTIONAL OPTIONAL
1   + - low
2   REFERENCE
3   CONTENT
```

Some examples of the use of margin text appear below:

```
\startlines
\inleft{to be}\quotation{To be or not to be} to me
\inright{or not}is rather famous english
```

```
\inmargin{to be}And just as it is meant to be
that quote will never perish
\stoplines
```

This will become:

**to be** "To be or not to be" to me
is rather famous english **or not**
**to be** And just as it is meant to be
that quote will never perish

**123** The mechanism of margin texts is rather complex. If you think of multiline margin texts and the alignment of these lines with the lines in the textbody you can imagine a few typographic problems. The number 123 next to this paragraph is not aligned but is typeset somewhat lower. This is done by adding the keyword `low`:

```
\inmargin[low]{\ssd 123}The mechanism of margin texts ...
```

It is possible to set up the way margin texts are typeset by means of the command:

**a rather marginal effect** With `align` we define the left or right alignment of the margin text. Default margin texts are right aligned. In this example alignment is `middle`.

We can also align on the left or right side automatically. In a double sided document design optimisation of the margin text may ask for more than one processing step. In the example below you see some of the possible setups.

**left** This is `left` aligned
**middle** but this goes in the `middle`. Don't forget that
**right** `right` in this sense, align means a ragged right margin.
**yes** Just to be complete, there is `yes`
**no** and `no`.
**inner** The outsiders `inner` and
**outer** `outer` adapt themselvs to a doublesided design.

The left and right margin can be set up separately by adding `[left]` or `[right]` as the first argument.

**that way we can nove quite some text into the margin** With `before` and `after` we can influence margin texts. Bij default the same line spacing is used as in the textbody. But when a narrower fontsize is used we can also adapt the interline spacing. For example:

```
\setupmargindata
  [style=\bfx\setupinterlinespace]
```

Page breaking and margin text are in conflict with each other. The reason is that TEX first typesets a complete page in order to be able to determine the right spot for page breaking. However the margin text is already typeset at that moment. In a next processing stage the margin texts are typeset correctly. If you want to force margin texts in a margin you can type `\inmargin[+]`.

The next command can be compared with the command like `\section`. Before the command is placed in the margin TEX looks if it can be placed on the actual page. If not, it is moved to the following page.

```
\margintext [.1.] [.2.] {.3.}
             OPTIONAL OPTIONAL
1   + - low
2   REFERENCE
3   CONTENT
```

The layout of your ascii–file will not interfere with the function of this command. This may seem obvious, but TEX programmers know that it is not the case. For example even commands that take care of index entries can be typed close to the margin texts.

The layout of your ascii–file will not interfere with the function of this command. You might not expect it to, but TEX programmers know that with TEX, the layout of the source usually interferes with for instance margin texts and index entries. In ConTEXt commands that take care of margin texts take care of this situation, so that index entries can be typed close to the margin texts and margin texts can be separated from the next paragraph by an empty line. The same cannot be said for other TEX macropackages.

```
\margintext{text in themargin}
\index{margintexts}

After experimenting a long time I have succeeded to filter
empty lines and commands that stand between body texts and
margin texts. It is amazing but the index entry really works.
```

Because of the close relation with the page design the margin width is set up by means of: `\setuplayout` (see **section 3.4**).

**Isn't this cute?** The command `\margintext` enables you to put texts in the margin that show completely different characteristics than that of the text body. You can typeset different margin texts with different characteristics like bodyfont, line spacing and offset.

```
\margintext{Isn't}
\margintext{this}
\margintext{cute?}
```

In the setup we see an optional argument. The number is determined by the order of definition.

```
\setupmargindata[1][align=right, line=1,style=slanted]
\setupmargindata[2][align=middle,line=2,style=boldslanted]
\setupmargindata[3][align=left,  line=3,style=bold]
```

This means that the second margintext in a row will start on line 2, and be typeset in a bold slanted font. One can explicitly force a margintext to go some place, by saying for instance:

```
\margintext[2]{this is the second one}
```

## 4.8 Subscript and superscript

There are three commands to create superscript and subscript outside the math mode:

```
\high {.*.}

*   CONTENT
```

```
\low {.*.}

*   CONTENT
```

```
\lohi [.1.] {.2.} {.3.}
       OPTIONAL
1   low
2   CONTENT
3   CONTENT
```

The next example illustrates the use of these commands:

```
You can walk on \high {high} heels or \low {low} heels but your height
is still the same.
```

This results in:

You can walk on $^{\text{high}}$ heels or $_{\text{low}}$ heels but your height is still the same.

These commands relate to the ^ and _ in math mode. In case of larger fontsizes like `\tfc`, the ^ and _ will not create the desired output. Compare the examples below:

```
test\high{test} test test$^{\rm test}$ test
{\bf test\high{test} test test$^{\bf test}$ test}
{\tfb test\high{test} test test$^{\tfb test}$ test}
```

This becomes:

test$^{\text{test}}$ test test$^{test}$ test
**test$^{\textbf{test}}$ test test$^{test}$ test**
test$^{\text{test}}$ test test$^{test}$ test

## 4.9 Columns

The TEX programmer knows that it is not easy to put text in columns. Gratefully a ConTEXt user is not bothered with the implementation of extensive macros.

You can typeset text in columns. Most commands can be used in a normal way without any problems. The floating object like tables or figures are somewhat limited. This is caused by the fact that TEX has limited capabilities for typesetting columns. For insiders: columns are produced with the primitives: `\output` and `\vsplit`.

The number of columns is unlimited, however TEXs memory can only handle upto about twenty to thirty or fourty columns.

The number of columns and the type setting of a vertical line as a column separator is set up by:

```
\setupcolumns [..,..=..,..]
                     *
                  OPTIONAL
*   n          =  NUMBER
    ntop       =  NUMBER
    rule       =  on off
    height     =  DIMENSION
    tolerance  =  verystrict strict tolerant verytolerant stretch
    distance   =  DIMENSION
    balance    =  yes no
    align      =  text inner outer left right flushleft flushright middle center normal no
                  yes
    blank      =  fixed halfline line flexible big medium small
    option     =  background
    direction  =  left right
    inherits from \setupframed
```

The n indicates the number of columns. The column text is enclosed by:

```
\startcolumns [..,..=..,..] ... \stopcolumns
                     *
                  OPTIONAL
*   inherits from \setupcolumns
```

The local setup of columns can be added directly after this command. A new column is forced by:

```
\column
```

The text below is typeset in two columns with a verytolerant alignment.

```
\startcolumns[rule=on,n=2,tolerance=verytolerant]
Thus, I came to the conclusion that the designer of a new
system must not only be the implementer and first
.
.
\bf D.E. Knuth
\stopcolumns
```

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.

The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.

But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

**D.E. Knuth**

This example makes it painfully obvious that spacing between lines is not on forehand equal. By default the line spacing in this document is big, which equals .75×\lineheight. Furthermore, the allowable stretch in line spacing makes vertical alignment practically impossible.

For this reason the default line spacing is equal to the lineskip and stretching is not allowed. When a switch in fontsize is desirable you should do so before starting the column mechanism. Font switches within columns will have a poor result. The next example shows a line spacing equal to the lineskip.

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.

The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.

But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

**D.E. Knuth**

This effect is reached by the (default) setup:

```
\setupcolumns[blank={fixed,line}]
```

In **section 3.5** typesetting on a grid is explained. This mechanism works quite well within columns.

TEX is not an easy to learn typesetting system or program. The problem is that “knowing everything is possible” leads to “wanting everything that is possible”. However using ConTEXt or TEX takes considerable learning time. And it is not feasible to explain every single detail in this manual. Therefore “doing” is the answer.

This text shows that one can do some tricks with columns. The frame is created by:

```
\unexpanded\def\FramedColumn#1{\ruledhbox{\box#1}}
```

```
\setupcolumns[command=\FramedColumn]
```

A less senseless display is:

```
\def\FramedColumn#1%
  {\hbox to \hsize
     {\ifodd\currentcolumn\unhbox\hss#1\else\unhbox#1\hss\fi}}
```

This time the columns will look like:

TEX is not an easy to learn typesetting system or program. The problem is that “knowing everything is possible” leads to “wanting everything that is possible”. However using ConTEXt or TEX takes considerable learning time. And it is not feasible to explain every single detail in this manual. Therefore “doing” is the answer.

A column can be manipulated as a whole. For example to create a background:

```
\setupfootnotes
  [location=columns,
    background=color,
    backgroundcolor=white]
\setuplayout
  [grid=yes]
```

This time the column will be typeset on a grid:

TEX is not an easy to learn typesetting system or program. The problem is that “knowing everything is possible” leads to “wanting everything that is possible”. However using ConTEXt or TEX takes considerable learning time. And it is not feasible to explain every single detail in this manual. Therefore “doing” is the answer.

## 4.10 Paragraphs in columns

In some cases you want to typeset a paragraph in columns. For example in a definition where you have a first column containing meaningful text and a second column containing meaningful text. In these cases you can use:

```
\defineparagraphs [.1.] [..,.=2.,..]

1   IDENTIFIER

2   n           =  NUMBER
    rule        =  on off
    height      =  fit DIMENSION
    before      =  COMMAND
    after       =  COMMAND
    inner       =  COMMAND
    distance    =  DIMENSION
    tolerance   =  verystrict strict tolerant verytolerant stretch
    align       =  inner outer left right flushleft flushright middle center normal no yes
```

This command defines a column layout that is recalled by its name.

The layout can be set up by:

```
\setupparagraphs [.1.] [.2.] [..,.=3.,..]
                       OPTIONAL
1   IDENTIFIER

2   NUMBER each

3   style       =  normal bold slanted boldslanted type cap small... COMMAND
    width       =  DIMENSION
    height      =  DIMENSION
    align       =  inner outer left right flushleft flushright middle center normal no yes
    tolerance   =  verystrict strict tolerant verytolerant stretch
    distance    =  DIMENSION
    before      =  COMMAND
    after       =  COMMAND
    inner       =  COMMAND
    command     =  COMMAND
    rule        =  on off
```

The width of non–specified columns is determined automatically. Distance relates to horizontal white space in front of a column. The next column is specified by:

We show a simple example of the use of paragraphs in columns.

```
\defineparagraphs[TwoColumns][n=2]
\setupparagraphs[TwoColumns][1][width=5cm]

\startTwoColumns
  This is the top left corner.
\TwoColumns
  In graphic environments the top right corner is also called the upper
  right corner.
\stopTwoColumns

\startTwoColumns
  In a similar way, the bottom left corner is called the lower left corner.
\TwoColumns
  Which leaves the bottom right corner, that is also known as lower right
  corner. Now what is the alternative name for the top left corner?
\stopTwoColumns
```

Here the `\TwoColumns` separates the columns. With a default setup this results in:

| | |
|---|---|
| This is the top left corner. | In graphic environments the top right corner is also called the upper right corner. |
| In a similar way, the bottom left corner is called the lower left corner. | Which leaves the bottom right corner, that is also known as lower right corner. Now what is the alternative name for the top left corner? |

We also could have used `\nextTwoColumns` instead of `\TwoColumns`. Sometimes this is more readable in your ascii text. An alternative specification is:

```
\TwoColumns first text \\ second text \\
```

You can add a command to the keywords `bottom` and `top`. These commands will be executed before or after the text. For example a column can be forced down by `[top=\vfill]`.

This is the right place to show a more complex example. The use of `paragraphs` is preferred over the use of columns because the text is kept together. If we want to score an item on two dimensions we need three columns:

```
\defineparagraphs [CombinedItem]     [n=3,rule=on]
\setupparagraphs  [CombinedItem] [2] [width=3em]
\setupparagraphs  [CombinedItem] [3] [width=7em]
```

The item itself is defined with `\defineenumeration` (see section ??):

```
\defineenumeration
  [SomeItem]
  [location=left,text=,width=3em,before=,after=]
```

The scoring is done on a scale that is typeset as an itemization (see section ??). An item might look like this in ascii:

```
\startCombinedItem
```

```
  \startSomeItem
    The student is able to write a detailed planning for the
    design and construction of a water purification plant.
  \stopSomeItem
\nextCombinedItem
  \startitemize[5,packed]
    \item yes \item no
  \stopitemize
\nextCombinedItem
  \startitemize[5,packed]
    \item self study \item class room \item simulation
  \stopitemize
\stopCombinedItem
```

And will result in:

| | | |
|---|---|---|
| **1**<br><br>The student is able to write a detailed planning for the design and construction of a water purification plant. | ◦ yes<br>◦ no | ◦ self study<br>◦ class room<br>◦ simulation |

When the scoring scales are identical over all items we can use macros:

```
\def\firstscale%
  {\startitemize[5,packed]
     \item yes \item no
   \stopitemize}

\def\secondscale%
  {\startitemize[5,packed]
   \item self study \item class room \item simulation
   \stopitemize}

\startCombinedItem
  \startSomeItem
    The student is able to write a detailed planning for the
    design and construction of a water purification plant.
  \stopSomeItem
\nextCombinedItem
  \firstscale
\nextCombinedItem
  \secondscale
\stopCombinedItem
```

Or even more sophisticated:

```
\def\startItem%
  {\startCombinedItem
   \startSomeItem}

\def\stopItem%
  {\stopSomeItem
   \nextCombinedItem \firstscale
```

```
      \nextCombinedItem \secondscale
      \stopCombinedItem}
   \startItem
     The student is able to write a detailed planning for the
     design and construction of a water purification plant.
   \stopItem
```

A definition like the one above can be very surprising. The commands in such a definition can interfere and result in undesirable output. We think of `\vtop`'s that align on the baseline and `\vbox` s that align under the baseline. Another example with framed texts show that ConTEXt takes care of most of the problems.

| | | |
|---|---|---|
| left | middle | right |

## 4.11 Tabulate

In a later chapter we will go into detail on typesetting tables. Consider this paragraph to be an appetizer. We use the term tabulate when a table is part of the running text. A simple tabulation looks like this:

```
\starttabulate[|l|p|]
\NC question \NC Sometimes it is surprising to notice that writers,
independently of each other, explore the same theme along similar lines.
Three of the four books mentioned here fall into this category. Which
books do not belong in this list? \NC \NR
\stoptabulate

\starttabulate[|l|l|l|]
\NC A. \NC This Perfect Day          \NC Ira Levin           \NC \NR
\NC B. \NC Opstaan op Zaterdag       \NC Jan Gerhart Toonder \NC \NR
\NC C. \NC Tot waar zal ik je brengen \NC Anton Koolhaas      \NC \NR
\NC D. \NC The City And The Stars     \NC Arthur Clarke       \NC \NR
\stoptabulate
```

This results in:

| | |
|---|---|
| question | Sometimes it is surprising to notice that writers, independently of each other, explore the same theme along similar lines. Three of the four books mentioned here fall into this category. Which books do not belong in this list? |

| | | |
|---|---|---|
| A. | This Perfect Day | Ira Levin |
| B. | Opstaan op Zaterdag | Jan Gerhart Toonder |
| C. | Tot waar zal ik je brengen | Anton Koolhaas |
| D. | The City And The Stars | Arthur Clarke |

With `\NC` we go to the next column and with `\NR` to the next row. Definitions like `[|l|p|]` and `[|l|l|l|]` are called a template. The set ups are similar to those of `\starttable` (see in ??).

The default template looks like this: `[|l|p|]`. The second column is typeset as a normal paragraph and with a width that is calculated automatically by TEX.

```
\starttabulate
\NC d: \NC avond, afond, avend, afend \NC \NR
\NC t: \NC avont, afont, avent, afent \NC \NR
\stoptabulate
```

This quotation from "Spellingsverandering van zin naar onzin" by G.C. Molewijk (1992) will look like this:[4]

| | |
|---|---|
| d: | avond, afond, avend, afend |
| t: | avont, afont, avent, afent |

## 4.12 Alignment

Horizontal and vertical alignment is set up by:

```
\setupalign [...,...]
                 *

*   width left right middle inner outer wide broad height bottom line reset hanging
    nothanging hyphenated nothyphenated lesshyphenation morehyphenation new old normal yes
    no flushleft flushright flushouter flushinner center hz nohz spacing nospacing tolerant
    verytolerant stretch
```

The keys `left`, `middle` and `right`, `inner` and `outer` apply to horizontal alignment and `bottom`, `height` and `line` to vertical alignment.

The key `right` results in the text being typeset ragged right. The keyword `broad` can be combined with `left`, `middle` and `right` which results in somewhat more rough alignments.

The option `line` lets the last line touch the bottom of the page while `height` aligns the baseline to the bottom.

Individual lines can be aligned with the commands:

```
\leftaligned {...}
               *

*   CONTENT
```

```
\midaligned {...}
              *

*   CONTENT
```

```
\rightaligned {...}
                *

*   CONTENT
```

[4] For the non–dutch readers: this book "Change of spelling, from sense to nonsense" is one of the most humorous books on the developments in a language one can imagine. If you ever come to studying dutch, you should give this book a try.

alignment over a number of lines is done by:

```
\startalignment [...,...] ... \stopalignment
                    *
                 OPTIONAL
*   inherits from \setupalign
```

The text below shows a number of examples of horizontal alignment.

The Brittish stubbornly stick to
driving at the left side of the road.

This can be considered a form conservatism,
or alternatively phrased: right–wing thinking.

However, a political drive–in–the–middle
compromise would definitely lead to accidents.

We done this with:

```
\leftaligned{The Brittish stubbornly stick to}
\leftaligned{driving at the left side of the road.}
\blank[medium]
\rightaligned{This can be considered a form conservatism,}
\rightaligned{or alternatively phrased: right||wing thinking.}
\blank[medium]
\midaligned{However, a political drive||in||the||middle}
\midaligned{compromise would definitely lead to accidents.}
```

The last words of a paragraph can be placed on the right hand side by the command `\wordright`, **so with:**

```
\wordright {...}
             *
*   CONTENT
```

When typesetting a paragraph, TEX tries several alternatives and decides which one to choose based on a system, of penalties. Normally TEX is very strict, but we can instruct TEX to be a bit more tolerant. This means that, instead of letting problematic situations remain unsolved —i.e. let words that cannot be hyphenated stick into the margin— TEX will add a bit more stretch and apply different penalties for successive hyphens.

Alignment can be set up by:

```
\setuptolerance [...,...]
                    *
*   horizontal vertical stretch space verystrict strict tolerant verytolerant
```

By default we use `[horizontal,verystrict]` for horizontal alignment and `[vertical,strict]` for vertical alignment.[5] A last resort is provided by the keyword `stretch`, which in unsolvable situations will stretch spaces, extending the ugliness even further.

[5] If you want a real ugly result, you should set the TEX variable `\pretolerance` to 10.000. It is up to you.

In double sided typesetting, alignment can be coupled to the left or right pages.

```
\startalignment[inner]
\quotation {Out of nowhere} is a rather normal way of saying that it is
not clear where something originates. It is typically a phrase that has
no counterpart, in the sense that nobody would comprehend the remark
\quotation {Into somewhere}.
\stopalignment

\startalignment[outer]
\quotation {Out of bounds} is a similar quote. There is no counterpart
\quotation {In of bounds}. Both examples demonstrate that in(ner) and
out(er) are not always counterparts.
\stopalignment
```

Results of the commands above depend on the location of the page (left or right). The commands lead to:

“Out of nowhere” is a rather normal way of saying that it is not clear where something originates. It is typically a phrase that has no counterpart, in the sense that nobody would comprehend the remark “Into somewhere”.

“Out of bounds” is a similar quote. There is no counterpart “In of bounds”. Both examples demonstrate that in(ner) and out(er) are not always counterparts.

## 4.13 New lines

A new line is forced by:[6]

```
\crlf
```

If you want to have lines show up the way you typed them in your source file you can use:

```
\startlines ... \stoplines
```

Default indenting is off. You can set up lines by:

```
\setuplines [..,..=..,..]

*   before     =  COMMAND
    after      =  COMMAND
    inbetween  =  COMMAND
    indenting  =  never none not no yes always first next small medium big normal odd even
                  DIMENSION
    space      =  yes no
```

If we set up `indenting=odd` for example we will obtain:

Come on, he said, give me a while,

[6] In titles, headers and margin texts \\ is available for introducing a new line.

and I will typeset you this text
with rivers like the river Nile

This was typed in the source file as:

```
\setupindenting[medium]
\setuplines[indenting=even]
\startlines
Come on, he said, give me a while,
and I will typeset you this text
with rivers like the river Nile
\stoplines
```

Lines can be numbered with:

```
\startlinenumbering [.*.] ... \stoplinenumbering

*   continue
```

A simple example of numbered lines might look like this:

```
\startlinenumbering
There is of course no problem with trying to prevent illegal copying of
\cap {cd}'s and records. However, why should artists benefit from these
measures, who themselves have no problems with copying themes, lyrics
and melodies?
\stoplinenumbering
```

this becomes:

1 There is of course no problem with trying to prevent illegal copying of CD's and records. How-
2 ever, why should artists benefit from these measures, who themselves have no problems with
3 copying themes, lyrics and melodies?

We can influence line numbering by:

```
\setuplinenumbering [..,.=*.,..]

*   conversion   =  numbers characters Characters romannumerals Romannumerals TEXT
    start        =  NUMBER
    step         =  NUMBER
    width        =  DIMENSION
    location     =  intext inmargin
    style        =  normal bold slanted boldslanted type cap small... COMMAND
    prefix       =  TEXT
    referencing  =  on off
```

With the variable `conversion` you set up the type of numbering. You may even use your own character, for example an em–dash (keyed in as `---`). In that case this character is set in front of each line.

In **chapter 12.5** we will explain how we can refer to a linenumber. The parameters `prefix` and `referencing` can be used to unfluence that proces.

In the example below we use the following setup:

```
\setuplinenumbering[conversion=numbers,step=2,location=intext]
```

and:

```
\setuplinenumbering[conversion=characters,step=1,location=intext]
```

a macro is a piece of text
random at first sight
a bunch of stupid tokens that
looks less that awful right

but when fed to TEX the program
you will be surprised
thanks to macros your text too
will look quite organized

You can also mark lines in order to refer to specific line numbers. This will be shown in in **chapter 12.5**.

## 4.14 New page

In some instances it is up to you to force, prevent or encourage a new page.

```
\page [...,...]
        *
*   yes makeup no preference bigpreference left right disable last quadruple even odd blank
    empty reset start stop
```

The possible set ups are explained in **table 4.3**. If no setup is used \page will result in a new page.

| **setup** | **result** |
|---|---|
| `yes` | force a new page |
| `makeup` | the same, without fill |
| `no` | when possible, avoid page break |
| `preference` | when possible, force page break |
| `bigpreference` | when possible, force page break, try harder |
| `left` | force a left page |
| `right` | force a right page |
| `disable` | ignore the next \page command |
| `last` | add last page(s) |
| `quadruple` | add pages until quadruple number of pages |
| `even` | go to the next even page |
| `odd` | go to the next odd page |
| `blank` | insert a completely blank page |
| `empty` | insert an empty page (with headers etc.) |
| `reset` | reset the disable command |

**Table 4.3** Setups of \page.

The setups `last` and `quadruple` can be used in double sided (reduced) typesetting. The first setup up will add pages until an even number is obtained, the second set up will add pages until the next quadruple is reached. When you want to overrule the automatic page numbering you type the pagenumber yourself:

```
\page[25]
```

You can also use a relative number like `[+4]`. You can use this feature when you want to be on the safe side and if you don't know at what page you are.

While generating empty pages you have to take doublesidedness into account, for example:

```
\page[right,empty,right]
```

## 4.15 Pagenumbers

At any location in the text the pagenumber can be set up with the command:

```
\setuppagenumber [..,.=.,..]
                      *

*   number  =  NUMBER
    state   =  start stop keep
```

The pagenumber position on the page is defined by:

```
\setuppagenumbering [..,.=.,..]
                         *

*   alternative       =  singlesided doublesided
    location          =  header footer left right middle margin marginedge inleft inright
    conversion        =  numbers characters Characters romannumerals Romannumerals
    style             =  normal bold slanted boldslanted type cap small... COMMAND
    left              =  TEXT
    right             =  TEXT
    way               =  bytext bycd:section
    text              =  TEXT
    numberseparator   =  TEXT
    textseparator     =  TEXT
    cd:sectionnumber  =  yes no
    separator         =  TEXT
    strut             =  yes no
    state             =  start stop
    width             =  DIMENSION
    command           =  \...#1
```

The position varies with the nature of the document. With `conversion` we state the way we want to display the number. With `location` we define pagenumber positions like the bottom or top, left or right side or in the margin. You can use combinations of these options. For example:

```
\setuppagenumbering[location={header,inmargin}]
```

| alternative=singlesided | alternative=doublesided |
|---|---|
| left, right | marginedge |
| middle | middle |
| margin | margin |

Another alternative is {singlesided,doublesided}. In this case headers and footers will be mirrored in a double–sided document. The backspace is not mirrored (see **figure 4.1**).

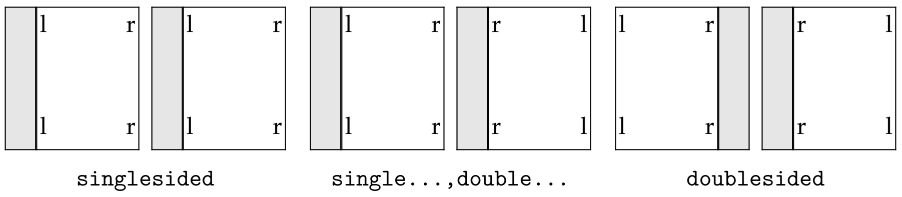


**Figure 4.1** Three ways to mirror.

You can assign text to the parameters left and right. These texts will encloses the pagenumber:

```
\setuppagenumbering[conversion=romannumerals,left={---},right={~--}]
```

This will lead to: – viii –. With style you define the font and with state pagenumbering is switched on and off.

Numbering can become very fancy when you use command to execute an operation. This command has an argument and will be executed every time a pagenumber is placed. A framed pagenumber can be obtained by:

```
\setuppagenumbering[command=\inframed]
```

or partially framed by:

```
\def\mypagenumber#1%
  {\inframed[frame=off,leftframe=on,rightframe=on]{#1}}

\setuppagenumbering[command=\mypagenumber]
```

In this we use \inframed instead of \framed, because the pagenumber must align with the texts of the headers and footers.

With textseparator you can define a separator between the section and pagenumber. Default this is a –. When the pagenumber is to appear at the margin the numberseparator is placed between the number and the footer text. Default this is a space with a width of 1em.

In interactive documents subpagenumbering is frequently used for hyperlinking. When every new section is started on a new page the footer text can be set up with:

```
\setupsubpagenumber
  [way=byparagraph]
\setupfootertexts
  [screen {\subpagenumber} of {\numberofsubpages}] []
```

The setup is done with:

```
\setupsubpagenumber [..,..=..,..]
                           *
*   way    =  bytext bycd:section
    state  =  start stop none
```

and the numbers themselves can be recalled by `\subpagenumber` and `\numberofsubpages`. These numbers are only reliable in headers and footers. In the case of interactive documents a more abstract definition can be used:

```
\setupfootertexts[][{\interactionbar[alternative=d]}]
```

In this case one can jump to the previous and following subpages. The subnumbering can be reset with `[reset]`.

In a similar fashion one has access to the page number and the total number of pages: `\pagenumber` and `\totalnumberofpages`.

## 4.16 Headers and footers

Text in the header and footer are set up with the commands:

```
\setupheadertexts [.1.] [.2.] [.3.]
                  OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

```
\setupfootertexts [.1.] [.2.] [.3.]
                  OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

A large number of setup arguments can be added. When the first setup is left out, the default value (`text`) specifies that the footer and header should be placed under or over the pagebody. The edge is located at the left side of the margin and is only used in interactive documents where an extended pagebody is needed.

The value `date` generates a date and `pagenumber` generates the pagenumber. Part, chapter and section titles can be summoned to appear in the header– and footer text by `part`, `chapter`, `paragraph` etc. By default the mark mechanism is active. Sectionnumbers can also be recalled: `chapternumber` etc.

Setting the `state` is done for the whole header, so one should use the one–argument version:

```
\setupheader[state=high]
```

Those who want more variations in headers and footers can use four instead of two arguments. Four arguments have only effect in double–sided documents.

```
\setupfootertexts
  [even left][even right]
  [odd left][odd right]
```

So there are different combinations of arguments possible:

```
\setupheadertexts
\setupheadertexts[mid text]
\setupheadertexts[left text][right text]
\setupheadertexts[left text][right text][left .][right .]
\setupheadertexts[location][left text][right text]
\setupheadertexts[location][left text][right text][left .][right .]
```

Instead of text, one can specify keywords like `chapter`, `date` or `pagenumber`. When the pagenumber is positioned in this way, one should also say:

```
\setuppagenumbering[location=]
```

The current setups of the headers and footers are cleared when no values are stated in `\setupfootertexts`. Problems can be expected when you use `[ ]` in your setup. These have to be enclosed in curly brackets:

```
\setupfootertexts[chapter][{\currentdate[month,year]}]
```

The type setting of head– and foot texts can be influenced by:

```
\setupheader [.1.] [..,.=2.,..]
              OPTIONAL
1   TEXT margin edge
2   state       =  normal stop start empty high none nomarking IDENTIFIER
    strut       =  yes no
    style       =  normal bold slanted boldslanted type cap small... COMMAND
    leftstyle   =  normal bold slanted boldslanted type cap small... COMMAND
    rightstyle  =  normal bold slanted boldslanted type cap small... COMMAND
    leftwidth   =  DIMENSION
    rightwidth  =  DIMENSION
    before      =  COMMAND
    after       =  COMMAND
```

and

```
\setupfooter [.1.] [..,.=2.,..]
              OPTIONAL
1   inherits from \setupheader
2   inherits from \setupheader
```

As with `\setup...texts` the first argument is optional. The keys `state`, `before` and `after` work on all parts of the pagebody, on the main text, the margins and edges.

When `...width` is set up the text is clipped at the given width. The key `strut` is important when footers or headers contain other objects than text. When `strut` is set to `no`, the object is not corrected for linedepth. You could use the command `\showstruts` to get some information on this phenomena.

The setups with `state` are explained in table ??. You should bear in mind that page numbering will always continue whether or not the pagenumbers are placed.

| **setup** | **result** |
|---|---|
| `normal` | visible |
| `none` | invisible, no whitespace |
| `empty` | one page invisble, whitespace |
| `high` | one page visible, no whitespace |
| `start` | visible |
| `nomarking` | leave out marks |
| `stop` | invisible, whitespace |

When setups are done between `\start` and `\stop` they will only work locally. This means that the setups are reset after `stop`. Headers and footers may appear even while you think new ones should appear. This is due to the way TEX determines valid breakpoints. One can never be certain when such an automatic break will occur. The solution is to force a new page by `\page` before `\stop`.

Headers and footers can be switched off on a page by means of:

```
\noheaderandfooterlines
```

Next to head– and footertexts there are also over– and bottomtexts. These are setup in a similar way:

```
\setuptoptexts [.1.] [.2.] [.3.]
                OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

```
\setuptexttexts [.1.] [.2.] [.3.]
                 OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

```
\setupbottomtexts [.1.] [.2.] [.3.]
                   OPTIONAL
1   text margin edge
2   TEXT date MARK pagenumber
3   TEXT date MARK pagenumber
```

```
\setuptop [.1.] [..,.2=.,..]
          OPTIONAL
1   inherits from \setupheader
2   inherits from \setupheader
```

```
\setuptext [.1.] [..,.2=.,..]
           OPTIONAL
1   inherits from \setupheader
2   inherits from \setupheader
```

```
\setupbottom [.1.] [..,.2=.,..]
             OPTIONAL
1   inherits from \setupheader
2   inherits from \setupheader
```

```
\notopandbottomlines
```

When the height of an area equals zero, no text is placed. By default the top and bottom area have zero height, so setting their text areas without setting the height has no effect.

At the instance of a new part or chapter we can deal in a different way with the headers and footers. Suppose that a default setup looks like this:

```
\setupheadertexts[pagenumber]
\setupfootertexts[chapter][paragraph]
```

At the first page of new chapters this may look not too good. Therefore we could state:

```
\setuphead[chapter][header=empty,footer=empty]
```

However if we use it in this way we loose the pagenumber. A more adequate solution is:

```
\definetext[chapter][footer][pagenumber]
```

with:

```
\setuphead[chapter][header=high,footer=chapter,page=right]
```

we obtain the desired effect. The pagenumber appears in the foot and the header disappears completely. These kind of commands are essential when you don't want to define all kinds of setups locally in a text, for example before every new chapter. This mechanism only works when going to a new page enabled.

```
\definetext [.1.] [.2.] [.3.] [.4.] [.5.]
                                OPTIONAL OPTIONAL
1  IDENTIFIER
2  header footer
3  TEXT
4  TEXT
5  TEXT
```

## 4.17 Footnotes

In some texts you can't do without footnotes. The footnote marker is placed in the text and the note itself is typeset at another location in the text, usually at the bottom of the page. Most often at the bottom of the page.

```
\footnote [.1.] {.2.}
          OPTIONAL
1  REFERENCE
2  CONTENT
```

A footnote number or –symbol is recalled with:

```
\note [.*.]
*  REFERENCE
```

An example of footnotes is given below.

```
The first compositions of the American composer Steve Reich will probably
only appreciated by the most \quote {purist} among those who like
minimal||music \footnote {A decent minimal is not so much characterized by
a minimal use of musical instruments, but more by subtle shifts in
polyphonic rhythms.}, his later works, like \quote {The Desert Music}, are
compositions for full orchestra, where the orchestra is extended with a for
Reich characteristic rhythm section \footnote {In most cases this section
consists of pianos, marimbas and xylophones.} and choir. Together
with John Adams, \footnote {His \quote {Fearful Symmetries} is a perfect
mix
of classic, jazz, swing and pop music.} Reich can be considered one of
today's leading composers. It is, however, a pity that they can only be
seen
\footnote {The nice thing about compositions like \quote {Drumming} and
\quote {Sextet} is de fact that \quotation {what the ear hears} differs
from what the \quotation {eye sees happening}.} and heard at the smaller
broad companies, like the \cap {VPRO}. \footnote{A non commercial Dutch
broadcast company.} \footnote {Sometimes also at other companies, because
somehow this kind of music is quite suited for impressive and|/|or
melodramatic documentaries.}
```

Undesired spaces are ignored. Spacing between two footnote numbers or symbols is taken care of. The result looks like this:

The first compositions of the American composer Steve Reich will probably only appreciated by the most 'purist' among those who like minimal–music[7], his later works, like 'The Desert Music', are compositions for full orchestra, where the orchestra is extended with a for Reich characteristic rhythm section[8] and choir. Together with John Adams,[9] Reich can be considered one of today's leading composers. It is, however, a pity that they can only be seen[10] and heard at the smaller broad companies, like the VPRO.[11] [12]

The type setting of the footnote can be setup with the command below that is defined in the setup area of your document.

```
\setupfootnotes [..,.=.,..]
                     *

*   conversion      =  numbers characters Characters romannumerals Romannumerals
    way             =  bytext bycd:section
    location        =  page TEXT columns firstcolumn lastcolumn high none
    rule            =  on off
    before          =  COMMAND
    after           =  COMMAND
    width           =  DIMENSION
    height          =  DIMENSION
    bodyfont        =  5pt ... 12pt small big
    style           =  normal bold slanted boldslanted type cap small... COMMAND
    distance        =  DIMENSION
    columndistance  =  DIMENSION
    margindistance  =  DIMENSION
    n               =  NUMBER
    numbercommand   =  \...#1
    textcommand     =  \...#1
    split           =  tolerant strict verystrict NUMBER
    textstyle       =  normal bold slanted boldslanted type cap small... COMMAND
    textcolor       =  IDENTIFIER
    interaction     =  yes no
    factor          =  NUMBER
    inherits from \setupframed
```

By default footnotes are placed at the bottom of a page. When using columns you can set `location` to `columns` so that the footnotes appear in the last column.

We can frame footnotes, place them in columns and decouple them from a page. The meaning of this last option is explained in an example.

[7] A decent minimal is not so much characterized by a minimal use of musical instruments, but more by subtle shifts in polyphonic rhythms.
[8] In most cases this section consists of pianos, marimbas and xylophones.
[9] His 'Fearful Symmetries' is a perfect mix of classic, jazz, swing and pop music.
[10] The nice thing about compositions like 'Drumming' and 'Sextet' is de fact that "what the ear hears" differs from what the "eye sees happening".
[11] A non commercial Dutch broadcast company.
[12] Sometimes also at other companies, because somehow this kind of music is quite suited for impressive and/or melodramatic documentaries.

```
\startlocalfootnotes[n=0]
  \placetable
    {A (latin) table.}
    \placelegend
      {\starttable[|l|r|]
       \HL
       \VL Nota \footnote {Bene} \VL Bene \footnote {Nota} \VL\FR
       \VL Bene \footnote {Nota} \VL Nota \footnote {Bene} \VL\LR
       \HL
       \stoptable}
      {\placelocalfootnotes}
\stoplocalfootnotes
```

The table enables the float placement mechanism, so we don't know on which page the table nor the footnotes will appear. So the footnotes are coupled to the table by using local footnotes.

[n=0]

| | |
|---|---|
| Nota[1] | Bene[2] |
| Bene[3] | Nota[4] |

1 Bene
2 Nota
3 Nota
4 Bene

**Table 4.4** A (latin) table.

```
\startlocalfootnotes ... \stoplocalfootnotes
```

```
\placelocalfootnotes [..,..=..,..]
                            *
                         OPTIONAL
*   inherits from \setupfootnotes
```

Footnotes can be placed at the end of a chapter or a document. The key `location` is set at `text` and we use the following command to place the footnotes:

```
\placefootnotes [..,..=..,..]
                       *
                    OPTIONAL
*   inherits from \setupfootnotes
```

When `n` is set at 2, you can display the footnotes in columns. This should be done at an early stage because TEX is using the dimensions of the footnotes to determine the page break. More information can be found in the source code of the ConTEXt module: `core-not.tex`.

The next example demonstrates that footnote numbers can be replaced by footnote symbols. In this example `conversion` is set at `set 3`.

note: use footnotes sparingly[13]
note: be brief[14]
note: no notes are even better[15]

Default the key `numbercommand` is set `\high`, but other setups are allowed. You can also work with:

```
\setupfootnotedefinition [..,..=..,..]

*   inherits from \setupdescriptions
```

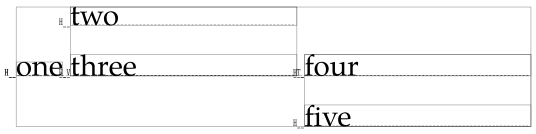
to define the exact way of how to display the footnotes, because the standard definition mechanism is used (see section ??).

## 4.18 Aligned boxes

TEX is basically aware of two kind of boxes: `\hbox` and `\vbox`. A horizontal `\hbox` can be considered a line, a `\vbox` a paragraph. There are two types of vertical boxes: a `\vbox` aligns on the baseline of the last line, while a `\vtop` aligns on the first line.

```
\hbox{\hbox{one} \vbox{two\par three} \vtop{four\par five}}
```

When we make the frames visible —in this case we said `\showboxes` in advance— the example above becomes:



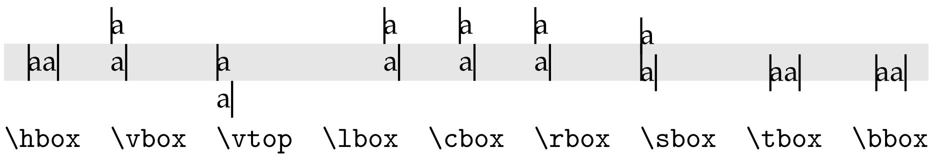
In addition ConTEXt provides a lot of alternative boxes, like: `\cbox`, `\lbox` and `\rbox`. These commands can be used while defining your own macros, but will seldom appear in the running text. Like in `\hbox` and `\vbox` the dimension of the width can be added.

```
\cbox{... text ...}
\lbox to 4cm{... text ...}
```

The reader is invited to experiment with these commands. A new line is forced with `\\`.

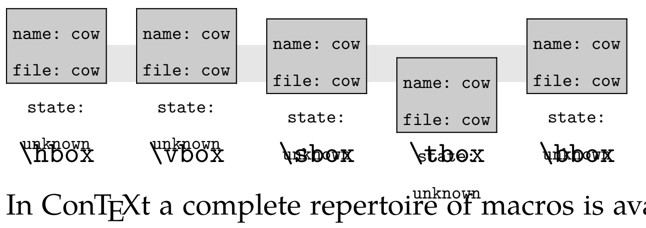
For some very dedicated purposes there is `\sbox`. This command is used to give a box the height of a strut. You may forget this command.

To another category of boxes belong `\tbox` and `\bbox`. Both are used within tables. Look at the example below that illustrates their use.



The `\tbox` and `\bbox` are also used in figures.

[13] During the development of ConTEXt the footnote mechanism was one of the first real challenges. And I'm challenged still since I just encountered documents with footnotes within footnotes.
[14] Why? See note[13].
[15] QED.



In ConTEXt a complete repertoire of macros is available that relies on boxes. For example we can add cutmarks to a box:

```
\setbox0=\vbox{The Final Cut\par --- \em Pink Floyd}
\makecutbox0 \box0
```

Be aware of the fact that such marks lie outside the boxes.

We can visualize boxes by using `\ruledhbox`, `\ruledvbox` and `\ruledvtop` instead of `\hbox`, `\vbox` and `\vtop`. With `\showmakeup` we can visualise everything automatically and we can get some insight on the features of ConTEXt and TEX.

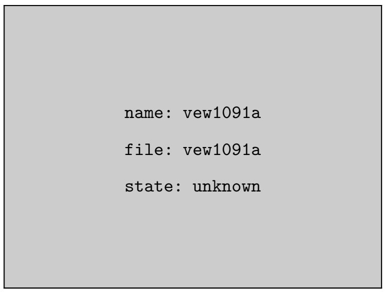
The next example shows that we can use TEX for more than only the straight forward typesetting. However, to be able to do this, one should have some insight in the manipulation of boxes. We use buffers to enhance comprehensibility.

```
\startbuffer[water]
Drink geen water \crlf direct uit de kraan! \blank

\start
  \tfx \setupinterlinespace Het drinkwater is tijdelijk niet betrouwbaar.
  Kook het water voor consumptie ten minste 2~minuten. Zodra het water
  weer betrouwbaar is, krijgt u bericht. \par
\stop

\blank[2*big]

\language[en] Do not drink water \crlf directly from the tap! \blank

\start
  \tfx \setupinterlinespace The water is temporarily unfit for drinking.
  Boil the water during at least 2~minutes before consumption. As soon
  as the water is reliable again, you will be notified. \par
\stop
\stopbuffer
```

This text is typeset in a framed box. We use two temporary boxes. The first determines the height of the second one. Instead of `\tfx\setupinterlinespace` you could use `\switchtobodyfont` to switch to a narrower bodyfont. (`[small]`). The `\par` is essential!

```
\framed[offset=\bodyfontsize]
  {\setbox0=\vbox
     {\hsize 16em\switchtobodyfont[ss]\getbuffer[water]}
   \setbox2=\vbox to \ht0
     {\vfill\externalfigure[vew1091a][width=5cm]\vfill}
   \hskip1em\box2\hskip1em\box0\hskip1em}
```

The result —an example of a drinking water warning— is shown below.



Drink geen water
direct uit de kraan!

Het drinkwater is tijdelijk niet betrouwbaar. Kook het water voor consumptie ten minste 2 minuten. Zodra het water weer betrouwbaar is, krijgt u bericht.

Do not drink water
directly from the tap!

The water is temporarily unfit for drinking. Boil the water during at least 2 minutes before consumption. As soon as the water is reliable again, you will be notified.

## 4.19 Makeup

A document may have a titlepage, a colofon and some pages that are not directly related to the main part of the document. Mostly these pages are not numbered and can do without headers and footers. Because their layout needs extra attention we prefer the word makeup for defining their specific layout.

The commands `\startstandardmakeup` and `\stopstandardmakeup` exclude text from the standard pagebody and its layout. Below a simple example is given. You will notice commands like `\vfill`, `\blank`, `\tf` and even `\crlf` and `\vskip`.

```
\startstandardmakeup
  \tfd Jobs around the house \blank[2*big]
  \tfb Part 1: Gas, water and  electricity \vfill
  \tfb J. Hagen \crlf A.F. Otten \blank
  \tfb Hasselt \crlf \currentdate[month,year]
\stopstandardmakeup
```

In double–sided documents an empty page is generated that functions as the backside of the title page. However sometimes this backside should also be typeset.

```
\startstandardmakeup[doublesided=no]
... the front
\stopstandardmakeup
\startstandardmakeup[page=no]
... the back
\stopstandardmakeup
```

Because double–sided typesetting is turned off, a backside page is not generated. And because the key `page` is `no` the next page does not get the layout of a right hand side page (this would be default).

With the command `\showframe` frames can be made visible (temporarily) around the made up text. This is very convenient during the typesetting of separate pages.

Next to the command `\startstandardmakeup` one can define his own layout with different dimensions by means of:

```
\definemakeup [.1.] [..,.2.,..]

1   IDENTIFIER

2   inherits from \setupmakeup
```

```
\setupmakeup [.1.] [..,.2.,..]

1   IDENTIFIER

2   width        =  DIMENSION
    height       =  DIMENSION
    voffset      =  DIMENSION
    hoffset      =  DIMENSION
    page         =  left yes right
    commands     =  COMMAND
    doublesided  =  yes no empty
    headerstate  =  normal stop start empty none nomarking
    footerstate  =  normal stop start empty none nomarking
    textstate    =  normal stop start empty none nomarking
    topstate     =  stop start
    bottomstate  =  stop start
    pagestate    =  stop start
    color        =  IDENTIFIER
```

The first command generates a `\start...stop`–pair between which the new typesetting commands can be typed. Bij default the result of this new layout is typeset on an empty page. The new layout is marked with `<<name>>`, for selection at a later stage (see section ??).

The commands that are provided after the key `commands` are executed immediately when a new layout is called. In this local layouts can be defined.

# 5 Typography

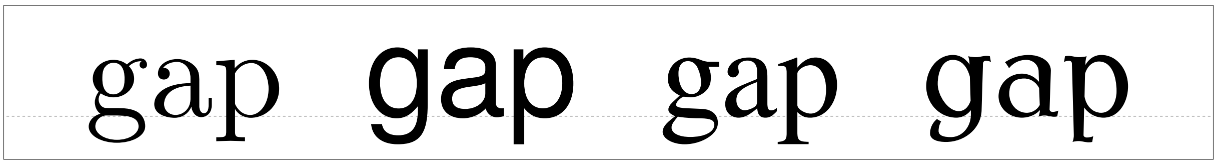
## 5.1 Introduction

Throughout the millennia humans have developed and adapted methods for storing facts and thoughts on a variety of different media. A very efficient way of doing this is using logograms, as the Chinese have done for ages. Another method is to represent each syllable in a word by a symbol, as the Japanese do when writing telegrams. However, the most common way of storing characters is by using a limited set of shapes representing basic sounds (a.k.a. phonemes). Such a collection is called an *alphabet*, and the shapes are called *letters*.

TEX is primarily meant for typesetting languages that use this third method. The other two methods can also be dealt with, but some extra effort is needed. In this chapter we will focus on languages that use alphabets, the other methods will be explained in later chapters.

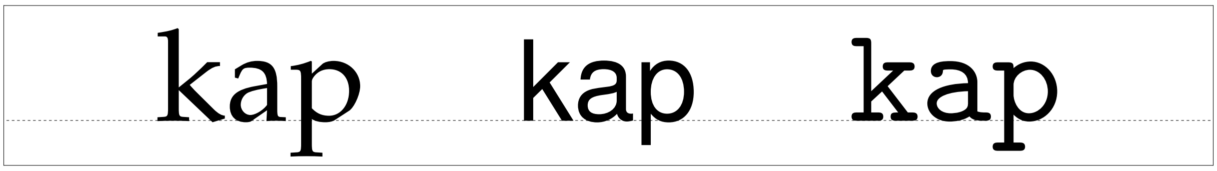
The shapes representing the characters that make up an alphabet are more or less standardized, and thereby can be recognized by readers even if their details differ. A collection of pictures representing character shapes is called a *font*, and the pictures in a font are called *glyphs*.

The example below shows (from left to right) a Computer Modern font, a Helvetica lookalike, a Times Roman lookalike and the Antiqua Torunska font, all scaled to 48pt.



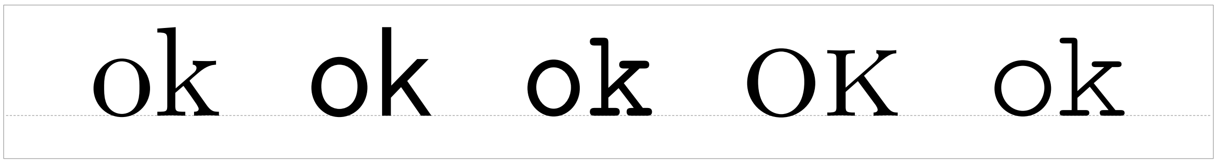
As you can see, quite some design variation is possible. It follows that when fonts from different sources (designers) are intermixed, the result is not always pleasing to look at. The term *font collection* refers to a set of fonts combined together in such a way that the overall appearance on a page looks good and reading is as comfortable as possible.

The next example shows an attempt at such a font collection: the fonts were picked such that the glyph sizes and the line thicknesses are roughly the same.

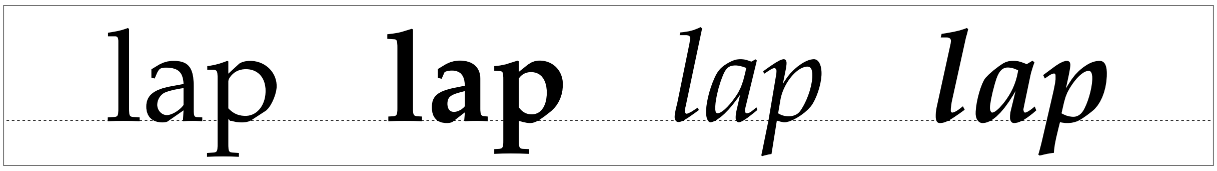


Fonts from a single source often already come in a few variations that are intended to be used together. Such a set of fonts with the same basic design is known as a *font family*. For example, Computer Modern is a font family, as is Lucida.

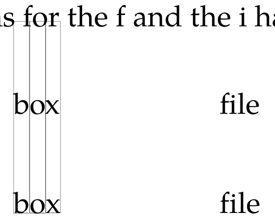
Within a font family there can be multiple *styles*. In the example below you see five font styles of the Latin Modern family: the Roman, Sans, Typewriter, Smallcaps and Variable Typewriter.



Within a given style, there can be multiple *alternatives* of a font. The example below shows a normal, a bold, an italic, and a bold italic alternative.



The distance between the individual glyphs in a word and the actual glyphs that are used depends on the combinations of these glyphs. In the top line of the next sample, the gap between the b and the o as well as the distance between the o and the x is slightly altered. This is called kerning. Further, the separate glyphs for the f and the i have been combined into a single one. This is called ligaturing.



The font shown here is Computer Modern, the default TEX font. This font is designed by Donald Knuth. The Computer Modern has many kerning pairs, while the Palatino–like font that is used for most of the text in this manual has only a few, while both have essentially the same list of ligatures.

Micro–typography like kerning pairs and ligatures are not to be altered by the user, but are part of the font design and the required data is stored inside the font file, together with the drawing routines for the actual pictures. It *is* possible for the user to alter fonts and interline spacing and some more aspects on the level of macro–typography. The choice of font is the main topic of this chapter.

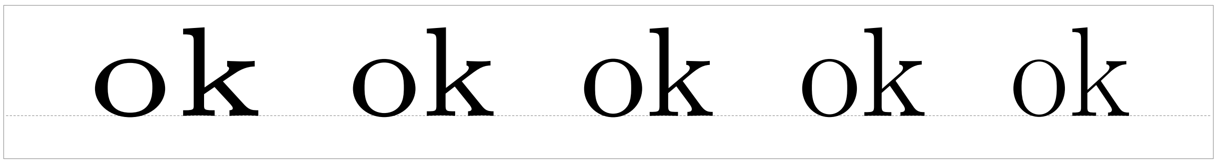
There are many different methods that can be used to classify fonts. There are classification systems based on the period in which the style was first developed; on the characteristics of the font; or the font application, like a newspaper or a book. Often, classification systems mix these characteristics to a certain point.

For example, the Computer Modern family can be classified as a ‘modern’ font. This is a classification that primarily indicates a period (late 18th century), but it also implies a particular shape: ‘modern’ fonts have a high contrast between thick and thin strokes, and their stress axis is perfectly vertical.

At the same time, specific fonts in the Computer Modern family can be classified as to their style: ‘serif’ (glyphs strokes have embellishments at the end), ‘sans serif’ (shapes end abruptly), or ‘monospaced’ (all glyphs have the same width).

The Computer Modern family is in fact inspired by one font in particular: ‘Modern 8a’ by the Monotype corporation. Knuth implemented Computer Modern in MetaFont using parameters so that he could generate a whole collection of fonts all closely matching each other in overall style (not necessarily in *style*). In ConTEXt you will normally use a reimplementation of Computer Modern using a more modern file format (Type 1 or OpenType). This new version is called ‘Latin Modern’, and also features an extended glyph set making it usable for languages that could not be typeset with Knuth’s original fonts.

Computer Modern is one of the few font families that comes with dedicated design sizes. The example below shows the differences of a 5, 7, 9, 12 and 17 point design scaled up to 48 points. Such nuances in font size are seldom seen these days.



As explained earlier, the general appearance of a font style can be classified according to many schemes, and the exact terminology used depends on the background of the user. In **table 5.1** you can see some examples of the terms that are used by various people to identify the three font styles that are most often found together within a single book design (such as for a software manual).

| **terms** | **intended usage** |
|---|---|
| regular, serif, roman | main text |
| support, sans | section headings |
| teletype, mono, type | `code examples` |

**Table 5.1** Some ways of identifying the font styles in a document design.

Within the lists of terms, the earlier names are normally used by typographers and book designers, the later ones are commonly used in TeX. In ConTeXt all of these terms can be used intermixed because they are all remapped to the same set of internal commands. As will be explained later, the command `\rm` is used to switch to the style used for the main text (this is usually a font style with serifs), `\ss` to switch to the support style (usually a style without serifs) and `\tt` to switch to the code example style (for which usually monospaced fonts are used).

Text can be typeset in different font sizes. The unit `pt`, short for 'printer's point', is normally used to specify the size of a font. There are a little over 72 points per inch (or a little under 2.85 points per millimeter, if you prefer metric units). Traditionally, font designers used to design a glyph collection for each point size, but nowadays most fonts have only a single design size of 10 points, or at most a small set of sizes with names indicating their proposed use, like *caption*, *text*, and *display*.

The next sections will go into the details of switching of font styles and fonts in your documents. Be warned that the font switching mechanism is rather complex. This is due to the different modes like math mode and text mode in ConTeXt. If you want to understand the mechanism fully, you will have to acquaint yourself with the concept of encoding vectors and obtain some knowledge on fonts and their peculiarities. See the next chapter for more information.

## 5.2 The mechanism

Font switching is one of the oldest features of ConTeXt because font switching is indispensable in a macro package. During the years extensions to the font switching mechanism were inevitable. The following starting points have been chosen during the development of this mechanism:

- It must be easy to change font *styles*, e.g., switching between roman (serif, regular), sans serif (support), teletype (monospaced) etc. (`\rm`, `\ss`, `\tt` etc.)
- More than one *alternative* set of glyphs shapes must be available like italic and bold (`\it` and `\bf`).
- Different font *families* like Latin Modern Roman and Lucida Bright must be supported.
- It must be possible to combine different families into font *collections*.
- Different sub– and super–scripts must be available. These script sizes have to be consistent across the switching of family, style and alternative.
- It should be possible to combine all of these requirements into a single definition unit called a *body font*.
- Changing the global font collection as well as the size must also be easy, and so sizes between 8pt and 14.4pt must be available by default.

Before reading further, please stop for a moment to make sure you thoroughly comprehend the above paragraphs. ConTEXt's terminology probably differs from what you are accustomed to, especially if you were previously a LaTEX user.

## 5.3 Font switching

The mechanism to switch from one style to another is somewhat complex, not in the least because the terminology is a bit fuzzy. A quick recap: we call a set of fonts from the same source, with the same basic design, like Lucida or Computer Modern Roman, a *family*. Within such a family, the members can be grouped according to characteristics such as the presence of serifs, or the variability of width. Such a group is called a *style*. Examples of styles within a family are: 'roman', 'sans serif' and 'teletype'. We saw already that there can be other classifications. Within a style there can be *alternatives*, like 'boldface' and 'italic'.

There are different ways to change into a new a style or alternative. You can use `\ss` to switch to a sans serif font style and `\bf` to get a bold alternative. When a different style is chosen, the alternatives adapt themselves to this style.

Often a document will be mostly typeset using just one combination of family and style. This is called the *bodyfont*. Consistent use of alternative commands like `\bf` and `\it` in the text will automatically result in the desired bold and italic alternatives when you change the family or style in the setup area of your input file, since these commands adapt to the specified family and style.

### 5.3.1 Font style switching

Switching to another font style is done by one of five two-letter commands that are listed in **table 5.2**.

The 'handwritten' and 'calligraphic' font styles are sometimes useful when dealing with very elaborate document layout definitions. In the ConTEXt distribution only the Lucida font family uses these styles; in any other font set they are simply ignored. You could use them in your own font setups if you so desire. See the next chapter for font setup definitions.

There is a sixth internal style that is only ever referred to as 'mm'. This style handles math fonts. It does not make sense to use this style directly so there is no command attached to it, but it is quite important internally so it makes sense to introduce it right away.

| **command** | **keyword equivalents** |
|---|---|
| `\rm` | serif, regular, roman, rm |
| `\ss` | sans, support, sansserif, ss |
| `\tt` | mono, type, teletype, tt |
| `\hw` | handwritten, hw |
| `\cg` | calligraphic, cg |
| – | mm |

**Table 5.2** Font style switching commands and their keyword equivalents. For more on keywords, see **subsection 5.3.3**.

### 5.3.2 Font alternative switching

The alternatives within a style are given in **table 5.3**. Not all fonts have both italic and slanted or the bold alternatives of each. Some other fonts do not have small caps or have only one set of digits. When an alternative is not known, ConTEXt will attempt to choose a suitable replacement automatically. For instance, the italic alternative may be used for if slanted is not available or vice versa.

| **command** | **keyword equivalents** |
|---|---|
| `\bf` | bold |
| `\it` | italic |
| `\bi` | bolditalic, italicbold |
| `\sl` | slanted |
| `\bs` | boldslanted, slantedbold |
| `\sc` | smallcaps |
| `\os` | mediaeval (from *oldstyle*) |
| `\tf` | normal (from *typeface*) |

**Table 5.3** Font alternative switching commands and their keyword equivalents. With `\os` you tell ConTEXt that you prefer mediaeval or old–style numbers as in 139 over 139.

Note that, while these alternatives can sometimes seem to be 'combined,' as in `bolditalic`, it is important to recognize that only one alternative can actually be active at a time. In this regard alternatives are like 'radio buttions.' `bolditalic` is in fact one predefined alternative, not a combination of two. Alternatives cannot be arbitrarily combined, or turned on and off independently of each other.

Besides these two-letter commands, there is a series of font selector commands with a size suffix attached. Some examples of that are:

```
\tfx \bfx \slx \itx
\tfa \tfb \tfc \tfd \tfxx
```

The `a` suffix selects a somewhat larger font size than the default. Each of the ordered alphabetic suffixes `a`, `b`, … select a somewhat larger actual font than the previous suffix. The `x` and `xx` suffixes select smaller and yet smaller versions. Note that these commands select font sizes relative to the default, not relative to whatever font size is currently in effect.

| | |
|---|---|
| `\bfx` | smallbold |
| `\itx` | smallitalic |
| `\bix` | smallbolditalic, smallitalicbold |
| `\slx` | smallslanted |
| `\bsx` | smallboldslanted, smallslantedbold |
| `\tfx` | small, smallnormal |

**Table 5.4** Small alternative switching commands and their keyword equivalents.

The 'small' (single `x` suffix) switches mentioned in **table 5.4**, such as `\tfx`, are always available. The availability of other commands like `\ita`, `\bfxx`, `\bfc`, etc. depends on the completeness of the font definition files. For the core ConTEXt fonts, you can count on at least `\tfa`, `\tfb`, `\tfc`, `\tfd`, and `\tfxx` being defined. For the others, just try and see what happens.

When you have chosen a larger character size, for example `\tfb`, then `\tf` equals `\tfb`, `\bf` equals `\bfb`, etc. This behavior is almost always preferable over returning to the original character size, but it may catch you off-guard.

More generic font scaling commands are also available:

```
\tx \txx
\setsmallbodyfont \setbigbodyfont
```

The command `\tx` adapts itself to both the style and the alternative. This command is rather handy when one wants to write macros that act like a chameleon. Going one more step smaller, is possible too: `\txx`. Using `\tx` when `\tx` is already given, is equivalent to `\txx`.

The commands `\setsmallbodyfont` and `\setbigbodyfont` switch to the 'small' and 'big' body font sizes. These relative sizes are defined via the 'body font environment', see **section 5.9**.

The various commands will adapt themselves to the actual setup of font and size. For example:

```
{\rm test {\sl test} {\bf test} \tfc test {\tx test} {\bf test}}
{\ss test {\sl test \tx test} {\bf test \tx test}}
```

will result in:

test *test* **test** test test **test**
test *test* *test* **test** **test**

When the `\rm` style is active, ConTEXt will interpret the command `\tfd` as if it was `\rmd`, when the style `\ss` is active, `\tfd` as is treated as `\ssd`. All default font setups use `tf`–setups so they will automatically adapt to the current font style.

The remainder of this section is for the sake of completeness. Use of the following commands in new documents is discouraged.

Frequent font switching leads to longer processing times. When no sub- or superscripts are used and you are very certain what font you want to use, you can perform fast font switches with: `\rmsl`, `\ssbf`, `\tttf`, etc.

The plain TEX compatible font switches `\vi`, `\vii`, `\viii`, `\ix`, `\x`, and `\xii` are also defined, these have local effects like `\tfx` and `\tfa`.

### 5.3.3 Switching font styles in setup commands

A number of ConTEXt commands use the parameter `style` to set the used font. The parameter mechanism is rather flexible so that within the parameter `style` you can use any of the font switching commands like `\bf` or `bf` or `\switchtobodyfont`, but also a number of keywords like

```
normal  bold  italic  bolditalic  slanted  boldslanted  type
small  smallbold  smallitalic  ...  smallslanted  ...  smalltype
capital
```

Most of these keywords have already been listed in the tables 5.3 and 5.4, but a few predefined ones have not been mentioned yet. These are displayed in **table 5.5**, together with the commands they execute. As is normal in ConTEXt, you can extend the list of accepted keywords by defining your own. This will be explained in?? in the next chapter.

| | |
|---|---|
| `\tt` | type, mono |
| `\ttx` | smalltype |
| `\ss` | sans, sansserif |
| `\ss \bf` | sansbold |
| `\setsmallbodyfont` | smallbodyfont |
| `\setbigbodyfont` | bigbodyfont |
| `\smallcapped` | cap, capital |
| `\WORD` | WORD |

**Table 5.5** Remaining font alternative keywords.

## 5.4 Emphasize

Within most macro–packages the command `\em` is available. This command behaves like a chameleon which means that it will adapt to the actual typeface. In ConTEXt `\em` has the following characteristics:

- a switch to *italic* or *slanted* is possible
- a switch within `\bf` results in ***bold italic*** or ***bold slanted*** (when available)
- a so called *italic correction* is performed automatically (\/)

The bold italic or bold slanted characters are supported only when `\bs` and `\bi` are available.

```
The mnemonic {\em em} means {\em emphasis}.
{\em The mnemonic {\em em} means {\em emphasis}.}
{\bf The mnemonic {\em em} means {\em emphasis}.}
{\em \bf The mnemonic {\em em} means {\em emphasis}.}
{\it The mnemonic em {\em means \bf emphasis}.}
{\sl The mnemonic em {\em means \bf emphasis}.}
```

This results in:

The mnemonic *em* means *emphasis*.
*The mnemonic* em *means* emphasis.
**The mnemonic *em* means *emphasis*.**
***The mnemonic* em *means* emphasis.**
*The mnemonic em* means **emphasis**.
*The mnemonic em* means **emphasis**.

The advantage of the use of `\em` over `\it` and/or `\sl` is that consistent typesetting is enforced.

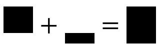
By default emphasis is set at *slanted*, but in this text it is set at *italic*. This setting is made via `\setupbodyfontenvironment`, see **section 5.9** for more details:

```
\setupbodyfontenvironment
   [default]
   [em=italic]
```

## 5.5 Line spacing

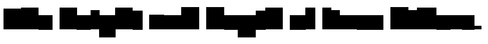
In TEX linespacing is determined by a number of variable dimensions like `\topskip`, `\parskip` and `\baselineskip`. However, in ConTEXt these variables are related to the bodyfont size.

A line has a height and a depth. The distance between two lines is normally equal to the sum of the maximum height and maximum depth:



This sum is in ConTEXt equal to 2.8ex, so almost three times the height of an x. This is about 1.2 times the bodyfont height. The proportion between maximum height and depth is .72 : .28 by default. Linespacing alters when a new bodyfont is used or when linespacing is defined explicitly by `\setupinterlinespace` (which is explained later):

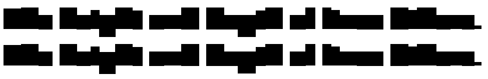
Sometimes a line does not have the maximum height or depth. The next example illustrates this:



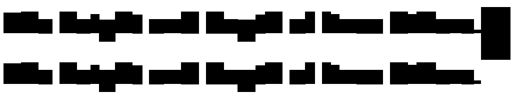
It says:

The height and depth of lines differs.

When we put two of these lines above each other we will get:



You can see that the distance is somewhat bigger that the sum of the height and depth of each separate line. This distance is called the baseline distance (`\baselineskip`) and is in this document 13.8292pt. If we add some extra height to the line we see this:



To prevent the lines from touching TEX adds a `\lineskip`, in our example 1.0pt. In a similar way TEX is taking care of the first line of a page to have at least a height of `\topskip` (here 11.0pt plus 55.0pt).

Linespacing is set up by:

```
\setupinterlinespace [.*.]
                       OPTIONAL
*   reset small medium auto big on off
```

```
\setupinterlinespace [..,.*=.,..]

*   height  =  NUMBER
    depth   =  NUMBER
    line    =  DIMENSION
    top     =  NUMBER
    bottom  =  NUMBER
```

Linespacing adapts to the size of the actual bodyfont automatically. This means that the user can leave this command untouched, unless a different linespacing is wanted. Instead of a factor one of the predetermined values `small` (1.0), `medium` (1.25) or `big` (1.5) can be given. Below an example is given of a text with a linespacing of 1.25: `\setupinterlinespace[medium]`.

> Whenever it comes to my mind that "everything that comes in quantities, will somehow survive", I also got the feeling that in a few hundred years people will draw the saddening conclusion that all those top–ten hits produced by computers represent the some of todays musical and instrumental abilities. Isn't it true that archaeologists can spend a lifetime on speculating about some old coins from the first century? On the other hand, the mere fact that one can have success with this type of non–music success of some top–hit musicians demonstrates both the listeners inability to rate the product and the lack of self criticism of the performers. In principle the future archaeologist will therefore draw the right conclusion.

When you make a font switch the linespacing is adapted when you give the command `\setupinterlinespace` without any setup parameters and also when you add the key `reset`, for example

```
\setupinterlinespace[reset,medium]
```

The text below is typeset in the fontsize `\tfa`, using the following input:

```
\start \tfa \setupinterlinespace
In books meant for children we often find
a somewhat ... when needed. \par \stop
```

In this example the `\par` is necessary because TEX operates on whole paragraphs. Within a group one has to close the paragraph explicitly with an empty line or `\par` otherwise TEX will have forgotten the linespacing before the paragraph is finished (as in that case, the paragraph is ended by the empty line after the `\stop`).

The word `height` is typeset inside a bare `\tfd` group, to illustrate why `\setupinterlinespace` is required.

In books meant for children we often find a somewhat bigger typeface, for instance because we are convinced that this enables them to read the book themselves. On the other hand, I can also imagine that it is a cheap way to increase the number of pages. Unfortunately scaling up will also uncover the lack of quality of the typesetting used and/or the lack of typographic knowledge of the user of such a system. The interline space sometimes differs on a line by line basis, and depends on the height of the current line. Therefore, when changing the style, something that should only be done on purpose, also change the baseline distance when needed.

Instead of a keyword, one can pass a key–value pair to define the characteristics of a line.

The default settings are:

```
\setupinterlinespace
  [height=.72,
   depth=.28,
   top=1.0,
   bottom=0.4,
   line=2.8ex]
```

The `height` and `depth` determine the ratio between the height and depth of a line. The baseline distance is set to 2.8ex. The parameters `top` and `bottom` specify the relation between the bodyfont size and the height of the first line and the depth of the last line on a page. They are related to TEX's `\topskip` and `\maxdepth`.

We will see later that instead of setting the spacing at the document level, i.e. for each font, you can set the spacing per body font environment:

```
\setupbodyfontenvironment
  [modern] [12pt]
  [interlinespace=14pt]
```

## 5.6 Capitals

Some words and abbreviations are typeset in capitals (uppercase). ConTEXt provides the following commands for changing both upper– and lowercase characters into capitals.

```
\cap {.*.}

*   CONTENT
```

```
\Cap {.*.}

*   CONTENT
```

```
\CAP {.*.}

*   CONTENT
```

```
\Caps {.. .*. ..}

*   WORD
```

The command `\cap` converts all letters to capitals at the size of `\tx`. If you switch to italic (`\it`), bold (`\bf`), etc. the capital letter will also change. Since `\cap` has a specific meaning in math mode, the formal implementation is called `\smallcapped`. However in text mode one can use `\cap`.

```
Capitals for \cap {UK} are \cap {OK} and capitals for \cap {USA} are
okay. But what about capitals in \cap {Y2K}.
```

this results in:

Capitals for UK are OK and capitals for USA are okay. But what about capitals in Y2K.

A `\cap` within a `\cap` will not lead to any problems:

```
\cap {People that have gathered their \cap {capital} at the cost of other
people are not seldom \nocap {decapitated} in revolutionary times.}
```

or:

PEOPLE THAT HAVE GATHERED THEIR CAPITAL AT THE COST OF OTHER PEOPLE ARE NOT SELDOM decapitated IN REVOLUTIONARY TIMES.

In this example you can see that `\cap` can be temporarily revoked by `\nocap`.

```
\nocap {.*.}

*   CONTENT
```

The command `\Cap` changes the first character of a word into a capital and `\CAP` changes letters that are preceded by `\\` into capital letters. With `\Caps` you can change the first character of several words into a capital letter.

```
\setupcapitals [..,.=.,..]

*   title  =  yes no
    sc     =  yes no
```

With this command the capital mechanism can be set up. The key `sc=yes` switches to real SMALL CAPS. The key `title` determines whether capitals in titles are changed.

Next to the former `\cap`–commands there are also:

```
\Word {.*.}
*   WORD
```

and

```
\Words {.. .*. ..}
*   WORD
```

These commands switch the first characters of a word or words into capitals. All characters in a word are changed with:

```
\WORD {.*.}
*   WORD
```

Let's end this section with real small capitals. When these are available the real small caps `\sc` are preferred over the pseudo–capital in abbreviations and logos.

```
In a manual on \TeX\ and Con\TeX t there is always the question whether
to type \cap{\TeX} and \cap{Con\TeX t} or {\sc \TeX} and {\sc Con\TeX t}.
Both are defined as a logo in the style definition so we type \type {\TEX}
and \type {\CONTEXT}, which come out as \TEX\ and \CONTEXT.
```

Results in:

In a manual on TEX and ConTEXt there is always the question whether to type TEX and CONTEXT or TEX and ConTEXt. Both are defined as a logo in the style definition so we type `\TEX` and `\CONTEXT`, which come out as TEX and ConTEXt.

IT IS ALWAYS POSSIBLE TO TYPESET TEXT IN SMALL CAPITALS. HOWEVER, REALIZE THAT LOWER CASE CHARACTERS DISCRIMINATE MORE AND MAKE FOR AN EASIER READ.

An important difference between `\cap` and `\sc` is that the latter command is used for a specific designed font type. The command `\cap` on the other hand adapts itself to the actual typeface: *KAP*, **KAP**, ***KAP***, etc.

## 5.7 Character spacing

Some typesetting packages stretch words (inter character spacing) to reach an acceptable alignment. In ConTEXt this not supported. On purpose! Words in titles can be stretched by:

```
\stretched {.*.}
*   WORD
```

```
\hbox to \hsize {\stretched{there\\is\\much\\stretch\\in ...}}
\hbox to 20em   {\stretched{... and\\here\\somewhat\\less}}
```

With `\\` you can enforce a space (`{}` is also allowed).

t h e r e i s m u c h s t r e t c h i n . . .
. . . a n d h e r e s o m e w h a t l e s s

These typographically non permitted actions are only allowed in heads. The macros that take care of stretching do this by processing the text character by character.

This chapter will not go into the details of underlining because using underlining for typographical purposes is a bad practice. Instead, the commands related to under- and over-lining are discussed in **section 14.5** (“**Underline**”).

## 5.8 Selecting bodyfonts

The bodyfont (main font), font style and size is set up with:

```
\setupbodyfont [...,*,...]

*   IDENTIFIER serif regular roman sans support sansserif mono type teletype handwritten
    calligraphic 5pt ... 12pt
```

In a running text a temporary font switch is done with the command:

```
\switchtobodyfont [...,*,...]

*   IDENTIFIER serif regular roman sans support sansserif mono type teletype handwritten
    calligraphic 5pt ... 12pt small big
```

This command doesn’t change the bodyfont in headers and footers. With `small` and `big` you switch to a smaller or larger font.

In most cases, the command `\setupbodyfont` is only used once: in the style definition, and font switching inside the document is done with `\switchtobodyfont`. Don’t confuse these two because that may lead to some rather strange but legitimate effects.

### 5.8.1 Body font sizes

Body font sizes actually consist of two components: the font size and a number of indirect parameters. Think of things like the font size used in headers, footers, footnotes, sub– and superscripts, as well as the interline space and a few others.

This is why in ConTEXt there is the concept of a *body font environment* (expressed as a dimension), and that is what you pass as an argument to `\setupbodyfont` or `\switchtobodyfont`. The definitions as presented above indicate `5pt ... 12pt` for the body font environment, but actually any dimension is acceptable.

The most frequently used sizes are predefined as body font environments: 4pt ... 12pt, 14.4pt, and 17.3pt. But when you use a different, not-yet-defined size specification —for example in a title page— ConTEXt will define a body font environment for that size automatically. While doing so, ConTEXt normally works with a precision of 1 decimal to prevent unnecessary loading of fontsizes with only small size differences.

Be warned that in this case, the results may be a less than ideal. The reason is that ConTEXt not just has to load the actual font, but it also has to guess at the various other settings like the relative font sizes and the interline space. It does so by using the values from the nearest smaller body font environment is that is already defined.

You can extend the list of predefined body font environments and even alter the precision in body font matching. See **section 5.9** for detailed information about how to tweak or define your own body font sizes.

To end this section, the example below demonstrates how the interline space is adapted automatically, when changing the size of the bodyfont. Consider this input:

```
{\switchtobodyfont[14.4pt] with these commands \par}
{\switchtobodyfont[12pt]   for font switching  \par}
{\switchtobodyfont[10pt]   it is possible to   \par}
{\switchtobodyfont[8pt]    produce an eye test: \par}
{\switchtobodyfont[6pt]    a x c e u i w m q p \par}
```

The actual ConTEXt behaviour is shown below on the left. On the right you can see what would have happened if the interline space were not automatically adapted.

with these commands
for font switching
it is possible to
produce an eye test:
a x c e u i w m q p

with these commands
for font switching
it is possible to
produce an eye test:
a x c e u i w m q p

## 5.8.2 Body font identifiers

In the definition block of `setupbodyfont` there was a list of words given besides the special marker `IDENTIFIER`. These words are the symbolic ConTEXt names for the font styles that we ran into earlier, with a few aliases so that you do not have to worry about the actual naming convention used. The symbolic names are mapped to two-letter internal style abbreviations that are used internally. See **table 5.2** for an overview.

Although the macro syntax does not say so, you can use two-letter internal style abbreviations (`ss`, `rm`) as well as the longer names, if you prefer.

We have seen already that there are other and easier ways to switch the font style, so if `\setupbodyfont` could only be used for this purpose it would not be all that useful. But luckily there is more: the optional `IDENTIFIER` can be a 'body font name' (aka 'typeface'). Such names have to be predefined, perhaps in a font support file, or simply on earlier lines in the style definition.

A 'typeface' is a symbolic name that links a single font style to actual font families. Such symbolic names are typically grouped together in a definition block that sets up values that link the four styles `\rm`, `\ss`, `\tt` and `\mm` to fonts in a 'font collection', and such definition blocks are called 'typescripts'.

ConTEXt expects you to define your own font setups, but there are quite a few examples predefined in various typescript files. Not all of those are perpetually loaded, so you usually have to execute a typescript explicitly to get the typeface names predefined. To this end, typescripts *themselves* also have names.

Executing a typescript is done by `\usetypescript`. We will get back to `\usetypescript` later because it is in fact a very flexible command, but let's discuss simple usage first.

```
\usetypescript [...,...] [...,...] [...,...]
                  1           2           3
                           OPTIONAL    OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   IDENTIFIER
```

A typical input sequence for selecting the predefined 'palatino' set of typefaces in MkII will look like this:

```
\usetypescript[palatino][ec]
\setupbodyfont[palatino,12pt]
```

In this example the typescript named `palatino` is asked for in the `ec` font encoding, and that defines a set of typefaces under the name `palatino`. These are then used by `\setupbodyfont` and eventually this makes pdfTEX load the free Type 1 font URW Palladio in the correct encoding. URW Palladio is a font that looks a lot like the commercial font Linotype Palatino by Hermann Zapf, which explains the name of the typescript and typefaces.

Font encodings will be handled fully in the **section 5.15**. For now, please take for granted the fact that pdfTEX needs a second argument to `\usetypescript` that specifies an encoding name, and that there is a fixed set of acceptable names that depends on the typescript that is being requested.

In XeTEX and MkIV the situation is a little bit different because fonts are reencoded to match Unicode whenever that is possible. That in turn means that XeTEX and MkIV prefer to use OpenType fonts over Type 1 fonts, so different typescript definitions are used behind the scenes, and the second argument to `\usetypescript` becomes optional.

For example,

```
\usetypescript[palatino]
\setupbodyfont[palatino,12pt]
```

will make XeTEX and LuaTEX load the OpenType font Pagella. This is a free font from the TEX Gyre project, that also looks just like the commercial font Linotype Palatino. You may as well leave the second argument in place: while it will always be ignored by LuaTEX, XeTEX will actually use that encoding if the typescript uses Type 1 fonts instead of the more modern OpenType or TrueType font formats.

All predefined typescripts attach meaning to (at least) the three basic text font styles(serif, sans, and mono), so you can e.g. do this:

```
\usetypescript[times][ec]
\setupbodyfont[times,sans,12pt]
```

and end up using the OpenType font TEX Gyre Heros or the Type 1 font URW Nimbus Sans L. Both fonts are very similar in appearance to Linotype Helvetica, by the way.

The typescripts that come with the ConTEXt distribution are placed in source files that have names that start with `type-`. Some of these files are automatically loaded when needed, but most have to be loaded explicitly. There is a list in **table 5.6**

Some of the internal building blocks for typescripts are themselves located in yet other files (font size and font map file information, for example). Normally, when ConTEXt has to load typescript information from files, it will try to save memory by only executing the typescript it needs at that moment and discarding all other information. If you have enough memory at your disposal, you can speed up typescript use considerably by adding

```
\preloadtypescripts
```

in your preamble or your `cont-usr.tex`. This will make ConTEXt store all the typescript information in internal token registers the first (and therefore only) time it loads the actual files.

| File | Loaded by pdfTEX | Loaded by XƎTEX | Loaded by MkIV | Description |
|---|---|---|---|---|
| type-akb | no | no | no | PostScript fonts using psnfss names (Type 1) |
| type-buy | no | no | no | Various commercial fonts (Type 1) |
| type-cbg | no | no | no | Greek free fonts (Type 1) |
| type-cow | no | no | no | The ConTEXt cow font (Type 1) |
| type-exp | no | no | no | Commercial Zapf fonts (OpenType) |
| type-fsf | no | no | no | Commercial Fontsite 500 fonts (Type 1) |
| type-ghz | no | no | no | Commercial Zapf fonts (Type 1) |
| type-gyr | no | no | no | The TEX Gyre project fonts (Type 1) |
| type-hgz | no | no | no | Commercial Zapf fonts (OpenType) |
| type-msw | no | no | no | Fonts that come with Microsoft Windows (Type 1) |
| type-omg | no | no | no | Omega free fonts (Type 1) |
| type-one | yes | no | no | Various free fonts (Type 1) |
| type-otf | no | yes | yes | Various free fonts (OpenType) |
| type-xtx | no | yes | no | Fonts that come with MacOSX (OpenType) |

**Table 5.6** The typescript source files that are part of ConTEXt.

Explicit loading one of those files is done via the macro `\usetypescriptfile`.

The predefined typescripts, the typefaces they define, the files in which they are contained in the ConTEXt distribution, and the encodings they support in MkII mode are listed in **table 5.7**. In the following section there is a table (**5.8**) that explains what font set each typescript attaches to each of the font styles.

```
\usetypescriptfile [...,*,...]

*   FILE
```

For example, the following

```
\usetypescriptfile[type-buy]
\usetypescript[lucida][texnansi]
\setupbodyfont[lucida,12pt]
```

will make pdfTEX use the Lucida Bright font family. Because this is a commercial font, this only works correctly if you have actually bought and installed the fonts. This uses the `texnansi` encoding because that is the preferred encoding of the actual fonts.

This is a good moment to explain a little trick: because the various `type-xxx` files define the building blocks for typescripts as well as the actual typescripts, it is sometimes possible to alter the effect of a typescript by loading an extra typescript file. For example,

```
\usetypescriptfile[type-gyr]
\usetypescript[palatino][ec]
```

| **Typescript** | **Typeface** | **File** | **Encodings** |
|---|---|---|---|
| antykwa-torunska | antykwa | type-one, type-otf | texnansi,ec,8r,t2a |
| fourier | fourier | type-one | ec |
| iwona | iwona | type-one, type-otf | texnansi,ec,8r,t2a |
| iwona-heavy | iwona-heavy | type-one, type-otf | texnansi,ec,8r,t2a |
| iwona-light | iwona-light | type-one, type-otf | texnansi,ec,8r,t2a |
| iwona-medium | iwona-medium | type-one, type-otf | texnansi,ec,8r,t2a |
| modern | modern | type-one, type-otf | texnansi,ec,qx,t5,default |
| modern-base | modern | type-one, type-otf | texnansi,ec,qx,t5,default,t2a/b/c |
| modernvariable | modernvariable | type-one, type-otf | texnansi,ec,qx,8r,t5 |
| palatino | palatino | type-one, type-otf | texnansi,ec,qx,8r,t5 |
| postscript | postscript | type-one, type-otf | texnansi,ec,qx,8r,t5 |
| times | times | type-one, type-otf | texnansi,ec,qx,8r,t5 |
| OmegaLGC | omlgc | type-omg | (unspecified) |
| cbgreek | cbgreek | type-cbg | (unspecified) |
| cbgreek-all | cbgreek-all | type-cbg | (unspecified) |
| cbgreek-medium | cbgreek-medium | type-cbg | (unspecified) |
| cow | cow | type-cow | default |
| sheep | sheep | type-cow | default |
| **lucida** | lucida | type-buy | texnansi,ec,8r |
| **lucidabfm** | lucida | type-buy | texnansi,ec,8r |
| **lucidabfm** | lucidabfm | type-buy | texnansi,ec,8r |
| **lucidaboldmath** | lucida | type-buy | texnansi,ec,8r |
| **lucidaboldmath** | lucidaboldmath | type-buy | texnansi,ec,8r |
| **optima** | optima | type-one | texnansi,ec,qx |
| **optima** | optima | type-ghz | texnansi,ec,qx |
| **optima-nova** | optima | type-ghz, type-hgz | texnansi,ec |
| **optima-nova-os** | optima-os | type-ghz, type-hgz | texnansi,ec |
| **palatino** | palatino | type-hgz | (cannot be used in MkII) |
| **palatino-informal** | palatino-informal | type-hgz | (cannot be used in MkII) |
| **palatino-light** | palatino-light | type-exp | (cannot be used in MkII) |
| **palatino-medium** | palatino-medium | type-exp | (cannot be used in MkII) |
| **palatino-normal** | palatino-normal | type-exp | (cannot be used in MkII) |
| **palatino-nova** | palatino | type-hgz | (cannot be used in MkII) |
| **palatino-sans** | palatino | type-hgz | (cannot be used in MkII) |

**Table 5.7** The typescripts. Typescripts that use commercial fonts are typeset in bold. Typescripts above the horizontal line are preloaded.

```
\setupbodyfont[palatino,12pt]
```

will result in pdfTEX using the Type 1 font Pagella from the TEX Gyre project instead of the older and less complete URW Palladio, because the definition of the building blocks for the `palatino` typescript that is in the `type-gyr` file overwrites the preloaded definition from the `type-one` file.

Two of the files in the ConTEXt distribution exist precisely for this reason:

`type-gyr.tex`
: maps the typical PostScript font names for the free URW fonts to the TEX Gyre set;

`type-akb.tex`
: maps the same names to the commercial Adobe fonts.

For the definitions in the second file to work, you also need to execute an extra typescript:

```
\usetypescriptfile [type-akb]
```

```
\usetypescript [adobekb] [ec]

\usetypescript [palatino] [ec]
\setupbodyfont[palatino,12pt]
```

### 5.8.3 Typeface definitions

Defining a typeface goes like this:

```
\starttypescript [palatino] [texnansi,ec,qx,t5,default]

\definetypeface[palatino] [rm] [serif][palatino] [default]
\definetypeface[palatino] [ss] [sans] [modern]   [default] [rscale=1.075]
\definetypeface[palatino] [tt] [mono] [modern]   [default] [rscale=1.075]
\definetypeface[palatino] [mm] [math] [palatino] [default]

\stoptypescript
```

This defines a typescript named `palatino` in five different encodings. When this typescript is executed via `\usetypescript`, it will define four typefaces, one of each of the four basic styles `rm`, `ss`, `tt`, and `mm`.

```
\definetypeface [.1.] [.2.] [.3.] [.4.] [.5.] [.6.]
                                        OPTIONAL OPTIONAL
1   TEXT

2   rm ss tt mm hw cg

3   IDENTIFIER

4   IDENTIFIER

5   IDENTIFIER

6   features  =  IDENTIFIER
    rscale    =  NUMBER
    encoding  =  IDENTIFIER
    text      =  IDENTIFIER
```

The third and fourth arguments to `\definetypeface` are pointers to already declared font sets; these are defined elsewhere. **Table 5.8** gives the full list of predefined typescripts (the first argument of `\starttypescript`) and font sets that are attached to the styles (the third and fourth argument of each `\definetypeface`).

The names in the third argument (like `serif` and `sans`) do *not* have the same meaning as the names used in `\setupbodyfont`. Inside `\setupbodyfont`, they were keywords that were internally remapped to one of the two-letter internal styles. Inside `\definetypeface`, they are nothing more than convenience names that are attached to a group of fonts by the person that wrote the font definition. They only reflect a grouping that the person believed that could be a single font style. Oftentimes, these names are identical to the official style keywords, just as the typescript and typeface names are often the same, but there can be (and sometimes are) different names altogether.

How to define your own font sets will be explained in the next chapter, but there are quite a few predefined font sets that come with ConTEXt; these are all listed in the four tables **5.9**, **5.10**, **5.11**, and **5.12**.

| **Typescript** | **Style rm** | **Style ss** | **Style tt** | **Style mm** |
|---|---|---|---|---|
| OmegaLGC | omega | – | omega | – |
| antykwa-torunska | antykwa-torunska | modern | modern | antykwa-torunska |
| cbgreek | cbgreek | cbgreek | cbgreek | – |
| cbgreek-all | cbgreek | cbgreek | cbgreek | – |
| cbgreek-medium | cbgreek | cbgreek | cbgreek | – |
| cow | cow | cow serif | modern | cow |
| fallback | modern | modern | modern | modern |
| fourier | fourier | modern | modern | fourier |
| iwona | modern | iwona | modern | iwona |
| iwona-heavy | modern | iwona-heavy | modern | iwona-heavy |
| iwona-light | modern | iwona-light | modern | iwona-light |
| iwona-medium | modern | iwona-medium | modern | iwona-medium |
| lucida | lucida | lucida | lucida | lucida |
| lucidabfm | lucida | lucida | lucida | lucida bfmath |
| lucidaboldmath | lucida | lucida | lucida | lucida boldmath |
| modern | modern | modern | modern | modern |
| modern-base | (computer-)modern | (computer-)modern | (computer-)modern | (computer-)modern |
| modernvariable | simple | modern | modern | modern |
| optima | palatino | optima-nova | modern | palatino |
| optima-nova | optima-nova sans | optima-nova | latin-modern | latin-modern |
| optima-nova-os | optima-nova-os sans | optima-nova-os | latin-modern | latin-modern |
| palatino | palatino-nova | palatino-sans | latin-modern | latin-modern |
| palatino | palatino | modern | modern | palatino |
| palatino-informal | palatino-nova | palatino-informal | latin-modern | latin-modern |
| palatino-light | palatino-nova | palatino-sans-light | latin-modern | latin-modern |
| palatino-medium | palatino-nova | palatino-sans-medium | latin-modern | latin-modern |
| palatino-normal | palatino-nova | palatino-sans-normal | latin-modern | latin-modern |
| palatino-nova | palatino-nova | palatino-sans | latin-modern | latin-modern |
| palatino-sans | palatino-nova | palatino-sans | latin-modern | latin-modern |
| postscript | times | helvetica | courier | times |
| sheep | sheep | sheep serif | modern | sheep |
| times | times | helvetica | modern | times |

**Table 5.8** The typescripts.
Unless stated otherwise, style **rm** uses a group named serif, style **ss** uses sans, style **tt** uses mono, and style **mm** uses math. A single dash in a cell means that the typescript does not define that style; you should refrain from using the style. The lucida, lucidabfm, and lucidaboldmath typescripts also define **hw** and **cg** as 'lucida handwring' and 'lucida calligraphy'. The modern-base typescript switches back to computer-modern for a few legacy encodings: t2a, t2b, and t2c.

For everything to work properly in MkII, the predefined font sets also have to have an encoding attached, you can look those up in the relevant tables as well.

The fifth argument to `\definetypeface` specifies specific font size setups (if any), these will be covered in section ?? in the next chapter. Almost always, specifying `default` will suffice.

The optional sixth argument is used for tweaking font settings like the specification of font features or adjusting parameters. In this case, the two `modern` font sets are loaded with a small magnification, this evens out the visual heights of the font styles.

A note for the lazy: if the sixth argument is not given and the fifth argument happens to be `default`, then the fifth argument can be omitted as well.

There are four possible keys in the sixth argument:

| **Identifier** | **file** | **Encodings** | **Supported styles** |
|---|---|---|---|
| modern | type-one | ec, qx, texnansi, t5 | serif, sans, mono, math, boldmath, bfmath |
| latin-modern | type-one | ec, qx, texnansi, t5 | serif, sans, mono, math, boldmath, bfmath |
| computer-modern | type-one | t2a/b/c | serif, sans, mono, math, boldmath, bfmath |
| simple | type-one | – synonyms only – | serif |
| concrete | type-one | – hardcoded – | serif |
| euler | type-one | – hardcoded – | math, boldmath, bfmath |
| ams | type-one | – hardcoded – | math |
| fourier | type-one | ec | math, serif |
| courier | type-one | 8r, ec, qx, texnansi, t5 | mono |
| helvetica | type-one | 8r, ec, qx, texnansi, t5 | sans |
| times | type-one | 8r, ec, qx, texnansi, t5 | serif, math |
| palatino | type-one | 8r, ec, qx, texnansi, t5 | serif, math |
| bookman | type-one | 8r, ec, qx, texnansi, t5 | serif |
| schoolbook | type-one | 8r, ec, texnansi, t5 | serif |
| chancery | type-one | 8r, ec, qx, texnansi | calligraphy |
| charter | type-one | 8r, ec, texnansi | serif |
| utopia | type-one | ec, texnansi | serif |
| antykwa-torunska | type-one | ec, qx, texnansi, t5, t2a/b/c, greek | serif, math |
| antykwa-torunska-light | type-one | ec, qx, texnansi, t5, t2a/b/c, greek | serif, math |
| antykwa-torunska-cond | type-one | ec, qx, texnansi, t5, t2a/b/c, greek | serif, math |
| antykwa-torunska-lightcond | type-one | ec, qx, texnansi, t5, t2a/b/c, greek | serif, math |
| antykwa-poltawskiego | type-one | 8r, ec, texnansi | serif |
| iwona | type-one | ec, qx, texnansi, t5 | sans, math |
| iwona-light | type-one | ec, qx, texnansi, t5 | sans, math |
| iwona-medium | type-one | ec, qx, texnansi, t5 | sans, math |
| iwona-heavy | type-one | ec, qx, texnansi, t5 | sans, math |
| iwona-cond | type-one | ec, qx, texnansi, t5 | sans |
| iwona-light-cond | type-one | ec, qx, texnansi, t5 | sans |
| iwona-medium-cond | type-one | ec, qx, texnansi, t5 | sans |
| iwona-heavy-cond | type-one | ec, qx, texnansi, t5 | sans |
| kurier | type-one | ec, qx, texnansi, t5 | sans, math |
| kurier-light | type-one | ec, qx, texnansi, t5 | sans, math |
| kurier-medium | type-one | ec, qx, texnansi, t5 | sans, math |
| pagella | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| palatino | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| termes | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| times | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| bonum | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| bookman | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| schola | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| schoolbook | type-gyr | ec, qx, texnansi, t5, t2a/b/c | serif |
| heros | type-gyr | ec, qx, texnansi, t5, t2a/b/c | sans |
| helvetica | type-gyr | ec, qx, texnansi, t5, t2a/b/c | sans |
| adventor | type-gyr | ec, qx, texnansi, t5, t2a/b/c | sans |
| cursor | type-gyr | ec, qx, texnansi, t5, t2a/b/c | mono |
| courier | type-gyr | ec, qx, texnansi, t5, t2a/b/c | mono |
| omega | type-omg | – hardcoded – | naskh, serif, mono |
| cbgreek | type-cbg | – hardcoded – | serif, sans, mono |
| cbgreek-medium | type-cbg | – hardcoded – | serif, sans, mono |
| cbgreek-all | type-cbg | – hardcoded – | serif, sans, mono |
| cow | type-cow | – hardcoded – | math, serif |
| sheep | type-cow | – hardcoded – | math, serif |

**Table 5.9** The predefined body font identifiers for free Type 1 and MetaFont fonts

| **Identifier** | **file** | **Encodings** | **Supported styles** |
|---|---|---|---|
| lucida | type-buy | 8r, ec, texnansi | serif, sans, mono, handwriting, calligraphy, math, boldmath, bfmath, casual, fax |
| informal | type-buy | – hardcoded – | casual, math |
| officina | type-buy | 8r, ec, texnansi | serif, sans |
| meta | type-buy | 8r, ec, texnansi | serif, sans, expert |
| meta-medium | type-buy | 8r, ec, texnansi | sans |
| meta-lf | type-buy | 8r, ec, texnansi | sans |
| meta-book | type-buy | 8r, ec, texnansi | sans |
| meta-book-lf | type-buy | 8r, ec, texnansi | sans |
| meta-bold | type-buy | 8r, ec, texnansi | sans |
| meta-bold-lf | type-buy | 8r, ec, texnansi | sans |
| meta-normal | type-buy | 8r, ec, texnansi | sans |
| meta-normal-lf | type-buy | 8r, ec, texnansi | sans |
| meta-medium | type-buy | 8r, ec, texnansi | sans |
| meta-medium-lf | type-buy | 8r, ec, texnansi | sans |
| meta-black | type-buy | 8r, ec, texnansi | sans |
| meta-black-lf | type-buy | 8r, ec, texnansi | sans |
| univers | type-buy | 8r, ec, texnansi | sans |
| univers-light | type-buy | 8r, ec, texnansi | sans |
| univers-black | type-buy | 8r, ec, texnansi | sans |
| mendoza | type-buy | 8r, ec, texnansi | serif |
| frutiger | type-buy | 8r, ec, texnansi | sans |
| kabel | type-buy | 8r, ec, texnansi | sans |
| thesans | type-buy | 8r, ec, texnansi | sans, mono, expert |
| sabon | type-buy | 8r, ec, texnansi | serif |
| stone | type-buy | ec, texnansi | serif, sans |
| stone-oldstyle | type-buy | – synonyms only – | serif, sans |
| industria | type-buy | ec, texnansi | sans |
| bauhaus | type-buy | ec, texnansi | sans |
| swift | type-buy | ec, texnansi | serif |
| swift-light | type-buy | – synonyms only – | serif |
| syntax | type-buy | ec, texnansi | sans |
| linoletter | type-buy | ec, texnansi | serif |
| zapfino | type-ghz | 8r, ec, texnansi | serif, handwriting |
| palatino-sans-light | type-exp | texnansi, ec | sans |
| palatino-sans-normal | type-exp | texnansi, ec | sans |
| palatino-sans-medium | type-exp | texnansi, ec | sans |
| opus | type-fsf | 8r, ec, texnansi | sans |
| typewriter | type-fsf | 8r, ec, texnansi | mono |
| garamond | type-fsf | 8r, ec, texnansi | serif |
| optima | type-ghz | 8r, ec, texnansi | sans |
| optima-nova | type-ghz | 8r, ec, texnansi | sans |
| optima-nova-os | type-ghz | 8r, ec, texnansi | sans |
| optima-nova-light | type-ghz | 8r, ec, texnansi | sans |
| optima-nova-medium | type-ghz | 8r, ec, texnansi | sans |
| palatino | type-ghz | 8r, ec, texnansi | serif |
| palatino-nova | type-ghz | 8r, ec, texnansi | serif |
| palatino-nova-os | type-ghz | 8r, ec, texnansi | serif |
| palatino-nova-light | type-ghz | 8r, ec, texnansi | serif |
| palatino-nova-medium | type-ghz | 8r, ec, texnansi | serif |
| aldus-nova | type-ghz | 8r, ec, texnansi | serif |
| melior | type-ghz | 8r, ec, texnansi | serif |
| verdana | type-msw | texnansi | sans |
| arial | type-msw | texnansi | sans |

**Table 5.10** The predefined body font identifiers for commercial Type 1 fonts

| Identifier | file | Supported styles | Identifier | file | Supported styles |
|---|---|---|---|---|---|
| modern | type-otf | serif, sans, mono, math, boldmath, bfmath | palatino | type-otf | serif, math |
| | | | times | type-otf | serif, math |
| | | | bookman | type-otf | serif |
| latin-modern | type-otf | serif, sans, mono, math, boldmath, bfmath | schoolbook | type-otf | serif |
| | | | chancery | type-otf | calligraphy |
| | | | helvetica | type-otf | sans |
| modern-vari | type-otf | mono | courier | type-otf | mono |
| latin-modern-vari | type-otf | mono | antykwa-torunska | type-otf | serif, math |
| modern-cond | type-otf | mono | antykwa-torunska-light | type-otf | serif, math |
| latin-modern-cond | type-otf | mono | antykwa-torunska-cond | type-otf | serif, math |
| computer-modern | type-otf | serif, sans, mono, math, boldmath, bfmath | antykwa-torunska-lightcond | type-otf | serif, math |
| | | | antykwa-poltawskiego | type-otf | serif |
| | | | iwona-light | type-otf | sans, math |
| concrete | type-otf | serif | iwona | type-otf | sans, math |
| euler | type-otf | math, boldmath, bfmath | iwona-medium | type-otf | sans, math |
| | | | iwona-heavy | type-otf | sans, math |
| ams | type-otf | math | iwona-cond | type-otf | sans |
| pagella | type-otf | serif | iwona-light-cond | type-otf | sans |
| termes | type-otf | serif | iwona-medium-cond | type-otf | sans |
| bonum | type-otf | serif | iwona-heavy-cond | type-otf | sans |
| schola | type-otf | serif | kurier | type-otf | sans, math |
| chorus | type-otf | serif | kurier-light | type-otf | sans, math |
| heros | type-otf | sans | kurier-medium | type-otf | sans, math |
| adventor | type-otf | sans | charter | type-otf | serif |
| cursor | type-otf | sans | gentium | type-xtx | serif |

**Table 5.11** The predefined body font identifiers for free Opentype fonts

| Identifier | file | Supported styles | Identifier | file | Supported styles |
|---|---|---|---|---|---|
| zapfino | type-hgz | serif, handwriting | times | type-xtx | serif |
| optima-nova | type-hgz | sans | palatino | type-xtx | serif |
| optima-nova-os | type-hgz | sans | helvetica | type-xtx | sans |
| optima-nova-light | type-hgz | sans | courier | type-xtx | mono |
| optima-nova-medium | type-hgz | sans | hoefler | type-xtx | serif |
| palatino-nova | type-hgz | serif | lucidagrande | type-xtx | sans |
| palatino-nova-os | type-hgz | serif | optima | type-xtx | sans |
| palatino-nova-light | type-hgz | serif | gillsans | type-xtx | sans |
| palatino-nova-medium | type-hgz | serif | gillsanslt | type-xtx | sans |
| palatino-sans | type-hgz | sans | zapfino | type-xtx | handwriting, serif |
| palatino-informal | type-hgz | sans | applechancery | type-xtx | calligraphy, serif |
| melior | type-hgz | serif | timesnewroman | type-xtx | serif |
| – all four-variant fonts – | type-xtx | Xserif | arial | type-xtx | sans |
| – all four-variant fonts – | type-xtx | Xsans | lucida | type-xtx | serif, sans, mono, handwriting, fax, calligraphy |
| – all four-variant fonts – | type-xtx | Xmono | | | |

**Table 5.12** The predefined body font identifiers for commercial Opentype fonts

| **key** | **default value** | explanation |
|---|---|---|
| `rscale` | 1 | a scaling factor for this typescript relative to the selected body font size |
| `encoding` | `\defaultencoding` | the encoding for the typeface, normally inherited from the typescript automatically |

| | |
|---|---|
| `features` | this applies a predefined font feature set (see **section 5.10**) |
| `text` | sets up the forced math text style |

If you look closely, in **table 5.12** you will notice three very special items: `Xserif`, `Xsans` and `Xmono`. These belong to a special XƎTEX-only trick called 'wildcard typescripts'.

XƎTEX offers some nice features in terms of automatically finding related fonts in a family, namely the italic, bold, and bolditalic alternatives. To take advantage of that, there's a set of wildcard typescripts that take an arbitrary Macintosh font name as input, and provide as many of the alternatives it can find. To set these typescripts (and the calling conventions) apart from the familiar ones, the typescripts are identified with `Xserif`, `Xsans`, and `Xmono`.

To call these special typescripts, it's most convenient to define a typeface that uses these features. The named font slot should contain the display name of the Regular alternative (not the family name) of the font in question. For example, you could have the following mix:

```
\starttypescript[myface]
\definetypeface[myface][rm][Xserif][Baskerville]   [default]
\definetypeface[myface][tt][Xmono] [Courier]       [default][rscale=.87]
\definetypeface[myface][ss][Xsans] [Optima Regular][default]
\stoptypescript
```

As you can see, you can activate relative scaling of face sizes. The above definitions look very much like any other typeface definition, except that the serif/sans/mono identifier is preceded with X, and that there is no underlying "Optima Regular" defined anywhere. Those missing bits of the definitions are handled by typescript and XƎTEX magic.

## 5.9 Body font environments

Earlier we saw that within a single body font there are in fact different font sizes such as super- and subscripts. The relations between these sizes are defined by *body font environments*.

For all regular font sizes, environments are predefined that fulfill their purpose adequately. However when you want to do some extra defining yourself there is:

```
\definebodyfontenvironment [.1.] [.2.] [..,.=.,..]
                           OPTIONAL         OPTIONAL

1   IDENTIFIER

2   5pt ... 12pt default

3   text            =  DIMENSION
    script          =  DIMENSION
    scriptscript    =  DIMENSION
    x               =  DIMENSION
    xx              =  DIMENSION
    a               =  DIMENSION
    b               =  DIMENSION
    c               =  DIMENSION
    d               =  DIMENSION
    small           =  DIMENSION
    big             =  DIMENSION
    interlinespace  =  DIMENSION
    em              =  normal bold slanted boldslanted type cap small... COMMAND
```

The first argument is optional, and specifier the typeface identifier that this particular body font environment setup is for. It defaults to the current typeface.

The second argument is the size of the body font environment that is being defined. This argument is not really optional, the macro syntax description is a little misleading.

The third argument once again is optional, and contains the actual settings as key-value pairs. If it is missing, defaults will be guessed at by ConTEXt itself. Although the macro syntax says the type is `DIMENSION`, floating point numbers are also acceptable. Such numbers are multipliers that are applied to the font size when the body font environment is applied.

| | |
|---|---|
| `text` | Math text size or multiplier (default is `1.0`) |
| `script` | Math script size (default is `0.7`) |
| `scriptscript` | Math scriptscript size (default is `0.5`) |
| `x` | The size used for commands like `\tfx` (default is `0.8`) |
| `xx` | The size used for the `\tfxx` command (default is `0.6`) |
| `a` | The size for commands like `\tfa` (default is `1.200`) |
| `b` | The size for commands like `\tfb` (default is `1.440`) |
| `c` | The size for commands like `\tfa` (default is `1.728`) |
| `d` | The size for commands like `\tfd` (default is `2.074`) |
| `big` | The 'larger' font size (default is `1.2`) |
| `small` | The 'smaller' font size (default is `0.8`) |
| `interlinespace` | Distance between lines in a paragraph (default is `2.8ex`) |
| `em` | The style to use for emphasis (default is `slanted`) |

So, when you want to have a somewhat bigger fontsize for just a few words (e.g. for a book title) you can type:

```
\definebodyfontenvironment [24pt]
\switchtobodyfont[24pt]
```

For longer stretches of text you will probably want to set up most of the values explicitly, using something like this

```
\definebodyfontenvironment
  [22pt]
  [        text=22pt,
         script=17.3pt,
   scriptscript=14.4pt,
              x=17.3pt,
             xx=14.4pt,
            big=28pt,
          small=17.3pt]
```

To tweak already defined sizes, there is an accompanying setup command with the same parameter conventions:

```
\setupbodyfontenvironment [.1.] [.2.] [..,.=3.,..]
                                 OPTIONAL        OPTIONAL
1   inherits from \definebodyfontenvironment
2   inherits from \definebodyfontenvironment
3   inherits from \definebodyfontenvironment
```

## 5.10 Font feature sets

As mentioned already, some fonts contain extra information besides the actual glyph shapes. In traditional TEX fonts, the extra information is roughly limited to kerning pairs and ligature information, and both of these 'features' are automatically applied to the text that is being typeset. In the odd case where one of the two needs to be suppressed, a little bit of macro trickery can do the job without too many complicating factors.

But with the new OpenType font format that is used by XETEX and LuaTEX, the list of possible features has increased enormously. OpenType fonts have not just kerning information and ligature information, but there can also be other features like optional oldstyle figures, caps and smallcaps glyphs, decorative swashes, etc. all inside a single font file.

Not only that, but some of these features are not even supposed to be active all the time. Certain features should only be activated if the user asks for it, while other features depend on the script and language that is in use for the text that is being typeset.

This is a big step forward in that there are now far fewer fonts needed to achieve the same level of quality than before, all that extra font information also poses a big challenge for macro writers. And add to that the fact that at the core, the two engines (XETEX and LuaTEX) handle OpenType fonts completely different from each other.

ConTEXt has a new subsystem called 'font features' to create order in this forest of features. The most important command is `\definefontfeature`. This command can be used to group various font features under a single symbolic name, that can then be used as e.g. the argument to the `features` key of `\definetypeface`.

```
\definefontfeature [.1.] [.2.] [.=3.]
                         OPTIONAL
1   TEXT
2   IDENTIFIER
3   compose   =  no yes
    mode      =  node base
    tlig      =  no yes
    trep      =  no yes
    script    =  IDENTIFIER
    language  =  IDENTIFIER
    ..tag..   =  no yes
```

```
\definefontfeature
   [default-base]
   [script=latn,language=dflt,liga=yes,kern=yes,tlig=yes,trep=yes]
```

As you can probably guess, the first argument is the symbolic name that is being defined. The second argument is a mix of a-hoc settings and OpenType font features.

| | |
|---|---|
| `compose` | Use fallback composition in MkIV (experimental, undocumented) |
| `protrusion` | Character protrusion in MkIV (see **section 5.14**) |
| `expansion` | Character expansion in MkIV (see **section 5.14**) |
| `script` | An OpenType script identifier |
| `language` | An OpenType script language identifier |
| `tlig` | A virtual feature for legacy (TEX-style) automatic ligatures (for compatibility, there is an alias for this key called `texligatures`) |
| `trep` | A virtual feature for legacy (TEX-style) automatic ligatures (for compatibility, there is an alias for this key called `texquotes`) (only works in MkIV) |
| `mode` | Processing mode for MkIV. `node` and `base` allowed, `base` is default |
| `<tag>` | Any OpenType feature tag is acceptable, but in MkIV only a 'known' subset actually has any effect, and then only in `node` mode. This list is given in **table 5.13**. In XeTEX, processing depends on the internal subengine that is used by XeTEX, and that is outside of ConTEXt's control. |

A few fontfeatures are predefined by context:

```
default    liga=yes,kern=yes,tlig=yes,trep=yes
smallcaps  liga=yes,kern=yes,tlig=yes,trep=yes,smcp=yes
oldstyle   liga=yes,kern=yes,tlig=yes,trep=yes,onum=yes
```

At the moment, `smallcaps` and `oldstyle` only work in XeTEX (in MkIV, it would need an extra `mode=node` pair).

## 5.11 Displaying the current font setup

With the command `\showbodyfont` an overview is generated of the available characters, and an overview of the different fontsizes within a family can be summoned with `\showbodyfontenvironment`.

```
\showbodyfont [...,*...]
                 OPTIONAL
*   inherits from \setupbodyfont
```

```
\showbodyfontenvironment [...,*...]
                            OPTIONAL
*   inherits from \setupbodyfont
```

Specifying actual `IDENTIFIER`s to these commands is currently unreliable because they internally are still counting on an older system of body font definitions, but you can safely use a size argument to get the information for the current font set.

Below an example of the possible output is shown, for `\showbodyfont[12pt]`

| | |
|---|---|
| `aalt` | Access All Alternates |
| `abvf` | Above-Base Forms |
| `abvm` | Above-Base Mark Positioning |
| `abvs` | Above-Base Substitutions |
| `afrc` | Alternative Fractions |
| `akhn` | Akhands |
| `blwf` | Below-Base Forms |
| `blwm` | Below-Base Mark Positioning |
| `blws` | Below-Base Substitutions |
| `c2pc` | Petite Capitals From Capitals |
| `c2sc` | Small Capitals From Capitals |
| `calt` | Contextual Alternates |
| `case` | Case-Sensitive Forms |
| `ccmp` | Glyph Composition/Decomposition |
| `cjct` | Conjunct Forms |
| `clig` | Contextual Ligatures |
| `cpsp` | Capital Spacing |
| `cswh` | Contextual Swash |
| `curs` | Cursive Positioning |
| `dflt` | Default Processing |
| `dist` | Distances |
| `dlig` | Discretionary Ligatures |
| `dnom` | Denominators |
| `expt` | Expert Forms |
| `falt` | Final glyph Alternates |
| `fina` | Terminal Forms |
| `fin2` | Terminal Forms #2 |
| `fin3` | Terminal Forms #3 |
| `frac` | Fractions |
| `fwid` | Full Width |
| `half` | Half Forms |
| `haln` | Halant Forms |
| `halt` | Alternate Half Width |
| `hist` | Historical Forms |
| `hkna` | Horizontal Kana Alternates |
| `hlig` | Historical Ligatures |
| `hngl` | Hangul |
| `hojo` | Hojo Kanji Forms |
| `hwid` | Half Width |
| `init` | Initial Forms |
| `isol` | Isolated Forms |
| `ital` | Italics |
| `jalt` | Justification Alternatives |
| `jp04` | JIS2004 Forms |
| `jp78` | JIS78 Forms |
| `jp83` | JIS83 Forms |
| `jp90` | JIS90 Forms |
| `kern` | Kerning |
| `lfbd` | Left Bounds |
| `liga` | Standard Ligatures |
| `ljmo` | Leading Jamo Forms |
| `lnum` | Lining Figures |
| `locl` | Localized Forms |
| `mark` | Mark Positioning |
| `medi` | Medial Forms |
| `med2` | Medial Forms #2 |
| `mgrk` | Mathematical Greek |
| `mkmk` | Mark to Mark Positioning |
| `mset` | Mark Positioning via Substitution |
| `nalt` | Alternate Annotation Forms |
| `nlck` | NLC Kanji Forms |
| `nukt` | Nukta Forms |
| `numr` | Numerators |
| `onum` | Old Style Figures |
| `opbd` | Optical Bounds |
| `ordn` | Ordinals |
| `ornm` | Ornaments |
| `palt` | Proportional Alternate Width |
| `pcap` | Petite Capitals |
| `pnum` | Proportional Figures |
| `pref` | Pre-base Forms |
| `pres` | Pre-base Substitutions |
| `pstf` | Post-base Forms |
| `psts` | Post-base Substitutions |
| `pwid` | Proportional Widths |
| `qwid` | Quarter Widths |
| `rand` | Randomize |
| `rkrf` | Rakar Forms |
| `rlig` | Required Ligatures |
| `rphf` | Reph Form |
| `rtbd` | Right Bounds |
| `rtla` | Right-To-Left Alternates |
| `ruby` | Ruby Notation Forms |
| `salt` | Stylistic Alternates |
| `sinf` | Scientific Inferiors |
| `size` | Optical Size |
| `smcp` | Small Capitals |
| `smpl` | Simplified Forms |
| `ss01` | Stylistic Set 1 |
| `ss02` | Stylistic Set 2 |
| `ss03` | Stylistic Set 3 |
| `ss04` | Stylistic Set 4 |
| `ss05` | Stylistic Set 5 |
| `ss06` | Stylistic Set 6 |
| `ss07` | Stylistic Set 7 |
| `ss08` | Stylistic Set 8 |
| `ss09` | Stylistic Set 9 |
| `ss10` | Stylistic Set 10 |
| `ss11` | Stylistic Set 11 |
| `ss12` | Stylistic Set 12 |
| `ss13` | Stylistic Set 13 |
| `ss14` | Stylistic Set 14 |
| `ss15` | Stylistic Set 15 |
| `ss16` | Stylistic Set 16 |
| `ss17` | Stylistic Set 17 |
| `ss18` | Stylistic Set 18 |
| `ss19` | Stylistic Set 19 |
| `ss20` | Stylistic Set 20 |
| `subs` | Subscript |
| `sups` | Superscript |
| `swsh` | Swash |
| `titl` | Titling |
| `tjmo` | Trailing Jamo Forms |
| `tnam` | Traditional Name Forms |
| `tnum` | Tabular Figures |
| `trad` | Traditional Forms |
| `twid` | Third Widths |
| `unic` | Unicase |
| `valt` | Alternate Vertical Metrics |
| `vatu` | Vattu Variants |
| `vert` | Vertical Writing |
| `vhal` | Alternate Vertical Half Metrics |
| `vjmo` | Vowel Jamo Forms |
| `vkna` | Vertical Kana Alternates |
| `vkrn` | Vertical Kerning |
| `vpal` | Proportional Alternate Vertical Metrics |
| `vrt2` | Vertical Rotation |
| `zero` | Slashed Zero |

**Table 5.13** The OpenType features that are understood by MkIV in `mode=node` processing mode

| [palatino] [12pt] | | | | | | | | | | | | | \mr : *Ag* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | \tf | \sc | \sl | \it | \bf | \bs | \bi | \tfx | \tfxx | \tfa | \tfb | \tfc | \tfd |
| \rm | Ag | AG | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |
| \ss | Ag | Ag | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |
| \tt | Ag | AG | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |

And the output of `\showbodyfontenvironment[12pt]` is:

| [palatino] [12pt] | | | | | | | |
|---|---|---|---|---|---|---|---|
| text | script | scriptscript | x | xx | small | big | interlinespace |
| 20.7pt | 14.4pt | 12pt | 17.3pt | 14.4pt | 17.3pt | 20.7pt | |
| 17.3pt | 12pt | 10pt | 14.4pt | 12pt | 14.4pt | 20.7pt | |
| 14.4pt | 11pt | 9pt | 12pt | 10pt | 12pt | 17.3pt | |
| 12pt | 9pt | 7pt | 10pt | 8pt | 10pt | 14.4pt | |
| 11pt | 8pt | 6pt | 9pt | 7pt | 9pt | 12pt | |
| 10pt | 7pt | 5pt | 8pt | 6pt | 8pt | 12pt | |
| 9pt | 7pt | 5pt | 7pt | 5pt | 7pt | 11pt | |
| 8pt | 6pt | 5pt | 6pt | 5pt | 6pt | 10pt | |
| 7pt | 6pt | 5pt | 6pt | 5pt | 5pt | 9pt | |
| 6pt | 5pt | 5pt | 5pt | 5pt | 5pt | 8pt | |
| 5pt | 5pt | 5pt | 5pt | 5pt | 5pt | 7pt | |
| 4pt | 4pt | 4pt | 4pt | 4pt | 4pt | 6pt | |

## 5.12 Math fonts

There are only a few font families in existence that can handle math properly because such fonts have to carry a complete set of characters and symbols for mathematical typesetting. Among these, the Computer Modern Roman distinguishes itself by its many design sizes; that really pays off when typesetting complicated math formulas.

Many TEX users have chosen TEX for its superb math typesetting.

This chapter will not go into any details but in math mode, the central concept is the *math family* (not to be confused with the *font families* discussed earlier). There are math families for `\bf`, `\it`, etc. as well as for the special math symbols. Within each family, there are always exactly three member fonts: `text`, `script` and `scriptscript`, or a normal, smaller and smallest font. The normal font size is used for running text and the smaller ones for sub and superscripts. The next example will show what the members of a math family can do.

```
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\rm 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\tf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\bf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\sl 6x^2$
```

When this is typeset you see this:

$\mathrm{x}^2 + \mathrm{x}^2 + x^2 + x^2 + \boldsymbol{x}^2 + \boldsymbol{x}^2 = 6x^2$
$\mathrm{x}^2 + \mathrm{x}^2 + x^2 + x^2 + \boldsymbol{x}^2 + \boldsymbol{x}^2 = 6\mathrm{x}^2$
$\mathrm{x}^2 + \mathrm{x}^2 + x^2 + x^2 + \boldsymbol{x}^2 + \boldsymbol{x}^2 = \boldsymbol{6x^2}$
$\mathrm{x}^2 + \mathrm{x}^2 + x^2 + x^2 + \boldsymbol{x}^2 + \boldsymbol{x}^2 = 6x^2$

As you can see, the alphabetic characters adapt to the selected font family but the symbols are all typeset in the same font regardless. Technically this means that the symbols are set in the fixed font family 0 whereas the alphabetic characters are typeset using variable family numbers.

Typesetting math formulas can also be done somewhat differently, as we will see in the next example.

```
$\tf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\sl\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bs\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\it\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bi\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
```

A new command is used: `\mf`, which stands for *math font*. This command takes care of the symbols in such a way that they are also set in the actually selected font, just like the characters.

$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$
$x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$
$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$
$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$
$\mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 + \mathrm{x}^2 = 6\mathrm{x}^2$

You should take into account that TEX typesets a formula as a whole. In some cases this means that setups at the end of the formula have an effect that starts already at the beginning of the formula.

For example, the exact location of `\mf` is not that important. We also could have typed:

```
$\bf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = \mf 6x^2$
```

There is much more to be said about math, but it is better to do that in chapter `??`, about math.

## 5.13 Em and Ex

In specifying dimensions we can distinguish physical units like `pt` and `cm` and internal units like `em` and `ex`. These last units are related to the actual fontsize. When you use these internal units in specifying for example horizontal and vertical spacing you don't have to do any recalculating when fonts are switched in the style definition.

Some insight in these units does not hurt. The width of an `em` is not the with of an M, but that of an — (an em–dash). When this glyph is not available in the font another value is used. **Table 5.14** shows some examples. We see that the width of a digit is about `.5em`. In Computer Modern Roman a digit is exactly half an em wide.

| `\tf` | `\bf` | `\sl` | `\tt` | `\ss` | `\tfx` |
|---|---|---|---|---|---|
| \|12\|<br>\|M\|<br>\| \|<br>\|—\| | **\|12\|**<br>**\|M\|**<br>\| \|<br>\|—\| | *\|12\|*<br>*\|M\|*<br>\| \|<br>\|—\| | `|12|`<br>`|M|`<br>\| \|<br>`|---|` | \|12\|<br>\|M\|<br>\| \|<br>\|—\| | \|12\|<br>\|M\|<br>\| \|<br>\|—\| |

**Table 5.14** The width of an `em`.

In most cases we use `em` for specifying width and `ex` for height. An `ex` equals the height of a lowercase x. **Table 5.15** shows some examples.

| `\tf` | `\bf` | `\sl` | `\tt` | `\ss` | `\tfx` |
|---|---|---|---|---|---|
| x | **x** | *x* | `x` | x | x |

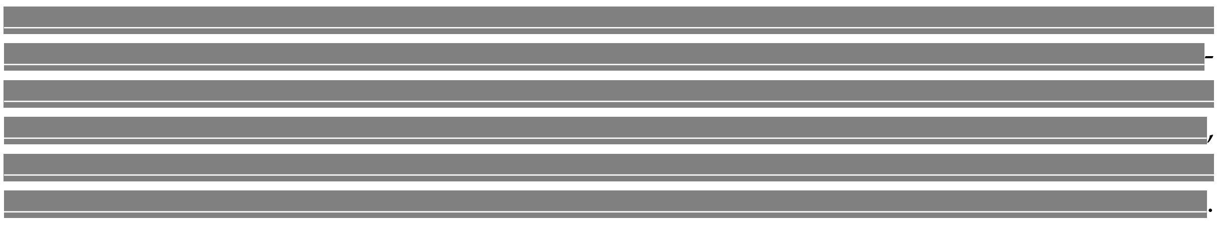
**Table 5.15** The height of an `ex`.

## 5.14 Font handling

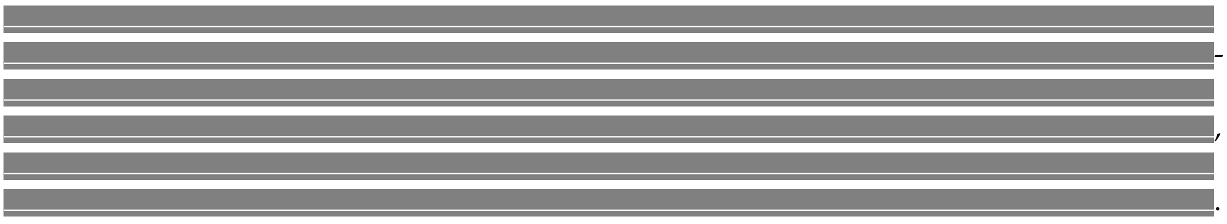
Almost all users of typesetting systems based on TEX do so because of the quality of the output it produces. pdfTEX (and through inheritance LuaTEX as well) contains a few extensions to the typesetting engine that make the output even better than the results achieved by Knuth's original TEX. Although the extensions are made available by pdfTEX, they are not limited to the pdf output, they will work with the dvi backend just as well. And when the extensions are defined but not enabled, then the typeset output is 100% identical to when the feature is not present at all.

### 5.14.1 Character protrusion

In the following fake paragraph, you can see a hyphenation point, a secondary sentence, separated by a comma, and a last sentence, ending with a period. Miraculously, this paragraph fits into lines. Although exaggerated, these lines demonstrate that visually the hyphen and punctuation characters make the margin look ragged.

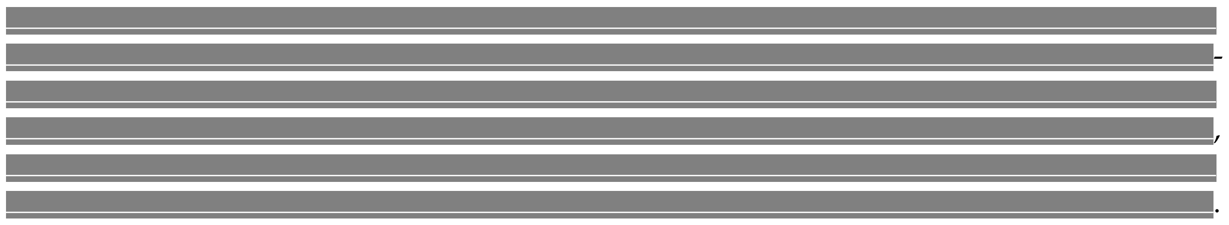


Before computers started to take over the traditional typesetter's job, it was common practice to move hyphens and punctuation into the margin, like in:



In this alternative, the margin looks less ragged, and this becomes more noticeable once you get aware of this phenomenon.

Sometimes, shifting the characters completely into the margin is too much for the sensitive eye, for instance with an italic font, where the characters already hang to the right. In such cases, we need to compromise.



pdfTEX (and LuaTEX, that has inherited this feature) has provisions to move characters into the margin when they end up at the end of a line. Such characters are called protruding characters. pdfTEX takes protruding into account when breaking a paragraph.

We will demonstrate protruding using a quote from Hermann Zapf's article "About micro–typography and the *hz*–program" in Electronic Publishing, vol 6 (3), 1993.

After TEX has typeset this paragraph (using a specific font size and line width) it may have constructed the following lines.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge
and information about the rules of typography from books, from computer magazines or the instruction manuals
which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was
in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by
their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic
from now on.

As you can see, the height and depth of the lines depend on the characters, but their width equals what TEX calls `\hsize`. However, the natural width of the lines may differ from `\hsize`.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge
and information about the rules of typography from books, from computer magazines or the instruction manuals
which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was
in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by
their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic
from now on.

Here the inter–word space is fixed to what TEX considers to be a space. This example also demonstrates that TEX does not have spaces, but stretches the white area between words to suit its demands. When breaking lines, TEX's mind is occupied by boxes, glue and penalties, or in more common language: (parts of) words, stretchable white space, and more or less preferred breakpoints.

Coming back to the use of
typefaces in electronic publishing:
many of the new typographers re-
ceive their knowledge and infor-
mation about the rules of typog-
raphy from books, from computer
magazines or the instruction man-
uals which they get with the pur-
chase of a PC or software. There
is not so much basic instruction,
as of now, as there was in the old
days, showing the differences be-
tween good and bad typographic
design. Many people are just fas-
cinated by their PC's tricks, and
think that a widely–praised pro-
gram, called up on the screen,
will make everything automatic
from now on.

This time we have enabled pdfTEX's protruding mechanism. The characters that stick into the margin are taken into account when breaking the paragraph into lines, but in the final result, they do not count in the width. Here we used an ugly three column layout so that we got a few more hyphens to illustrate the principle.

When that same text is typeset in the traditional way in two columns, it looks like this:

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

As you can see, the hyphens and punctuation fit snugly into the line and as a result the line endings look a bit ragged. With protrusion turned on, it looks like this:

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Now the punctuation protrudes a little into the margin. Although the margin is now geometrically uneven it looks straighter to the human eye because not so much whitespace 'pushes into' the text.

### 5.14.2 Font expansion

In typesetting the two characters hz are tightly connected to Hermann Zapf and the next couple of pages we will discuss a method for optimizing the look and feel of a paragraph using a mechanism that is inspired by his work. Although official qualified in pdfTEX as font adjusting, we will use the short qualification hz since this is how it is called in the pdfTEX community.

First, here is again the same example text that was used in the previous section, typeset using normal TEX–comptibale font settings:

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

The example below shows hz in action. This paragraph is typeset with hz enabled and has a more even spacing than the text above.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

The average reader will not notice the trick, but those sensitive to character shapes will see that some glyphs are widened slightly and others are narrowed slightly. Ideally the programs that built the glyph should be defined in such a way that this goes unnoticed, but in practice glyph programs are not that clever and so a brute force horizontal scaling is applied. As long as the used percentage is small, the distortion will go unnoticed and the paragraph will look slightly better because the whitespace distribution is more even.

### 5.14.3 Other font handlings

In addition to the two handlings documented in the previous paragraphs (protruding and hz), ConTEXt also provides the `noligs` handling (handy when one processes xml), `flexspacing` and `prespacing` (meant for languages like French that need spacing around for instance `:` and `;`). These handlings are experimental.

### 5.14.4 How to use font handlings

Before we go into the details of the actual extensions, let's see what is provided by ConTEXt as the user–level interface. The ConTEXt interface to those new features is through a subsystem called 'font handling', and at the top that subsystem is seamlessly integrated into the normal alignment macros.

For example, assuming the system is set up already to support protrusion, you can simply say

```
\setupalign[hanging]
```

to turn protrusion on. However, this will only work correctly if a number of special setups have taken place internally. The command `\setupalign` only toggles a switch, and the required setups have to be done elsewhere.

The list of font handling–related keys for `\setupalign` is:

| | |
|---|---|
| `hanging` | turns on character protrusion |
| `nohanging` | turns off character protrusion |
| `hz` | turns on font expansion |
| `nohz` | turns off font expansion |
| `spacing` | turns on special spacing rules |
| `nospacing` | turns off special spacing |

Largely because of the tight connection with the font itself, the method of defining and setting font handling is a little different between pdfTEX and MkIV.

### 5.14.5 Setting up font handlings in MkII

Now, let's move on to how to set up the system for font handling properly. Most of the underlying features of pdfTEX cannot be turned *merely* on or off, it is possible to tweak the machinery on the font as well as on the individual glyph level. You can define those settings all on your own, but ConTEXt comes with a handy set of predefined values.

| **name** | `\setupalign` | **description** |
|---|---|---|
| `pure` | `hanging` | full protrusion of only selected punctuation |
| `normal` | `hanging` | partial protrusion of punctuation and some asymmetrical letters |
| `hz` | `hz` | variable correction of character widths |
| `quality` | `hanging,hz` | combination of hz and pure |
| `highquality` | `hanging,hz` | combination of hz and normal |
| `flexspacing` | `spacing` | automatic extra spacing around various punctuation characters |
| `prespacing` | `spacing` | like flexspacing, but ignoring . and , and with smaller effects |
| `noligs` | `--` | suppresses ligatures; because this is irreversible it is not controlled via `\setupalign` |

You need to be aware of the fact that at the moment that you actually define a font, you need to tell what handling you want to apply.

Note: setting up font handling involves a few low-level font definition commands, so you may want to read the chapter about font definitions first.

Say that we want to hang only the serif fonts and say that we use Palatino as main typeface.

```
\setupfontsynonym [Serif] [handling=pure]
\definetypeface [palatino] [rm] [serif] [palatino] [default]
```

In the above example, the font loader is instructed to treat fonts with the virtual name `Serif` in a special way by applying the font handling named `pure`. After that, the typeface collection `palatino` is (re)defined and by that process the font tagged as `Serif` will get the 'hanging' settings attached it.

Now enable this typeface collection can be enabled by:

```
\setupbodyfont [palatino]
```

and finally, don't forget to turn on hanging by:

```
\setupalign [hanging]
```

However, this only takes care of the `Serif` font. Normally, that is the virtual name for the combination `\rm\tf`. If you also want the bold variants to hang, you have to add an extra line:

```
\setupfontsynonym [SerifBold] [handling=pure]
```

And so on for all the alternatives. This is tedious, so ConTEXt provides a shortcut. If you want to set all serif weights at once, you can call on a predefined typescript component before defining the typeface:

```
\usetypescript [serif] [handling] [pure]
```

for hanging punctuation, or for all characters:

```
\usetypescript [serif] [handling] [normal]
```

The full example then becomes:

```
\usetypescript [serif] [handling] [pure]
\definetypeface [palatino] [rm] [serif] [palatino] [default]
\setupbodyfont [palatino]
\setupalign [hanging]
```

The first argument can be one of three named typescript groups: `serif` (for the virtual font synonyms whose names begin with `Serif`), `sans` (for `Sans`), or `mono` (for `Mono`). The second argument should always be `handling`. The third argument has to be one of named font handlings that are listed in the table at the start of this section.

The typescripts that are used in these examples work by altering the font synonyms for virtual symbolic font names like `Serif` and `SerifBold` en bloc. They will even work with your own typescripts if (but only if) these typescripts use the same font naming conventions as the ConTEXt core.

The definition of font handlings is actually a two-step process. A named font handling consists of one or more handling vectors that have to be defined first, those are then combined under a single name.

This is not the right place to describe how to define the low-level vector definitions in detail, for that you are referred to the documented source of the main handling definition file `hand-def.tex`. But to give you an idea of what it looks like, here is a small excerpt of that file. The `pure` handling vector is defined as:

```
\startfonthandling [pure]

  \defineprotrudefactor , 0 1
  \defineprotrudefactor . 0 1
  \defineprotrudefactor : 0 1
  \defineprotrudefactor ; 0 1
  \defineprotrudefactor - 0 1

  \defineprotrudefactor hyphen 0 1
  \defineprotrudefactor endash 0 .5
  \defineprotrudefactor emdash 0 .33 % .5

\stopfonthandling
```

The `pure` font handling itself is then defined as follows:

```
\definefonthandling [pure] [pure] [type=hanging]
```

The `hz` setup runs along the same lines. First here is a vector:

```
\startfonthandling [hz]

  \defineadjustfactor A .5
  \defineadjustfactor B .7
  \defineadjustfactor C .7
  ...

\stopfonthandling
```

And then the definition of the hz handling is as follows:

```
\definefonthandling [hz] [hz,extended] [type=hz]
```

To wrap this up, here is the macro syntax for the font handling definition and setup.

```
\definefonthandling [.1.] [...,2,...] [.3.]

1   IDENTIFIER

2   IDENTIFIER

3   type    =  hanging hz spacing tag
    right   =  NUMBER
    left    =  NUMBER
    factor  =  NUMBER
    min     =  NUMBER
    max     =  NUMBER
    step    =  NUMBER
```

As you can see, the `\definefonthandling` command accepts three arguments. The first is the handling to be defined, the second is a list of handling vectors to be used, and the third sets up a number of settings.

| | |
|---|---|
| `type` | the type of this font handling feature, for use by `\setupalign` |
| `right` | used by `type=hanging`, default 1 |
| `left` | used by `type=hanging`, default 1 |
| `factor` | used by `type=spacing`, default 1 |
| `min` | used by `type=hz`, default 20 |
| `max` | used by `type=hz`, default 20 |
| `step` | used by `type=hz`, default 5 |

On top of the list at the beginning of this paragraph, a few more elaborate font handlings are also predefined:

```
\definefonthandling [purebold]         [pure] [type=hanging]
\definefonthandling [pureitalic]       [pure] [type=hanging,right=1.5]
\definefonthandling [pureslanted]      [pure] [type=hanging,right=1.5]
\definefonthandling [purebolditalic]   [pure] [type=hanging,right=1.5]
\definefonthandling [pureboldslanted]  [pure] [type=hanging,right=1.5]
```

The `right` parameter (there is also `left`) is a multiplication factor that is applied to the values in the associated vector. Such definitions can be more extensive, like:

```
\definefonthandling
  [normalitalic]
  [punctuation,alpha,extended]
  [type=hanging,right=1.5]
```

Here we have combined three vectors into one handling. For these extended font handlings, there are no predefined typescripts, so you either have to use the font synonyms directly, or define your own typescripts. Now, if you think this is overly complicated, you are probably right. Normally you will just invoke protruding handlings defined previously, but the mechanisms are there to fine–tune the handlings to your precise wishes.

In case you want to alter some of the settings of an already defined font handling, there is

```
\setupfonthandling [.1.] [.2.]

1   IDENTIFIER
2   inherits from \definefonthandling
```

The first argument is the handling to be altered, the second sets up the settings.

## 5.14.6 Setting up font handlings in MkIV

In MkIV, font handling is merged with the font features (because these already have a low-level connection to the font), so you can set up the font-side of things with the sixth argument of `\definetypeface`, like so:

```
\definefontfeature
    [hz] [default]
    [protrusion=pure, mode=node, script=latn]
\definetypeface [palatino] [rm] [serif] [palatino] [default] [features=hz]
\setupbodyfont [palatino]
\setupalign [hanging]
```

or by redefining the feature set that is used by the typescript you are using and then (re-)executing the typescript, like so:

```
\definefontfeature
    [default] [default]
    [protrusion=pure, expansion=quality, mode=node, script=latn]
\usetypescript[palatino]
\setupbodyfont [palatino]
\setupalign [hanging]
```

There is a list of predefined font handling feature values that you can use:

For protrusion, there is:

| **name** | **\setupalign** | **description** |
|---|---|---|
| pure | hanging | full protrusion of only selected punctuation |
| punctuation | hanging | partial protrusion of punctuation |
| alpha | hanging | partial of some asymmetrical letters |
| quality | hanging | the combination of punctuation and alpha |

For expansion, there is:

| **name** | **\setupalign** | **description** |
|---|---|---|
| quality | hz | variable correction of character widths |

These are defined in the file `font-ext.lua`. The low–level definitions look like

```
fonts.protrusions.vectors['pure'] = {
    [0x002C] = { 0, 1    }, -- comma
    [0x002E] = { 0, 1    }, -- period
    [0x003A] = { 0, 1    }, -- colon
    [0x003B] = { 0, 1    }, -- semicolon
    [0x002D] = { 0, 1    }, -- hyphen
    [0x2013] = { 0, 0.50 }, -- endash
    [0x2014] = { 0, 0.33 }, -- emdash
}
fonts.protrusions.classes['pure'] = {
    vector = 'pure', factor = 1
}
```

That was the complete definition of `protrusion=pure`. The key `classes` has the same function as the macro call `\definefonthandling` in MkII. It references the named vector `pure` and sets up a parameter.

For `protrusion`, there is only the one parameter `factor`, but for `expansion` there are a few more:

```
\startLUA
fonts.expansions.classes['quality'] = {
    stretch = 2, shrink = 2, step = .5, vector = 'default', factor = 1
}
fonts.expansions.vectors['default'] = {
    [byte('A')] = 0.5,
    [byte('B')] = 0.7,
    ... -- many more characters follow
}
\stopLUA
```

As you can see, the definition order of vector vs. class is not important, and the format of the vector is a little different. The use of `byte()` is just so that that keying in hex numbers can be avoided. The values are bare numbers instead of hashes because there is only one per-character parameter involved with character expansion.

Also note that the values for the parameters `stretch`, `shrink` and `step` are divided by a factor 10 compared to the MkII definition.

In MkIV, there is no support for the `spacing` key to `\setupalign` yet. That is because the low–level features in pdfTEX are not present in LuaTEX, and there is no replacement yet. The font handling `noligs` is, of course, replaced by the OpenType font feature tags for ligatures: simply leave all of the relevant font features turned off.

## 5.15 Encodings and mappings

This section only applies to pdfTEX. If you are exclusively using XeTEX or MkIV, you can safely ignore the following text.

Not every language uses the (western) Latin alphabet. Although in most languages the basic 26 characters are somehow used, they can be combined with a broad range of accents placed in any place.

In order to get a character representation, also called glyph, in the resulting output, you have to encode it in the input. This is no problem for `a..z`, but other characters are accessed by name, for instance `\eacute`. The glyph é can be present in the font but when it's not there, TEX has to compose the character from a letter e and an accent ´.

In practice this means that the meaning of `\eacute` depends on the font and font encoding used. There are many such encodings, each suited for a subset of languages.

| **encoding** | **usage** | **status** |
|---|---|---|
| `8r` | a (strange) mixture of encodings | useless |
| `default` | the 7 bit ascii encoding as used by plain TEX | obsolete |
| `ec` | the prefered encoding of TEX distributions | okay |
| `greek` | an encoding for modern greek | okay |
| `qx` | an encoding that covers most eastern european languages | okay |
| `t2a` | a cyrillic TEX font encoding | ? |
| `t2b` | another cyrillic TEX font encoding | ? |
| `t2c` | another another cyrillic TEX font encoding | ? |
| `t5` | an encoding dedicated to vietnamese (many (double) accents) | okay |
| `texnansi` | a combination of TEX and Adobe standard encoding | okay |

These encodings are font related as is demonstrated in **figure 5.1**, **5.2**, **5.3**, **and 5.4**. Here we used the `\showfont` command.

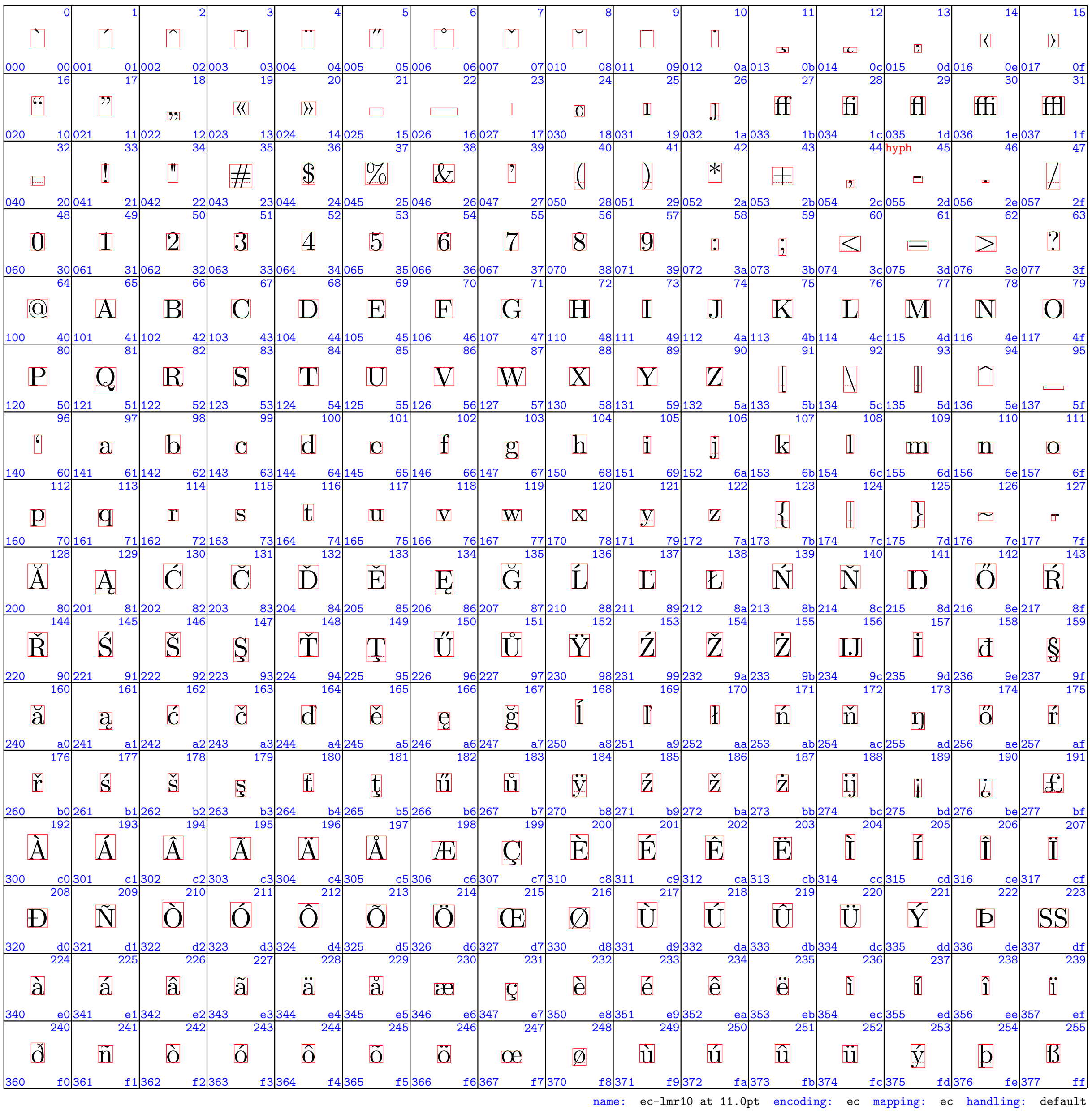

name: ec-lmr10 at 11.0pt encoding: ec mapping: ec handling: default

**Figure 5.1** The Latin Modern Roman font in ec encoding.

The situation is even more complicated than it looks, since the font may be virtual, that is, built from several fonts.

The advantage of using specific encodings is that you can let TEX hyphenate words in the appropriate way. The hyphenation patterns are applied to the internal data structures that represent the sequence of glyphs. In spite of what you may expect, they are font–dependent! Even more confusing: they not only depend on the font encoding, but also on the mapping from lower to uppercase characters, or more precise, on the existence of such a mapping.

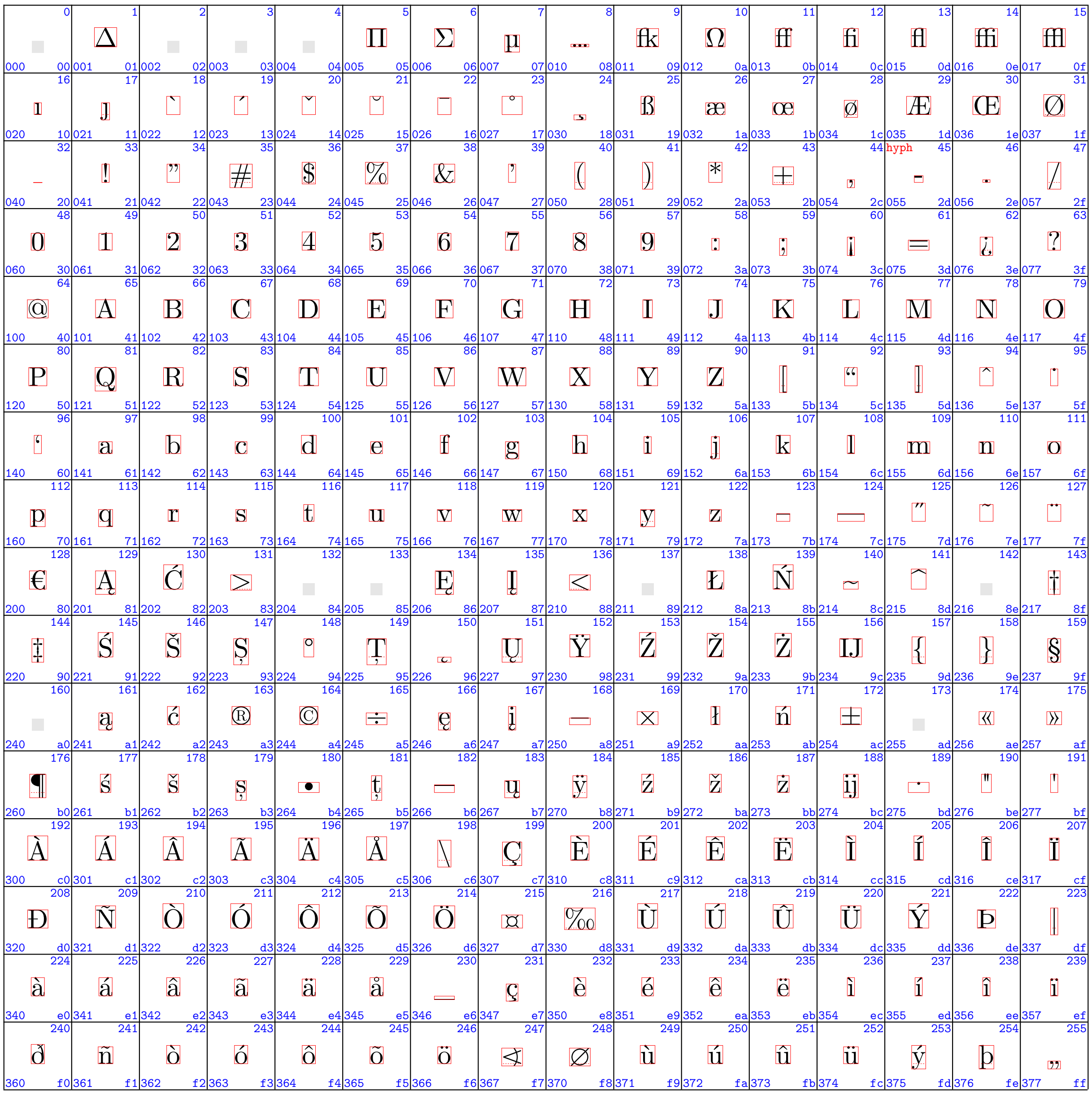
Unless you want to play with these encodings and mappings, in most cases you can forget their details and rely on what other TEX experts tell you to do. Normally switching from one to another encoding and/or mapping takes place with the change in fonts or when some special

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 000 00 | 1 € 001 01 | 2 002 02 | 3 003 03 | 4 ⁄ 004 04 | 5 ˙ 005 05 | 6 ˝ 006 06 | 7 ˛ 007 07 | 8 ﬂ 010 08 | 9 011 09 | 10 012 0a | 11 ﬀ 013 0b | 12 ﬁ 014 0c | 13 015 0d | 14 ﬃ 016 0e | 15 ﬄ 017 0f |
| 16 ı 020 10 | 17 ȷ 021 11 | 18 ` 022 12 | 19 ´ 023 13 | 20 ˇ 024 14 | 21 ˘ 025 15 | 22 ¯ 026 16 | 23 ˚ 027 17 | 24 ¸ 030 18 | 25 ß 031 19 | 26 æ 032 1a | 27 œ 033 1b | 28 ø 034 1c | 29 Æ 035 1d | 30 Œ 036 1e | 31 Ø 037 1f |
| 32 040 20 | 33 ! 041 21 | 34 " 042 22 | 35 # 043 23 | 36 $ 044 24 | 37 % 045 25 | 38 & 046 26 | 39 ’ 047 27 | 40 ( 050 28 | 41 ) 051 29 | 42 * 052 2a | 43 + 053 2b | 44 , 054 2c | hyph 45 - 055 2d | 46 . 056 2e | 47 / 057 2f |
| 48 0 060 30 | 49 1 061 31 | 50 2 062 32 | 51 3 063 33 | 52 4 064 34 | 53 5 065 35 | 54 6 066 36 | 55 7 067 37 | 56 8 070 38 | 57 9 071 39 | 58 : 072 3a | 59 ; 073 3b | 60 < 074 3c | 61 = 075 3d | 62 > 076 3e | 63 ? 077 3f |
| 64 @ 100 40 | 65 A 101 41 | 66 B 102 42 | 67 C 103 43 | 68 D 104 44 | 69 E 105 45 | 70 F 106 46 | 71 G 107 47 | 72 H 110 48 | 73 I 111 49 | 74 J 112 4a | 75 K 113 4b | 76 L 114 4c | 77 M 115 4d | 78 N 116 4e | 79 O 117 4f |
| 80 P 120 50 | 81 Q 121 51 | 82 R 122 52 | 83 S 123 53 | 84 T 124 54 | 85 U 125 55 | 86 V 126 56 | 87 W 127 57 | 88 X 130 58 | 89 Y 131 59 | 90 Z 132 5a | 91 [ 133 5b | 92 \ 134 5c | 93 ] 135 5d | 94 ^ 136 5e | 95 _ 137 5f |
| 96 ‘ 140 60 | 97 a 141 61 | 98 b 142 62 | 99 c 143 63 | 100 d 144 64 | 101 e 145 65 | 102 f 146 66 | 103 g 147 67 | 104 h 150 68 | 105 i 151 69 | 106 j 152 6a | 107 k 153 6b | 108 l 154 6c | 109 m 155 6d | 110 n 156 6e | 111 o 157 6f |
| 112 p 160 70 | 113 q 161 71 | 114 r 162 72 | 115 s 163 73 | 116 t 164 74 | 117 u 165 75 | 118 v 166 76 | 119 w 167 77 | 120 x 170 78 | 121 y 171 79 | 122 z 172 7a | 123 { 173 7b | 124 \| 174 7c | 125 } 175 7d | 126 ˜ 176 7e | 127 ¨ 177 7f |
| 128 Ł 200 80 | 129 ' 201 81 | 130 ‚ 202 82 | 131 ƒ 203 83 | 132 „ 204 84 | 133 … 205 85 | 134 † 206 86 | 135 ‡ 207 87 | 136 ˆ 210 88 | 137 ‰ 211 89 | 138 Š 212 8a | 139 ‹ 213 8b | 140 Œ 214 8c | 141 Ž 215 8d | 142 ˆ 216 8e | 143 – 217 8f |
| 144 ł 220 90 | 145 ‘ 221 91 | 146 ’ 222 92 | 147 “ 223 93 | 148 ” 224 94 | 149 • 225 95 | 150 – 226 96 | 151 — 227 97 | 152 ˜ 230 98 | 153 ™ 231 99 | 154 š 232 9a | 155 › 233 9b | 156 œ 234 9c | 157 ž 235 9d | 158 ~ 236 9e | 159 Ÿ 237 9f |
| 160 240 a0 | 161 ¡ 241 a1 | 162 ¢ 242 a2 | 163 £ 243 a3 | 164 ¤ 244 a4 | 165 ¥ 245 a5 | 166 ¦ 246 a6 | 167 § 247 a7 | 168 ¨ 250 a8 | 169 © 251 a9 | 170 ª 252 aa | 171 « 253 ab | 172 ¬ 254 ac | 173 - 255 ad | 174 ® 256 ae | 175 ¯ 257 af |
| 176 ° 260 b0 | 177 ± 261 b1 | 178 ² 262 b2 | 179 ³ 263 b3 | 180 ´ 264 b4 | 181 µ 265 b5 | 182 ¶ 266 b6 | 183 · 267 b7 | 184 ¸ 270 b8 | 185 ¹ 271 b9 | 186 º 272 ba | 187 » 273 bb | 188 ¼ 274 bc | 189 ½ 275 bd | 190 ¾ 276 be | 191 ¿ 277 bf |
| 192 À 300 c0 | 193 Á 301 c1 | 194 Â 302 c2 | 195 Ã 303 c3 | 196 Ä 304 c4 | 197 Å 305 c5 | 198 Æ 306 c6 | 199 Ç 307 c7 | 200 È 310 c8 | 201 É 311 c9 | 202 Ê 312 ca | 203 Ë 313 cb | 204 Ì 314 cc | 205 Í 315 cd | 206 Î 316 ce | 207 Ï 317 cf |
| 208 Ð 320 d0 | 209 Ñ 321 d1 | 210 Ò 322 d2 | 211 Ó 323 d3 | 212 Ô 324 d4 | 213 Õ 325 d5 | 214 Ö 326 d6 | 215 × 327 d7 | 216 Ø 330 d8 | 217 Ù 331 d9 | 218 Ú 332 da | 219 Û 333 db | 220 Ü 334 dc | 221 Ý 335 dd | 222 Þ 336 de | 223 ß 337 df |
| 224 à 340 e0 | 225 á 341 e1 | 226 â 342 e2 | 227 ã 343 e3 | 228 ä 344 e4 | 229 å 345 e5 | 230 æ 346 e6 | 231 ç 347 e7 | 232 è 350 e8 | 233 é 351 e9 | 234 ê 352 ea | 235 ë 353 eb | 236 ì 354 ec | 237 í 355 ed | 238 î 356 ee | 239 ï 357 ef |
| 240 ð 360 f0 | 241 ñ 361 f1 | 242 ò 362 f2 | 243 ó 363 f3 | 244 ô 364 f4 | 245 õ 365 f5 | 246 ö 366 f6 | 247 ÷ 367 f7 | 248 ø 370 f8 | 249 ù 371 f9 | 250 ú 372 fa | 251 û 373 fb | 252 ü 374 fc | 253 ý 375 fd | 254 þ 376 fe | 255 ÿ 377 ff |

name: texnansi-lmr10 at 11.0pt encoding: texnansi mapping: texnansi handling: default

**Figure 5.2** The Latin Modern Roman font in texnansi encoding.

output encoding is needed, for instance in pdf annotations and/or unicode vectors that enable searching in documents. So, to summarize this: encodings and mappings depend on the fonts used as well have consequences for the language specific hyphenation patterns. Fortunately ConTEXt handles this for you automatically.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 000 00 | Δ 1 001 01 | 2 002 02 | 3 003 03 | 4 004 04 | Π 5 005 05 | Σ 6 006 06 | µ 7 007 07 | … 8 010 08 | fk 9 011 09 | Ω 10 012 0a | ff 11 013 0b | fi 12 014 0c | fl 13 015 0d | ffi 14 016 0e | ffl 15 017 0f |
| ı 16 020 10 | ȷ 17 021 11 | ` 18 022 12 | ´ 19 023 13 | ˇ 20 024 14 | ˘ 21 025 15 | ¯ 22 026 16 | ˚ 23 027 17 | ¸ 24 030 18 | ß 25 031 19 | æ 26 032 1a | œ 27 033 1b | ø 28 034 1c | Æ 29 035 1d | Œ 30 036 1e | Ø 31 037 1f |
| ␣ 32 040 20 | ! 33 041 21 | ” 34 042 22 | # 35 043 23 | $ 36 044 24 | % 37 045 25 | & 38 046 26 | ’ 39 047 27 | ( 40 050 28 | ) 41 051 29 | * 42 052 2a | + 43 053 2b | , 44 054 2c | hyph - 45 055 2d | . 46 056 2e | / 47 057 2f |
| 0 48 060 30 | 1 49 061 31 | 2 50 062 32 | 3 51 063 33 | 4 52 064 34 | 5 53 065 35 | 6 54 066 36 | 7 55 067 37 | 8 56 070 38 | 9 57 071 39 | : 58 072 3a | ; 59 073 3b | ¡ 60 074 3c | = 61 075 3d | ¿ 62 076 3e | ? 63 077 3f |
| @ 64 100 40 | A 65 101 41 | B 66 102 42 | C 67 103 43 | D 68 104 44 | E 69 105 45 | F 70 106 46 | G 71 107 47 | H 72 110 48 | I 73 111 49 | J 74 112 4a | K 75 113 4b | L 76 114 4c | M 77 115 4d | N 78 116 4e | O 79 117 4f |
| P 80 120 50 | Q 81 121 51 | R 82 122 52 | S 83 123 53 | T 84 124 54 | U 85 125 55 | V 86 126 56 | W 87 127 57 | X 88 130 58 | Y 89 131 59 | Z 90 132 5a | [ 91 133 5b | “ 92 134 5c | ] 93 135 5d | ˆ 94 136 5e | ˙ 95 137 5f |
| ‘ 96 140 60 | a 97 141 61 | b 98 142 62 | c 99 143 63 | d 100 144 64 | e 101 145 65 | f 102 146 66 | g 103 147 67 | h 104 150 68 | i 105 151 69 | j 106 152 6a | k 107 153 6b | l 108 154 6c | m 109 155 6d | n 110 156 6e | o 111 157 6f |
| p 112 160 70 | q 113 161 71 | r 114 162 72 | s 115 163 73 | t 116 164 74 | u 117 165 75 | v 118 166 76 | w 119 167 77 | x 120 170 78 | y 121 171 79 | z 122 172 7a | – 123 173 7b | — 124 174 7c | ˝ 125 175 7d | ˜ 126 176 7e | ¨ 127 177 7f |
| € 128 200 80 | Ą 129 201 81 | Ć 130 202 82 | > 131 203 83 | 132 204 84 | 133 205 85 | Ę 134 206 86 | Į 135 207 87 | < 136 210 88 | 137 211 89 | Ł 138 212 8a | Ń 139 213 8b | ~ 140 214 8c | ⁀ 141 215 8d | 142 216 8e | † 143 217 8f |
| ‡ 144 220 90 | Ś 145 221 91 | Š 146 222 92 | Ş 147 223 93 | ° 148 224 94 | Ţ 149 225 95 | ˛ 150 226 96 | Ų 151 227 97 | Ÿ 152 230 98 | Ź 153 231 99 | Ž 154 232 9a | Ż 155 233 9b | IJ 156 234 9c | { 157 235 9d | } 158 236 9e | § 159 237 9f |
| 160 240 a0 | ą 161 241 a1 | ć 162 242 a2 | ® 163 243 a3 | © 164 244 a4 | ÷ 165 245 a5 | ę 166 246 a6 | į 167 247 a7 | − 168 250 a8 | × 169 251 a9 | ł 170 252 aa | ń 171 253 ab | ± 172 254 ac | 173 255 ad | « 174 256 ae | » 175 257 af |
| ¶ 176 260 b0 | ś 177 261 b1 | š 178 262 b2 | ş 179 263 b3 | • 180 264 b4 | ţ 181 265 b5 | ‒ 182 266 b6 | ų 183 267 b7 | ÿ 184 270 b8 | ź 185 271 b9 | ž 186 272 ba | ż 187 273 bb | ij 188 274 bc | · 189 275 bd | " 190 276 be | ' 191 277 bf |
| À 192 300 c0 | Á 193 301 c1 | Â 194 302 c2 | Ã 195 303 c3 | Ä 196 304 c4 | Å 197 305 c5 | \ 198 306 c6 | Ç 199 307 c7 | È 200 310 c8 | É 201 311 c9 | Ê 202 312 ca | Ë 203 313 cb | Ì 204 314 cc | Í 205 315 cd | Î 206 316 ce | Ï 207 317 cf |
| Ð 208 320 d0 | Ñ 209 321 d1 | Ò 210 322 d2 | Ó 211 323 d3 | Ô 212 324 d4 | Õ 213 325 d5 | Ö 214 326 d6 | ¤ 215 327 d7 | ‰ 216 330 d8 | Ù 217 331 d9 | Ú 218 332 da | Û 219 333 db | Ü 220 334 dc | Ý 221 335 dd | Þ 222 336 de | \| 223 337 df |
| à 224 340 e0 | á 225 341 e1 | â 226 342 e2 | ã 227 343 e3 | ä 228 344 e4 | å 229 345 e5 | _ 230 346 e6 | ç 231 347 e7 | è 232 350 e8 | é 233 351 e9 | ê 234 352 ea | ë 235 353 eb | ì 236 354 ec | í 237 355 ed | î 238 356 ee | ï 239 357 ef |
| ð 240 360 f0 | ñ 241 361 f1 | ò 242 362 f2 | ó 243 363 f3 | ô 244 364 f4 | õ 245 365 f5 | ö 246 366 f6 | ∢ 247 367 f7 | ∅ 248 370 f8 | ù 249 371 f9 | ú 250 372 fa | û 251 373 fb | ü 252 374 fc | ý 253 375 fd | þ 254 376 fe | „ 255 377 ff |

name: qx-lmr10 at 11.0pt encoding: qx mapping: qx handling: default

**Figure 5.3** The Latin Modern Roman font in qx encoding.

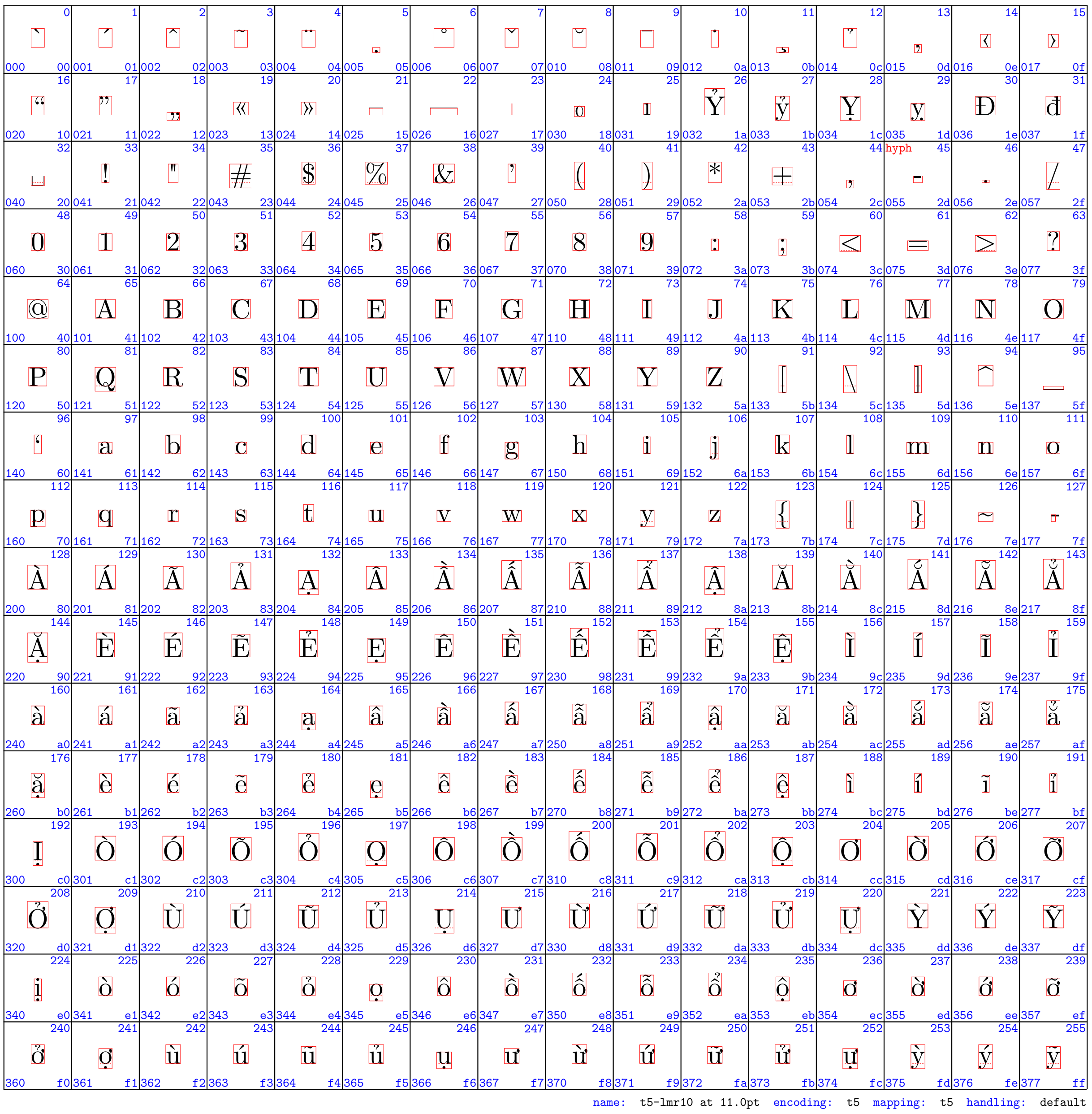

name: t5-lmr10 at 11.0pt encoding: t5 mapping: t5 handling: default

**Figure 5.4** The Latin Modern Roman font in t5 encoding.

If you want to know to what extent a font is complete and characters need to be composed on the fly, you can typeset a a couple of tables. The (current) composition is shown by `\showaccents`, as shown in **figure 5.5**

```
—— ec ec-uplr8a at 11.0pt:  composed bottom char raw ——————————
\'   á b́ ć d́ é f́ ǵ h́ í ȷ́ ḱ ĺ ḿ ń ó ṕ q́ ŕ ś t́ ú v́ ẃ x́ ý ź
     Á B́ Ć D́ É F́ Ǵ H́ Í J́ Ḱ Ĺ Ḿ Ń Ó Ṕ Q́ Ŕ Ś T́ Ú V́ Ẃ X́ Ý Ź
\`   à b̀ c̀ d̀ è f̀ g̀ h̀ ì ȷ̀ k̀ l̀ m̀ ǹ ò p̀ q̀ r̀ s̀ t̀ ù v̀ ẁ x̀ ỳ z̀
     À B̀ C̀ D̀ È F̀ G̀ H̀ Ì J̀ K̀ L̀ M̀ Ǹ Ò P̀ Q̀ R̀ S̀ T̀ Ù V̀ Ẁ X̀ Ỳ Z̀
\^   â b̂ ĉ d̂ ê f̂ ĝ ĥ î ˆ k̂ l̂ m̂ n̂ ô p̂ q̂ r̂ ŝ t̂ û v̂ ŵ x̂ ŷ ẑ
     Â B̂ Ĉ D̂ Ê F̂ Ĝ Ĥ Î Ĵ K̂ L̂ M̂ N̂ Ô P̂ Q̂ R̂ Ŝ T̂ Û V̂ Ŵ X̂ Ŷ Ẑ
\~   ã b̃ c̃ d̃ ẽ f̃ g̃ h̃ ĩ ȷ̃ k̃ l̃ m̃ ñ õ p̃ q̃ r̃ s̃ t̃ ũ ṽ w̃ x̃ ỹ z̃
     Ã B̃ C̃ D̃ Ẽ F̃ G̃ H̃ Ĩ J̃ K̃ L̃ M̃ Ñ Õ P̃ Q̃ R̃ S̃ T̃ Ũ Ṽ W̃ X̃ Ỹ Z̃
\"   ä b̈ c̈ d̈ ë f̈ g̈ ḧ ï ȷ̈ k̈ l̈ m̈ n̈ ö p̈ q̈ r̈ s̈ ẗ ü v̈ ẅ ẍ   z̈
     Ä B̈ C̈ D̈ Ë F̈ G̈ Ḧ Ï J̈ K̈ L̈ M̈ N̈ Ö P̈ Q̈ R̈ S̈ T̈ Ü V̈ Ẅ Ẍ Ÿ Z̈
\H   a̋ b̋ c̋ d̋ e̋ f̋ g̋ h̋ i̋ ȷ̋ k̋ l̋ m̋ n̋ ő p̋ q̋ r̋ s̋ t̋ ű v̋ w̋ x̋ y̋ z̋
     A̋ B̋ C̋ D̋ E̋ F̋ G̋ H̋ I̋ J̋ K̋ L̋ M̋ N̋ Ő P̋ Q̋ R̋ S̋ T̋ Ű V̋ W̋ X̋ Y̋ Z̋
\r   å b̊ c̊ d̊ e̊ f̊ g̊ h̊ i̊ ȷ̊ k̊ l̊ m̊ n̊ o̊ p̊ q̊ r̊ s̊ t̊ ů v̊ ẘ x̊ ẙ z̊
     Å B̊ C̊ D̊ E̊ F̊ G̊ H̊ I̊ J̊ K̊ L̊ M̊ N̊ O̊ P̊ Q̊ R̊ S̊ T̊ Ů V̊ W̊ X̊ Y̊ Z̊
\v   ǎ b̌ č ď ě f̌ ǧ ȟ ǐ ǰ ǩ ľ m̌ ň ǒ p̌ q̌ ř š   ǔ v̌ w̌ x̌ y̌ ž
     Ǎ B̌ Č Ď Ě F̌ Ǧ Ȟ Ǐ J̌ Ǩ Ľ M̌ Ň Ǒ P̌ Q̌ Ř Š Ť Ǔ V̌ W̌ X̌ Y̌ Ž
\u   ă b̆ c̆ d̆ ĕ f̆ ğ h̆ ĭ ȷ̆ k̆ l̆ m̆ n̆ ŏ p̆ q̆ r̆ s̆ t̆ ŭ v̆ w̆ x̆ y̆ z̆
     Ă B̆ C̆ D̆ Ĕ F̆ Ğ H̆ Ĭ J̆ K̆ L̆ M̆ N̆ Ŏ P̆ Q̆ R̆ S̆ T̆ Ŭ V̆ W̆ X̆ Y̆ Z̆
\=   ā b̄ c̄ d̄ ē f̄ ḡ h̄ ī ȷ̄ k̄ l̄ m̄ n̄ ō p̄ q̄ r̄ s̄ t̄ ū v̄ w̄ x̄ ȳ z̄
     Ā B̄ C̄ D̄ Ē F̄ Ḡ H̄ Ī J̄ K̄ L̄ M̄ N̄ Ō P̄ Q̄ R̄ S̄ T̄ Ū V̄ W̄ X̄ Ȳ Z̄
\.   ȧ ḃ ċ ḋ ė ḟ ġ ḣ i ȷ̇ k̇ l̇ ṁ ṅ ȯ ṗ q̇ ṙ ṡ ṫ u̇ v̇ ẇ ẋ ẏ ż
     Ȧ Ḃ Ċ Ḋ Ė Ḟ Ġ Ḣ İ J̇ K̇ L̇ Ṁ Ṅ Ȯ Ṗ Q̇ Ṙ Ṡ Ṫ U̇ V̇ Ẇ Ẋ Ẏ Ż
\b   a̱ ḇ c̱ ḏ e̱ f̱ g̱ ẖ i̱ j̱ ḵ ḻ m̱ ṉ o̱ p̱ q̱ ṟ s̱ ṯ u̱ v̱ w̱ x̱ y̱ ẕ
     A̱ Ḇ C̱ Ḏ E̱ F̱ G̱ H̱ I̱ J̱ Ḵ Ḻ M̱ Ṉ O̱ P̱ Q̱ Ṟ S̱ Ṯ U̱ V̱ W̱ X̱ Y̱ Ẕ
\d   ạ ḅ c̣ ḍ ẹ f̣ g̣ ḥ ị j̣ ḳ ḷ ṃ ṇ ọ p̣ q̣ ṛ ṣ ṭ ụ ṿ ẉ x̣ ỵ ẓ
     Ạ Ḅ C̣ Ḍ Ẹ F̣ G̣ Ḥ Ị J̣ Ḳ Ḷ Ṃ Ṇ Ọ P̣ Q̣ Ṛ Ṣ Ṭ Ụ Ṿ Ẉ X̣ Ỵ Ẓ
\k   ą b̨ c̨ d̨ ę f̨ g̨ h̨ į j̨ k̨ l̨ m̨ n̨ ǫ p̨ q̨ r̨ s̨ t̨ ų v̨ w̨ x̨ y̨ z̨
     Ą B̨ C̨ D̨ Ę F̨ G̨ H̨ Į J̨ K̨ L̨ M̨ N̨ Ǫ P̨ Q̨ R̨ S̨ T̨ Ų V̨ W̨ X̨ Y̨ Z̨
\c   a̧ b̧ ç ḑ ȩ f̧ ģ ḩ i̧ j̧ ķ ļ m̧ ņ o̧ p̧ q̧ ŗ ş   u̧ v̧ w̧ x̧ y̧ z̧
     A̧ B̧ Ç Ḑ Ȩ F̧ Ģ Ḩ I̧ J̧ Ķ Ļ M̧ Ņ O̧ P̧ Q̧ Ŗ Ş   U̧ V̧ W̧ X̧ Y̧ Z̧
```

**Figure 5.5** Output of `\showaccents` for the current (palatino) font in pdfTEX

# 6 Fonts

## 6.1 Introduction

This chapter will cover the details of defining fonts and collections of fonts, and it will explain how to go about installing fonts in both MkII and MkIV. It helps if you know what a font is, and are familiar with the ConTEXt font switching macros.

The original ConTEXt font model was based on plain TEX, but evolved into a more extensive one primarily aimed at consistently typesetting Pragma ADE's educational documents. The fact that pseudo caps had to be typeset in any font shape in the running text as well as superscripts, has clearly determined the design. The font model has been relatively stable since 1995.

Currently there are three layers of font definitions:

- simple font definitions: such definitions provide `\named` access to a specific font in a predefined size
- body font definitions: these result in a coherent set of fonts, often from a same type foundry or designer, that can be used intermixed as a 'style'
- typescript definitions: these package serif, sans serif, mono spaced and math and other styles in such a way that you can conveniently switch between different combinations

These three mechanisms are actually build on top of each other and all rely on a low level mapping mechanism that is responsible for resolving the real font file name and the specific font encoding used.

When TEX users install one of the TEX distributions, like TEX-live, automatically a lot of fonts will be installed on their system. Unfortunately it is not that easy to get a clear picture of what fonts are there and what is needed to use them. And although the `texmf` tree is prepared for commercial fonts, adding newly bought fonts is not trivial. To compensate this, ConTEXt MkII comes with `texfont.pl`, a program that can install fonts for you. And if the global setup is done correctly, MkIV and XƎTEX can use the fonts installed in your operation system without the need for extra installation work.

## 6.2 Font files and synonyms

In ConTEXt, whenever possible you should define symbolic names for fonts. The mapping from such symbolic names onto real font names can be done such that it takes place unnoticed by the user. This is good, since the name can depend on the encoding, in which case the name is obscure and hard to remember. The trick is knowing how to use the `\definefontsynonym` command.

The first argument is the synonym that is being defined or redefined. This synonym should be something simpler than the original name that's easier to use. Redefinition is not only allowed but often very useful. The second argument is the replacement of the synonym. This replacement can be a real font name, but it can also be another synonym. The optional third argument can be used for to specify font settings.

```
\definefontsynonym [.1.] [.2.] [.3.]
                                OPTIONAL
1   TEXT
2   IDENTIFIER
3   encoding  =  IDENTIFIER
    features  =  IDENTIFIER
    handling  =  IDENTIFIER
    mapping   =  IDENTIFIER
```

**Figure 6.1** none

There is no limit on the number of synonyms that can chained together, but the last one in the chain has to be a valid font name. ConTEXt knows it has reached the bottom level when there is no longer any replacement possible.

Font settings actually take place at the bottom level, since they are closely related to specific instances of fonts. Any settings that are defined higher up in the chain percolate down, unless they are already defined at the lower level.

| | |
|---|---|
| `encoding` | The font file encoding for tfm-based (MkII) fonts. |
| `handling` | The font handling for MkII (see previous chapter). |
| `features` | The font handling for MkIV and XeTEX (see previous chapter). |
| `mapping` | letter case change mapping for MkII that may be used in special cases; never actually used in the ConTEXt core. See **chapter 10** on languages for details. |

Here is an example of the use of font synonyms:

```
\definefontsynonym [Palatino] [uplr8t]   [encoding=ec]
```

In this example, the argumnet `uplr8t` is the real font (the actual file name is `uplr8t.tfm`, but file extensions are normally omitted), and it contains the metrics for the Type 1 font URW Palladio L in EC encoding. From now on, the name `Palatino` can be used in further font definitions to identify this font, instead of the dreadfully low–level (and hard to remember) name `uplr8t` and its accompanying encoding.

## 6.2.1 Font names

In pdfTEX, the real font is the name of the TEX metrics file, minus the extension, as we saw already. In XeTEX and MkIV a font name is a bit more complex, because in both cases OpenType fonts can be accessed directly by their official font name (but with any embedded spaces stripped out) as well as via the disk file name.

In these two systems, ConTEXt first attempts to find the font using the official font name. If that doesn't work, then it tries to use the font by file name as a fallback. Since this is not very efficient and also because it may generate (harmless, but alarming looking) warnings, it is possible to force ConTEXt into one or the other mode by using a prefix, so you will most often see synonym definitions like this:

```
\definefontsynonym [MSTimes]       [name:TimesNewRoman] [features=default]
\definefontsynonym [Iwona-Regular] [file:Iwona-Regular] [features=default]
```

In XƎTEX, the `file` prefix implies that XƎTEX will search for an OpenType font (with extension `otf` or `ttf`) and if that fails it will try to find a TEX font (with extension `tfm`). In MkIV, the list is a little longer: OpenType (`otf`, `ttf`), Type 1 (`afm`), Omega (`ofm`), and finally TEX (`tfm`).

The use of aliases to hide the complexity of true font names is already very useful, but ConTEXt goes further than that. An extra synonym level is normally defined that attaches this font name to a generic name like `Serif` or `Sans`.

```
\definefontsynonym [Serif] [Palatino]
```

An important advantage of using names like `Serif` in macro and style definitions is that it can easily be remapped onto a completely different font than `Palatino`. This is often useful when you are experimenting with a new environment file for a book or when you are writing a ConTEXt module.

In fact, inside an environment file it is useful to go even further and define new symbolic names that map onto `Serif`.

```
\definefontsynonym [TitleFont] [Serif]
```

By using symbolic names in the main document and in style and macro definitions, you can make them independent of a particular font and let them adapt automatically to the main document fonts. That is of course assuming these are indeed defined in terms of `Serif`, `Sans`, etcetera. All the ConTEXt predefined typescripts are set up this way, and you are very much encouraged to stick to the same logic for your own font definitions as well.

The list of 'standard' symbolic names is given in **table 6.1**

## 6.2.2 Adjusting font settings

As mentioned earlier, the items in the third argument of `\definefontsynonym` percolate down the chain of synonyms. Occasionally, you may want to splice some settings into that chain, and that is where `\setupfontsynonym` comes in handy.

```
\setupfontsynonym [.1.] [.=2.]

1   IDENTIFIER
2   inherits from \definefontsynonym
```

For example, the predefined MkII typescripts for font handling that we saw in the previous chapter contain a sequence of commands like this:

```
\setupfontsynonym [Serif]       [handling=pure]
\setupfontsynonym [SerifBold]   [handling=pure]
\setupfontsynonym [SerifItalic] [handling=pure]
...
```

The first line means 'adjust the `handling` setting of only the `Serif` font to `pure`'. Any font lower down in the synonym chain won't receive this setting. Another example might be setting a font variant for SMALL CAPS:

```
\setupfontsynonym [SerifSmallCaps]       [features=smallcaps]
```

| name | style, alternative | explanation |
|---|---|---|
| `Blackboard` | `--` | Used by the `\bbd` macro |
| `Calligraphic` | `--` | Used by the `\cal` macro |
| `Fraktur` | `--` | Used by the `\frak` macro |
| `Gothic` | `--` | Used by the `\goth` macro |
| `OldStyle` | `--` | Used by the `\os` macro |
| `MPtxtfont` | `--` | The default font for MetaPost |
| `Calligraphy` | `cg,tf` | |
| `Handwriting` | `hw,tf` | |
| `MathRoman(Bold)` | `mm,mr(bf)` | |
| `MathItalic(Bold)` | `mm,mi(bf)` | |
| `MathSymbol(Bold)` | `mm,sy(bf)` | |
| `MathExtension(Bold)` | `mm,ex(bf)` | |
| `MathAlpha(Bold)` | `mm,ma(bf)` | |
| `MathBeta(Bold)` | `mm,mb(bf)` | |
| `MathGamma(Bold)` | `mm,mc(bf)` | |
| `MathDelta(Bold)` | `mm,md(bf)` | |
| `Mono` | `tt,tf` | |
| `MonoBold` | `tt,bf` | |
| `MonoItalic` | `tt,it` | |
| `MonoBoldItalic` | `tt,bi` | |
| `MonoSlanted` | `tt,sl` | |
| `MonoBoldSlanted` | `tt,bs` | |
| `MonoCaps` | `tt,sc` | |
| `Sans` | `ss,tf` | |
| `SansBold` | `ss,bf` | |
| `SansItalic` | `ss,it` | |
| `SansBoldItalic` | `ss,bi` | |
| `SansSlanted` | `ss,sl` | |
| `SansBoldSlanted` | `ss,bs` | |
| `SansCaps` | `ss,sc` | |
| `Serif` | `rm,tf` | |
| `SerifBold` | `rm,bf` | |
| `SerifItalic` | `rm,it` | |
| `SerifBoldItalic` | `rm,bi` | |
| `SerifSlanted` | `rm,sl` | |
| `SerifBoldSlanted` | `rm,bs` | |
| `SerifCaps` | `rm,sc` | |

**Table 6.1** Standard symbolic font names, and the style–alternative pair they belong to.

## 6.3 Simple font definitions

The most simple font definition takes place with `\definefont`.

```
\definefont [.1.] [.2.] [.3.]
                        OPTIONAL
1   IDENTIFIER
2   FILE
3   TEXT
```

This macro defines a font with the same name as the first argument and you can use its name as an identifier to select that font. The second argument works in the same way as the second argument to `\definefontsynonym`: you can use either a font synonym or a real font. There is an optional third argument that can be either a bare number like `1.5` , or a named setup (see section ??). In case of a bare number, that is a local setting for the interline space. In case of a setup, that setup can do whatever it wants.

For instance:

```
\loadmapfile [koeieletters]
\definefont  [ContextLogo] [koeielogos at 72pt]
\ContextLogo \char 2
```

will result in

If you want a fixed size font like in the example above, you can define a font using the primitive TEX `at` or `scaled` modifiers.

Be warned that `at` is often useful, but `scaled` is somewhat unreliable since it scales the font related to its internal design size, and that is often unknown. Depending on the design size is especially dangerous when you use symbolic names, since different fonts have different design sizes, and designers differ in their ideas about what a design size is. Compare for instance the 10pt instance of a Computer Modern Roman with Lucida Bright (which more looks like a 12pt then).

```
\definefont [TitleFont] [Serif scaled 2400]
```

Hardcoded sizes can be useful in many situations, but they can be annoying when you want to define fonts in such a way that their definitions adapt themselves to their surroundings. That is why ConTEXt provides an additional way of scaling:

```
\definefont [TitleFont] [Serif sa 2.4]
```

The `sa` directive means as much as ‘scaled at the body font size’. Therefore this definition will lead to a 24pt scaling when the (document) body font size equals 10pt. Because the definition has a lazy nature, the font size will adapt itself to the current body font size.

There is an extra benefit to using `sa` instead of `at`. Instead of a numeric multiplier, you can also use the identifiers that were defined in the body font environment that specified the related dimensions. For example, this scales the font to the `b` size, being 1.440 by default:

```
\definefont [TitleFont] [Serif sa b]
```

In fact, if you use a bare name like in

```
\definefont [TitleFont] [Serif]
```

it will internally be converted to

```
\definefont [TitleFont] [Serif sa *]
```

which in turn expands into the current actual font size, after the application of size corrections for super– and subscripts etc.

For example

```
\definefont [TitleFont] [Sans]
{\TitleFont test} and {\tfc \TitleFont test}
```

gives

test and test

A specialized alternative to `sa` that is sometimes useful is `mo`. Here the size maps onto to body font size only after it has passed through an optional size remapping. Such remappings are defined by the macro `\mapfontsize`:

```
\mapfontsize [.1.] [.2.]

1  DIMENSION
2  DIMENSION
```

Such remapping before applying scaling is sometimes handy for math fonts, where you may want to use slightly different sizes than the ones given in the body font environment. In the ConTEXt distribution, this happens only with the Math Times fonts, where the predefined typescript contains the following lines:

```
\mapfontsize [5pt]    [6.0pt]
\mapfontsize [6pt]    [6.8pt]
\mapfontsize [7pt]    [7.6pt]
\mapfontsize [8pt]    [8.4pt]
\mapfontsize [9pt]    [9.2pt]
\mapfontsize [10pt]   [10pt]
\mapfontsize [11pt]   [10.8pt]
\mapfontsize [12pt]   [11.6pt]
\mapfontsize [14.4pt] [13.2pt]
```

As we have seen, `\definefont` creates a macro name for a font switch. For ease of use, there is also a direct method to access a font:

```
\definedfont [.*.]

*  inherits from \definefont
```

Where the argument has exactly the same syntax as the second argument to `\definefont`. In fact, this macro executes `\definefont` internally, and then immediately switches to the defined font.

## 6.4 Defining body fonts

In older versions of ConTEXt, the model for defining fonts that will be described in this section was the top–level user interface. These days, typescripts are used at the top–level, and the body font definitions are wrapped inside of those.

Most commercial fonts have only one design size, and when you create a typescript for such fonts, you can simply reuse the predefined size definitions. Later on we will see that this means you can just refer to a `default` definition.

Still, you may need (or want) to know the details of body font definitions if you create your own typescripts, especially if the fonts are not all that standard. For example, because Latin Modern comes in design sizes, there was a need to associate a specific font with each bodyfont size. You may find yourself in a similar situation when you attempt to create a typescript for a 'professional' commercial font set.

The core of this intermediate model is the `\definebodyfont` command that is used as follows:

```
\definebodyfont [10pt] [rm] [tf=tir at 10pt]
```

This single line actually defines two font switches `\tf` for use after a `\rm` command, and `\rmtf` for direct access.

As one can expect, the first implementation of a font model in TEX is also determined and thereby complicated by the fact that the Computer Modern Roman fonts come in design sizes. As a result, definitions can look rather complex and because most TEX users start with those fonts, font definitions are considered to be complex.

Another complicating factor is that in order to typeset math, even more (font) definitions are needed. Add to that the fact that sometimes fonts with mixed encodings have to be used, i.e. with the glyphs positioned in different font slots, and you can understand why font handling in TEX is often qualified as 'the font mess'. Flexibility simply has its price.

Like most other TEX users, Hans Hagen started out using the Computer Modern Roman fonts. Since these fonts have specific design sizes, ConTEXt supports extremely accurate `\definebodyfont` definitions with specific font names and sizes for each combination. The following is an example of that:

```
\definebodyfont [12pt] [rm]
    [ tf=cmr12,
     tfa=cmr12 scaled \magstep1,
     tfb=cmr12 scaled \magstep2,
     tfc=cmr12 scaled \magstep3,
     tfd=cmr12 scaled \magstep4,
      bf=cmbx12,
      it=cmti12,
      sl=cmsl12,
      bi=cmbxti10 at 12pt,
      bs=cmbxsl10 at 12pt,
      sc=cmcsc10 at 12pt]
```

It should be clear to you that for fonts with design sizes, similar `\definebodyfont` commands will have to be written for each of the requested body font sizes. But many commercial fonts do not come in design sizes at all. In fact, many documents have a rather simple design and use only a couple of fonts for all sizes.

The previous example used the available TEX–specifications `scaled` and `at`, but (as we say already) ConTEXt supports special keyword that is a combination of both: `sa` (scaled at).

For example, for the Helvetica Type 1 font definition we could define:

```
\definebodyfont [12pt] [ss]
  [tf=hv  sa 1.000,
   bf=hvb sa 1.000,
   it=hvo sa 1.000,
   sl=hvo sa 1.000,
  tfa=hv  sa 1.200,
  tfb=hv  sa 1.440,
  tfc=hv  sa 1.728,
  tfd=hv  sa 2.074,
   sc=hv  sa 1.000]
```

The scaling is done in relation to the bodyfont size. In analogy with TEX's `\magstep` we can use `\magfactor`: instead of `sa 1.440` we could specify `sa \magfactor2`.

If you are happy with the relative sizes as defined in the body font environment (and there is no reason not to), the `\definebodyfont` can be four lines shorter. That is because ConTEXt predeclares a whole collection of names that combine the styles `rm`, `ss`, `tt`, `tf`, `hw` and `cg` with the alternatives `bf`, `it`, `sl`, `bi`, `bs`, and `sc` with the postfixes `a`, `b`, `c`, `d`, `x` and `xx`.

For the combination of `ss` and `sl`, the following identifiers are predeclared:

```
\ss    \ssa   \ssb   \ssc   \ssd    \ssx  \ssxx
\sl   \sla  \slb  \slc  \sld   \slx \slxx
\sssl  \sssla \ssslb \ssslc \sssld
```

And because there are no more sizes in the definition any more, we can just as well combine all of the requested sizes in a single `\definebodyfont` by using a list of sizes as the first argument. This means exactly the same as repeating that whole list five (or more) times, but saves a lot of typing:

```
\definebodyfont [12pt,11pt,10pt,9pt,8pt] [ss]
  [tf=hv  sa 1.000,
   bf=hvb sa 1.000,
   it=hvo sa 1.000,
   sl=hvo sa 1.000,
   sc=hv  sa 1.000]
```

Because the font names (may) depend on the encoding vector, we had better use the previously discussed method for mapping symbolic names. So, any one of the three following lines can be used, but the third one is best:

```
\definebodyfont [10pt,11pt,12pt] [ss] [tf=hv        sa 1.000]
\definebodyfont [10pt,11pt,12pt] [ss] [tf=Helvetica sa 1.000]
\definebodyfont [10pt,11pt,12pt] [ss] [tf=Sans      sa 1.000]
```

And in the actual ConTEXt core, the default body fonts are in fact defined with commands like this:

```
\definebodyfont [default] [rm]
  [ tf=Serif       sa 1,
    ...
    it=SerifItalic sa 1,
    ... ]
```

We saw that `\tf` is the default font. Here `\tf` is defined as `Serif sa 1` which means that it is a serif font, scaled to a normal font size. This `Serif` is mapped elsewhere on for example `Palatino` which in turn is mapped on the actual filename `uplr8t`, as demonstrated earlier.

```
\definebodyfont [...,...] [...] [..,..=..,..]
                  1         2          3
                         OPTIONAL
1   5pt ... 12pt small big
2   rm ss tt hw cg mm
3   tf  =  FILE
    bf  =  FILE
    sl  =  FILE
    it  =  FILE
    bs  =  FILE
    bi  =  FILE
    sc  =  FILE
    mr  =  FILE
    ex  =  FILE
    mi  =  FILE
    sy  =  FILE
    ma  =  FILE
    mb  =  FILE
    mc  =  FILE
    md  =  FILE
```

The macro syntax for `\definebodyfont` is a bit abbreviated. Besides the two–letter keys that are listed for the third argument, it is also possible to assign values to font identifiers with the alphabetic suffixes `a` through `d` like `tfa` as well as the ones with an `x` or `xx` suffix like `bfx`. You can even define totally new keywords, if you want that.

As an example we will define a bigger fontsize of `\tf`:

```
\definebodyfont [10pt,11pt,12pt] [rm]
    [tfe=Serif at 48pt,
     ite=SerifItalic at 48pt]
\tfe Big {\it Words}.
```

This becomes:

Big *Words*.

Note that there is a small trick here: the assignment to `ite` is needed for the command `\it` to work properly. Without that, the command `\it` would run the 'normal' version of `it` and that has a size of 11pt.

The keywords `mr`, `ex`, `mi`, `sy`, `ma`, `mb`,`mc` and `md` all relate to math families. As was already hinted at in **table 6.1**, these have extended relatives suffixed by `bf` for use within bold math environments.

Calls of `\definebodyfont` for the `mm` style look quite different from the other styles, because they set up these special keywords, and nothing else. The first four keys are required in all math setups just to do basic formula typesetting, the other four (`ma` …`md`) can be left undefined. Those are normally used for fonts with special symbols or alphabets like the AMS symbol fonts `msam` and `msbm`.

Here is what a setup for a fairly standard `mm` could look like:

```
\definebodyfont [10pt] [mm]
    [mr=cmr10,
     ex=cmex10,
     mi=cmmi10,
     sy=cmsy10]
\definebodyfont [17.3pt,14.4pt,12pt,11pt,10pt,9pt] [mm]
    [ma=msam10 sa 1,
     mb=msbm10 sa 1]
```

The keys `mc` and `md` are left undefined. This example explicitly shows how multiple `\definebodyfonts` are combined by ConTEXt automatically and that there is no need to do everything within a single definition (in fact this was already implied by the `tfe` trick above.)

Apart from the calling convention as given in the macro syntax that has already been shown, there are a few alternative forms of `\definebodyfont` that can be used to defined and call body fonts by name:

```
\definebodyfont [.1.] [.2.] [.3.]

1  IDENTIFIER
2  inherits from \setupbodyfont
3  inherits from \setupbodyfont
```

This was used in the default serif font defintion shown above: the first argument to `\definebodyfont` was the identifier `default` because these definitions were to be used from within other definitions.

An actual size will be provided by the commands at the top–level in the calling chain, the third argument in that `\definebodyfont` call will also be `default` instead of actually specifying settings.

```
\definebodyfont [.1.] [.2.] [.3.]

1   inherits from \setupbodyfont
2   inherits from \setupbodyfont
3   IDENTIFIER
```

The use of the `default` actually happens deep inside ConTEXt so there is clear code that can be shown, but if it was written out, a call would for example look like this:

```
\definebodyfont
  [17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
  [rm,ss,tt,mm]
  [default]
```

To end this section: for advanced TEX users there is the dimension–register `\bodyfontsize`. This variable can be used to set fontwidths. The number (rounded) points is available in `\numberofpoints\bodyfontsize`.

This way of defining fonts has been part of ConTEXt from the beginning, but as more complicated designs started to show up, we felt the need for a more versatile mechanism.

## 6.5 Typescripts and typefaces

On top of the existing traditional font module, ConTEXt now provides a more abstract layer of typescripts and building blocks for definitions and typefaces as font containers. The original font definition files have been regrouped into such typescripts thereby reducing the number of files involved.

As we saw earlier, 'using' a typescript is done via the a call to the macro `\usetypescript`. Here is the macro syntax setup again:

```
\usetypescript [...,1...] [...,2...] [...,3...]
                           OPTIONAL   OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   IDENTIFIER
```

Typescripts are in fact just organized definitions, and 'using' a typescript therefore actually means nothing more than executing the set of definitions that is contained within a particular typescript.

The main defining command for typescripts is a start–stop pair that wraps the actual macro definitions.

```
\starttypescript [...] [...] [...]
   ....
\stoptypescript
```

As with `\usetypescript`, there can be up to three arguments, and these two sets of arguments are linked to eachother: the values of the first and second argument in the call to `\starttypescript` of

```
\starttypescript [palatino] [texnansi,ec,qx,t5,default]
  ...
\stoptypescript
```

are what make the MkII-style call to `\usetypescript`

```
\usetypescript [palatino] [ec]
...
```

possible and meaningful: the first argument in both cases is the same so that this matches, and the second argument of `\usetypescript` appears in the list that is the second argument of `\starttypescript`, so this also matches. ConTEXt will execute all matching blocks it knows about: there may be more than one.

To perform the actual matching, ConTEXt scans through the list of known `\starttypescript` blocks for each of the combinations of items in the specified arguments of `\usetypescript`. These blocks can be preloaded definitions in TEX's memory, or they may come from a file.

There is a small list of typescript files that is tried always, and by using `\usetypescriptfile` you actually add extra ones at the end of this list.

The automatically loaded files for the three possible engines are, in first to last order:

| pdftex | xetex | luatex | explanation |
|---|---|---|---|
| `type-tmf` | `type-tmf` | `type-tmf` | Core TEX community fonts |
| `type-siz` | `type-siz` | `type-siz` | Font size setups |
| `type-one` | | | Type 1 free fonts |
| | `type-otf` | `type-otf` | OpenType free fonts |
| | `type-xtx` | | MacOSX font support |
| `type-akb` | | | Basic Adobe Type 1 mappings |
| `type-loc` | `type-loc` | `type-loc` | A user configuration file |

Extra arguments to `\usetypescript` are ignored, and that is why that same two-argument call to `\usetypescript` works correctly in MkIV as well, even tough the typescript itself uses only a single argument:

```
\starttypescript [palatino]
  ...
\stoptypescript
```

On the other hand, extra arguments to `\starttypescript` are not ignored: a `\starttypescript` with two specified arguments will not be matched by a `\usetypescript` that has only one specified argument.

However, you can force any key at all to match by using the special keyword `all` in your `\usetypescript` or `\starttypescript`. We will see later that this use of a wildcard is sometimes handy.

### 6.5.1 A typescript in action

Before we can go on and explain how to write `\starttypescript` blocks, we have to step back for a moment to the macro `\definetypeface`, and especially to the third, fourth and fifth argument:

```
\starttypescript [palatino] [texnansi,ec,qx,t5,default]
\definetypeface[palatino] [rm] [serif] [palatino] [default]
...
```

Remember how in the previous chapter there were the tables that listed all the predefined combinations? It was said there that these '...are nothing more than convenience names that are attached to a group of fonts by the person that wrote the font definition'.

Here is how that works: these arguments of `\definetypeface` are actually used as parts of `\usetypescript` calls. To be preciese, inside the macro definition of `\definetypeface`, there are the following lines:

```
\def\definetypeface
  ...
  \usetypescript[#3,map][#4][name,default,\typefaceencoding,special]
  \usetypescript[#3][#5][size]
  ...
```

In our example `#3` is `serif`, `#4` is `palatino`, and `#5` is default. The value of `\typefaceencoding` is inherited from the calling `\usetypescript`. That means that the two lines expand into:

```
\usetypescript[serif,map][palatino] [name,default,ec,special]
\usetypescript[serif][default][size]
```

And those typescripts will be searched for. This example is using MkII, so the list of typescript files is `type-tmf`, `type-siz`, `type-one`, `type-akb`, and `type-loc`. The first two arguments of `\usetypescript` are handled depth first, so first all 'serif' typescripts are tried against all the files in the list and then all the 'map' typescripts.

Not all of the seached typescript blocks are indeed present in the list of files that have to be scanned, but a few are, and one apparently even more than once:

| | | | |
|---|---|---|---|
| `type-tmf.tex` | serif | palatino | name |
| `type-one.tex` | serif | palatino | texnansi,ec,8r,t5 |
| `type-one.tex` | serif | palatino | ec,texnansi,8r |
| `type-one.tex` | map | all | – |
| `type-siz.tex` | serif | default | size |

All of the found blocks are executed, so let's look at them in order

```
\starttypescript [serif] [palatino] [name]
    \definefontsynonym [Serif]             [Palatino]
    \definefontsynonym [SerifBold]         [Palatino-Bold]
    \definefontsynonym [SerifItalic]       [Palatino-Italic]
    \definefontsynonym [SerifSlanted]      [Palatino-Slanted]
    \definefontsynonym [SerifBoldItalic]   [Palatino-BoldItalic]
    \definefontsynonym [SerifBoldSlanted]  [Palatino-BoldSlanted]
    \definefontsynonym [SerifCaps]         [Palatino-Caps]
```

```
\stoptypescript
```

This block has mapped the standard symbolic names to names in the 'Palatino' family, one of the standard font synonym actions as explained in the beginning of this chapter.

```
\starttypescript [serif] [palatino] [texnansi,ec,8r,t5]
\definefontsynonym [Palatino]
    [\typescriptthree-uplr8a]  [encoding=\typescriptthree]
\definefontsynonym [Palatino-Italic]
    [\typescriptthree-uplri8a] [encoding=\typescriptthree]
\definefontsynonym [Palatino-Bold]
    [\typescriptthree-uplb8a]  [encoding=\typescriptthree]
\definefontsynonym [Palatino-BoldItalic]
    [\typescriptthree-uplbi8a] [encoding=\typescriptthree]
\definefontsynonym [Palatino-Slanted]
    [\typescriptthree-uplr8a-slanted-167] [encoding=\typescriptthree]
\definefontsynonym [Palatino-BoldSlanted]
    [\typescriptthree-uplb8a-slanted-167] [encoding=\typescriptthree]
\definefontsynonym [Palatino-Caps]
    [\typescriptthree-uplr8a-capitalized-800] [encoding=\typescriptthree]

\loadmapfile[\typescriptthree-urw-palatino.map]
\stoptypescript
```

This maps the Palatino names onto the actual font files. Some further processing is taking place here: the calling `\usetypescript` that was called from within the `\definetypeface` knows that it wants `ec` encoding. Because this is the third argument, it becomes the replacement of `\typescriptthree`. The body of the typescript therefore reduces to:

```
\definefontsynonym[Palatino]            [ec-uplr8a]                 [encoding=ec]
\definefontsynonym[Palatino-Italic]     [ec-uplri8a]                [encoding=ec]
\definefontsynonym[Palatino-Bold]       [ec-uplb8a]                 [encoding=ec]
\definefontsynonym[Palatino-BoldItalic] [ec-uplbi8a]                [encoding=ec]
\definefontsynonym[Palatino-Slanted]    [ec-uplr8a-slanted-167]     [encoding=ec]
\definefontsynonym[Palatino-BoldSlanted][ec-uplb8a-slanted-167]     [encoding=ec]
\definefontsynonym[Palatino-Caps]       [ec-uplr8a-capitalized-800][encoding=ec]

\loadmapfile[ec-urw-palatino.map]
```

Incidentally, this also loads a font map file. In earlier versions of ConTEXt, this was done by separate typescripts in the file `type-map.tex`, but nowadays all map loading is combined with the definition of the synonyms that link to the true fonts on the harddisk. This way, there is a smaller chance of errors creeping in. See **section 6.9** for more details on font map files.

The third match is a block that sets sets up 'TeXPalladioL' font synonyms. These will not actually be used, but it is a match so it will be executed anyway.

```
\starttypescript [serif] [palatino] [ec,texnansi,8r]
\definefontsynonym[TeXPalladioL-BoldItalicOsF]
    [\typescriptthree-fplbij8a][encoding=\typescriptthree]
...
\stoptypescript
```

The next matched entry loads the font map files for the default fonts:

```
\starttypescript [map] [all]
    \loadmapfile[original-base.map]
    \loadmapfile[original-ams-base.map]
\stoptypescript
```

this will not really be needed for the palatino `\rm` typescript, but it ensures that even if there is something horribly wrong with the used typescripts, at least pdfTEX will be able to find the Latin Modern (the default font set) on the harddisk.

The last match is the missing piece of the font setup:

```
\starttypescript [serif] [default] [size]
  \definebodyfont
    [4pt,5pt,6pt,7pt,8pt,9pt,10pt,11pt,12pt,14.4pt,17.3pt]
    [rm] [default]
\stoptypescript
```

and now the typescript is complete.

As explained earlier, that last block references a named `\definebodyfont` that is defined in `type-unk.tex`:

```
\definebodyfont [default] [rm]
  [tf=Serif sa 1,
   bf=SerifBold sa 1,
   it=SerifItalic sa 1,
   sl=SerifSlanted sa 1,
   bi=SerifBoldItalic sa 1,
   bs=SerifBoldSlanted sa 1,
   sc=SerifCaps sa 1]
```

similar `default` blocks are defined for the other five font styles also.

Looking back, you can see that the Palatino-specific typescripts did actually do anything except definining font synonyms, loading a map file, and calling a predefined `bodyfont`.

### 6.5.2 Some more information

As we saw already, typescripts and its invocations have up to three specifiers. An invocation matches the script specification when the three arguments have common keywords, and the special keyword `all` is equivalent to any match.

Although any keyword is permitted in any of the three arguments, the current definitions (and macros like `\definetypeface`) make heavy use of some keys in particular:

| **pattern** | **application** |
|---|---|
| `[serif] [*] [*]` | serif fonts |
| `[sans] [*] [*]` | sans serif fonts |
| `[mono] [*] [*]` | mono spaced fonts |
| `[math] [*] [*]` | math fonts |
| `[*] [*] [size]` | size specifications |
| `[*] [*] [name]` | symbolic name mapping |

| | |
|---|---|
| `[*] [*] [special]` | special settings |
| `[*] [all] [*]` | default case(s) |
| `[map] [*] [*]` | map file specifications |

When you take a close look at the actual files in the distribution you will notice a quite a few other keywords. One in particular is worth mentioning: instead of the predefined sizes in `default`, you can use the `dtp` size scripts with their associated body font environments by using

```
\usetypescript [all] [dtp] [size]
```

or

```
\definetypeface[palatino] [rm] [serif] [palatino] [dtp]
```

In the top–level typescript for the palatino, we had a bunch of `\definetypeface` commands, as follows:

```
\definetypeface [funny] [rm] [serif] [palatino] [default] [encoding=texnansi]
\definetypeface [funny] [ss] [sans]  [palatino] [default] [encoding=texnansi]
\definetypeface [funny] [tt] [mono]  [palatino] [default] [encoding=texnansi]
\definetypeface [funny] [mm] [math]  [palatino] [default] [encoding=texnansi]
```

Once these commands are executed (wether or not as part of a typescript), `\funny` will enable this specific collection of fonts. In a similar way we can define a collection `\joke`.

```
\definetypeface [joke] [rm] [serif] [times]     [default] [encoding=texnansi]
\definetypeface [joke] [ss] [sans]  [helvetica] [default] [rscale=0.9,
                                                            encoding=texnansi]
\definetypeface [joke] [tt] [mono]  [courier]   [default] [rscale=1.1,
                                                            encoding=texnansi]
\definetypeface [joke] [mm] [math]  [times]     [default] [encoding=texnansi]
```

And the familiar Computer Modern Roman as `\whow`:

```
\definetypeface [whow] [rm] [serif] [modern] [latin-modern] [encoding=ec]
\definetypeface [whow] [ss] [sans]  [modern] [latin-modern] [encoding=ec]
\definetypeface [whow] [tt] [mono]  [modern] [latin-modern] [encoding=ec]
\definetypeface [whow] [mm] [math]  [modern] [latin-modern] [encoding=ec]
```

Now has become possible to switch between these three font collections at will. Here is a sample of some text and a little bit of math:

```
Who is {\it fond} of fonts?
Who claims that $t+e+x+t=m+a+t+h$?
Who {\ss can see} {\tt the difference} here?
```

When typeset in `\funny`, `\joke`, and `whow`, the samples look like:

Who is *fond* of fonts?
Who claims that $t + e + x + t = m + a + t + h$?
Who can see the difference here?

Who is *fond* of fonts?
Who claims that $t + e + x + t = m + a + t + h$?
Who can see `the difference` here?

Who is fond of fonts?
Who claims that $t + e + x + t = m + a + t + h$?
Who can see the difference here?

With `\showbodyfont` you can get an overview of this font.

| | | | | | | | [funny] | | | | | | \mr : *Ag* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | \tf | \sc | \sl | \it | \bf | \bs | \bi | \tfx | \tfxx | \tfa | \tfb | \tfc | \tfd |
| \rm | Ag | AG | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |
| \ss | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag |
| \tt | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag |

**Figure 6.2** The `funny` typeface collection.

| | | | | | | | [joke] | | | | | | \mr : *Ag* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | \tf | \sc | \sl | \it | \bf | \bs | \bi | \tfx | \tfxx | \tfa | \tfb | \tfc | \tfd |
| \rm | Ag | AG | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |
| \ss | Ag | AG | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |
| \tt | Ag | Ag | *Ag* | *Ag* | **Ag** | ***Ag*** | ***Ag*** | Ag | Ag | Ag | Ag | Ag | Ag |

**Figure 6.3** The `joke` typeface collection.

| | | | | | | | [whow] | | | | | | \mr : *Ag* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | \tf | \sc | \sl | \it | \bf | \bs | \bi | \tfx | \tfxx | \tfa | \tfb | \tfc | \tfd |
| \rm | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag |
| \ss | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag |
| \tt | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag | Ag |

**Figure 6.4** The `whow` typeface collection.

When defining the joke typeface collection, we used a scale directive. The next sample demonstrates the difference between the non scaled and the scaled alternatives.

Who is *fond* of fonts?
Who claims that $t + e + x + t = m + a + t + h$?
Who can see `the difference` here?

Who is *fond* of fonts?
Who claims that $t + e + x + t = m + a + t + h$?
Who can see `the difference` here?

It may not be immediately clear from the previous examples, but a big difference between using typeface definitions and the old method of redefining over and over again, is that the new method uses more resources. This is because each typeface gets its own name space assigned. As an intentional side effect, the symbolic names also follow the typeface. This means that for instance:

```
\definefont[MyBigFont][Serif sa 1.5] \MyBigFont A bit larger!
```

will adapt itself to the currently activated serif font shape, here `\funny`, `\joke` and `\whow`.

A bit larger!
A bit larger!
A bit larger!

### 6.5.3 A bit more about math

Math is kind of special in the sense that it has its own set of fonts, either or not related to the main text font. By default, a change in style, for instance bold, is applied to text only.

```
$        \sqrt{625} =     5\alpha$
$\bf     \sqrt{625} =     5\alpha$
$        \sqrt{625} = \bf 5\alpha$
$\bfmath \sqrt{625} =     5\alpha$
```

The difference between these four lines is as follows:

$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\boldsymbol{\sqrt{625} = 5\alpha}$

In order to get a bold $\alpha$ symbol, we need to define bold math fonts.[16] Assuming the font's typescripts support bold math, the most convenient way of doing this is the following:

```
\definetypeface [whow] [mm]
  [math,boldmath] [modern] [default] [encoding=texnansi]
```

Bold math looks like this:

$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\boldsymbol{\sqrt{625} = 5\alpha}$

The definitions are given on the next page. Such definitions are normally collected in the project bound file, for instance called `typeface.tex`, that is then manually added to the list of typescript files:

```
\usetypescriptfile[typeface] % project scripts
```

It is also possible to avoid typescripts. When definitions are used only once, it makes sense to use a more direct method. We will illustrate this with a bit strange example.

Imagine that you want some math formulas to stand out, but that you don't have bold fonts. In that case you can for instance scale them. A rather direct method is the following.

[16] Bold math is already prepared in the core modules, so normally one can do with less code

```
\definebodyfont
  [funny]
  [12pt,11pt,10pt,9pt,8pt,7pt] [mm]
  [mrbf=MathRoman     mo 2,
   exbf=MathExtension mo 2,
   mibf=MathItalic    mo 2,
   sybf=MathSymbol    mo 2]
```

Our math sample will now look like:

$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\boldsymbol{\sqrt{625} = 5\alpha}$

We can also use an indirect method:

```
\definebodyfont
  [smallmath] [mm]
  [mrbf=MathRoman     mo .5,
   exbf=MathExtension mo .5,
   mibf=MathItalic    mo .5,
   sybf=MathSymbol    mo .5]

\definebodyfont
  [funny]
  [12pt,11pt,10pt,9pt,8pt,7pt]
  [mm] [smallmath]
```

This method is to be preferred when we have to define more typefaces since it saves keystrokes.

$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\sqrt{625} = 5\alpha$
$\boldsymbol{\sqrt{625} = 5\alpha}$

For efficiency reasons, the font definitions (when part of a typeface) are frozen the first time they are used. Until that moment definitions will adapt themselves to changes in for instance scaling and (mapped) names. Freezing definitions is normally no problem because typefaces are defined for a whole document and one can easily define more instances. When you redefine it, a frozen font is automatically unfrozen.

## 6.6 Predefined font, style and alternative keywords

Some of the internal commands are worth mentioning because they define keywords and you may want to add to the list.

Font size switching is done with keywords like `twelvepoint` and commands like `\twelvepoint` or `\xii`, which is comparable to the way it is done in plain TEX. These commands are defined with:

```
\definebodyfontswitch [fourteenpointfour] [14.4pt]
```

```
\definebodyfontswitch [twelvepoint]       [12pt]
\definebodyfontswitch [elevenpoint]       [11pt]
\definebodyfontswitch [tenpoint]          [10pt]
\definebodyfontswitch [ninepoint]         [9pt]
\definebodyfontswitch [eightpoint]        [8pt]
\definebodyfontswitch [sevenpoint]        [7pt]
\definebodyfontswitch [sixpoint]          [6pt]
\definebodyfontswitch [fivepoint]         [5pt]
\definebodyfontswitch [fourpoint]         [4pt]
\definebodyfontswitch [xii]  [12pt]
\definebodyfontswitch [xi]   [11pt]
\definebodyfontswitch [x]    [10pt]
\definebodyfontswitch [ix]   [9pt]
\definebodyfontswitch [viii] [8pt]
\definebodyfontswitch [vii]  [7pt]
\definebodyfontswitch [vi]   [6pt]
```

But be warned that `\xi` is later redefined as a greek symbol.

The keys in `\setupbodyfont` are defined in terms of:

```
\definefontstyle [rm,roman,serif,regular]    [rm]
\definefontstyle [ss,sansserif,sans,support] [ss]
\definefontstyle [tt,teletype,type,mono]     [tt]
\definefontstyle [hw,handwritten]            [hw]
\definefontstyle [cg,calligraphic]           [cg]
```

In many command setups we encounter the parameter `style`. In those situations we can specify a key. These keys are defined with `\definealternativestyle`. The third argument is only of importance in chapter and section titles, where, apart from `\cap`, we want to obey the font used there.

```
\definealternativestyle [mediaeval]          [\os]                 []
\definealternativestyle [normal]             [\tf]                 []
\definealternativestyle [bold]               [\bf]                 []
\definealternativestyle [type]               [\tt]                 []
\definealternativestyle [mono]               [\tt]                 []
\definealternativestyle [slanted]            [\sl]                 []
\definealternativestyle [italic]             [\it]                 []
\definealternativestyle [boldslanted,
                         slantedbold]        [\bs]                 []
\definealternativestyle [bolditalic,
                         italicbold]         [\bi]                 []
\definealternativestyle [small,
                         smallnormal]        [\tfx]                []
\definealternativestyle [smallbold]          [\bfx]                []
\definealternativestyle [smalltype]          [\ttx]                []
\definealternativestyle [smallslanted]       [\slx]                []
\definealternativestyle [smallboldslanted,
                         smallslantedbold]   [\bsx]                []
```

```
\definealternativestyle [smallbolditalic,
                          smallitalicbold]    [\bix]             []
\definealternativestyle [sans,
                          sansserif]          [\ss]              []
\definealternativestyle [sansbold]           [\ss\bf]           []
\definealternativestyle [smallbodyfont]      [\setsmallbodyfont] []
\definealternativestyle [bigbodyfont]        [\setbigbodyfont]  []
\definealternativestyle [cap,
                          capital]            [\smallcapped]     [\smallcapped]
\definealternativestyle [smallcaps]          [\sc]              [\sc]
\definealternativestyle [WORD]               [\WORD]            [\WORD]
```

In section 5.4 we have already explained how *emphasizing* is defined. With oldstyle digits this is somewhat different. We cannot on the forehand in what font these can be found. By default we have the setup:

```
\definefontsynonym [OldStyle] [MathItalic]
```

As we see they are obtained from the same font as the math italic characters. The macro `\os` fetches the runtime setting by executing `\symbolicfont{OldStyle}`, which is just a low-level version of `\definedfont[OldStyle sa *]`. A few other macros behave just like that:

| **macro** | **synonym** | **default value** |
|---|---|---|
| `\os` | OldStyle | MathItalic (lmmi10) |
| `\frak` | Fraktur | eufm10 |
| `\goth` | Gothic | eufm10 |
| `\cal` | Calligraphic | cmsy10 (lmsy10) |
| `\bbd` | Blackboard | msbm10 |

In addition to all the alrady mentioned commands there are others, for example macros for manipulating accents. These commands are discussed in the file `font-ini`. More information can also be found in the file `core-fnt` and specific gimmicks in the file `supp-fun`. So enjoy yourself.

## 6.7 Symbols and glyphs

Some day you may want to define your own symbols, if possible in such a way that they nicely adapt themselves to changes in style and size. A good example are the € symbols. You can take a look in `symb-eur.tex` to see how such a glyph is defined.

```
\definefontsynonym [EuroSerif]     [eurose]
\definefontsynonym [EuroSerifBold] [euroseb]
...
\definefontsynonym [EuroSans]      [eurosa]
\definefontsynonym [EuroSansBold]  [eurosab]
...
\definefontsynonym [EuroMono]      [euromo]
\definefontsynonym [EuroMonoBold]  [euromob]
```

Here we use the free Adobe euro fonts, but there are alternatives available. The symbol itself is defined as:

```
\definesymbol [euro] [\getglyph{Euro}{\char160}]
```

You may notice that we only use the first part of the symbolic name. ConTEXt will complete this name according to the current style. You can now access this symbol with `\symbol[euro]`

| | `\tf` | `\bf` | `\sl` | `\it` | `\bs` | `\bi` |
|---|---|---|---|---|---|---|
| Serif | € | € | € | € | € | € |
| Sans | € | € | € | € | € | € |
| Mono | € | € | € | € | € | € |

More details on defining symbols and symbol sets can be found in the documentation of the symbol modules.

## 6.8 Encodings

**TODO:** Add macro syntax definition blocks

Until now we assumed that an a will become an a during type setting. However, this is not always the case. Take for example ä or æ. This character is not available in every font and certainly not in the Computer Modern Typefaces. Often a combination of characters `\"a` or a command `\ae` will be used to produce such a character. In some situation TEX will combine characters automatically, like in `fl` that is combined to fl and not fl. Another problem occurs in converting small print to capital print and vice versa.

Below you see an example of the `texnansi` mapping:

```
\startmapping[texnansi]
  \definecasemap 228 228 196  \definecasemap 196 228 196
  \definecasemap 235 235 203  \definecasemap 203 235 203
  \definecasemap 239 239 207  \definecasemap 207 239 207
  \definecasemap 246 246 214  \definecasemap 214 246 214
  \definecasemap 252 252 220  \definecasemap 220 252 220
  \definecasemap 255 255 159  \definecasemap 159 255 159
\stopmapping
```

This means so much as: in case of a capital the character with code 228 becomes character 228 and in case of small print the character becomes character 196.

These definitions can be found in `enco-ans`. In this file we can also see:

```
\startencoding[texnansi]
  \defineaccent " a 228
  \defineaccent " e 235
  \defineaccent " i 239
  \defineaccent " o 246
  \defineaccent " u 252
  \defineaccent " y 255
\stopencoding
```

and

```
\startencoding[texnansi]
  \definecharacter ae 230
  \definecharacter oe 156
  \definecharacter o  248
  \definecharacter AE 198
\stopencoding
```

As a result of the way accents are placed over characters we have to approach accented characters different from normal characters. There are two methods: TEX does the accenting itself *or* prebuild accentd glyphs are used. The definitions above take care of both methods. Other definitions are sometimes needed. In the documentation of the file `enco-ini` more information on this can be found.

## 6.9 Map files

**TODO:** This section is too informal

If you're already sick of reading about fonts, you probably don't want read this section. But alas, dvi post processors and pdfTEX will not work well if you don't provide them `map` files that tell them how to handle the files that contain the glyphs.

In its simplest form, a definition looks as follows:

```
usedname < texnansi.enc < realname.pfb
```

This means as much as: when you want to include a file that has the `tfm` file `usedname`, take the outline file `realname.pfb` and embed it with the `texnansi` encoding vector. Sometimes you need more complicated directives and you can leave that to the experts. We try to keep up with changes in the map file syntax, the names of fonts, encodings, locations in the TEX tree, etc. However, it remains a troublesome area.

It makes sense to take a look at the `cont-sys.rme` file to see what preferences make sense. If you want to speed up the typescript processing, say (in `cont-sys.tex`:

```
\preloadtypescripts
```

If you want to change the default encoding, you should add something:

```
\setupencoding [default=texnansi]
```

You can let ConTEXt load the map files for pdfTEX:

```
\autoloadmapfilestrue
```

The following lines will remove existing references to map files and load a few defaults.

```
\resetmapfiles
\loadmapfile[original-base.map]
\loadmapfile[original-ams-base.map]
\loadmapfile[original-public-lm.map]
```

As said, map files are a delicate matter.

## 6.10 Installing fonts

**TODO:** Document use of MkIVand XƎTEX and in particular OSFONTDIR

Most TEX distributions come with a couple of fonts, most noticeably the Computer Modern Roman typefaces. In order to use a font, TEX has to know its characteristics. These are defined in `tfm` and `vf` files. In addition to these files, on your system you can find a couple of more file types.

| suffix | content |
|---|---|
| tfm | TEX specific font metric files that, in many cases, can be generated from `afm` files |
| vf | virtual font files, used for building glyph collections from other ones |
| afm | Adobe font metric files that are more limited than `tfm` files (especially for math fonts) |
| pfm | Windows specific font metric files, not used by TEX applications |
| pfb | files that contain the outline specification of the glyphs fonts, also called Type 1 |
| enc | files with encoding vector specifications |
| map | files that specify how and what font files are to be included |

On your disk (or cdrom) these files are organized in such a way that they can be located fast.[17] The directory structure normally is as follows:

```
texmf / fonts  / tfm    / vendor    / name / *.tfm
               / afm    / vendor    / name / *.afm
               / pfm    / vendor    / name / *.pfm
               / vf     / vendor    / name / *.vf
               / type1  / vendor    / name / *.pfb
      / pdftex / config /                    *.cfg
               / config /                    *.map
               / config / encoding /         *.enc
```

The `texmf-local` or even better `texmf-fonts` tree normally contains your own fonts, so that you don't have to reinstall them when you reinstall the main tree. The `pdftex` directory contains the files that pdfTEX needs in order to make decisions about the fonts to include. The `enc` files are often part of distributions, as is the configuration `cfg` file. When you install new fonts, you often also have to add or edit `map` files.

ConTEXt comes with a Perl script `texfont.pl` that you can use to install new fonts. Since its usage is covered by a separate manual, we limit ourselves to a short overview.

Say that you have just bought a new font. A close look at the files will reveal that you got at least a bunch of `afm` and `pfb` files and if you're lucky `tfm` files.

Installing such a font can be handled by this script. For this you need to know (or invent) the name of the font vendor, as well as the name of the font. The full set of command line switches is given below:[18]

[17] If you have installed teTEX or fpTEX (possibly from the TEXlive cdrom) you will have many thousands of font files on your system.

| switch | meaning |
|---|---|
| `fontroot` | texmf font root (automatically determined) |
| `vendor` | vendor name (first level directory) |
| `collection` | font collection (second level directory) |
| `encoding` | encoding vector (default: texnansi) |
| `sourcepath` | when installing, copy from this path |
| `install` | copy files from source to font tree |
| `makepath` | when needed, create the paths |
| `show` | run tex on `*.tex` afterwards |

You seldom need to use them all. In any case it helps if you have a local path defined already. The next sequence does the trick:

```
texfont --ve=FontFun --co=FirstFont --en=texnansi --ma --in
```

This will generate the `tfm` files from the `afm` files, and copy them to the right place. The Type 1 files (`pfb`) will be copied too. The script also generates a `map` file. When this is done successfully, a TEX file is generated and processed that shows the font maps. If this file looks right, you can start using the fonts. The TEX file also show you how to define the fonts.

This script can also do a couple of more advanced tricks. Let us assume that we have bought (or downloaded) a new font package in the files `demofont.afm` and `demofont.pfb` which are available on the current (probably scratch) directory. First we make sure that this font is installed (in our case we use a copy of the public Iwona Regular):

```
texfont --ve=test --co=test --ma --in demofont
```

We can now say:

```
\loadmapfile[texnansi-test-test.map]
\definefontsynonym[DemoFont][texnansi-demofont]
\ruledhbox{\definedfont[DemoFont at 50pt]Interesting}
```

Interesting

From this font, we can derive a slanted alternative by saying:

```
texfont --ve=test --co=test --ma --in --sla=.167 demofont
```

The map file is automatically extended with the entry needed.

```
\definefontsynonym[DemoFont-Slanted][texnansi-demofont-slanted-167]
\ruledhbox{\definedfont[DemoFont-Slanted at 50pt]Interesting}
```

Interesting

We can also create a wider version:

```
texfont --ve=test --co=test --ma --in --ext=1.50 demofont
```

When you use the `--make` and `--install` switch, the directories are made, fonts installed, and entries appended to the map file if needed.

```
\definefontsynonym[DemoFont-Extended][texnansi-demofont-extended-1500]
\ruledhbox{\definedfont[DemoFont-Extended at 50pt]Interesting}
```

[18] there are a couple of more switches described in the manual `mtexfonts`.

Interesting

Instead of using pseudo caps in TEX by using `\kap`, you can also create a pseudo small caps font.

```
texfont --ve=test --co=test --ma --in --cap=0.75 demofont
```

This method is much more robust but at the cost of an extra font.

```
\definefontsynonym[DemoFont-Caps][texnansi-demofont-capitalized-750]
\ruledhbox{\definedfont[DemoFont-Caps at 50pt]Interesting}
```

Interesting

| switch | meaning |
|---|---|
| `extend=factor` | stretch the font to the given factor |
| `narrow=factor` | shrink the font to the given factor |
| `slant=factor` | create a slanted font |
| `caps=factor` | replace lowercase characters by small uppercase ones |
| `test` | use test/test as vendor/collection |

When manipulating a font this way, you need to provide a file name. Instead of a factor you can give the keyword `default` or a `*`.

```
texfont --test --auto --caps=default demofont
```

The previous example runs create fonts with the rather verbose names:

```
demofont
demofont-slanted-167
demofont-extended-150
demofont-capitalized-750
```

This naming scheme makes it possible to use more instances without the risk of conflicts.

In the distribution you will find an example batch file `type-tmf.dat` which creates metrics for some free fonts for the encoding specified. When you create the default font metrics this way, preferably `texmf-fonts`, you have a minimal font system tuned for you prefered encoding without the risk for name clashes. When you also supply `--install`, the font outlines will be copied from the main tree to the fonts tree, which sometimes is handy from the perspective of consistency.

## 6.11 Getting started

**TODO:** This section needs to be modernized

The way TEX searches for files (we're talking web2c now) is determined by the configuration file to which the `TEXMFCNF` environment variable points (the following examples are from my own system):

```
set TEXMFCNF=T:/TEXMF/WEB2C
```

When searching for files, a list of directories is used:

```
set TEXMF={$TEXMFFONTS,$TEXMFPROJECT,$TEXMFLOCAL,!!$TEXMFMAIN}
```

Here we've added a font path, which itself is set with:

```
set TEXMFMAIN=E:/TEX/TEXMF
set TEXMFLOCAL=E:/TEX/TEXMF-LOCAL
set TEXMFFONTS=E:/TEX/TEXMF-FONTS
```

Now you can generate metrics and map files. The batch file is searched for at the ConTEXt data path in the texmf tree or on the local path.

```
texfont --encoding=ec --batch type-tmf.dat
```

If you want to play with encoding, you can also generate more encodings, like `8r` or `texnansi`.

```
texfont --encoding=texnansi --batch type-tmf.dat
texfont --encoding=8r       --batch type-tmf.dat
```

After a while, there will be generated `tfm`, `vf`, and `map` files. If you let ConTEXt pass the map file directives to pdfTEX, you're ready now. Otherwise you need to add the names of the mapfiles to the file `pdftex.cfg`. You can best add them in front of the list, and, if you use ConTEXt exclusively, you can best remove the other ones.

As a test you can process the TEX files that are generated in the process. These also give you an idea of how well the encoding vectors match your expectations.

Now, the worst that can happen to you when you process your files, is that you get messages concerning unknown `tfm` files or reports on missing fonts when pdfTEX writes the file. In that case, make sure that you indeed *have* the right fonts (generated) and/or that the map files are loaded. As a last resort you can load all map files by saying:

```
\usetypescript [map] [all]
```

and take a look at the log file and see what is reported.

In due time we will provide font generation scripts for installation of other fonts as well as extend the typescript collection.

## 6.12 Remarks

It really makes sense to take a look at the font and type definition files (`font-*.tex` and `type-*.tex`). There are fallbacks defined, as well as generic definitions. Studying styles and manual source code may also teach you a few tricks.

# 7 Colors

## 7.1 Introduction

Judicious use of color can enhance your document's layout. For example. in interactive documents color can be used to indicate hyperlinks or other aspects that have no meaning in paper documents, or background colors can be used to indicate screen areas that are used for specific information components.

In this chapter we describe the ConTEXt color support. We will also pay attention to backgrounds and overlays because these are related to the color mechanism.

## 7.2 Color

One of the problems in typesetting color is that different colors may result in identical gray shades. We did some research in the past on this subject and we will describe the ConTEXt facilities on this matter and the way ConTEXt forces us to use color consistently. Color should not be used indiscriminately, therefore you first have to activate the color mechanism:

```
\setupcolors[state=start]
```

Other color parameters are also available:

```
\setupcolors [..,.*=.,..]

*   state       =  start stop global local
    conversion  =  yes no always
    reduction   =  yes no
    rgb         =  yes no
    cmyk        =  yes no
    mpcmyk      =  yes no
    mpspot      =  yes no
    textcolor   =  IDENTIFIER
    split       =  c m y k p s no IDENTIFIER
    criterium   =  all none
```

The parameter `state` can also be set at `local` or `global`. If you do not know whether the use of color will cross a page boundary, then you should use `global` or `start` to keep track of the color. We use `local` in documents where color will never cross a page border, as is the case in many screen documents. This will also result in a higher processing speed. (For most documents it does not hurt that much when one simply uses `start`).

By default both the rgb and cmyk colorspaces are supported. When the parameter `cmyk` is set at `no`, then the cmyk color specifications are automatically converted to rgb. The reverse is done when `rgb=no`. When no color is allowed the colors are automatically converted to weighted grayshades. You can set this conversion with `conversion`. When set to `always`, all colors are converted to gray, when set to `yes`, only gray colors are converted.

Colors must be defined. For some default color spaces, this is done in the file `colo-<<xxx>>.tex`. After definition the colors can be recalled with their mnemonic name `<<xxx>>`. By default the file `colo-rgb.tex` is loaded. In this file we find definitions like:

```
\definecolor [darkred]   [r=.5, g=.0, b=.0]
\definecolor [darkgreen] [r=.0, g=.5, b=.0]
........... ..........  .................
```

A file with color definitions is loaded with:

```
\setupcolor[rgb]
```

Be aware of the fact that there is also a command `\setupcolors` that has a different meaning. The `rgb` file is loaded by default.

Color must be activated like this:

```
\startcolor[darkgreen]
We can use as many colors as we like. But we do have to take into
account that the reader is possibly \color [darkred] {colorblind}. The
use of color in the running text should always be carefully considered.
The reader easily tires while reading multi||color documents.
\stopcolor
```

In the same way you can define cmyk colors and grayshades:

```
\definecolor [cyan] [c=1,m=0,y=0,k=0]
\definecolor [gray] [s=0.75]
```

gray can also be defined like this:

```
\definecolor [gray] [r=0.75,r=0.75,b=0.75]
```

When the parameter `conversion` is set at `yes` the color definitions are automatically downgraded to the `s`–form: `[s=.75]`. The `s` stands for 'screen'. When `reduction` is `yes`, the black component of a cmyk color is distilled from the other components.

One of the facillities of color definition is the heritage mechanism:

```
\definecolor [important] [red]
```

These definitions enable you to use colors consistently. Furthermore it is possible to give all important issues a different color, and change colors afterwards or even in the middle of a document.

So, next to `\setupcolors` we have the following commands for defining colors:

```
\definecolor [.1.] [..,.2.,..]
                       =
1   IDENTIFIER

2   r  =  TEXT
    g  =  TEXT
    b  =  TEXT
    c  =  TEXT
    m  =  TEXT
    y  =  TEXT
    k  =  TEXT
    s  =  TEXT
    h  =  TEXT
    t  =  TEXT
    a  =  TEXT
    p  =  TEXT
    e  =  TEXT
```

A color definition file is loaded with:

```
\setupcolor [.*.]

*   IDENTIFIER
```

Typesetting color is done with:

```
\color [.1.] {.2.}

1   TEXT

2   CONTENT
```

```
\startcolor [.*.] ... \stopcolor

*   IDENTIFIER
```

A complete palette of colors is generated with:

```
\showcolor [.*.]

*   IDENTIFIER
```

**Figure 7.1** shows the colors that are standard available (see `colo-rgb.tex`).

The use of color in TEX is not trivial. TEX itself has no color support. Currently color support is implemented using TEX's low level `\mark`'s and `\special`'s. This means that there are some limitations, but in most cases these go unnoticed.

It is possible to cross page boundaries with colors. The headers and footers and the floating figures or tables will stil be set in the correct colors. However, the mechanism is not robust.

In this sentence we use colors within colors. Aesthetically this is bad.

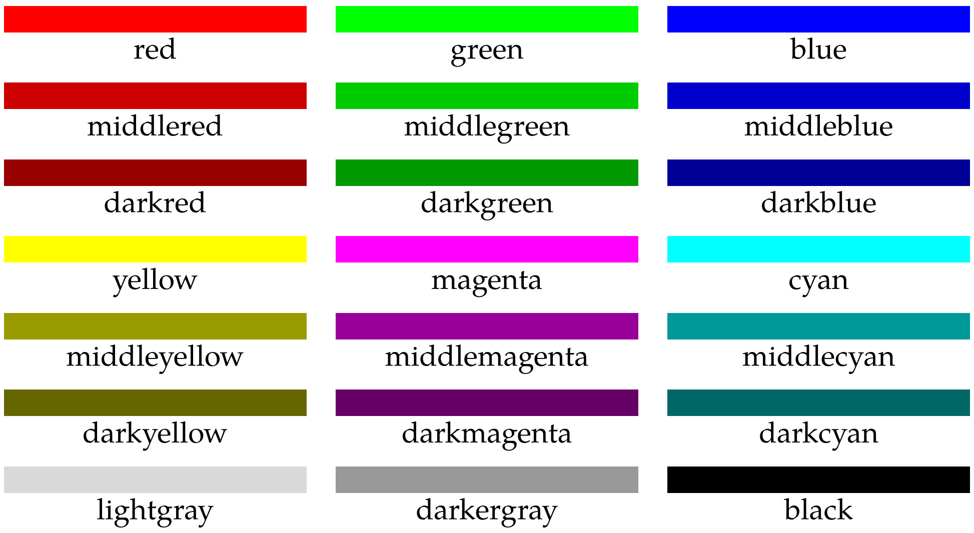


**Figure 7.1** Some examples of colors.

As soon as a color is defined it is also available as a command. So there is a command `\darkred`. These commands do obey grouping. So we can say `{\darkred this is typeset in dark red}`.

There are a number of commands that have the parameter `color`. In general, when a `style` can be set, `color` can also be set.

The default color setup is:

```
\setupcolors [conversion=yes, reduction=no, rgb=yes, cmyk=yes]
```

This means that both colorspaces are supported and that the $k$–component in cmyk colors is maintained. When `reduction=yes`, the $k$–component is 'reduced'. With `conversion=no` equal color components are converted to gray shades.

## 7.3 Grayscales

When we print a document on a black and white printer we observe that the differences between somes colors are gone. **Figure 7.2** illustrates this effect.

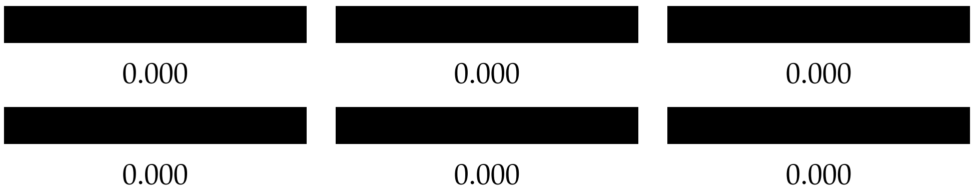


**Figure 7.2** Three cyan variations with equal gray shades.

In a black and white print all blocks look the same but the three upper blocks have different cyan based colors. The lower blocks simulate grayshades. We use the following conversion formula:

$$gray = .30 \times red + .59 \times green + .11 \times blue$$

A color can be displayed in gray with the command:

```
\graycolor [.*.]

*   TEXT
```

The actual values of a color can be recalled by the commands `\colorvalue{<<name>>}` and `\grayvalue{<<name>>}`.

We can automatically convert all used colors in weighted grayshades.

```
\setupcolors [conversion=always]
```

## 7.4 Colorgroups and palettes

TEX itself has hardly any built–in graphical features. However the ConTEXt color mechanism is designed by looking at the way colors in pictures are used. One of the problems is the effect we described in the last section. On a color printer the picure may look fine, but in black and white the results may be disappointing.

In TEX we can aproach this problem systematically. Therefore we designed a color mechanism that can be compared with that in graphical packages.

We differentiate between individual colors and colorgroups. A colorgroup contains a number of gradations of a color. By default the following colorgroups are defined.

undefined

The different gradations within a colorgroup are represented by a number. A colorgroup is defined with:

```
                    1     2
\definecolorgroup [...] [...] [x:y:z=,..]
                        OPTIONAL    3

1   IDENTIFIER

2   rgb cmyk gray s

3   TRIPLET
```

An example of a part of the rgb definition is:

```
\definecolorgroup
  [blue][rgb]
  [1.00:1.00:1.00,
   0.90:0.90:1.00,
   ..............,
   0.40:0.40:1.00,
      0.30:0.30:1.00]
```

The [rgb] is not mandatory in this case, because ConTEXt expects rgb anyway. This command can be viewed as a range of color definitions.

```
\definecolor [blue:1] [r=1.00, g=1.00, b=1.00]
\definecolor [blue:2] [r=0.90, g=0.90, b=1.00]
..............
\definecolor [blue:7] [r=0.40, g=0.40, b=1.00]
\definecolor [blue:8] [r=0.30, g=0.30, b=1.00]
```

A color within a colorgroup can be recalled with `<<name>>:<<number>>`, for example: `blue:4`.

There is no maximum to the number of gradations within a colorgroup, but on the bases of some experiments we advise you to stay within 6 to 8 gradations. We can explain this. Next to colorgroups we have palettes. A pallet consists of a limited number of *logical* colors. Logical means that we indicate a color with a name. An example of a palette is:

undefined

The idea behind palettes is that we have to avoid colors that are indistinguishable in black and white print. A palette is defined by:

```
\definepalet
  [example]
  [strange=red:3,
       top=green:1,
        .....
    bottom=yellow:8]
```

We define a palette with the command:

```
\definepalet [.1.] [..,.=2.,..]

1   IDENTIFIER
2   IDENTIFIER  =  IDENTIFIER
```

ConTEXt contains a number of predefined palettes. Within a palette we use the somewhat abstract names of quarks: *top*, *bottom*, *up*, *down*, *strange* and *charm*. There is also *friend* and *rude* because we ran out of names. Be aware of the fact that these are just examples in the rgb definition file and based on our own experiments. Any name is permitted.

The system of colorgroups and palettes is based on the idea that we compose a palette from the elements of a colorgroup with different numbers. Therefore the prerequisite is that equal numbers should have an equal grayshade.

undefined

When a palette is composed we can use the command:

```
\setuppalet [.*.]

*   IDENTIFIER
```

After that we can use the colors of the chosen palette. The logical name can be used in for example `\color[strange]{is this not strange}`.

An example of the use of palettes is shown in the verbatim typesetting of TEX code. Within this mechanism colors with names like `prettyone`, `prettytwo`, etc. are used. There are two palettes, one for color and one for gray:

```
\definecolor [colorprettyone] [r=.9, g=.0, b=.0]
\definecolor [grayprettyone]  [s=.3]
```

These palettes are combined into one with:

```
\definepalet
  [colorpretty]
  [  prettyone=colorprettyone,    prettytwo=colorprettytwo,
   prettythree=colorprettythree, prettyfour=colorprettyfour]
```

```
\definepalet
  [graypretty]
  [  prettyone=grayprettyone,     prettytwo=grayprettytwo,
   prettythree=grayprettythree,  prettyfour=grayprettyfour]
```

Now we can change all colors by resetting the palette with:

```
\setuptyping[palet=colorpretty]
```

Each filter can be set differently:

```
\definepalet [MPcolorpretty] [colorpretty]
\definepalet [MPgraypretty]  [graypretty]
```

As you can see a palette can inherit its properties from another palette. This example shows something of the color philosophy in ConTEXt: you can treat colors as abstractions and group them into palettes and change these when necessary.

On behalf of the composition of colorgroups and palettes there are some commands available to test whether the colors are distinguishable.

```
\showcolorgroup [.1.] [...,2...]

1   IDENTIFIER

2   horizontal vertical name value NUMBER
```

```
\showpalet [.1.] [...,2...]

1   IDENTIFIER

2   horizontal vertical name value
```

```
\comparecolorgroup [.*.]

*   IDENTIFIER
```

```
\comparepalet [.*.]

*   IDENTIFIER
```

The overviews we have shown thusfar are generated by the first two commands and the gray values are placed below the baseline. On the left there are the colors of the grayshades.

undefined

This overview is made with `\comparecolorgroup[green]` and the one below with `\comparepalet[gamma]`.

undefined

The standard colorgroups and palettes are composed very carefully and used systematically for coloring pictures. These can be displayed adequately in color and black and white.

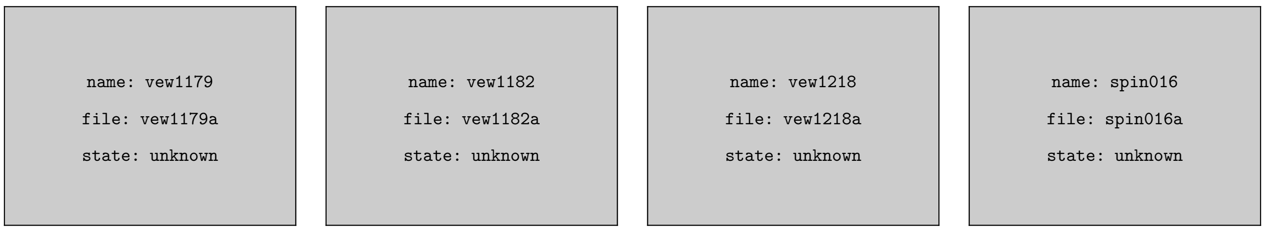


**Figure 7.3** Some examples of the use of color.

# 8 Verbatim text

Text can be displayed in verbatim (typed) form. The text is typed between the commands:

```
\startTYPING ... \stopTYPING
```

Like in:

```
\starttyping
In this text there are enough examples of verbatim text. The command
definitions and examples are typeset with the mentioned commands. Like in
this example.
\stoptyping
```

For in–line typed text the command `\type` is available.

```
\type {.*.}

*   CONTENT
```

A complete file can be added to the text with the command:

```
\typefile [.1.] {.2.}
          OPTIONAL
1   IDENTIFIER
2   CONTENT
```

The style of typing is set with:

```
\setuptyping [.1.] [..,.=2.,..]
              OPTIONAL
1   file typing IDENTIFIER

2   space      =  on off
    page       =  yes no
    option     =  slanted normal commands color none
    text       =  yes no
    icommand   =  COMMAND
    vcommand   =  COMMAND
    ccommand   =  COMMAND
    before     =  COMMAND
    after      =  COMMAND
    margin     =  DIMENSION standard yes no
    evenmargin =  DIMENSION
    oddmargin  =  DIMENSION
    blank      =  DIMENSION small medium big standard halfline line
    escape     =  SETUP:CHARACTER
    space      =  on off
    tab        =  NUMBER yes no
    page       =  yes no
    indentnext =  yes no
    style      =  normal bold slanted boldslanted type cap small... COMMAND
    color      =  IDENTIFIER
    palet      =  IDENTIFIER
    lines      =  yes no hyphenated
    empty      =  yes all no
    numbering  =  line file no
    bodyfont   =  5pt ... 12pt small big
```

This setup influences the display verbatim (`\starttyping`) and the verbatim typesetting of files (`\typefile`) and buffers (`\typebuffer`). The first optional argument can be used to define a specific verbatim environment.

```
\setuptyping[file][margin=default]
```

When the key `space=on`, the spaces are shown:

```
No␣alignment␣is␣to␣be␣preferred
over␣␣␣aligning␣␣␣by␣␣␣means␣␣of
spaces␣or␣the␣s␣t␣r␣e␣t␣c␣h␣i␣n␣g␣of␣words
```

A very special case is:

```
\definetyping
  [broadtyping]

\setuptyping
  [broadtyping]
  [oddmargin=-1.5cm,evenmargin=-.75cm]
```

This can be used in:

```
\startbroadtyping
A verbatim line can be very long and when we don't want to hyphenate we
typeset it in the margin on the uneven pages.
\stopbroadtyping
```

At a left hand side page the verbatim text is set in the margin.

```
A verbatim line can be very long and when we don't want to hyphenate we
typeset it in the margin on the uneven pages.
```

An in–line verbatim is set up by:

```
\setuptype [..,.*=.,..]

*   space    =  on off
    option   =  slanted normal none
    style    =  normal bold slanted boldslanted type cap small... COMMAND
    color    =  IDENTIFIER
```

When the parameter `option` is set at `slanted` all text between `<<` and `>>` is typeset in a *`slanted letter`*. This feature can be used with all parameters. In this way `\type{aa<<bb>>cc}` will result in: `aa<<bb>>cc`.

For reasons of readability you can also use other characters than `{` and `}` as *outer* parenthesis. You can choose your own non–active (a non–special) character, for example: `\type+like this+` or `\type-like that-`. Furthermore you can use the mentioned `<<` and `>>`, as in `\type<<like this>>` or even `\type<like that>`.

The parameter `option=commands` enables you to process commands in a typed text. In this option `\` is replaced by `/`. This option is used for typesetting manuals like this one. For example:

```
\seethis <</rm : this command has no effect>>
 /vdots
\sihtees <</sl : neither has this one>>
```

The double `<<` and `>>` overtake the function of `{}`.

Within the type–commands we are using `\tttf`. When we would have used `\tt`, the `\sl` would have produced a slanted and `\bf` a bold typeletter. Now this will not happen:

```
\seethis <</rm : this command has no effect>>
 /vdots
\sihtees <</sl : neither has this one>>
```

One of the most interesting options of typesetting verbatim is a program source code. We will limit the information on this topic and refer readers to the documentation in the files `verb-<<xxx>>.tex` and `cont-ver.tex`. In that last file you can find the following lines:

```
\definetyping [MP]  [option=MP]
\definetyping [PL]  [option=PL]
\definetyping [JS]  [option=JS]
\definetyping [TEX] [option=TEX]
```

Here we see that it is possible to define your own verbatim environment. For that purpose we use the command:

```
\definetyping [.1.] [..,.2=.,..]

1   inherits from \setuptyping

2   inherits from \setuptyping
```

The definitions above couple such an environment to an option.

```
\startMP
beginfig (12) ;
  MyScale = 1.23 ;
  draw unitsquare scaled MyScale shifted (10,20) ;
endfig ;
\stopMP
```

In color (or reduced gray) this will come out as:

```
beginfig (12) ;
  MyScale = 1.23 ;
  draw unitsquare scaled MyScale shifted (10,20) ;
endfig ;
```

These environments take care of typesetting the text in such a way that the typographics match the chosen language. It is possible to write several filters. Languages like MetaPost, MetaFont, Perl, JavaScript, sql, and off course TEX are supported. By default color is used to display these sources, where several palettes take care of the different commands. That is why you see the parameter `palet` in `\setuptyping`. One can use font changes or even own commands instead, by assigning the appropriate values to the `icommand` (for identifiers), `vcommand` (for variables) and `ccommand` parameters (for the rest). By default we have:

```
\setuptyping [icommand=\ttsl, vcommand=, ccommand=\tf]
```

We have some alternatives for `\type`. When typesetting text with this command the words are not hyphenated. Hyphenation is performed however when one uses:

```
\typ {.*.}

*   CONTENT
```

When you are thinking of producing a manual on TEX you have a command that may serve you well:

```
\tex {.*.}

*   CONTENT
```

This command places a \ in front of typed text.

# 9 Backgrounds and Overlays

## 9.1 Text backgrounds

In a number of commands, for example `\framed`, you can use backgrounds. A background may have a color or a screen (pure gray). By default the `backgroundscreen` is set at `0.95`. Usable values lie between 0.70 and 1.00.

Building screens in TEX is memory consuming and may cause error messages. The screens are therefore build up externally by means of PostScript or pdf instructions. This is set up with:

```
\setupscreens [..,.*=.,..]

*   method      =  dot rule external
    resolution  =  NUMBER
    factor      =  NUMBER
    screen      =  NUMBER
```

The parameter `factor` makes only sense when the method `line` or `dot` is chosen. The parameter `screen` determines the 'grid' of the screen. Text on a screen of 0.95 is still readable.

Visually the TEX screens are comparable with PostScript screens. When memory and time are non issues TEX screens come out more beautiful than postscript screens. There are many ways to implement screens but only the mentioned methods are implemented.

Behind the text in the pagebody screens can be typeset. This is done by enclosing the text with the commands:

```
\startbackground
\stopbackground
```

We have done so in this text. Backgrounds can cross page boundaries when necessary. Extra vertical whitespace is added around the text for reasons of readability.

```
\startbackground {.*.} ... \stopbackground

*   CONTENT
```

The background can be set up with:

```
\setupbackground [..,.*=.,..]

*   leftoffset    =  DIMENSION
    rightoffset   =  DIMENSION
    topoffset     =  DIMENSION
    bottomoffset  =  DIMENSION
    before        =  COMMAND
    after         =  COMMAND
    state         =  start stop
    inherits from \setupframed
```

The command `\background` can be used in combination with for example placeblocks:

```
\placetable
  {Just a table.}
  \background
  \starttable[|c|c|c|]
  \HL
  \VL red  \VL green   \VL blue   \VL \AR
  \VL cyan \VL magenta \VL yellow \VL \AR
  \HL
  \stoptable
```

The command `\background` expects an argument. Because a table is 'grouped' it will generate `{}` by itself and no extra braces are necessary.

```
\background {.*.}

*   CONTENT
```

A fundamental difference between colors and screens is that screens are never converted. There is a command `\startraster` that acts like `\startcolor`, but in contrast to the color command, ConTEXt does not keep track of screens across page boundaries. This makes sense, because screens nearly always are used as simple backgrounds.

## 9.2 Layout backgrounds

In interactive or screen documents the different screen areas may have different functions. Therefore the systematic use of backgrounds may seem obvious. It is possible to indicate all areas or compartments of the pagebody (screenbody). This is done with:

```
\setupbackgrounds [.1.] [...,2,...] [..,..=3..,..]
                  OPTIONAL  OPTIONAL
1   top header TEXT footer bottom page paper leftpage rightpage

2   leftedge leftmargin TEXT rightmargin rightedge

3   state   =  start stop cd:repeat
    inherits from \setupframed
```

Don't confuse this command with `\setupbackground` (singular). A background is only calculated when something has changed. This is more efficient while generating a document. When you want to calculate each background separately you should set the parameter `state` at `repeat`. The page background is always recalculated, since it provides an excellent place for page dependent buttons.

After `\setupbackgrounds` without any arguments the backgrounds are also re–calculated.

A specific part of the layout is identified by means of an axis (see **figure 9.1**).

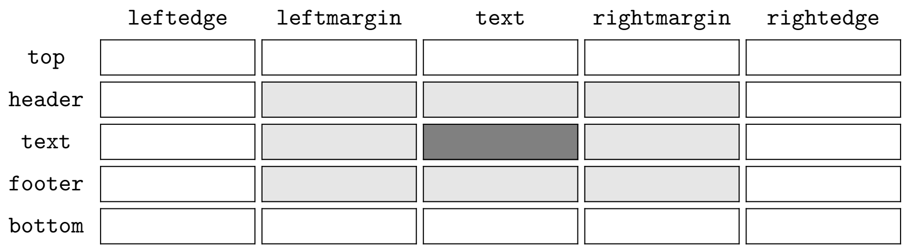


**Figure 9.1** The coordinates in `\setupbackgrounds`.

You are allowed to provide more than one coordinate at a time, for example:

```
\setupbackgrounds
  [header,text,footer]
  [text]
  [background=screen]
```

or

```
\setupbackgrounds
  [text]
  [text,rightedge]
  [background=color,backgroundcolor=MyColor]
```

Some values of the paremeter page, like `offset` and `corner` also apply to other compartments, for example:

```
\setupbackgrounds
  [page]
  [offset=.5\bodyfontsize
   depth=.5\bodyfontsize]
```

When you use menus in an interactive or screen document alignment is automatically adjusted for offset and/or depth. It is also possible to set the parameter page to the standard colors and screens.

If for some reason an adjustment is not generated you can use `\setupbackgrounds` (without an argument). In that case ConTEXt will calculate a new background.

## 9.3 Overlays

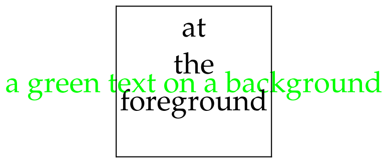
TEX has only limited possibilities to enhance the layout with specific features. In ConTEXt we have the possibility to 'add something to a text element'. You can think of a drawing made in some package or other ornaments. What we technically do is lay one piece of text over another piece text. That is why we speak of 'overlays'.

When we described the backgrounds you saw the paremeters `color` and `screen`. These are both examples of an overlay. You can also define your own background:

```
\defineoverlay[gimmick][\green a green text on a background]
```

```
\framed
  [height=2cm,background=gimmick,align=middle]
  {at\\the\\foreground}
```

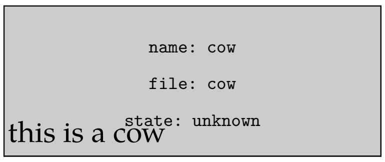
This would look like this:



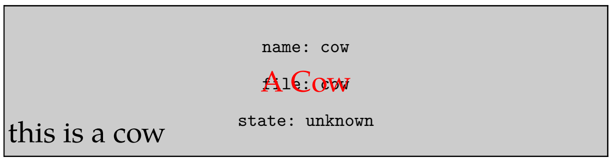
An overlay can be anything:

```
\defineoverlay
  [gimmick]
  [{\externalfigure[cow][width=\overlaywidth,height=\overlayheight]}]
\framed
  [height=2cm,width=5cm,background=gimmick,align=right]
  {\vfill this is a cow}
```

We can see that in designing an overlay the width and height are available in macros. This enables us to scale the figure.



We can combine overlays with one another or with a screen and color.



The TEX definitions look like this:

```
\defineoverlay
  [gimmick]
  [{\externalfigure[cow][width=\overlaywidth,height=\overlayheight]}]
\defineoverlay
  [nextgimmick]
  [\red A Cow]
\framed
  [height=2cm,width=.5\textwidth,
   background={screen,gimmick,nextgimmick},align=right]
  {\vfill this is a cow}
```

# 10 Language specific issues

## 10.1 Introduction

One of the more complicated corners of ConTEXt is the department that deals with languages. Fortunately users will seldom notice this, but each language has its own demands and we put quite some effort in making sure that most of the issues on hyphenation rules and accented and non latin characters could be dealt with. For as long as it does not violate the ConTEXt user interface, we also support existing input schemes.

In the early days TEX was very American oriented, but since TEX version 3 there is (simultaneous) support for multiple languages. The input of languages with many accents —sometimes more accents per character— may look rather complicated, depending on the use of dedicated input encodings or special TEX commands.

The situation is further complicated by the fact that specific input does not have a one–to–one relation with the position of a glyph in a font. We discussed this in section ??. It is important to make the right choices for input and font encoding.

In this chapter we will deal with hyphenation and language specific labels. More details can be found in the language definition files (`lang-<<xxx>>`), the font files (`font-<<xxx>>`) and the encoding files (`enco-<<xxx>>`). There one can find details on how to define commands that deal with accents and special characters as covered in a previous chapter, sorting indexes, providing support for Unicode, and more.

## 10.2 Automatic hyphenating

Each language has its own hyphenation rules. As soon as you switch to another language, ConTEXt will activate the appropriate set of hyphenation patterns for that language. Languages are identified by their official two character identifiers, like: Dutch (`nl`), English (`en`), German (`de`) and French (`fr`). A language is chosen with the following command:[19]

```
\language [.*.]

*   nl fr en uk de es cz ..
```

Some short cut commands are also available. They can be used enclosed in braces:

```
\nl  \en  \de  \fr  \sp  \uk  \pl  \cz  ...
```

The command `\language[nl]` can be compared with `\nl`. The first command is more transparant. The two character commands may conflict with existing commands. Take, for example, Italian and the code for *italic* type setting. For this reason we use capitals for commands that may cause any conflicts. One may also use the full names, like `czech`.

At any instance you can switch to another language. In the example below we switch from English to French and vice versa.

[19] In case of any doubt please check if the hyphenation patterns are included in the `fmt`–file.

```
The French composer {\fr Olivier Messiaen} wrote \quote {\fr Quatuor pour
la fin du temps} during the World War II in a concentration camp. This
may well be one of the most moving musical pieces of that period.
```

We use these language switching commands if we cannot be certain that an alternative hyphenation pattern is necessary.

The French composer Olivier Messiaen wrote ‘Quatuor pour la fin du temps’ during the World War II in a concentration camp. This may well be one of the most moving musical pieces of that period.

How far do we go in changing languages. Borrowed words like perestrojka and glasnost are often hyphenated okay, since these are Russian words used in an English context. When words are incorrectly hyphenated you can define an hyphenation pattern with the TEX–command:

```
\hyphenation{<<ab-bre-via-tion>>}
```

You can also influence the hyphenation in a text by indicating the allowed hyphenation pattern in the word: at the right locations the command `\-` is added: `al\-lo\-wed`.

## 10.3 Definitions and setups

When a format file is generated the hyphenation pattern one needs should be added to this file. The definition and installation of a language is therefore not transparant for the user. We show the process to give some insight in the mechanism. An example:[20]

```
\installlanguage
  [en]
  [spacing=broad,
   leftsentence=---,
   rightsentence=---,
   leftsubsentence=---,
   rightsubsentence=---,
   leftquote=\upperleftsinglesixquote,
   rightquote=\upperrightsingleninequote,
   leftquotation=\upperleftdoublesixquote,
   rightquotation=\upperrightdoubleninequote,
   date={month,\ ,day,{,\ },year},
   default=en,
   state=stop]
```

and:

```
\installlanguage
  [uk]
  [default=en,
   state=stop]
```

With the first definition you define the language component. You can view this definition in the file `lang-ger.tex`, the german languages. Languages are arranged in language groups.

[20] The somewhat strange name `\upperleftsinglesixquote` is at least telling us what the quote will look like.

This arrangement is of no further significance at the moment. Since language definitions are preloaded, users should not bother about setting up such files.

The second definition inherits its set up from the English installation. In both definitions `state` is set at `stop`. This means that no patterns are loaded yet. That is done in the files `cont-<<xx>>`, the language and interface specific ConTEXt versions. As soon as `state` is set at `start`, a new pattern is loaded, which can only be done during the generation of a format file.

We use some conventions in the file names of the patterns `lang-xx.pat` and the exceptions `lang-xx.hyp`. Normally a language is installed with a two character code. However there are three character codes, like deo for hyphenating 'old deutsch' and `nlx` the Dutch extended characterset, or 8–bit encoding. On distributions that come with patterns, the filenames mentioned can be mapped onto the ones available on the system. This happens in the file `cont-usr.tex`.

After installation you are not bound to the two character definitions. Default the longer (English) equivalents are defined:

```
\installlanguage[german][de]
```

```
\installlanguage [.1.] [..,.2=.,..]

1   IDENTIFIER

2   spacing              =  packed broad
    lefthyphenmin        =  NUMBER
    righthyphenmin       =  NUMBER
    state                =  start stop
    leftsentence         =  COMMAND
    rightsentence        =  COMMAND
    leftsubsentence      =  COMMAND
    rightsubsentence     =  COMMAND
    leftquote            =  COMMAND
    rightquote           =  COMMAND
    leftquotation        =  COMMAND
    rightquotation       =  COMMAND
    leftspeech           =  COMMAND
    middlespeech         =  COMMAND
    rightspeech          =  COMMAND
    limittext            =  TEXT
    date                 =  TEXT
    compoundhyphen       =  COMMAND
    leftcompoundhyphen   =  COMMAND
    rightcompoundhyphen  =  COMMAND
    default              =  IDENTIFIER
```

```
\setuplanguage [.1.] [..,.2=.,..]

1   nl fr en uk de es cz ..

2   inherits from \installlanguage
```

The setup in these commands relate to the situations that are shown below.

```
\currentdate
|<|all right there we go|>|
|<| |<|all right|>| there we go|>|
|<|all right |<|there|>| we go|>|
\quote{all right there we go}
\quotation{all right there we go}
\quotation{\quote{all right} there we go}
\quotation{all right \quote{there} we go}
```

This becomes:

September 27, 2013
—all right there we go—
— —all right— there we go—
—all right —there— we go—
'all right there we go'
"all right there we go"
"'all right' there we go"
"all right 'there' we go"

We will discuss `||` in one of the next sections.

## 10.4 Date

Typesetting a date is also language specific so we have to pay some attention to dates here. When the computer runs at the actual time and date the system date can be recalled with:

```
\currentdate [...,*...]

*   inherits from \date
```

The sequence in which `day`, `month` and `year` are given is not mandatory. The pattern `[day,month,year]` results in 27 September 2013. We use `\currentdate[weekday,month,day,{,},year]` to obtain Friday September 27,2013.

A short cut looks like this: `[dd,mm,yy]` and will result in 270913. Something like `[d,m,y]` would result in 27September2013 and with `[referral]` you will get a 20130927. Combinations are also possible. Characters can also be added to the date pattern. The date 27–09–13 is generated by the pattern `[dd,--,mm,--,yy]`.

A date can be (type)set with the command:

```
\date [..,..=..,..] [...,...]
            1           2
        OPTIONAL    OPTIONAL
1   d  =  NUMBER
    m  =  NUMBER
    y  =  NUMBER
2   day month year weekday d m y w dd mm yy space -- day+ d+ dd+ referral TEXT
```

The first (optional) argument is used to specify the date:

```
\date[d=10,m=3,y=1996][weekday,month,day, year]
```

When no argument is given you will obtain the actual date. When the second argument is left out the result equals that of `\currentdate`. The example results in:

Sunday March 10 1996

## 10.5 Labels and heads

In some cases ConTEXt will generate text labels automatically, for example the word **Figure** is generated automatically when a caption is placed under a figure. These kind of words are called textlabels. Labels can be set with the command:

```
\setuplabeltext [.1.] [.2.]
                 OPTIONAL
1   nl fr en uk de es cz ..
2   IDENTIFIER  =  TEXT
```

Relevant labels are: `table`, `figure`, `chapter`, `appendix` and comparable text elements. An example of such a set up is:

```
\setuplabeltext[en][chapter=Chapter ]
\setuplabeltext[nl][hoofdstuk=Hoofdstuk ]
```

The space after `Chapter` is essential, because otherwise the chapternumber will be placed right after the word Chapter (Chapter1 instead of Chapter 1). A labeltext can recalled with:

```
\labeltext {.*.}
*   CONTENT
```

Some languages, like Chinese, use split labels. These can be passed as a comma separated list, like `chapter={left,right}`.

Titleheads for special sections of a document, like abbreviations and appendices are set up with:

```
\setupheadtext [.1.] [.2.]
                OPTIONAL
1   nl fr en uk de es cz ..
2   IDENTIFIER  =  TEXT
```

Examples of titleheads are `Content`, `Tables`, `Figures`, `Abbreviations`, `Index` etc. An example definition looks like:

```
\setupheadtext[content=Content]
```

A header can be recalled with:

```
\headtext {.*.}
*   CONTENT
```

Labels and titleheads are defined in the file `lang-<<xxx>>`. You should take a look in these files to understand the use of titleheads and labels.

The actual language that is active during document generation does not have to be the same language that is used for the labels. For this reason next to `\language` we have:

```
\mainlanguage [.*.]

*   nl fr en uk de es cz ..
```

When typesetting a document, there is normally one main language, say `\mainlanguage[en]`. A temporary switch to another language is then accomplished by for instance `\language[nl]`, since this does not influence the labels and titles. language.

## 10.6 Language specific commands

German TEX users are accustomed to entering `"e` and getting ë typeset in return. This and a lot more are defined in `lang-ger` using the compound character mechanism built in ConTEXt. Certain two or three character combinations result in one glyph or proper hyphenation. The example below illustrates this. Some macros are used that will not be explained here. Normally, users can stick to simply using the already defined commands.

```
\startlanguagespecifics[de]
  \installcompoundcharacter "a  {\moveaccent{-.1ex}\"a\midworddiscretionary}
  \installcompoundcharacter "s  {\SS}
  .....
  \installcompoundcharacter "U  {\smashaccent\"U}
  \installcompoundcharacter "Z  {SZ}
  .....
  \installcompoundcharacter "ck {\discretionary {k-}{k}{ck}}
  \installcompoundcharacter "TT {\discretionary{TT-}{T}{TT}}
  .....
  \installcompoundcharacter "`  {\handlequotation\c!leftquotation}
\stoplanguagespecifics
```

The command `\installcompoundcharacter` takes care of the German type setting, `"a` is converted to "a, `"U` in "U, `"ck` for the right hyphenation, etc. One can add more definitions, but this will violate portability. In a Polish ConTEXt the `/` is used instead of a `"`.

## 10.7 Automatic translation

It is possible to translate a text automatically in the actual language. This may be comfortable when typesetting letterheads. The example below illustrates this.

```
\translate [..,.*=.,..]

*   IDENTIFIER  =  TEXT
```

```
It depends on the actual language whether a labeltext is type set in
English {\en as an \translate [en=example, fr=exemple], \fr or in French
as an \translate}.
```

The second command call `\translate` uses the applied values. That is, `\translate` with no options uses the options of the last call to `\translate`.

It depends on the actual language whether a labeltext is type set in English as an example, or in French as an exemple.

## 10.8 Composed words

Words consisting of two separate words are often separated by an intra word dash, as in x–axis. This dash can be placed between `| |`, for example `|--|`. This command, which does not begin with a `\`, serves several purposes. When `||` is typed the default intra word dash is used, which is `--`. This dash is set up with:

```
\setuphyphenmark [..=..]

*   sign  =  -- --- - ~ ( ) = /
```

The `| |` is also used in word combinations like (intra)word, which is typed as `(intra|)|word`. The mechanism is not foolproof but it serves most purposes. In case the hyphenation is incorrect you can hyphenate the first word of the composed one by hand: `(in\-tra|)|word`.

| **input** | **normal** | **hyphenated** |
|---|---|---|
| `intra\|\|word` | intra–word | in-tra–word |
| `intra\|-\|word` | intra-word | in-tra-word |
| `intra\|(\|word)` | intra(word) | in-tra(word) |
| `(intra\|)\|word` | (intra)word | (in-tra-word |
| `intra\|--\|word` | intra–word | in-tra–word |
| `intra\|~\|word` | intra word | in-tra-word |

**Table 10.1** Hyphenation of composed words.

The main reason behind this mechanism is that TEX doesn't really know how to hyphenate composed words and how to handle subsentences. TEX know a lot about math, but far less about normal texts. Using this command not only serves consistency, but also makes sure that TEX can break compound words at the right places. It also keeps boundary characters at the right place when a breakpoint is inserted.

## 10.9 Pattern files manual

**TODO:** A large part of this section is obsolete

TEX has two mysterious commands that the average user will never or seldom meet:

```
\hyphenation{as-so-ciates}
\patterns   {.ach4}
```

Both commands can take multiple strings, so in fact both commands should be plural. The first command can be given any time and can be used to tell TEX that a word should be hyphenated in a certain way. The second command can only be issued when TEX is in virgin mode, i.e. starting with a clean slate. Normally this only happens when a format is generated.

The second command is more mysterious than the first one and its entries are a compact way to tell TEX between what character sequences it may hyphenate words. The numbers represent weights and the (often long) lists of such entries are generated with a special program called `patgen`. Since making patterns is work for specialists, we will not go into the nasty details here.

In the early stage of ConTEXt development it came with its own pattern files. Their names started with `lang-` and their suffixes were `pat` and `hyp`.

However, when ConTEXt went public, I was convinced to drop those files and use the files already available in distributions. This was achieved by using the ConTEXt filename remapping mechanism. Although those files are supposed to be generic, this is not always the case, and it remains a gamble if they work with ConTEXt. Even worse, their names are not consistent and the names of some files as well as locations in the tree keep changing. The price ConTEXt users pay for this is lack of hyphenation until such changes are noticed and taken care of. Because constructing the files is an uncoordinated effort, all pattern files have their own characteristics, most noticably their encoding.

After the need to adapt the name mapping once again, I decided to get back to providing ConTEXt specific pattern files. Pattern cooking is a special craft and TEX users may call themselves lucky that it's taken care of. So, let's start with thanking all those TEX experts who dedicate their time and effort to get their language hyphenated. It's their work we will build (and keep building) upon.

In the process of specific ConTEXt support, we will take care of:

- consistent naming, i.e. using language codes when possible as a prelude to a more sophisticated naming scheme, taking versions into account
- consistent splitting of patterns and hyphenation exceptions in files that can be recognized by their suffix
- making the files encoding independent using named glyphs
- providing a way to use those patterns in plain TEX as well

Instead of using a control sequence for the named glyphs, we use a different notation:

```
[ssharp] [zcaron] [idiaeresis]
```

The advantage of this notation is that we don't have to mess with spacing so that parsing and cleanup with scripts becomes more robust. The names conform to the ConTEXt way of naming glyphs and the names and reverse mappings are taken from the encoding files in the ConTEXt distribution, so you need to have ConTEXt installed.

The ConTEXt pattern files are generated by a Ruby script. Although the converting is rather straightforward, some languages need special treatment, but a script is easily adapted. If you want a whole bunch of pattern files, just say:

```
ctxtools --patterns all
```

or, if you want one language:

```
ctxtools --patterns nl
```

If for some reason this program does not start, try:

```
texmfstart ctxtools --patterns nl
```

When things run well, this will give you four files:

| | |
|---|---|
| `lang-nl.pat` | the patterns in an encoding indepent format |
| `lang-nl.hyp` | the hyphenation exceptions |
| `lang-nl.log` | the conversion log (can be deleted afterwards) |
| `lang-nl.rme` | the preambles of the files used (copyright notices and such) |

If you redistribute the files, it makes sense to bundle the `rme` files as well, unless the originals are already in the distribution. It makes no sense to keep the log files on your system. When the file `lang-all.xml` is present, the info from that file will be used and added to the pattern and hyphenation files. In that case no `rme` and `log` file will be generated, unless `--log` is provided.

In the Dutch pattern file you will notice entries like the following:

```
e[ediaeresis]n3
```

So, instead of those funny (encoding specific) `^^fc` or (format specific) `\"e` we use names. Although this looks ConTEXt dependent it is rather easy to map those names back to characters, especially when one takes into account that most languages only have a few of those special characters and we only have to deal with lower case instances.

The ConTEXt support module `supp-pat.tex` is quite generic and contains only a few lines of code. Actually, most of the code is dedicated to the simple xml handler. Loading a pattern meant for EC encoded fonts in another system than ConTEXt is done as follows:

```
\bgroup

  \input supp-pat

  \lccode"E4="E4  \definepatterntoken adiaeresis ^^e4
  \lccode"F6="F6  \definepatterntoken odiaeresis ^^f6
  \lccode"FC="FC  \definepatterntoken ediaeresis ^^fc
  \lccode"FF="FF  \definepatterntoken ssharp     ^^ff

  \enablepatterntokens
  \enablepatternxml

  \input lang-de.pat
  \input lang-de.hyp

\egroup
```

In addition to this one may want to set additional lower and uppercase codes. In $\varepsilon$-TEX these are stored with the language.

Just for completeness we provide the magic command to generate the xml variants:

```
ctxtools --patterns --xml all
```

This will give you files like:

```
<?xml version='1.0' standalone='yes'?>

<!-- some comment -->

<patterns>
... e&ediaeresis;n3 ...
</patterns>
```

This is also accepted as input but for our purpose it's probably best to stick to the normal method. The pattern language is a TEX specific one anyway.

## 10.10 Installing languages

Installing a language in ConTEXt should not take too much effort assuming the language is supported. Language specific labels are grouped in `lang-*` files, like `lang-ger.tex` for the germanic languages.

Patterns will be loaded from the files in the general TEX distribution unless `lang-nl.pat` is found, in which case ConTEXt assumes that you prefer the ConTEXt patterns. In that case, run

```
ctxtools --patterns all
```

You need to move the files to the ConTEXt base path that you can locate with:

```
textools --find context.tex
```

You can also use `kpsewhich`, but the above method does an extensive search. Of course you can also generate the files on a temporary location. Now it's time to generate the formats:

```
texexec --make --all
```

Since XETEX needs patterns in utf-8 encoding, we provide a switch for achieving that:

```
texexec --make --all --utf8
```

Beware: you need to load patterns for each language and encoding combination you are going to use. You can configure your local `cont-usr` file to take care of this. When an encoding does not have the characters that are needed, you will get an error. When using the non ConTEXt versions of teh patterns this may go unnoticed because the encoding is hard coded in the file. Of course it will eventually get noticed when the hyphenations come out wrong.

The ConTEXt distribution has a file lang-all.xml that holds the copyright and other notes of the patterns. A discription looks like:

```
<description language='nl'>
  <sourcefile>nehyph96.tex</sourcefile>
  <title>TeX hyphenation patterns for the Dutch language</title>
  <copyright>
    <year>1996</year>
    <owner> Piet Tutelaers (P.T.H.Tutelaers@tue.nl)</owner>
    <comment>8-bit hyphenation patterns for TeX based upon the new
      Dutch spelling, officially since 1 August 1996. These
      patterns follow the new hyphenation rules in the
      `Woordenlijst Nederlandse Taal, SDU Uitgevers, Den Haag
      1995' (the so called `Groene Boekje') described in
      section 5.2 (Het afbreekteken)</comment>
  </copyright>
</description>
```

*This file is 'work in process': more details will be added and comments will be enriched.*

## 10.11 Commands

You can at any moment add additional hyphenation exceptions to the language specific dictionaries. For instance:

```
\language[nl] \hyphenation{pa-tiën-ten}
```

Switching to another language is done with the `\language` command. The document language is set with `\mainlanguage`.

If you want to let TEX know that a word should be hyphenated in a special way, you use the `\-` command, for instance:

```
Con\-TeXt
```

Compound words are not recognized by the hyphenation engine, so there you need to add directives, like:

```
the ConTeXt|-|system
```

If you are using xml as input format, you need to load the hyphenation filter module. Here we assume that utf encoding is used:

```
\useXMLfilter[utf,hyp]
```

In your xml file you can now add:

```
<hyphenations language='nl' regime='utf'>
  <hyphenation>pa-tiën-ten</hyphenation>
  <hyphenation>pa-tiën-ten-or-ga-ni-sa-tie</hyphenation>
  <hyphenation>pa-tiën-ten-plat-form</hyphenation>
</hyphenations>
```

This filter also defines some auxiliary elements. Explicit hyphenation points can be inserted as follows:

```
Zullen we hier af<hyphenate/>bre<hyphenate/>ken of niet?
```

The compound token can be anything, but keep in mind that some tokens are treated special (see other manuals).

```
Wat is eigenlijk een patiënten<compound token="-"/>platform?
```

A language is set with:

```
nederlands <language code="en">english</language> nederlands
```

If you set attribute `scope` to `global`, labels (as used for figure captions and such) adapt to the language switch. This option actually invokes `\mainlanguage`.

## 10.12 Languages

When users in a specific language area use more than one font encoding, patterns need to be loaded multiple times. In theory this means that one can end up with more instances than TEX can host. However, the number of sensible font encodings is limited as is the number of languages that need hyphenation. Now that memory is cheap and machines are fast, preloading a lot of pattern files is no problem. The following table shows the patterns that are preloaded in the version of ConTEXt that is used to process this file.

**FIXME:** `\showpatterns` doesn't exist anymore

*In the (near) future the somewhat arcane `pl0` and `il2` encodings will go away since they are only used for Polish and Czech/Slovak computer modern fonts, which can be replaced by Latin Modern alternatives. Also, a new dense encoding may find its way into this list.*

## 10.13 Hyphenation

While hypenating, TEX has to deal with ligatures as well. While Thomas, Taco and I were discussing the best ways to neutralize the ancient greek patterns, Taco Hoekwater came up with the following explanation.[21]

fi fl ffi ffl

Any direct use of a ligature (as accessed by `\char` or through active characters) is wrong and will create faulty hypenation. Normally, when TeX sees 'office', it has the six tokens `office` and it knows from the patterns that it can hyphenate between the `ff`. It will build an internal list of four nodes, like this:

```
[char, o  , ffi ]
[lig , ffi, c   ,[f,f,i]]
[char, c  , e   ]
[char, e  , NULL]
```

As you can see from the `ffi` line, it has remembered the original characters. While hyphenating, it temporarily changes back to that, then re-instates the ligature afterwards.

If you feed it the ligature directly, like so:

```
[char, o   , ffi ]
[char, ffi , c   ]
[char, c   , e   ]
[char, e   , NULL]
```

it cannot do that. It tries to hyphenate as if the `ffi` was a character, and the result is wrong hyphenation.

[21] Thomas Schmitz is responsible for the associated third party module.

# 11 Text elements

## 11.1 Introduction

The core of ConTeXt is formed by the commands that structures the text. The most common structuring elements are chapters and sections. The structure is visualized by means of titles and summarized in the table of contents.

A text can be subdivided in different ways. As an introduction we use the methods of H. van Krimpen, K. Treebus and the Collectief Gaade. First we examine the method of van Krimpen:

1. French title
2. title
3. history & copyright
4. mission
5. preface/introduction
6. . . .
7. list of illustrations
8. acknowledgement
9. errata
10. the content
11. notes
12. literature
13. register(s)
14. colofon

The French title is found at the same spread as the back of the cover, or first empty sheet. In the colofon we find the used font, the names of the typesetter and illustrator, the number of copies, the press, the paper, the binding, etc.

The subdivision of Treebus looks like this:

1. French title
2. titlepage
3. colofon
4. copyright
5. mission
6. preface (1)
7. table of content
8. list of illustrations
9. introduction/preface (2)
10. . . .
11. epilogue
12. appendices
13. summaries
14. notes
15. literature
16. used words
17. addenda
18. register
19. acknowledgement photos
20. (colofon)

In this way of dividing a text the colofon is printed on the back of the titlepage. The first preface is written by others and not by the author.

The last text structure is that of the Collectief Gaade:

1. French title
2. series title
3. title
4. copyright
5. mission
6. blank
7. preface
8. table of content
9. introduction
10. . . .
11. appendices
12. notes
13. list of illustrations
14. used words
15. bibliography
16. colofon
17. register

Since there seems to be no standardized way of setting up a document, ConTEXt will only provide general mechanisms. These are designed in such a way that they meet the following specifications:

1. In a text the depth of sectioning seldom exceeds four. However, in a complex manuals more depth can be useful. In paper documents a depth of six may be very confusing for the reader but in electronic documents we need far more structure. This is caused by the fact that a reader cannot make a visual representation of the electronic book. Elements to indicate this structure are necessary to be able to deal with the information.
2. Not every level needs a number but in the background every level is numbered to be able to refer to these unnumbered structuring elements.
3. The names given to the structuring elements must be a logical ones and must relate to their purpose.
4. It is possible to generate tables of contents and registers at every level of the document and they must support complex interactivity.
5. A document will be divided in functional components like introductions and appendices with their respective (typographical) characteristics.
6. The hyphenation of titles must be handled correctly.
7. Headers and footers are supported based on the standard labels used in a document. For example `chapter` in a book and `procedure` in a manual.
8. A ConTEXt user must be able to design titles without worrying about vertical and horizontal spacing, referencing and synchronisation.

These prerequisites have resulted in a heavy duty mechanism that works in the background while running ConTEXt. The commands that are described in the following sections are an example of an implementation. We will also show examples of self designed titles.

## 11.2 Subdividing the text

A text is divided in chapters, sections, etc. with the commands:

```
\part [...,...] {...}
          1         2
       OPTIONAL
1  REFERENCE
2  CONTENT
```

```
\chapter [...,...] {...}
             1        2
          OPTIONAL
1  REFERENCE
2  CONTENT
```

```
\section [...,...] {...}
            1         2
          OPTIONAL
1   REFERENCE
2   CONTENT
```

```
\subsection [...,...] {...}
               1         2
             OPTIONAL
1   REFERENCE
2   CONTENT
```

```
\subsubsection [...,...] {...}
                  1         2
                OPTIONAL
1   REFERENCE
2   CONTENT
```

and

```
\title [...,...] {...}
          1         2
        OPTIONAL
1   REFERENCE
2   CONTENT
```

```
\subject [...,...] {...}
            1         2
          OPTIONAL
1   REFERENCE
2   CONTENT
```

```
\subsubject [...,...] {...}
               1         2
             OPTIONAL
1   REFERENCE
2   CONTENT
```

```
\subsubsubject [...,...] {...}
                  1         2
                OPTIONAL
1   REFERENCE
2   CONTENT
```

The first series of commands (`\chapter` ...) generate a numbered head, with the second series the titles are not numbered. There are a few more levels available than those shown above.

By default `\part` generates *no* title because most of the times these require special attention and a specific design. In the background however the partnumbering is active and carries out several initialisations. The other elements are set up to typeset a title.

| level | numbered title | unnumbered title |
|---|---|---|
| 1 | `\part` | |
| 2 | `\chapter` | `\title` |
| 3 | `\section` | `\subject` |
| 4 | `\subsection` | `\subsubject` |
| 5 | `\subsubsection` | `\subsubsubject` |

**Table 11.1** The structuring elements.

A structuring element has two arguments. The first argument, the reference, makes it possible to refer to the chapter or section from another location of the document. In **chapter 12** this mechanism is described in full. A reference is optional and can be left out.

```
\section{Subdividing a text}
```

ConTEXt generates automatically the numbers of chapters and sections. However there are situations where you want to enforce your own numbering. This is also supported.

```
\setuphead[subsection][ownnumber=yes]
\subsection{399}{The old number}
\subsection[someref]{400}{Another number}
```

In this example an additional argument appears. In the background ConTEXt still uses its own numbering mechanism, so operations that depend upon a consistent numbering still work okay. The extra argument is just used for typesetting the number. This user–provided number does not have to be number, it may be anything, like ABC-123.

### 11.2.399 The old number

### 11.2.400 Another number

You can automatically place titles of chapters, sections or other structuring elements in the header and footer with the marking mechanism. Titles that are too long can be shortened by:

```
\nomarking {.*.}

*   CONTENT
```

For example:

```
\chapter{Influences \nomarking{in the 20th century:} an introduction}
```

The text enclosed by `\nomarking` is replaced by dots in the header or footer. Perhaps an easier strategy is to use the automatic marking limiting mechanism. The next command puts the chapter title left and the section title right in the header. Both titles are limited in length.

```
\setupheadertexts[chapter][section]
\setupheader[leftwidth=.4\hsize,rightwidth=.5\hsize]
```

A comparable problem may occur in the table of contents. In that case we use `\nolist`:

```
\chapter{Influences in the 20th century\nolist{: an introduction}}
```

When you type the command \\ in a title a new line will be generated at that location. When you type `\crlf` in a title you will enforce a new line only in the table of contents. For example:

```
\chapter{Influences in the 20th century:\crlf an introduction}
```

This will result in a two line title in the table of context, while the title is only one line in the text.

It is possible to define your own structuring elements. Your 'own' element is derived from an existing text element.

```
\definehead [.1.] [.2.]

1   IDENTIFIER
2   SECTION
```

An example of a definition is:

```
\definehead[category][subsubject]
```

From this moment on the command `\category` behaves just like `\subsubject`, i.e., `\category` *inherits* the default properties of `\subsubject`. For example, `\category` is not numbered.

A number of characteristics available with `\setuphead` are described in **section 11.3**. Your own defined structuring elements can also be set up. The category defined above can be set up as follows:

```
\setuphead[category][page=yes]
```

This setup causes each new instance of category to be placed at the top of a new page.

We can also block the sectionnumbering with `\setupheads[sectionnumber=no]`. Sectionnumbering will stop but ConTEXt will continue the numbering on the background. This is necessary to be able to perform local actions like the generating local tables of content.

In defining your own structuring elements there is always the danger that you use existing TEX or ConTEXt commands. It is of good practice to use capitals for your own definitions. For example:

```
\definehead[WorkInstruction][section]
```

## 11.3 Variations in titles

The numbering and layout of chapters, sections and subsections can be influenced by several commands. These commands are also used in the design of your own heads. We advise you to start the design process in one of the final stages of your document production process. You will find that correct header definitions in the setup area of your source file will lead to a very clean source without any layout commands in the text.

The following commands are at your disposal:

```
\setuphead [...,...] [..,..=..,..]
              1              2

1   SECTION

2   style              =  normal bold slanted boldslanted type cap small... COMMAND
    textstyle          =  normal bold slanted boldslanted type cap small... COMMAND
    numberstyle        =  normal bold slanted boldslanted type cap small... COMMAND
    color              =  IDENTIFIER
    textcolor          =  IDENTIFIER
    numbercolor        =  IDENTIFIER
    number             =  yes no
    ownnumber          =  yes no
    page               =  left right yes
    continue           =  yes no
    header             =  none empty high nomarking
    text               =  none empty high nomarking
    footer             =  none empty high nomarking
    before             =  COMMAND
    inbetween          =  COMMAND
    after              =  COMMAND
    alternative        =  normal inmargin middle TEXT
    hang               =  none broad fit line NUMBER
    command            =  \...#1#2
    numbercommand      =  \...#1
    textcommand        =  \...#1
    deepnumbercommand  =  \...#1
    deeptextcommand    =  \...#1
    prefix             =  + - TEXT
    placehead          =  yes no empty
    incrementnumber    =  yes no LIST FILE
    resetnumber        =  yes no
    file               =  IDENTIFIER
    expansion          =  yes no command
    margintext         =  yes no
    inherits from \setupheads
```

Later we will cover many of the parameters mentioned here. This command can be used to set up one or more heads, while the next can be used to set some common features.

```
\setupheads [..,..=..,..]
                  *

*   sectionnumber  =  yes NUMBER no
    alternative    =  normal margin middle TEXT paragraph
    separator      =  TEXT
    stopper        =  TEXT
    align          =  inner outer left right flushleft flushright middle center normal no yes
    aligntitle     =  yes float no
    tolerance      =  verystrict strict tolerant verytolerant stretch
    indentnext     =  yes no
    command        =  \...#1#2
    margin         =  DIMENSION
```

The number of a title can be set up with:

```
\setupheadnumber [.1.] [.2.]

1   SECTION

2   NUMBER +cd:number -cd:number
```

This command accepts absolute and relative numbers, so [12], [+2] and [+]. The relative method is preferred, like:

```
\setuphead[chapter][+1]
```

This command is only used when one writes macros that do tricky things with heads. A number can be recalled by:

```
\headnumber [.*.]
             OPTIONAL
*   SECTION
```

and/or:

```
\currentheadnumber
```

For example:

```
\currentheadnumber    = 0
\headnumber[chapter]  = 11
\headnumber[section]  = 11.3
```

When you want to use the titlenumber in calculations you must use the command `\currentheadnumber`. This number is calculated by and available after:

```
\determineheadnumber [.*.]

*   SECTION
```

When headers and footers use the chapter and section titles they are automatically adapted at a new page. The example below results in going to new right hand side page for each chapter.

```
\setuphead
  [chapter]
  [page=right,
   after={\blank[2*big]}]
```

In extensive documents you can choose to start sections on a new page. The title of the first section however should be placed directly below the chapter title. You can also prefer to start this first section on a new page. In that case you set `continue` at `no`. **Figure 11.1** shows the difference between these two alternatives.

```
\setuphead
  [section]
  [page=yes,continue=no,
   after=\blank]
```

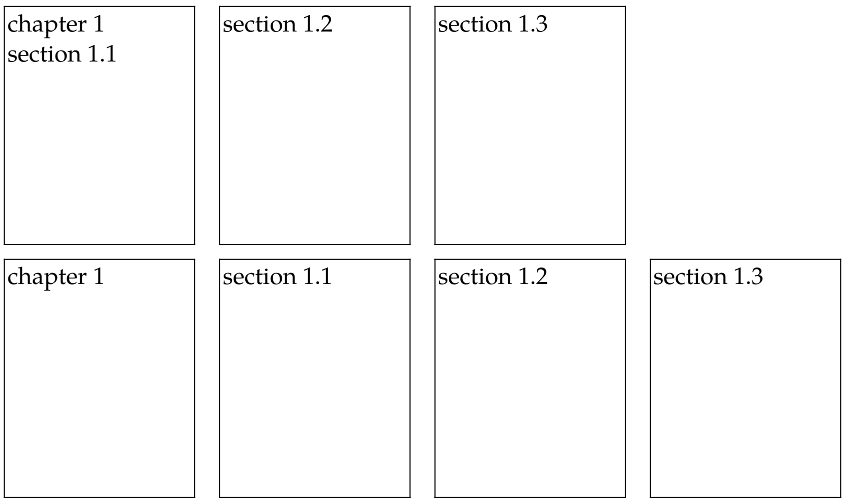


**Figure 11.1** Two alternatives for the first section.

It is also possible that you do not want any headers and footers on the page where a new chapter begins. In that case you should set `header` at `empty`, `high`, `nomarking` or an identification of a self defined header (this is explained in **section 4.16**).

By default the titles are typeset in a somewhat larger font. You can set the text and number style at your own chosen bodyfont. When the titles make use of the same body font (serif, sans, etc.) as the running text you should use neutral identifications for these fonts. So you use `\tfb` instead of `\rmb`. Font switching is also an issue in titles. For example if we use `\ssbf` instead of `\ss\bf` there is a chance that capitals and synonyms are not displayed the way they should. So you should always use the most robust definitions for fontswitching. Commands like `\kap` adapt their behaviour to these switchings.

A chapter title consists of a number and a text. It is possible to define your own command that typesets both components in a different way.

### 11.3.1 Title alternative equals normal

### 11.3.2 Title alternative equals inmargin

### Title alternative equals middle

These titles were generates by:

```
\setupheads[alternative=normal]
\subsection{Title alternative equals normal}
\setupheads[alternative=inmargin]
\subsection{Title alternative equals inmargin}
\setupheads[alternative=middle]
\subsubject{Title alternative equals middle}
```

In this manual we use a somewhat different title layout. The design of such a title is time consuming, not so much because the macros are complicated, but because cooking up something original takes time. In the examples below we will show the steps in the design process.

```
\unexpanded\def\HeadTitle#1#2%
```

```
  {\hbox to \hsize
     {\hfill % the % after {#1} suppresses a space
      \framed[height=1cm,width=2cm,align=left]{#1}%
      \framed[height=1cm,width=4cm,align=right]{#2}}}
```

```
\setuphead[subsection][command=\HeadTitle]
```

### 11.3.3 Title

A reader will expect the title of a section on the left hand side of the page, but we see an alternative here. The title is at the right hand side. One of the advantages of using `\framed` is, that turning `frame=on`, some insight can be gained in what is happening.

### 11.3.4 Another title

This alternative looks somewhat better. The first definition is slightly altered. This example also shows the features of the command `\framed`.

```
\unexpanded\def\HeadTitle#1#2%
  {\hbox to \hsize \bgroup
   \hfill
   \setupframed[height=1cm,offset=.5em,frame=off]
   \framed[width=2cm,align=left]{#1}%
   \framed[width=4cm,align=right,leftframe=on]{#2}%
   \egroup}
```

```
\setuphead
  [subsection]
  [command=\HeadTitle,
   style=\tfb]
```

We see that the font is set with the command `\setuphead`. These font commands should not be placed in the command `\HeadTitle`. You may wonder what happens when ConTEXt encounters a long title. Here is the answer.

### 11.3.5 A somewhat longer title

Since we have fixed the height at 1cm, the second line of the title end up below the frame. We will solve that problem in the next alternative. A `\tbox` provides a top aligned box.

```
\unexpanded\def\HeadTitle#1#2%
  {\hbox to \hsize \bgroup
   \hfill
   \setupframed[offset=.5em,frame=off]
   \tbox{\framed[width=3cm,align=left]{#1}}%
   \tbox{\framed[width=4cm,align=right,leftframe=on]{#2}}%
   \egroup}
```

```
\setuphead
  [subsection]
  [command=\HeadTitle]
```

This definition results in a title and a number that align on their first lines (due to `\tbox`).

11.3.6 | A considerably longer title

When the title design becomes more complex you have to know more of TEX. Not every design specification can be foreseen.

```
\setuphead[subsubject]    [alternative=text,style=bold]
\setuphead[subsubsubject] [alternative=text,style=slantedbold]
```

**Titles in the text** ***Why are titles in the text more difficult to program in TEX than we may expect beforehand.*** The answer lies in the fact that ConTEXt supports the generation of parallel documents. These are documents that have a printable paper version and an electronic screen version. These versions are coupled and thus hyperlinked by their titles. This means that when you click on a title you will jump to the same title in the other document. So we *couple* document versions:

```
\coupledocument
  [screenversion]
  [repman-e]
  [chapter,section,subsection,subsubsection,part,appendix]
  [The Reporting Manual]
\setuphead
  [chapter,section,subsection,subsubsection,part,appendix]
  [file=screenversion]
```

The first argument in `\coupledocument` identfies the screen document and the second argument specifies the file name of that document. The third argument specifies the coupling and the fourth is a description. After generating the documents you can jump from one version to another by just clicking the titles. This command only preloads references, the actual coupling is achieved by `\setuphead` command. Because titles in a text may take up several lines some heavy duty manipulation is necessary when typesetting such titles as we will see later.

## 11.4 Meta–structure

You can divide your document in functional components. The characteristics of the titles may depend in what component the title is used. By default we distinguish the next functional components:

- frontmatter
- bodypart
- appendices
- backmatter

Introductions and extroductions are enclosed by `\start ... \stop` constructs. In that case the titles will not be numbered like the chapters, but they are displayed in the table of contents. Within the component 'bodypart' there are no specific actions or layout manipulations, but in the 'appendices' the titles are numbered by letters (A, B, C, etc.).

```
\startfrontmatter
```

```
    \completecontent
    \chapter{Introduction}    <</Roman in content, no number>>
  \stopfrontmatter

  \startbodymatter
    \chapter{First}           <</Roman number 1, in content>>
      \section{Alfa}          <</Roman number 1.1, in content>>
      \section{Beta}          <</Roman number 1.2, in content>>
    \chapter{Second}          <</Roman number 2, in content>>
      \subject{Blabla}        <</Roman no number, not in content>>
  \stopbodymatter

  \startappendices
    \chapter{Index}           <</Roman letter A, in content>>
    \chapter{Abbreviations}   <</Roman letter B, in content>>
  \stopappendices

  \startbackmatter
    \chapter{Acknowlegdement} <</Roman no number, in content>>
    \title{Colofon}           <</Roman no number, not in content>>
  \stopbackmatter
```

When this code is processed, you will see that commands like `\title` and `\subject` never appear in the table of content and never get a number. Their behaviour is not influenced by the functional component they are used in. The behaviour of the other commands depend on the setup within such a component. Therefore it is possible to adapt the numbering in a functional component with one parameter setup.

## 11.5 Alternative mechanisms

Not every document can be structured in chapters and sections. There are documents with other numbering mechanisms and other ways to indicate levels in the text. The title mechanism supports these documents.

At the lowest level, the macros of ConTEXt do not work with chapters and sections but with sectionblocks. The chapter and section commands are predefined sectionblocks. In dutch this distinction is more clear, since there we have `\hoofdstuk` and `\paragraaf` as instances of 'secties'.

```
\definesectionblock [.1.] [..,.2.,..]
                               OPTIONAL
1  inherits from \setupsectionblock
2  inherits from \setupsectionblock
```

```
\setupsectionblock [.1.] [..,.=2.,..]

1   IDENTIFIER

2   number  =  yes no
    page    =  yes right
    before  =  COMMAND
    after   =  COMMAND
```

```
\definesection [.*.]

*   IDENTIFIER
```

```
\setupsection [.1.] [.2.] [..,.=3.,..]
                     OPTIONAL
1   IDENTIFIER

2   IDENTIFIER

3   conversion      =  numbers characters Characters romannumerals Romannumerals
    previousnumber  =  yes no
```

By default there are four sectionblocks:

```
\definesectionblock [bodypart]      [headnumber=yes]
\definesectionblock [appendices]    [headnumber=yes]
\definesectionblock [introductions] [headnumber=no]
\definesectionblock [extroductions] [headnumber=no]
```

We see that numbering is set with these commands. When numbering is off local tables of contents can not be generated. When numbers are generated but they do not have to be displayed you can use `\setupheads[sectionnumber=no]`.

By default every sectionblock starts at a new (right hand side) page. This prevents markings from being reset too early. A new page is enforced by `page`.

In ConTEXt there are seven levels in use but more levels can be made available.

```
\definesection [section-1]
\definesection [section-2]
.............. ..........
\definesection [section-7]
```

There are a number of titles predefined with the command `\definehead`. We show here some of the definitions:

```
\definehead [part]    [section=section-1]
\definehead [chapter] [section=section-2]
\definehead [section] [section=section-3]
```

The definition of a subsection differs somewhat from the others, since the subs inherit the characteristics of a section:

```
\definehead
```

```
  [subsection]
  [section=section-4,
   default=section]
```

The definitions of unnumbered titles and subjects are different because we don't want any numbering:

```
\definehead
  [title]
  [coupling=chapter,
   default=chapter,
   incrementnumber=no]
```

The unnumbered title is coupled to the numbered chapter. This means that in most situations the title is handled the same way as a chapter. You can think of the ways new pages are generated at each new unnumbered title or chapter. Characteristics like the style and color are also inherited.

There is more to consider. The predefined sectionblocks are used in appendices, because these have a different numbering system.

```
\setupsection
  [section-2]
  [appendixconversion=Character, % Watch the capital
   previousnumber=no]
\setuphead
  [part]
  [placehead=no]
\setuphead
  [chapter]
  [appendixlabel=appendix,
   bodypartlabel=chapter]
```

This means that within an `appendix` conversion from number to character takes place, but only at the level of section 2. Furthermore the titles that are related to `section-2` do not get a prefix in front of the number. The prefix consists of the separate numbers of the sectionblocks:

```
<section-1><separator><section-2><separator><section-3> <</rm etc.>>
```

By default section 2 (appendix) will be prefixed by the partnumber and a separator (.) and this is not desirable at this instance. At that level we block the prefix mechanism and we prevent that in lower levels (section 3 ...) the partnumber is included.

In the standard setup of ConTEXt we do not display the part title. You can undo this by saying:

```
\setuphead[part][placehead=yes]
```

Chapters and appendices can be labeled. This means that the titles are preceded with a word like *Chapter* or *Appendix*. This is done with `\setuplabeltext`, for example:

```
\setuplabeltext[appendix=Appendix~]
```

The look of the titles are defined by `\setuphead`. ConTEXt has set up the lower level section headings to inherit their settings from the higher level. The default setups for ConTEXt are therefore limited to:

```
\setuphead
  [part,chapter]
  [align=normal,
   continue=no,
   page=right,
   head=nomarking,
   style=\tfc,
   before={\blank[2*big]},
   after={\blank[2*big]}]
```

```
\setuphead
  [section]
  [align=normal,
   style=\tfa,
   before={\blank[2*big]},
   after=\blank]
```

With nomarking, we tell ConTEXt to ignore markings in running heads at the page where a chapter starts. We prefer \tfc, because this enables the title to adapt to the actual bodyfont. The {} around \blank are essential for we do not want any conflicts with [ ].

Earlier we saw that new structuring elements could be defined that inherit characteristics of existing elements. Most of the time this is sufficient:

```
\definehead[topic]    [section][style=bold,before=\blank]
\definehead[category][subject][style=bold,before=\blank]
```

One of the reasons that the mechanism is rather complex is the fact that we use the names of the sections as setups in other commands. The marking of category can be compared with that of subject, but that of subject can not be compared with that section. During the last few years it appeared that subject is used for all sorts of titles in the running text. We don't want to see these in headers and footers.

While setting the parameter criterium in lists and registers and the way of numbering, we can choose persection or persubject. For indicating the level we can use the parameter section as well as subject. So we can alter the names of sections in logical ones that relate to their purpose. For example:

```
\definehead [handbook]       [section=section-1]
\definehead [procedure]      [section=section-2]
\definehead [subprocedure]   [section=section-3]
\definehead [instruction]    [procedure]
```

After this we can set up the structuring elements (or inherit them) and generate lists of procedures and instructions. We will discuss this feature in detail in one of the later chapters.

# 12 References

## 12.1 Table of contents

The table of contents is very common in books and is used to refer to the text that lies ahead. Tables of content are generated automatically by:

```
\placecontent
```

The table of contents shows a list of chapters and sections but this depends also on the location where the table of contents is summoned. Just in front of a chapter we will obtain a complete table. But just after the chapter we will only obtain a list of relevant sections or subsections. The same mechanism also works with sections and subsections.

```
\chapter{Mammals}
\placecontent
\section{Horses}
```

A table of contents is an example of a combined list. Before discussing combined lists we go into single lists. A single list is defined with:

```
            1     2         3
\definelist [...] [...] [..,..=..,..]
                  OPTIONAL  OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   inherits from \setuplist
```

An example of such a definition is:

```
\definelist[firstlevel]
```

Such a list is recalled with:

```
\placelist[firstlevel]
```

Each list may have its own set up:

```
\setuplist[firstlevel][width=2em]
```

Lists can be set up simultaneously, for example:

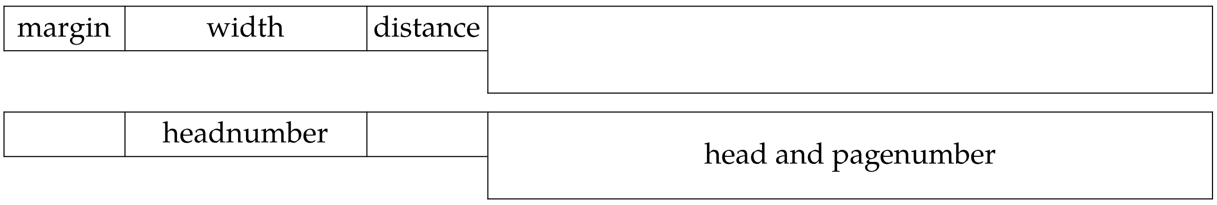
```
\setuplist[firstlevel,secondlevel][width=2em]
```

To generate a list you type:

```
               1            2
\placelist [...,...] [..,..=..,..]
                        OPTIONAL
1   IDENTIFIER
2   inherits from \setuplist
```

The layout of a list is determined by the values of `alternative` (see **table 12.1**), `margin`, `width` and `distance`. The alternatives `a`, `b` and `c` are line oriented. A line has the following construct:

```
               1          2
\setuplist [...,...] [..,..=..,..]

1   IDENTIFIER

2   state                =  start stop
    alternative          =  a b c ... none command
    coupling             =  on off
    criterium            =  SECTION local previous current all
    pageboundaries       =  LIST
    style                =  normal bold slanted boldslanted type cap small... COMMAND
    numberstyle          =  normal bold slanted boldslanted type cap small... COMMAND
    textstyle            =  normal bold slanted boldslanted type cap small... COMMAND
    pagestyle            =  normal bold slanted boldslanted type cap small... COMMAND
    color                =  IDENTIFIER
    command              =  \...#1#2#3
    numbercommand        =  \...#1
    textcommand          =  \...#1
    pagecommand          =  \...#1
    interaction          =  cd:sectionnumber TEXT pagenumber all
    before               =  COMMAND
    after                =  COMMAND
    inbetween            =  COMMAND
    left                 =  TEXT
    right                =  TEXT
    label                =  yes no
    prefix               =  yes no none
    pagenumber           =  yes no
    headnumber           =  yes no
    cd:sectionnumber     =  yes no
    aligntitle           =  yes no
    margin               =  DIMENSION
    width                =  DIMENSION fit
    height               =  DIMENSION fit broad
    depth                =  DIMENSION fit broad
    distance             =  DIMENSION
    separator            =  TEXT
    stopper              =  TEXT
    symbol               =  none 1 2 3 ...
    expansion            =  yes no command
    maxwidth             =  DIMENSION
    inherits from \setupframed
```



In a paper document it is sufficient to set up `width`. In an interactive document however the `width` determines the clickable area.[22]

In alternative `d` the titles in the table will be type set as a continuous paragraph. In that case the `before` and `after` have no meaning. The `distance`, that is 1em at a minimum, relates to the distance to the next element in the list. The next set up generates a compact table of contents:

[22] This also depends on the value assigned to `interaction`.

```
\setuplist
  [chapter]
  [before=\blank,after=\blank,style=bold]
\setuplist
  [section]
  [alternative=d,left=(,right=),pagestyle=slanted,prefix=no]
```

Since both lists are defined already when defining the sectioning command, we do not define them here. The parameter `prefix` indicates whether the preceding level indicator numbering is used. In this alternative the prefix is not used. Alternative `d` looks like this:

(12.1) Table of contents *215* (12.2) Synonyms *225* (12.3) Sorting *227* (12.4) Marking *228* (12.5) Cross references *231* (12.6) Predefined references *236* (12.7) Registers *237*

When `alternative` is set to `d`, an element in the list has the following construction:

| left | headnumber | right | head | page | distance |
|---|---|---|---|---|---|

When you define a title you also define a list. This means that there are standard lists for chapters, sections and subsections, etc. available.

These (sub)sections can be combined into one combined list. The default table of contents is such a combined list:

```
\definecombinedlist
  [content]
  [part,
   chapter,section,subsection,subsubsection,
   subsubsubsection,subsubsubsubsection]
  [level=subsubsubsubsection,
   criterium=local]
```

The alternative setups equals that of the separate lists.

```
\definecombinedlist [.1.] [...,2...] [..,..3..,..]
                                      OPTIONAL
1   IDENTIFIER
2   LIST
3   inherits from \setupcombinedlist
```

```
\setupcombinedlist [.1.] [..,..2..,..]
1   IDENTIFIER
2   level   =  1 2 3 4 SECTION current
    inherits from \setuplist
```

These commands themselves generate the commands:

The first command places a title at the top of the list. This title is unnumbered because we do not want the table of contents as an element in the list. In the next section we will discuss lists where the numbered title `\chapter` is used.

| alternative | display |
|---|---|
| a | number – title – pagenumber |
| b | number – title – spaces – pagenumber |
| c | number – title – dots – pagenumber |
| d | number – title – pagenumber (continuous) |
| e | title (framed) |
| f | title (left, middle or right aligned) |
| g | title (centered) |

**Table 12.1** Alternatives in combined lists.

Possible alternatives are summed up in **table 12.1**. There are a number of possible variations and we advise you to do some experimenting when you have specific wishes. The three parameters `width`, `margin` and `style` are specified for all levels òr for all five levels separately.

```
\setupcombinedlist
  [content]
  [alternative=c,
   aligntitle=no,
   width=2.5em]
```

The parameter `aligntitle` forces entries with no section number (like titles, subjects and alike) to be typeset onto the left margin. Otherwise the title is aligned to the numbered counterparts (like chapter, section and alike). Compare:

title
12 chapter

with:

title
12 chapter

You can also pass setup parameters to the `\place...` commands. For example:

```
\placecontent[level=part]
```

In this situation only the parts are used in the displayed list. Instead of an identifier, like part or chapter, you can also use a number. However this suggests that you have some insight in the level of the separate sections (part=1, chapter=2 etc.)

A table of contents may cross the page boundaries at an undesired location in the list. Pagebreaking in tables of content can hardly be automated. Therefore it is possible to adjust the pagebreaking manually. The next example illustrates this.

```
\completecontent[pageboundaries={2.2,8.5,12.3.3}]
```

This kind of 'fine–tuning' should be done at the end of the production proces. When the document is revised you have to evaluate the pagebreaking location. ConTEXt produces terminal feedback to remind you when these kind of commands are in effect.

Before a list can be generated the text should be processed twice. When a combined list is not placed after the text is processed twice you probably have asked for a local list.

There are two commands to write something directly to a list. The first command is used to add an element and the second to add a command:

```
\writetolist [.1.] {.2.} {.3.}

1   SECTION IDENTIFIER
2   CONTENT
3   CONTENT
```

```
\writebetweenlist [.1.] {.2.}

1   SECTION IDENTIFIER
2   CONTENT
```

We supply a simple example:

```
\writebetweenlist [section] {\blank}
\writetolist      [section] {---} {from here temporary}
\writebetweenlist [section] {\blank}
```

The next command is used in situations where information goes into the title but should not go into the list.

```
\nolist {.*.}

*   CONTENT
```

Consider for example the following example:

```
\definehead[function][ownnumber=yes]
\function{A-45}{manager logistics \nolist{(outdated)}}
\placelist[function][criterium=all]
```

When we call for a list of functions, we will get (. . . ) instead of (outdated). This can be handy for long titles. Keep in mind that each head has a corresponding list.

In an interactive document it is common practice to use more lists than in a paper document. The reason is that the tables of content is also a navigational tool. The user of the interactive document arrives faster at the desired location when many subtables are used, because clicking is the only way to get to that location.

In designing an interactive document you can consider the following setup (probably in a different arrangement):

```
\setuplayout[rightedge=3cm]
\setupinteraction[state=start,menu=on]
\setupinteractionmenu[right][state=start]
\startinteractionmenu[right]
```

```
  \placecontent
    [level=current, criterium=previous,
     alternative=f, align=right,
     interaction=all,
     before=, after=]
\stopinteractionmenu
```

These definitions make sure that a table of contents is typeset at every page (screen) in the right edge. The table displays the sections one level deeper than the actual level. So, for each section we get a list of subsections.

When you produce an interactive document with a table of contents at every level you can make a (standard) button that refers to `[previouscontent]`. This reference is generated automatically.

The list elements that are written to a list are not expanded (that is, commands remain commands). When expansion is needed you can set the parameter `expansion`. Expansion is needed in situations where you write variable data to the list. This is seldom the case.

In a more extensive document there may occur situations where at some levels there are no deeper levels available. Then the table of contents at that level is not available either. In that case you need more information on the list so you can act upon it. You can have access to:

| | |
|---|---|
| `\listlength` | the number of items |
| `\listwidth` | the maximum width of a list element |
| `\listheight` | the maximum height of a list element |

These values are determined by:

```
\determinelistcharacteristics [...,...] [..,..=..,..]
                                              OPTIONAL
1   IDENTIFIER
2   inherits from \setuplist
```

We end this section with an overview of the available alternatives. The first three alternatives are primarily meant for paper documents. The `criterium` parameter determines what lists are typeset, so in the next example, the sections belonging to the current chapter are typeset.

```
\placelist
  [section]
  [criterium=chapter,alternative=a]
```

12.1 Table of contents 215
12.2 Synonyms 225
12.3 Sorting 227
12.4 Marking 228
12.5 Cross references 231
12.6 Predefined references 236
12.7 Registers 237

```
\setuplabeltext[en][section={ugh }]
\placelist
```

```
  [section]
  [criterium=chapter,alternative=a,
   label=yes,width=2cm]
```

ugh 12.1 Table of contents 215
ugh 12.2 Synonyms 225
ugh 12.3 Sorting 227
ugh 12.4 Marking 228
ugh 12.5 Cross references 231
ugh 12.6 Predefined references 236
ugh 12.7 Registers 237

```
\placelist
  [section]
  [criterium=chapter,alternative=b]
```

```
12.1  Table of contents                                    215
12.2  Synonyms                                             225
12.3  Sorting                                              227
12.4  Marking                                              228
12.5  Cross references                                     231
12.6  Predefined references                                236
12.7  Registers                                            237
```

```
\placelist
  [section]
  [criterium=chapter,alternative=b,
   pagenumber=no,width=fit,distance=1em]
```

12.1 Table of contents
12.2 Synonyms
12.3 Sorting
12.4 Marking
12.5 Cross references
12.6 Predefined references
12.7 Registers

```
\placelist
  [section]
  [criterium=chapter,alternative=c,
   chapternumber=yes,margin=1.5cm]
```

```
      12.1  Table of contents . . . . . . . . . . . . . . . . . 215
      12.2  Synonyms  . . . . . . . . . . . . . . . . . . . . . 225
      12.3  Sorting . . . . . . . . . . . . . . . . . . . . . . 227
      12.4  Marking . . . . . . . . . . . . . . . . . . . . . . 228
      12.5  Cross references  . . . . . . . . . . . . . . . . . 231
      12.6  Predefined references . . . . . . . . . . . . . . . 236
      12.7  Registers . . . . . . . . . . . . . . . . . . . . . 237
```

```
\placelist % note the spaces on each side of the colon
  [section]
  [criterium=chapter,alternative=c,
   chapternumber=yes,separator={ : },width=fit]
```

12.1Table of contents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215
12.2Synonyms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 225
12.3Sorting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 227
12.4Marking . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
12.5Cross references . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 231
12.6Predefined references . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
12.7Registers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237

```
\placelist
  [section]
  [criterium=chapter,alternative=d]
```

12.1 Table of contents 215 12.2 Synonyms 225 12.3 Sorting 227 12.4 Marking 228
12.5 Cross references 231 12.6 Predefined references 236 12.7 Registers 237

```
\placelist
  [section]
  [criterium=chapter,alternative=d,
   distance=2cm]
```

12.1 Table of contents 215 12.2 Synonyms 225 12.3 Sorting 227
12.4 Marking 228 12.5 Cross references 231 12.6 Predefined
references 236 12.7 Registers 237

```
\placelist
  [section]
  [criterium=chapter,alternative=d,
   left={(},right={)}]
```

(12.1) Table of contents 215 (12.2) Synonyms 225 (12.3) Sorting 227 (12.4) Marking 228 (12.5) Cross references 231 (12.6) Predefined references 236 (12.7) Registers 237

```
\placelist
  [section]
  [criterium=chapter,alternative=e]
```

Table of contents

Synonyms

Sorting

Marking

Cross references

Predefined references

Registers

```
\placelist
```

```
  [section]
  [criterium=chapter,alternative=e,
   width=\textwidth,background=screen]
```

Table of contents

Synonyms

Sorting

Marking

Cross references

Predefined references

Registers

```
\placelist
  [section]
  [criterium=chapter,alternative=e,
   width=4cm]
```

Table of contents

Synonyms

Sorting

Marking

Cross references

Predefined references

Registers

```
\placelist
  [section]
  [criterium=chapter,alternative=f]
```

Table of contents
Synonyms
Sorting
Marking
Cross references
Predefined references
Registers

```
\placelist
  [section]
  [criterium=chapter,alternative=g]
```

Table of contents
Synonyms

Table of contents 12

Within a list entry, each element can be made interactive. In most cases, in screen documents, the option all is the most convenient one. Alternative e is rather well suited for screen documents and accepts nearly all parameters of `\framed`. In the next example we use a symbol instead of a sectionnumber. The parameter `depth` applies to this symbol.

```
\placelist
  [section]
  [criterium=chapter,alternative=a,
   pagenumber=no,distance=1em,
   symbol=3,height=1.75ex,depth=.25ex,numbercolor=gray]
```

When using color, don't forget to enable it. In the last example, All alternatives provide the means to hook in commands for the section number, text and pagenumber. Real complete freedom is provided by alternative `none`.

```
\placelist
  [section]
  [criterium=chapter,alternative=none,
   numbercommand=\framed,
   textcommand=\framed,pagecommand=\framed]
```

```
\unexpanded\def\ListCommand#1#2#3%
  {at page {\bf #3} we discuss {\bf #2}}
```

```
\placelist
  [section]
  [criterium=chapter,alternative=none,
   command=\ListCommand]
```

This alternative still provides much of the built–in functionality. Alternative `command` leaves nearly everything to the macro writer.

```
\unexpanded\def\ListCommand#1#2#3%
  {At p~#3 we discuss {\em #2}; }
```

```
\placelist
  [section]
  [criterium=chapter,alternative=command,
   command=\ListCommand]
```

At p 215 we discuss *Table of contents*; At p 225 we discuss *Synonyms*; At p 227 we discuss *Sorting*; At p 228 we discuss *Marking*; At p 231 we discuss *Cross references*; At p 236 we discuss *Predefined references*; At p 237 we discuss *Registers*;

As an alternative for `none`, we can use `horizontal` and `vertical`. Both commands have their spacing tuned for typesetting lists in for instance menus.

## 12.2 Synonyms

In many texts we use abbreviations. An abbreviation has a meaning. The abbreviation and its meaning have to be used and typeset consistently throughout the text. We do not like to see ABC and in the next line an ABC. For this reason it is possible to define a list with the used abbreviations and their meanings. This list can be recalled and placed at the beginning or end of a book for the convenience of the reader.

The use of abbreviations is an example of the synonym mechanism. A new category of synonyms is defined with the command:

```
\definesynonyms [.1.] [.2.] [.3.] [.4.]
                                  OPTIONAL
1   SINGULAR NAME
2   PLURAL NAME
3   COMMAND
4   COMMAND
```

The way the list is displayed can be influenced by:

```
\setupsynonyms [.1.] [..,.=.,..]
                          2
1   IDENTIFIER
2   textstyle     =  normal bold slanted boldslanted type cap small... COMMAND
    synonymstyle  =  normal bold slanted boldslanted type cap small... COMMAND
    location      =  left right top serried inmargin inleft inright
    width         =  DIMENSION
    state         =  start stop
    criterium     =  all used
    conversion    =  yes no
    expansion     =  yes no command
    command       =  \...#1#2#3
```

Abbreviations are defined with the command:

```
\definesynonyms[abbreviation][abbreviations][\infull]
```

We will explain the optional fourth argument later. After this definition a new command `\abbreviation` is available. An example of the use of abbreviations is:

```
\abbreviation {UN}  {United Nations}
\abbreviation {UK}  {United Kingdom}
\abbreviation {USA} {United States of America}
```

The meaning can be used in the text by:

```
\infull{abbreviation}
```

It is also possible to add commands in the abbreviation. In that case the command must be typed literally between the `[ ]`:

```
\abbreviation [TEX] {\TeX} {The \TeX\ Typesetting System}
```

Recalling such an abbreviation is done with `\TEX` and the meaning can be fetched with `\infull{TEX}`. In a running text we type `\TEX\` and in front of punctuation `\TEX`.

A synonym is only added to a list when it is used. When you want to display all defined synonyms (used and not used) you have to set the parameter `criterium` at `all`. By setting `state` at `stop` you will prevent list elements to be the added to the list even when they are used. This can be a temporary measure:

```
\setupsynonyms[abbreviation][state=stop]
\abbreviation {NIL} {Not In List}
\setupsynonyms[abbreviation][state=start]
```

Here we left out the optional first argument, in which case the abbreviation itself becomes the command (`\NIL`). So, in this case the next two definitions are equivalent:

```
\abbreviation [NIL] {NIL} {Not In List}
\abbreviation {NIL} {Not In List}
```

The formal definition of a synonym looks like this:

A list of synonyms is generated by:

The next command generates a list with a title (`\chapter`):

Here we see why we typed the plural form during the definition of the synonym. The plural is also used as the title of the list and the first character is capitalized. The title can be altered with `\setuphead` (see **section 11.3**).

Synonyms are only available after they are used. There are instances when the underlying mechanism cannot preload the definitions. When you run into such troubles, you can try to load the meaning of the synonyms with the command:

For instance, the meaning of abbreviations can be loaded with `\loadabbreviations`. In order to succeed, the text has to be processed at least once. Don't use this command if things run smoothly.

Next to the predefined abbreviations we also defined the si–units as synonyms. These must be loaded as a separate module. We will discuss this in **section 18.4**.

The attentive reader has seen that the command `\definesynonyms` has four arguments. The fourth argument is reserved for a command with which you can recall the synonym. In this way the synonyms are protected from the rest of the ConTEXt commands and there will be no conflicts using them.

```
\definesynonyms[Function][Functions][\FunctionName][\FunctionNumber]
```

We could define some functions like:

```
\Function [0001] {0001a} {Lithographer}
\Function [0002] {0002x} {Typesetter}
```

Than we can recall number and name by `\FunctionName` (Lithographer and Typesetter) and `\FunctionNumber` (0001a and 0002x), so:

```
The \FunctionName{0001} has functionnumber \FunctionNumber{0001}.
```

## 12.3 Sorting

Another instance of lists with synonyms is the sorted list. A sorted list is defined with:

```
\definesorting [.1.] [.2.] [.3.]
                            OPTIONAL
1   SINGULAR NAME
2   PLURAL NAME
3   COMMAND
```

The list is set up with:

```
\setupsorting [.1.] [..,.=2.,..]

1   IDENTIFIER
2   before     =  COMMAND
    after      =  COMMAND
    command    =  \...#1
    state      =  start stop
    criterium  =  all used
    style      =  normal bold slanted boldslanted type cap small... COMMAND
    expansion  =  yes no command
```

After the definition the next command is available. The `<<sort>>` indicates the name for the list you defined.

In accordance to lists there are two other commands available:

The title can be set up with `\setuphead`:

An example of sorting is:

```
\definesorting[city][cities]
\setupsorting[city][criterium=all]

\city {London}
\city {Berlin}
\city {New York}
\city {Paris}
\city {Hasselt}

\placelistofcities[] % temp, mkiv bugfix
```

The definition is typed in the setup area of your file or in an environment file. The cities can be typed anywhere in your text and the list can be recalled anywhere.

Berlin
Hasselt
London
New York
Paris

Another instance of the sorting command is that where we must type the literal text of the synonym in order to be able to sort the list. For example if you want a sorted list of commands you should use that instance. The predefined command `\logo` is an example of such a list.

```
\logo [TEX]   {\TeX}
\logo [TABLE] {\TaBlE}
```

When you use the alternative with the `[ ]` ConTEXt automatically defines a command that is available throughout your document. In the example above we have `\TABLE` and `\TEX` for recalling the logo. For punctuation we use `\TABLE`.

We advise you to use capital letters to prevent interference with existing ConTEXt and/or TEX commands.

Like in synonyms, a sorted list is only available after an entry is used. When sorting leads to any problems you can load the list yourself:

When we add a command in the third argument during the definition of the sorted list we may recall sorted list with this command. In this way the sorted lists can not interfere with existing commands (see **section 12.2**).

## 12.4 Marking

There is a feature to add 'invisible' marks to your text that can be used at a later stage. Marks can be used to place chapter or section titles in page headers or footers.

A mark is defined with:

```
\definemarking [.1.] [.2.]
                     OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
```

The second optional argument will be discussed at the end of this section. After the definition texts can be marked by:

```
\marking [.1.] {.2.}
1   IDENTIFIER
2   CONTENT
```

and recalled by:

```
\getmarking [.1.] [.2.]

1   IDENTIFIER

2   first last previous both all current
```

In analogy with the TEX–command `\mark`, we keep record of three other marks per mark (see **table 12.2**).

| **marks** | **location** |
|---|---|
| `previous` | the last of the previous page |
| `first` | the first of the actual page |
| `last` | the last of the actual page |
| `both` | first — last |
| `all` | previous — first — last |

**Table 12.2** Recorded marks, completed with some combinations.

When you use a combination of marks (`both` and `all`) marks are separated by an —. This separator can be set up with:

```
\setupmarking [.1.] [.2.]

1   IDENTIFIER

2   state      =  start stop
    separator  =  COMMAND
    expansion  =  yes no
```

The use of marks can be blocked with the parameter `state`. The parameter `expansion` relates to the expansion mechanism. By default expansion is inactive. This means that a command is stored as a command. This suits most situations and is memory effective. When you use altering commands in the mark you should activate the expansion mechanism.

Marks are initialised by:

```
\resetmarking [.*.]

*   IDENTIFIER
```

At the beginning of a chapter the marks of sections, subsections, etc. are reset. If we do not reset those marks would be active upto the next section or subsection.

Assume that a word list is defined as follows (we enforce some pagebreaks on purpose):

```
\definemarking[words]

\marking[words]{first}first word ...
\marking[words]{second}second word ...
```

```
\page
\marking[words]{third}third word ...
\marking[words]{fourth}fourth word ...
\page
\marking[words]{fifth}fifth word ...
\page
```

The results are shown in table 12.3.

| page | previous | first | last |
|---|---|---|---|
| 1 | — | first | second |
| 2 | second | third | fourth |
| 3 | fourth | fifth | fifth |

**Table 12.3** The reordering of marks.

While generating the title of chapters and sections `first` is used. The content of the marks can be checked easily by placing the mark in a footer:

```
\setupfootertexts
  [{\getmarking[words][first]}]
  []
```

or all at once:

```
\setupfootertexts
  [{\getmarking[words][previous]} --
   {\getmarking[words][first]} --
   {\getmarking[words][last]}]
  []
```

A more convenient way of achieving this goal, is the following command. The next method also takes care of empty markings.

```
\setupfootertexts[{\getmarking[words][all]}][]
```

Commands like `\chapter` generate marks automatically. When the title is too long you can use the command `\nomarking` (see section 11.2) or pose limits to the length. In ConTEXt the standard method to place marks in footers is:

```
\setupfootertexts[chapter][sectionnumber]
```

In case you defined your own title with `\definehead`, the new title inherits the mark from the existing title. For example when we define `\category` as follows:

```
\definehead[category][subsection]
```

After this command it does not matter whether we recall the mark by `category` or `subsection`. In this way we can also set up the footer:

```
\setupfootertexts[chapter][category]
```

There are situations where you really want a separate mark mechanism `category`. We could define such a mark with:

```
\definemarking[category]
```

However, we do want to reset marks so we have to have some information on the level at which the mark is active. The complete series of commands would look something like this:

```
\definehead[category][subsection]
\definemarking[category]
\couplemarking[category][subsection]
```

Note that we do this only when we both use category and subsection! After these commands it is possible to say:

```
\setupfootertexts[subsection][category]
```

The command `\couplemarking` is formally defined as:

```
\couplemarking [.1.] [.2.]

1  IDENTIFIER
2  IDENTIFIER
```

Its counterpart is:

```
\decouplemarking [.*.]

*  IDENTIFIER
```

It is obvious that you can couple marks any way you want, but it does require some insight in the ways ConTEXt works.

## 12.5 Cross references

We can add reference points to our text for cross referencing. For example we can add reference points at chapter titles, section titles, figures and tables. These reference points are typed between `[ ]`. It is even allowed to type a list of reference points separated by a comma. We refer to these reference points with the commands:

```
\in {.1.} {.2.} [.3.]

1  CONTENT
2  CONTENT
3  REFERENCE
```

```
\at {.1.} {.2.} [.3.]

1  CONTENT
2  CONTENT
3  REFERENCE
```

```
\about {.¹.} [.².]

1   CONTENT
2   REFERENCE
```

A cross reference to a page, text (number) or both can be made with:

```
\pagereference [.*.]

*   REFERENCE
```

```
\textreference [.¹.] {.².}

1   REFERENCE
2   CONTENT
```

```
\reference [.¹.] {.².}

1   REFERENCE
2   CONTENT
```

The command `\in` provides the number of a chapter, section, figure, table, etc. The command `\at` produces a pagenumber and `\about` produces a complete title. In the first two calls, the second argument is optional, and when given, is put after the number or title.

In the example below we refer to sections and pages that possess reference points:

```
In section~\in[cross references], titled \about[cross references], we
describe how a cross reference can be defined. This section starts
at page~\at[cross references] and is part of chapter~\in[references].
```

This becomes:

In section **12.5**, titled “**Cross references**”, we describe how a cross reference can be defined. This section starts at page **231** and is part of chapter **12**.

Here is another variation of the same idea:

```
In \in{section}[cross references], titled \about[cross references], we
describe how a cross reference can be defined. This section starts
at \at{page}[cross references] and is part of \in{chapter}[references].
```

We prefer this way of typing the cross references, especially in interactive documents. The clickable area is in this case not limited to the number, but also includes the preceding word, which is more convenient, especially when the numbering is disabled. In the first example you would have obtained a symbol like◀ that is clickable. This symbol indicates the direction of the cross reference: forward▶ or backward◀.

The direction of a hyperlink can also be summoned by the command `\somewhere`. In this way we find chapters or other text elements **before** and discuss somewhere **later** the descriptions.

```
\somewhere {.1.} {.2.} [.3.]

1   CONTENT
2   CONTENT
3   REFERENCE
```

This command gets two texts. The paragraph will be typed like this:

```
The direction of a hyperlink can also be summoned by the command
\type {\somewhere}. In this way we find chapters or other text elements
\somewhere {before} {after} [text elements] and discuss somewhere
\somewhere {previous} {later} [descriptions] the descriptions.
```

The next command does not need any text but will generate it itself. The generated texts can be defined with `\setuplabeltext` (see **page 193**).

```
\atpage [.*.]

*   REFERENCE
```

At the locations where we make reference points we can also type a complete list of reference points in a comma delimited list:

```
\chapter[first,second,third]{First, second and third}
```

Now you can cross reference to this chapter with `\in[first]`, `\in[second]` or `\in[third]`. In a large document it is difficult to avoid the duplication of labels. Therefore it is advisable to bring some order to your reference point definitions. For example, in this manual we use: `[fig:first]`, `[int:first]`, `[tab:first]` etc. for figures, intermezzos and tables respectively.

ConTEXt can do this for you automatically. Using the command `\setupreferencing`, you can set for instance `prefix=alfa`, in which case all references will be preceded by the word `alfa`. A more memory efficient approach would be to let ConTEXt generate a prefix itself: `prefix=+`. Prefixing can be stopped with `prefix=-`.

In many cases, changing the prefix in many places in the document is not an example of clearness and beauty. For that reason, ConTEXt is able to set the prefix automatically for each section. When for instance you want a new prefix at the start of each new chapter, you can use the command `\setuphead` to set the parameter `prefix` to `+`. The chapter reference itself is not prefixed, so you can refer to them in a natural way. The references within that chapter are automatically prefixed, and thereby local. When a chapter reference is given, this one is used as prefix, otherwise a number is used. Say that we have defined:

```
\setuphead[chapter][prefix=+]

\chapter[texworld]{The world of \TeX}
```

In this chapter, we can safely use references, without the danger of clashing with references in other chapters. If we have a figure:

```
\placefigure[here][fig:worldmap]{A map of the \TeX\ world}{...}
```

In the chapter itself we can refer to this figure with:

```
\in {figure} [fig:worldmap]
```

but from another chapter, we should use:

```
\in {figure} [texworld:fig:worldmap]
```

In general, when ConTEXt tries to resolve a reference in `\in`, `\at` etc., it first looks to see whether it is a local reference (with prefix). If such a reference is not available, ConTEXt will look for a global reference (without prefix). If you have some trouble understanding the mechanism during document production you can visualize the reference with the command `\version[temporary]`.

There are situations where you want to make a global reference in the middle of document. For example when you want to refer to a table of contents or a register. In that case you can type `-:` in the reference point label that *no* prefix is needed: you type `[-:content]`. Especially in interactive documents the prefix–mechanism is of use, since it enables you to have documents with thousands of references, with little danger for clashes. In the previous example, we would have got a global reference by saying:

```
\placefigure[here][-:fig:worldmap]{A map of the \TeX\ world}{...}
```

The generation of references can be started, stopped and influenced with the command:

```
                     *
\setupreferencing [..,..=..,..]

*   state             =  start stop
    cd:sectionnumber  =  yes no
    prefix            =  + - TEXT
    interaction       =  label TEXT all symbol
    width             =  DIMENSION
    left              =  COMMAND
    right             =  COMMAND
    convertfile       =  yes no small big
    separator         =  TEXT
    autofile          =  yes no page
    global            =  yes no
```

In this command the parameter `\<<section>>number` relates to the way the page numbers must be displayed. In interactive documents, we can refer to other documents. In that case, when the parameter `convertfile` is set to `yes`, external filenames are automatically converted to uppercase, which is sometimes needed for cdrom distributions. We will go into details later.

References from another document can be loaded with the command:

```
                    *
\usereferences [...,...]

*   FILE
```

With `left` and `right` you can define what is written around a reference generated by `\about`. Default these are quotes. The parameter `interaction` indicates whether you want references to be displayed like *section 1.2*, *section*, *1.2* or as a symbol, like ⧓.

What exactly is a cross reference? Earlier we saw that we can define a reference point by typing a logical label at the titles of chapters, sections, figures, etc. Then we can summon the numbers of chapters, sections, figures, etc. or even complete titles at another location in the document. For some internal purposes the real pagenumber is also available. In the background real pagenumbers play an important role in the reference mechanism.

In the examples below we discuss in detail how the reference point definitions and cross referencing works in ConTEXt.

```
\reference[my reference]{{Look}{at}{this}}
```

The separate elements can be recalled by `\ref`:

| | | |
|---|---|---|
| p | the typeset pagenumber | `\ref[p][my reference]` |
| t | the text reference | `\ref[t][my reference]` |
| r | the real pagenumber | `\ref[r][my reference]` |
| s | the subtext reference | `\ref[s][my reference]` |
| e | the extra text reference | `\ref[e][my reference]` |

In a paper document the reference is static: a number or a text. In an interactive document a reference may carry functionality like hyperlinks. In addition to the commands `\in` and `\at` that we discussed earlier we have the command `\goto`, which allows us to jump. This command does not generate a number or a text because this has no meaning in a paper version.

ConTEXt supports interactivity which is integrated into the reference mechanism. This integration saved us the trouble of programming a complete new set of interactivity commands and the user learns how to cope with these non–paper features in a natural way. In fact there is no fundamental difference in referring to chapter 3, the activation of a JavaScript, referring to another document or the submitting of a completed form.

A direct advantage of this integration is the fact that we are not bound to one reference, but we can define complete lists of references. This next reference is legal:

```
... see \in{section}[flywheel,StartVideo{flywheel 1}] ...
```

As expected this command generates a section number. And in an interactive document you can click on *section nr* and jump to the correct location. At the moment that location is reached a video titled *flywheel 1* is started. In order to reach this kind of comfortable referencing we cannot escape a fully integrated reference mechanism.

Assume that you want to make a cross reference for a general purpose. The name of the reference point is not known yet. In the next example we want to start a video from a general purpose menu:

```
\startinteractionmenu[right]
  \but [previouspage]  previous \\
  \but [nextpage]      next     \\
  \but [ShowAVideo]    video    \\
  \but [CloseDocument] stop     \\
\stopinteractionmenu
```

Now we can activate a video at any given moment by defining `ShowAVideo`:

```
\definerreference[ShowAVideo][StartVideo{a real nice video reel}]
```

This reference can be redefined or erased at any moment:

```
\definereference[ShowAVideo][]
```

```
\definereference [.1.] [...,2...]

1   IDENTIFIER
2   REFERENCE
```

```
\startlinenumbering
A special case of referencing is that of referring to linenumbers.
\startline [line:a] Different line numbering mechanism can be used
interchangeably. \startline [line:b] This leads to confusing input.
\stopline [line:a] \startline [line:c] Doesn't it? \stopline [line:c]
\stopline [line:b] A cross reference to a line can result in one line
number or a range of lines. \someline[line:d] {A cross reference is
specified by \type {\inline} where the word {\em line(s)} is
automatically added.} Here we have three cross references: \inline
[line:a], \inline [line:b], \inline[line:c] and \inline {as the last
reference} [line:d].
\stoplinenumbering
```

With `\startlines..\stoplines` you will obtain the range of lines in a cross reference and in case of `\someline` you will get the first line number. In this example we see that we can either let ConTEXt generate a label automatically, or privide our own text between braces.

1 A special case of referencing is that of referring to linenumbers. Different line numbering mech-
2 anism can be used interchangeably. This leads to confusing input. Doesn't it? A cross refer-
3 ence to a line can result in one line number or a range of lines. [redacted]
4 [line:d] A cross reference is specified by `\inline` where the word *line(s)* is automatically
5 added. Here we have three cross references: **lines 1–2**, **line 2**, **line 2** and as the last reference
6 ??.

```
\startlines ... \stoplines
```

```
\someline [.*.]

*   REFERENCE
```

```
\inline [.*.]

*   REFERENCE
```

## 12.6 Predefined references

One can imagine that it can be cumbersome and even dangerous for consistency when one has many references which the same label, like **figure** in `\in{figure}[somefig]`. For example, you may want to change each **figure** into `Figure` afterwards. The next command can both save time and force consistency:

```
\definereferenceformat [.1.] [..,.2.,..]

1   IDENTIFIER

2   left   =  TEXT
    right  =  TEXT
    text   =  TEXT
    label  =  IDENTIFIER
```

Given the following definitions:

```
\definereferenceformat [indemo]  [left=(,right=),text=demo]
\definereferenceformat [indemos] [left=(,right=),text=demos]
\definereferenceformat [anddemo] [left=(,right=),text=and]
```

we will have three new commands:

```
\indemo [demo:b]
\indemo {some text} [demo:b]
\indemos {some text} [demo:b] \indemo {and more text} [demo:c]
\indemos [demo:b] \anddemo [demo:c]
```

These will show up as:

**demo(BB)**
**some text(BB)**
**some text(BB) and more text(CC)**
**demos(BB) and(CC)**

Instead of using the `text` parameter, one can use `label` and recall a predefined label. The parameter `command` can be used to specify the command to use (`\in` by default).

## 12.7 Registers

A book without a register is not likely to be taken seriously. Therefore we can define and generate one or more registers in ConTEXt. The index entries are written to a separate file. The Perl script TEXutil converts this file into a format TEX can typeset.

A register is defined with the command:

```
\defineregister [.1.] [.2.]

1   SINGULAR NAME

2   PLURAL NAME
```

There are a number of commands to create register entries and to place registers. One register is available by default:

```
\defineregister[index][indices]
```

An entry is created by:

An entry has a maximum of three levels. The subentries are separated by a + or &. We illustrate this with an example.

```
\index{car}
\index{car+wheel}
\index{car+engine}
```

When index entries require special typesetting, for example `\sl` and `\kap` we have to take some measures, because these kind of commands are ignored during list generation and sorting. In those cases we can use the extended version. Between `[ ]` we type the literal ascii–string which will determine the alphabetical order.

For example we have defined logos or abbreviations like UN, UK and USA (see **section 12.2**), then an index entry must look like this:

```
\index[UN]{\UN}
\index[UK]{\UK}
\index[USA]{\USA}
```

If we do not do it this way UN, UK and USA will be placed under the `\`.

A cross reference within a register is created with:

This command has an extended version also with which we can input a 'pure' literal ascii string.

A register is generated and placed in your document with:

The next command results in register with title:

The register can be set up with the command `\setupregister`. When you use the command `\version[temporary]` during processing, the entries and their locations will appear in the margin (see section ??).

```
\setupregister [.1.] [.2.] [..,.=3.,..]
                      OPTIONAL
1   SINGULAR NAME

2   IDENTIFIER

3   n                   =  NUMBER
    balance             =  yes no
    align               =  inner outer left right flushleft flushright middle center normal no
                           yes
    style               =  normal bold slanted boldslanted type cap small... COMMAND
    pagestyle           =  normal bold slanted boldslanted type cap small... COMMAND
    textstyle           =  normal bold slanted boldslanted type cap small... COMMAND
    indicator           =  yes no
    coupling            =  yes no
    cd:sectionnumber    =  yes no
    criterium           =  SECTION local all
    distance            =  DIMENSION
    symbol              =  1 2 ... n a ... none
    interaction         =  pagenumber TEXT
    expansion           =  yes no command
    referencing         =  on off
    command             =  \...#1
    location            =  left middle right
    maxwidth            =  DIMENSION
    unknownreference    =  empty none
    alternative         =  a b A B
    prefix              =  both first none
    compress            =  no yes
    deeptextcommand     =  \...#1
```

By default a complete register is generated. However it is possible te generate partial registers. In that case the parameter `criterium` must be set. With `indicator` we indicate that we want a letter in the alphabetical ordering of the entries. When `referencing=on` is a pagereference is generated for every letter indicator, for example `index:a` or `index:w`. We can use these automatically generated references to refer to the page where for instance the a–entries start.

The commands we have mentioned thus far allow us to use a spacious layout in our source file. This means we can type the entries like this:

```
\chapter{Here we are}

\section{Where we are}
\index{here}
\index{where}

Wherever you are ...
```

Between `\chapter` and `\section` we should not type any text because the vertical spacing might be disturbed by the index entries. The empty line after the entry has no consequences. In case there are problems we always have the option to write index entries to the list by the more direct command:

There the `expansion` mechanism can be activated. Default expansion is inactive (see **page 229**).

In this reference manual there is a register with commands. This register is defined and initialised with:

```
\defineregister [macro] [macros]
\setupregister  [macro] [indicator=no]
```

And we can find entries like:

```
\macro{\tex{chapter}}
\macro{\tex{section}}
```

In case we want a register per chapter we can summon the accompanying register with the command below (the command `\tex` will place a \ in front of a word, but is ignored during sorting):[23]

```
\placeregister[macro]
  [criterium=chapter,n=2,before=,after=]
```

and we will obtain:

**TODO:** next example was borked

```
\start % dit moet, anders krijgen we dubbele letter-referenties
\setupregister[macro][referencing=off,align=]
\getbuffer
\stop  % register macro wordt immers ook aan het eind opgeroepen
```

A warning is due. The quality of the content of a register is completely in your hands. A bad selection of index entries leads to an inadequate register that is of no use to the reader.

Every entry shows one or more pagenumbers. With `symbol` we can define some alternatives. With `distance` the horizontal spacing between word and number or symbol is set.

| symbol | display |
|---|---|
| a | a b c d |
| n | 1 2 3 4 |
| 1 | ● ● ● ● |
| 2 | ▬ ▬ ▬ ▬ |

**Table 12.4** Alternatives for pagenumbers in registers.

Most of the time the layout of a register is rather simple. Some manuals may need some form of differentiating between entries. The definition of several registers may be a solution. However the layout can contribute to a better use of the register:

```
\index           {entry}
\index[key]      {entry}
\index[form::]   {entry}
\index[form::key]{entry}
\index           {form::entry}
\index[key]      {form::entry}
```

[23] Of course, `\placemacro` and `\completemacros` are also available.

```
\index[form::]   {form::entry}
\index[form::key]{form::entry}
```

The first two alternatives are known, but the rest is new and offers some control over the way the entry itself is typeset. The specification between `[ ]` relates to the pagenumber, the specification in front of the entry relates to the entry itself.

```
\setupregister[index][form][pagestyle=bold,textstyle=slanted]
```

Without any problems we can use different appearances for pagenumber and entry.

```
\setupregister[index][nb][pagestyle=bold]
\setupregister[index][hm][pagestyle=slanted]
```

With for example:

```
\index[nb::]{squareroot}
\index[hm::root]{$\srqt{2}$}
```

The index entries we have discussed so far indicate the one page where the entry is made, but we can also indicate complete ranges of pages using:

The entries in between, which are of the same order, are not placed in the register.

```
\startregister[endless]{endless}
...... an endless story ......
\stopregister[endless]
```

An extensive index entry, i.e. an entry with a large number of appearances, may have an uncomfortably long list of pagenumbers. Especially in interactive documents this leads to endless back and forth clicking. For this purpose we designed the feature of linked index entries. This means that you can couple identical entries into a list that enables the user to jump from entry to (identical) entry without returning to the register. The coupling mechanism is activated by:

```
\setupregister[index][coupling=yes]
```

In this way a mechanism is activated that places references in the register (◀◀▶) as well as in the text (◀ **word** ▶) depending on the availability of alternatives. A jump from the register will bring you to the first, the middle or the last appearance of the entry.

This mechanism is only working at the first level; subentries are ignored. Clicking on the word itself will bring you back to the register. Because we need the clickable word in the text we use the following command for the index entry itself:

For example `\coupledindex{where}`. The couplings must be loaded with the command:

```
\coupleregister [.*.]

*   IDENTIFIER
```

Normally this command is executed automatically when needed, so it's only needed in emergencies.

# 13 Descriptions

## 13.1 Introduction

In a document we can find text elements that bring structure to a document. We have already seen the numbered chapter and section titles, but there are more elements with a recognizable layout. We can think of numbered and non–numbered definitions, itemizations and citations. One of the advantages of TEX and therefore of ConTEXt is that coding these elements enables us to guarantee a consistent design in our document, which in turn allows us to concentrate on the content of our writing.

In this chapter we will discuss some of the elements that will bring structure to your text. We advise you to experiment with the commands and their setups. When applied correctly you will notice that layout commands in your text are seldom necessary.

## 13.2 Definitions

Definitions of concepts and/or ideas, that are to be typeset in a distinctive way, can be defined by `\definedescription`.

```
\definedescription [.1.] [..,.2=.,..]
                             OPTIONAL
1   IDENTIFIER
2   inherits from \setupdescriptions
```

The first argument of this command contains the name. After the definition a new command is available.

An example of the definition is:

```
\definedescription[definition][location=top,headstyle=bold]

\definition{icon}

An icon is a representation of an action or the name of a computer
program. Icons are frequently used in operating systems on several
computer platforms. \par
```

Several alternatives are displayed below:

| | |
|---|---|
| **icon** | An icon is a representation of an action or the name of a computer program. Icons are frequently used in operating systems on several computer platforms. |
| **icon** | Some users of those computer platforms are using these icons with an almost religious fanaticism. This brings the word icon almost back to its original meaning. |

**icon** An icon should be recognizable for every user but they are designed within a cultural and historical setting. In this fast and ever changing era the recognizability of icons is relative.

**icon** The 8–bit principle of computers was the reason that non–Latin scriptures were hardly supported by the operating systems. Not long ago this changed.

**icon** What for some languages looked like a handicap has now become a feature. Thousands of words and concepts are already layed down in characters. These characters therefore can be considered icons.

**icon** It is to be expected that people with expressive languages overtake us in computer usage because they are used to thinking in concepts.

**icon** The not–so–young generation remembers the trashcan in the earlier operating systems used to delete files. We in Holland were lucky that the text beneath it said: trashcan. A specific character for the trashcan would have been less sensitive misinterpretation, than the rather American–looking garbage receptacle unknown to many young people.

In the fifth example the definition is placed `serried` and defined as:

```
\definedescription
  [definition]
  [location=serried,headstyle=bold,width=broad,sample={icon}]

\definition{icon}

What for some languages looked like a handicap has now become a feature.
Thousands of words and concepts are already layed down in characters.
These characters therefore can be considered icons. \par
```

In the seventh example we have set `hang` at `broad`. This parameter makes only sense when we set the label at the right or left. When we set `width` at `fit` or `broad` instead of a number, the width of the sample is used. With `fit`, no space is added, with `broad`, a space of `distance` is inserted. When no sample is given the with of the defined word is used. The parameter `align` specifies in what way the text is aligned. When the definition is placed in the margin or typeset in a serried format, the parameter `margin` is of importance. When set to `standard` or `ja`, the marging follows the document setting. Alternatively you can pass a dimension.

Some characteristics of the description can be specified with:

```
\setupdescriptions [...,...] [..,..=..,..]
                        1            2
                    OPTIONAL
1   IDENTIFIER

2   style          =  normal bold slanted boldslanted type cap small... COMMAND
    color          =  IDENTIFIER
    width          =  fit broad DIMENSION
    distance       =  DIMENSION
    sample         =  TEXT
    text           =  TEXT
    closesymbol    =  TEXT
    closecommand   =  \...#1
    closesymbol    =  TEXT
    titleleft      =  TEXT
    titleright     =  TEXT
    titledistance  =  DIMENSION
    titlestyle     =  normal bold slanted boldslanted type cap small... COMMAND
    titlecolor     =  IDENTIFIER
    align          =  inner outer left right flushleft flushright middle center normal no yes
    margin         =  standard yes no DIMENSION
    location       =  left right top serried inmargin inleft inright hanging
    headstyle      =  normal bold slanted boldslanted type cap small... COMMAND
    headcolor      =  IDENTIFIER
    headcommand    =  COMMAND
    hang           =  fit broad NUMBER
    before         =  COMMAND
    inbetween      =  COMMAND
    after          =  COMMAND
    indentnext     =  yes no
    indenting      =  never none not no yes always first next small medium big normal odd
                      even DIMENSION
    command        =  COMMAND
```

The setup of a description can be changed with the command below. This has the same construct as `\definedescription`:

```
\setupdescriptions[<<name>>][<<setups>>]
```

When a description consists of more than one paragraph, use:

```
\startdefinition{icon}

An icon is a painting of Jesus Christ, Mother Mary or other holy figures.
These paintings may have a special meaning for some religious people.

For one reason or the other the description icon found its way to the
computer world where it leads its own life.

\stopdefinition
```

These commands will handle empty lines adequately.

## 13.3 Enumeration

Sometimes you will encounter text elements you would like to number, but they do not fit into the category of figures, tables, etc. Therefore ConTEXt has a numbering mechanism that we use for numbering text elements like questions, remarks, examples, etc. Such a text element is defined with:

```
\defineenumeration [...,...] [...] [..,..=..,..]
                     1         2          3
                             OPTIONAL   OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   inherits from \setupenumerations
```

After such a definition, the following commands are available:

```
\<<name>>
\sub<<name>>
\subsub<<name>>
\subsubsub<<name>>
```

Where *name* stands for any chosen name.

The numbering can take place at four levels. Conversion is related to the last level. If you specify a text, then this will be a label that preceeds every generated number. A number can be set and reset with the command:

```
\set<<enumeration>>{value}
\reset<<enumeration>>
```

You can use the `start` parameter in the setup command to explictly state a startnumber. Keep in mind that the enumeration commands increase the number, so to start at 4, one must set the number at 3. Numbers and subnumbers and be explictly increased with the commands:

```
\next<<enumeration>>
\nextsub<<enumeration>>
\nextsubsub<<enumeration>>
```

The example below illustrates the use of `\enumeration`. After the shown commands the content of a remark can be typed after `\remark`.

```
\defineenumeration
  [remark]
  [location=top,
   text=Remark,
   between=\blank,
   before=\blank,
   after=\blank]
```

Some examples of remarks are:

**Remark 1**

After definition the ‘remark’ is available at four levels: `\remark`, `\subremark`, `\subsubremark` and `\subsubsubremark`.

**Remark 2**

This command looks much like the command `\definedescription`.

The characteristics of numbering are specified with `\setupenumerations`. Many parameters are like that of the descriptions because numbering is a special case of descriptions.

```
\setupenumerations[<<name>>][<<setups>>]
```

```
\setupenumerations [...,...] [..,..=..,..]
                       1 OPTIONAL     2
1   IDENTIFIER
2   inherits from \setupdescriptions
```

The characteristics of sub and subsub enumerations can be set too. For example:

```
\setupenumerations[example][headstyle=bold]
\setupenumerations[subexample][headstyle=slanted]
```

Just like the description command there is a `\start−\stop` construction for multi paragraph typesetting.

Sometimes the number is obsolete. For example when we number per chapter and we have only *one* example in a specific chapter. In that case you can indicate with a `[-]` that you want no number to be displayed.

**Remark**

Because this remark was recalled by `\remark[-]` there is *no* number. Just as with other commands, we can also pass a reference label between `[ ]`. Also, we can setup the enumeration to stop numbering by setting `number` to `no`.

The numbering command can be combined usefully with the feature to move textblocks. An example is given in **section 15.4**. In that example we also demonstrate how to couple one numbered text to another. These couplings only have a meaning in interactive documents where cross references (hyperlinks) can be useful.

The numbering of text elements can appear in different forms. In that case we can let one numbered text element inherit its characteristic from another. We illustrate this in an example.

```
\defineenumeration[first]

\first  The numbering \type {first} is unique. We see that one
argument is sufficient. By default label and number are placed at the left
hand side.

\defineenumeration[second][first][location=right]

\second  The \type {second} inherits its counters from \type {first},
but is placed at the right hand side. In case of three arguments the first
one is the copy and the second the original.

\defineenumeration[third][location=inright]
\defineenumeration[fourth][location=inright]

\third The numbered elements \type {third} and \type {fourth} are both
unique and are placed in right margin.

\fourth  Both are defined in one command but they do have own
```

```
counters that are in no way coupled.

\defineenumeration[fifth][first]
\defineenumeration[sixth][first]

\fifth The elements \type {fifth} and \type {sixth} inherit the properties
and counters of \type {first}.

\sixth  Note: inheriting of \type{second} is not allowed because \type
{second} is not an original! \par
```

It may seem very complex but the text below may shed some light on this issue:

**first 1**

The numbering `first` is unique. We see that one argument is sufficient. By default label and number are placed at the left hand side.

**second 2**

The `second` inherits its counters from `first`, but is placed at the right hand side. In case of three arguments the first one is the copy and the second the original.

**third 1**

The numbered elements `third` and `fourth` are both unique and are placed in right margin.

**fourth 1**

Both are defined in one command but they do have own counters that are in no way coupled.

**fifth 3**

The elements `fifth` and `sixth` inherit the properties and counters of `first`.

**sixth 4**

Note: inheriting of `second` is not allowed because `second` is not an original!

It is possible to couple a numbered text element to another. For example we may couple questions and answers. In an interactive document we can click on a question which will result in a jump to the answer. And vice versa. The counters must be synchronised. Be aware of the fact that the counters need some resetting now and then. For example at the beginning of each new chapter. This can be automated by setting the parameter `way` to `bychapter`.

```
\definedescription [question] [coupling=answer]
\definedescription [answer]   [coupling=question]
```

## 13.4 Indenting

Indented itemizations, like dialogues, can be typeset with the command defined by

```
\defineindenting [.1.] [..,.=2.,..]

1   IDENTIFIER

2   inherits from \setupindentations
```

After this command `\<<name>>`, `\sub<<name>>` and `\subsub<<name>>` are available.

The parameters can be set up with the command:

## 13.5 Numbered labels

There is another numbering mechanism that is used for numbering specific text labels that also enables you to refer to these labels. For example, when you want to refer in your text to a number of transparencies that you use in presentations the next command can be used:

```
\definelabel [.1.] [..,.=2.,..]

1   IDENTIFIER

2   text       =  TEXT
    location   =  inmargin intext
    way        =  bytext bycd:section
    blockway   =  yes no
    headstyle  =  normal bold slanted boldslanted type cap small... COMMAND
    headcolor  =  IDENTIFIER
    before     =  COMMAND
    after      =  COMMAND
```

Where the parameter `location` is set at `intext` and `inmargin`. After this definition the following commands are available:

```
\reset<<name>>
\increment<<name>>
\next<<name>>
\current<<name>>[reference]
```

The `[reference]` after `currentname` is optional. After

```
\definelabel[video][text=video,location=inmargin]
```

This defines **video 1**`\video`, that results in a numbered label *video* in the margin. The command `\currentvideo` would have resulted in the number 0. The label can also be recalled with:

In our case, saying `\video` results in the marginal note concerning a video. The values of `before` and `after` are executed around the label (which only makes sense for in–text labels.

## 13.6 Itemize

Items in an itemization are automatically preceded by symbols or by enumerated numbers or characters. The symbols and the enumeration can be set up (see **table 13.1**). The layout can also be influenced. Itemization has a maximum of four levels.

| setup | result |
|---|---|
| n | 1, 2, 3, 4 |
| a | a, b, c, d |
| A | A, B, C, D |
| KA | A, B, C, D |
| r | i, ii, iii, iv |
| R | I, II, III, IV |
| KR | I, II, III, IV |
| m | 1, 2, 3, 4 |
| g | $\alpha, \beta, \gamma$ |
| G | A, B, Γ |

| setup | result |
|---|---|
| 1 | dot (●) |
| 2 | dash (–) |
| 3 | star ($\star$) |
| 4 | triangle ($\triangleright$) |
| 5 | circle ($\circ$) |
| 6 | big circle ($\bigcirc$) |
| 7 | bigger circle (◯) |
| 8 | square ($\square$) |

**Table 13.1** Item separator identifications in itemizations.

The command to itemize is:

```
\startitemize[<<setups>>]
\item ........
\item ........
\stopitemize
```

So you can do things like this:

```
Which of these theses are true?

\startitemize[A]
\item The difference between a village and a city is the existence of
      a townhall.
\item The difference between a village and a city is the existence of
      a courthouse.
\stopitemize
```

This will lead to:

Which of these theses are true?

A. The difference between a village and a city is the existence of a townhall.

B. The difference between a village and a city is the existence of a courthouse.

The symbols used under `1` to `8` can be defined with the command `\definesymbol` (see section ??) and the conversion of the numbering with `\defineconversion` (see section ??). For example:

```
Do the following propositions hold some truth?

\definesymbol[1][$\diamond$]

\startitemize[1]
\item The city of Amsterdam is built on wooden poles.
\item The city of Rome was built in one day.
```

```
\stopitemize
```

results in:

Do the following propositions hold some truth?

◇ The city of Amsterdam is built on wooden poles.

◇ The city of Rome was built in one day.

The keys `n`, `a`, etc. are related to the conversions. This means that all conversions are accepted. Take for example:

α. a `g` for Greek characters
β. a `G` for Greek capitals

When the setup and the `[ ]` are left out then the default symbol is typeset.

The indentation and horizontal whitespace is set up locally or globally with:

These arguments may appear in different combinations, like:

```
What proposition is true?

\startitemize[a,packed][stopper=:]
\item 2000 is a leap-year
\item 2001 is a leap-year
\item 2002 is a leap-year
\item 2003 is a leap-year
\stopitemize
```

this will become:

What proposition is true?

a: 2000 is a leap-year
b: 2001 is a leap-year
c: 2002 is a leap-year
d: 2003 is a leap-year

Both argument are optional. The key `packed` is one of the most commonly used:

```
What proposition is true?

\startitemize[n,packed,inmargin]
\item[ok] 2000 is a leap-year
\item 2001 is a leap-year
\item 2002 is a leap-year
\item 2003 is a leap-year
\stopitemize
```

will result in:

What proposition is true?

1. 2000 is a leap-year
2. 2001 is a leap-year
3. 2002 is a leap-year
4. 2003 is a leap-year

It happens very often that an itemization is preceded by a sentence like "*. . . can be seen below:*". In that case we add the key `intro` and the introduction sentence will be 'connected' to the itemization. After this setup a pagebreak between sentence and itemization is discouraged.

```
\startitemize[n,packed,inmargin,intro]
```

The setup of the itemization commands are presented in **table 13.2**.

| **setup** | **result** |
|---|---|
| `standard` | default setup |
| `packed` | no white space between items |
| `joinedup` | no white space before and after itemization |
| `paragraph` | no white space before an itemization |
| `<<n>>*serried` | little horizontal white space after symbol |
| `<<n>>*broad` | extra horizontal white space after symbol |
| `inmargin` | item separator in margin |
| `atmargin` | item separator at the margin |
| `stopper` | punctuation after item separator |
| `intro` | no pagebreak |
| `columns` | two columns |

**Table 13.2** Setup of `\setupitemize`.

In the last example we saw a reference point behind the command `\item` for future cross referencing. In this case we could make a cross reference to **answer 1** with the command `\in[ok]`.

The enumeration may be continued by adding the key `continue`, for example:

```
\startitemize[continue]
\item 2005 is a leap-year
\stopitemize
```

This would result in a rather useless addition:

5. 2005 is a leap-year

Another example illustrates that `continue` even works at other levels of itemizations:

- **supported image formats in pdfTEX**
  a. png
  b. eps
  c. pdf
- **non supported image formats in pdfTEX**
  a. jpg
  b. gif
  c. tif

This was typed as (in this document we have set `headstyle=bold`):

```
\startitemize[1,packed]
\head  supported image formats in \PDFTEX \par
```

```
        \startitemize[a]
        \item png \item eps \item pdf
        \stopitemize
\head  non supported image formats in \PDFTEX \par
        \startitemize[continue]
        \item jpg \item gif \item tif
        \stopitemize
\stopitemize
```

When we use the key `columns` the items are typeset in two columns. The number of columns can be set by the keys `one`, `two` (default), `three` or `four`.

```
\startitemize[n,columns,four]
\item png \item tif \item jpg \item eps \item pdf
\item gif \item pic \item bmp \item bsd \item jpe
\stopitemize
```

We can see that we can type the items at our own preference.

| | | | |
|---|---|---|---|
| 1. png | 4. eps | 7. pic | 10. jpe |
| 2. tif | 5. pdf | 8. bmp | |
| 3. jpg | 6. gif | 9. bsd | |

In such a long enumerated list the horizontal space between itemseparator and text may be too small. In that case we use the key `broad`, here `2*broad`:

| | | | |
|---|---|---|---|
| I. png | IV. eps | VII. pic | X. jpe |
| II. tif | V. pdf | VIII. bmp | |
| III. jpg | VI. gif | IX. bsd | |

The counterpart of `broad` is `serried`. We can also add a factor. Here we used `2*serried`.

- What format is this?

We can abuse the key `broad` for very simple tables. It takes some guessing to reach the right spacing.

This results in a rather strange example:

```
\startitemize[4*broad,packed]
\sym {yes} this is a nice format
\sym {no}  this is very ugly
\stopitemize
```

| | |
|---|---|
| yes | this is a nice format |
| no | this is very ugly |

The parameter `stopper` expects a character of your own choice. By default it is set at a period. When no level is specified and the `[ ]` are empty the actual level is activated. In section ?? we will discuss this in more detail. Stoppers only apply to ordered (numbered) list.

There are itemizations where a one line head is followed by a text block. In that case you use `\head` instead of `\item`. You can specify the layout of `\head` with the command `\setupitemize`. For example:

```
\setupitemize[each][headstyle=bold]
\startitemize[n]
\head A title head in an itemization

      After the command \type{\head} an empty line is mandatory. If you
      leave that out you will get a very long header.

\stopitemize
```

This becomes:

1. **A title head in an itemization**

   After the command `\head` an empty line is mandatory. If you leave that out you will get a very long header.

If we would have used `\item` the head would have been typeset in a normal font. Furthermore a pagebreak could have been introduced between head and textblock. This is not permitted when you use `\head`.

```
\head [...,*,...]
          OPTIONAL
*   REFERENCE
```

When you want to re-use the last number instead of increasing the next item you can use `\sub`. This feature is used in discussion documents where earlier versions should not be altered too much for reference purposes.

1. This itemization is preceded by `\startitemize[n,packed]`.
+ 1. This item is preceded by `\sub`, the other items by `\item`.
2. The itemization is ended by `\stopitemize`.

The most important commands are:

```
\item [...,*,...]
          OPTIONAL
*   REFERENCE
```

```
\sub [...,*,...]
         OPTIONAL
*   REFERENCE
```

In addition to `\item` there is `\sym`. This command enables us to type an indented text with our own symbol.

```
\sym {.*.}

*   CONTENT
```

Another alternative to `\item` is `\mar`. The specified argument is set in the margin (by default a typeletter) and enables us to comment on an item.

```
\mar [...,...] {...}
        1        2
      OPTIONAL
1   REFERENCE
2   CONTENT
```

Some at first sight rather strange alternatives are:

```
\its [...,...]
         *
      OPTIONAL
*   REFERENCE
```

```
\ran {...}
       *
*   CONTENT
```

These acronyms are placeholders for `items` and `range`. We illustrate most of these commands with an example that stems from a ntg questionnaire:

```
no          yes
◦   ◦   ◦   ◦   ◦     I can not do without TEX.
◦   ◦   ◦   ◦   ◦     I will use TEX forever.
◦   ◦   ◦   ◦   ◦     I expect an alternative to TEX in the next few years.
◦   ◦   ◦   ◦   ◦     I use TEX and other packages.
◦   ◦   ◦   ◦   ◦     I hardly use TEX.
◦   ◦   ◦   ◦   ◦     I am looking for another system.
```

The source is typed below. Look at the setup, it is local.

```
\startitemize[5,packed][width=8em,distance=2em,items=5]

\ran {no\hss yes}

\its I can not do without \TeX.
\its I will use \TeX\ forever.
\its I expect an alternative to \TeX\ in the next few years.
\its I use \TeX\ and other packages.
\its I hardly use \TeX.
\its I am looking for another system.

\stopitemize
```

For the interactive version there is:

```
\but [...]
       *
*   REFERENCE
```

This command resembles `\item` but produces an interactive symbol that executes the reference sequence specified.

The example below shows a combination of the mentioned commands. We also see the alternative `\nop`.

- **he got a head ache**
  1. of all the items
     he had to learn at school
  ++ 2. because the marginal explanation
  + 2. of the substantial content
  \# turned out to be mostly symbolic

This list was typed like this:

```
\startitemize
\head  he got a head ache

      \startitemize[n,packed]
      \item     of all the items
      \nop      he had to learn at school
      \mar{++}  because the marginal explanation
      \sub      of the substantial content
      \sym{\#}  turned out to be mostly symbolic
      \stopitemize
\stopitemize
```

With the no–operation command:

```
\nop
```

During the processing of itemizations the number of items is counted. This is the case with all versions. The next pass this information is used to determine the optimal location to start a new page. So do not despair when at the first parse your itemizations do not look the way you expected. When using TEXexec this is all taken care of.

We have two last pieces of advises. When items consist of two or more paragraphs always use `\head` instead of `\item`, especially when the first paragraph consists only one line. The command `\head` takes care of adequate pagebreaking between two paragraphs. Also, always use the key `[intro]` when a one line sentence preceeds the itemization. This can be automated by:

```
\setupitemize[each][autointro]
```

## 13.7 Items

A rarely used variant of producing lists is the command `\items`. It is used to produce simple, one level, vertical or horizontal lists. The command in its simplest form looks like this:

```
\items{<<alternative 1>>,<<alternative 2>>,...,<<alternative N>>}
```

Instead of an alternative you can also type `-`. In that case space is reserved but the item is not set. The layout of such a list is set with the command:

```
\setupitems [..,.*=.,..]

*   location   =  left right inmargin top bottom
    symbol     =  1 2 ... n a ... TEXT none
    width      =  DIMENSION
    n          =  NUMBER unknown
    before     =  COMMAND
    inbetween  =  COMMAND
    align      =  inner outer left right flushleft flushright middle center normal no yes
    after      =  COMMAND
```

The number (n) as well as the width are calculated automatically. When you want to do this yourself you can use the previous command or you pass the options directly. We show some examples.

```
\items[location=left]{png,eps,pdf}
```

◦ png
◦ eps
◦ pdf

```
\items[location=bottom]{png,eps,pdf}
```

◦ png
◦ eps
◦ pdf

```
\items[location=right,width=2cm]{png,eps,pdf}
```

◦ png
◦ eps
◦ pdf

```
\items[location=top,width=6cm,align=left]{png,eps,pdf}
```

◦png
◦eps
◦pdf

```
\items[location=inmargin]{png,eps,pdf}
```

◦ png
◦ eps
◦ pdf

```
\items[location=left,n=2,symbol=5]{jpg,tif}
```

◦ jpg
◦ tif

```
\items[symbol=3,n=6,width=\hsize,location=top]{png,eps,pdf,jpg,tif}
```

| | |
|---|---|
| ⋆ | |
| ⋆ | png |
| ⋆ | eps |
| ⋆ | pdf |
| ⋆ | jpg |
| ⋆ | tif |

The setup just after `\items` have the same effect as those of `\setupitems`:

```
\items [..,..=..,..] {.. ... ..}
              1              2
           OPTIONAL
1   inherits from \setupitems
2   CONTENT
```

## 13.8 Citations

The use of quotes depends on the language of a country: ‚Nederlands’, ‘English’, ‚Deutsch‘, «Français». The consistent use of single and double quotes is supported by a number of commands. A citation in the running text is typeset by:

```
\startquotation [...,...] ... \stopquotation
                    *
*   left middle right
```

This command can be compared with `\startnarrower` and has the same setup parameters. The quotes are placed around the text and they fall outside the textblock:

> “In commercial advertising ‘experts’ are quoted. Not too long ago I saw a commercial where a washing powder was recommended by the Dutch Society of Housewives. The remarkable thing was that there was a spokesman and not a spokeswoman. He was introduced as the “director”. It can’t be true that the director of the Society of Housewives is a man. Can it?”

In this example we see two other commands:

```
\startquotation
In commercial advertising \quote {experts} are quoted. Not too
long ago I saw a commercial where a washing powder was recommended
by the Dutch Society of Housewives. The remarkable thing was that
there was a spokesman and not a spokeswoman. He was introduced as
the \quotation {director}. It can't be true that the director of the
Society of Housewives is a man. Can it?
\stopquotation
```

The command `\quotation` produces double quotes and `\quote` single quotes.

```
\quote {...}
         *
*   CONTENT
```

```
\quotation {.*.}

*   CONTENT
```

These commands adapt to the language. In Dutch, English, German and French texts other quotes are activated. The body font is set with:

```
\setupquote [..,.=.,..]
                 *
*   before    =  COMMAND
    after     =  COMMAND
    style     =  normal bold slanted boldslanted type cap small... COMMAND
    color     =  IDENTIFIER
    location  =  TEXT margin
```

The location of a period, inside or outside a citation is somewhat arbitrary. The opinions on this issue differ considerately.

He said: “That is a bike” to which she replied: “Take a hike”.

The quotes are language dependent. Therefore it is of some importance that language switching is done correctly.

```
\quotation {He answered: \fr \quotation {Je ne parle pas fran\c cais}.}
\quotation {He answered: \quotation {\fr Je ne parle pas fran\c cais}.}
\quotation {\fr Il r\'epondait: \quotation{Je ne parle pas fran\c cais}.}
\fr \quotation {Il r\'epondait: \quotation{Je ne parle pas fran\c cais}.}
```

Watch the subtle difference.

“He answered: « Je ne parle pas français ».”
“He answered: “Je ne parle pas français”.”
“Il répondait: « Je ne parle pas français ».”
« Il répondait: « Je ne parle pas français ». »

When we want different quotes, we can change them. This is a language related setting.

```
\setuplanguage
  [en]
  [leftquote=\upperleftsinglesixquote,
   leftquotation=\upperleftdoublesixquote]
```

Fo rconsistency, such a setting can best be put into the local system file `cont-sys.tex`, together with other local settings. The following quotes are available:

| | | | |
|---|---|---|---|
| `\lowerleftsingleninequote` | ‚ | `\lowerrightsingleninequote` | ‚ |
| `\lowerleftdoubleninequote` | „ | `\lowerrightdoubleninequote` | „ |
| `\upperleftsingleninequote` | ’ | `\upperrightsingleninequote` | ’ |
| `\upperleftdoubleninequote` | ” | `\upperrightdoubleninequote` | ” |
| `\upperleftsinglesixquote` | ‘ | `\upperrightsinglesixquote` | ‘ |
| `\upperleftdoublesixquote` | “ | `\upperrightdoublesixquote` | “ |

# 14 Lines and frames

## 14.1 Introduction

TEX has an enormous capacity in handling text, but is very weak at handling graphical information. Lines can be handled adequately as long as you use vertical or horizontal lines. However, you can do graphical work with TEX by combining TEX and MetaPost.

In this chapter we introduce a number of commands that relate to drawing straight lines in your text. We will see a very sophisticated command `\framed` that can be used in many ways. The parameters of this command are also available in other commands.

## 14.2 Single lines

The simplest way to draw a line in ConTEXt is:

```
\hairline
```

For example:

```
\hairline
In what fairy tale is the wolf cut open and filled with stones? Was it in
{Little Red Riding-hood} or in \quote {The wolf and the seven goats}.
\hairline
```

This will become:

---

In what fairy tale is the wolf cut open and filled with stones? Was it in Little Red Riding-hood or in ‘The wolf and the seven goats’.

---

It does not look good at all. This is caused by the fact that a drawn line gets its own vertical whitespace. In **section 14.4** we will show how to alter this.

The effects of the command `\hairline` is best illustrated when we visualize `\strut`’s. We did so by saying `\showstruts` first.

A strut is a character with a maximum height and depth, but no width. The text in this example is surrounded by two strutted lines.

It is also possible to draw a line over the width of the actual paragraph:

```
\thinrule
```

Or more than one lines by:

```
\thinrules [..=..]
             OPTIONAL
*   inherits from \setupthinrules
```

For example:

```
\startitemize
\item question 1 \par \thinrules[n=2]
\item question 2 \par \thinrules[n=2]
\stopitemize
```

If you leave out a \par (or empty line), the thin rules come after the text. Compare

- question 1

- question 2

with

- question 1

- question 2

The last example was keyed in as:

```
\startitemize
\item question 1 \thinrules[n=2]
\item question 2 \thinrules[n=2]
\stopitemize
```

The parameters are set with:

```
\setupthinrules [.*=.]

*   interlinespace   =  small medium big
    n                =  NUMBER
    before           =  COMMAND
    inbetween        =  COMMAND
    after            =  COMMAND
    color            =  IDENTIFIER
    backgroundcolor  =  IDENTIFIER
    height           =  DIMENSION max
    depth            =  DIMENSION max
    alternative      =  a b c d
    rulethickness    =  DIMENSION
    color            =  IDENTIFIER
    background       =  color
    backgroundcolor  =  IDENTIFIER
```

You can draw thin vertical or horizontal lines with the commands:

```
\vl [.*.]

*   NUMBER
```

```
\hl [.*.]

*   NUMBER
```

The argument is optional. To `\vl` (|) you may pass a factor that relates to the actual height of a line and to `\hl` ( — ) a width that relates to the width of an em. So `\vl[2]` produces a rule with a height of two lines.

## 14.3 Fill in rules

On behalf of questionnaires there is the command:

```
\fillinline [..,.1=.,..] .2.
                 OPTIONAL
1   inherits from \setupfillinlines

2   NOTHING
```

With the accompanying setup command:

```
\setupfillinlines [..,.*=.,..]

*   width     =  DIMENSION
    margin    =  DIMENSION
    distance  =  DIMENSION
    before    =  COMMAND
    after     =  COMMAND
```

The example:

```
\fillinline[n=2,width=2cm]{name} \par
\fillinline[n=2,width=2cm]{address} \par
```

Leads to the next list:

name __________

address __________

An alternative is wanting the fill–in rule at the end of a paragraph. Then you use the commands:

```
\fillinrules [..,.=.,..] {...} {...}
                  1          2     3
              OPTIONAL           OPTIONAL
1   inherits from \setupfillinrules
2   CONTENT
3   CONTENT
```

```
\setupfillinrules [..,.=.,..]
                       *
*   width           =  fit broad DIMENSION
    distance        =  DIMENSION
    before          =  COMMAND
    after           =  COMMAND
    style           =  normal bold slanted boldslanted type cap small... COMMAND
    n               =  NUMBER
    interlinespace  =  small medium big
    separator       =  TEXT
```

The next example will show the implications:

```
\fillinline[width=3cm] Consumers in this shopping mall are frequently
confronted with questionnaires. Our hypothesis is that consumers rather
shop somewhere else than answer these kind of questionnaires. Do you
agree with this?
```

In this example we could of course have offered some alternatives for answering this question. By setting the width to `broad`, we get

Consumers in this shopping mall are frequently confronted with questionnaires. Our hypothesis is that consumers rather shop somewhere else than answer these kind of questionnaires. Do you agree with this? ______________

The next set of examples demonstrate how we can influence the layout.

```
\fillinrules[n=2,width=fit]{first}
\fillinrules[n=2,width=broad]{first}
\fillinrules[n=2,width=3cm]{first}
\fillinrules[n=2,width=fit,distance=.5em,separator=:]{first}
\fillinrules[n=2,width=broad,distance=.5em]{first}{last}
```

first

first

first

first:

first

last

## 14.4 Text lines

A text line is drawn just before and/or after a paragraph. The upper line may also contain text. The command is:

```
\textrule [.1.] {.2.}
           OPTIONAL OPTIONAL
1   top bottom
2   CONTENT
```

An example:

```
\textrule[top]{Instruments}
Some artists mention the instruments that they use during the production
of their \kap{CD}. In Peter Gabriel's \quote {Digging in the dust} he used
the {\em diembe}, {\em tama} and {\em surdu}. The information on another
song mentions the {\em doudouk}. Other \quote {unknown} instruments are
used on his \kap{cd} \quote {Passion}.
\textrule
```

This will result in:

**Instruments**

Some artists mention the instruments that they use during the production of their CD. In Peter Gabriel's 'Digging in the dust' he used the *diembe*, *tama* and *surdu*. The information on another song mentions the *doudouk*. Other 'unknown' instruments are used on his CD 'Passion'.

The behaviour of textlines is set up with the command below. With the parameter `width` you set the length of the line in front of the text.

```
\setuptextrules [..,..=..,..]
                      *
*   location   =  left inmargin
    before     =  COMMAND
    after      =  COMMAND
    inbetween  =  COMMAND
    width      =  DIMENSION
    distance   =  DIMENSION
    bodyfont   =  5pt ... 12pt small big
    color      =  IDENTIFIER
    style      =  normal bold slanted boldslanted type cap small... COMMAND
    rulecolor  =  IDENTIFIER
```

These is also a `\start`–`\stop` alternative. This one also honors the `bodyfont` parameter.

```
\starttextrule [.1.] {.2.} ... \stoptextrule
               OPTIONAL OPTIONAL
1   top bottom
2   CONTENT
```

## 14.5 Underline

Underlining text is not such an ideal method to banner your text. Nevertheless we introduced this feature in ConTEXt. Here is how it works. We use:

```
\underbar {.*.}

*   CONTENT
```

A disadvantage of this command is that words can no longer be hyphenated. This is a nasty side–effect. But we do support nested underlining.

The spaces in the last paragraph were also underlined. If we do not want that in this paragraph we use:

```
\underbars {.. .*. ..}

*   WORD
```

From the input we can see that the hyphen results from the compound word.

```
\underbar {A disadvantage of this command is that words can \underbar
{no} longer be hyphenated. This is a nasty side||effect. But we do
support \underbar {nested} underlining.}

\underbars {The spaces in the last paragraph were also underlined. If
we do not want that in this paragraph we use:}
```

The counterpart of these commands are:

```
\overbar {.*.}

*   CONTENT
```

```
\overbars {.. .*. ..}

*   WORD
```

You may wonder for what reasons we introduced these commands. The reasons are mainly financial:

| | |
|---|---|
| product 1 | 1.420 |
| product 2 | 3.182 |
| total | $\overline{4.602}$ |

This financial overview is made with:

```
\starttabulate[|l|r|]
\NC product 1 \NC          1.420  \NC \NR
\NC product 2 \NC          3.182  \NC \NR
\NC total     \NC \overbar{4.602} \NC \NR
\stoptabulate
```

The number of parameters in these commands is limited:

```
\setupunderbar [..,.*=.,..]

*   alternative    =  a b c
    rulethickness  =  DIMENSION
    bottomoffset   =  DIMENSION
    topoffset      =  DIMENSION
    rulecolor      =  IDENTIFIER
```

The alternatives are: alternative a, alternative b, alternative c while another line thickness results in: 1pt line, 2pt line.

A part of the text can be ~~striked~~ with the command:

```
\overstrike {.*.}

*   CONTENT
```

This command supports no nesting. Single ~~words~~ ~~are~~ ~~striked~~ with:

```
\overstrikes {.. .*. ..}

*   WORD
```

## 14.6 Framing

Texts can be framed with the command: `\framed`. In its most simple form the command looks like this:

```
\framed{A button in an interactive document is a framed text
with specific characteristics.}
```

The becomes:

A button in an interactive document is a framed text with specific characteristics.

The complete definition of this command is:

```
\framed [..,.=.,..] {...}
                1        2
         OPTIONAL
1   inherits from \setupframed
2   CONTENT
```

You may notice that all arguments are optional.

```
\framed
  [height=broad]
  {A framed text always needs special attention as far as the spacing
   is concerned.}
```

Here is the output of the previous source code:

A framed text always needs special attention as far as the spacing is concerned.

For the height, the values `fit` and `broad` have the same results. So:

```
\hbox
  {\framed[height=broad]{Is this the spacing we want?}
   \hskip1em
   \framed[height=fit]  {Or isn't it?}}
```

will give us:

Is this the spacing we want?  Or isn't it?

To obtain a comparable layout between framed and non–framed framing can be set on and off.

| | | |
|---|---|---|
| yes | no | yes |
| no | yes | no |

The rulethickness is set with the command `\setuprulethickness` (see section ??).

A framed text is typeset 'on top of' the baseline. When you want real alignment you can use the command `\inframed`.

```
to \framed{frame} or to be \inframed{framed}
```

or:

to frame or to be framed

It is possible to draw parts of the frame. In that case you have to specify the separate sides of the frame with `leftframe=on` and the alike.

We will now show some alternatives of the command `\framed`. Please notice the influence of `offset`. When no value is given, the offset is determined by the height and depth of the `\strut`, that virtual character with a maximum height and depth with no width. When exact positioning is needed within a frame you set `offset` at `none` (see also **tables 14.1, 14.2 and 14.3**). Setting the `offset` to `none` or `overlay`, will also disable the strut.

```
width=fit
width=broad
width=8cm,height=1.5em
offset=5pt
offset=0pt
offset=none
offset=overlay
width=8cm,height=1.5em,offset=0pt
width=8cm,height=1.5em,offset=none
```

The commands `\lbox` (ragged left), `\cbox` (ragged center) and `\rbox` (ragged right) can be combined with `\framed`:

| left of the middle | just in the middle | right of the middle |
|---|---|---|
| `\lbox` | `\cbox` | `\rbox` |

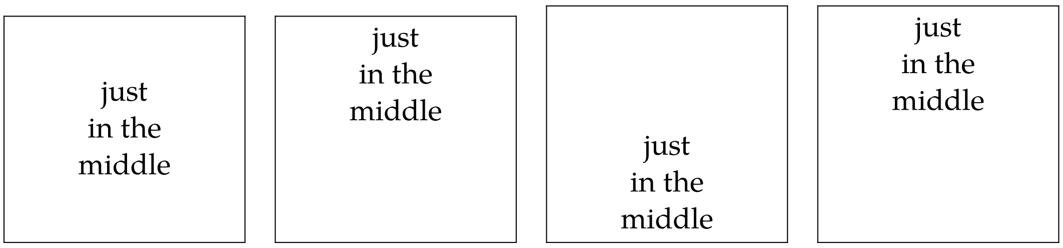
The second text is typed as follows:

```
\framed
  [width=.2\hsize,height=3cm]
  {\cbox to 2.5cm{\hsize2.5cm just\\in the\\middle}}
```

There is a more convenient way to align a text, since we have the parameters `align` and `top` and `bottom`. In the next one shows the influence of `top` and `bottom` (the second case is the default).

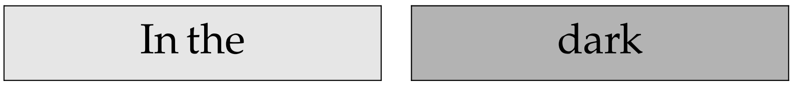
```
\setupframed[width=.2\hsize,height=3cm,align=middle]
\startcombination[4]
  {\framed[bottom=\vss,top=\vss]{just\\in the\\middle}}
  {\type{top=\vss}\crlf\type{bottom=\vss}}
  {\framed[bottom=\vss,top=]     {just\\in the\\middle}}
  {\type{top=}     \crlf\type{bottom=\vss}}
  {\framed[bottom=,top=\vss]     {just\\in the\\middle}}
```

```
    {\type{top=\vss}\crlf\type{top=}}
    {\framed[bottom=,top=]        {just\\in the\\middle}}
    {\type{top=}    \crlf\type{bottom=}}
  \stopcombination
```

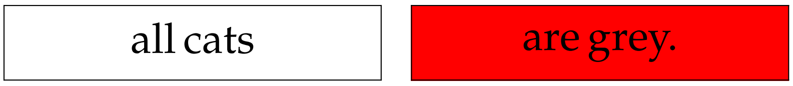


top=\vss bottom=\vss | top= bottom=\vss | top=\vss top= | top= bottom=

In the background of a framed text you can place a screen or a coloured background by setting `background` at `color` or `screen`. Don't forget to activate the the colour mechanism by saying (`\setupcolors[state=start]`).



background=screen | background=screen backgroundscreen=0.7



background=color | background=color backgroundcolor=red

There is also an option to enlarge a frame or the background by setting the `frameoffset` and/ or `backgroundoffset`. These do not influence the dimensions. Next to screens and colours you can also use your own kind of backgrounds. This mechanism is described in **section 9.3**.

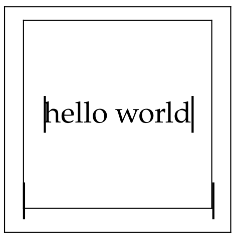
The command `\framed` itself can be an argument of `\framed`. We will obtain a framed frame.

```
\framed
  [width=3cm,height=3cm]
  {\framed[width=2.5cm,height=2.5cm]{hello world}}
```

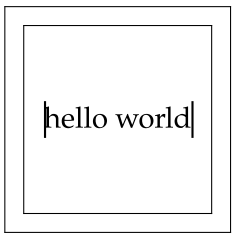
In that case the second frame is somewhat larger than expected. This is caused by the fact that the first framed has a strut. This strut is placed automatically to enable typesetting one framed text next to another. We suppress `\strut` with:

```
\framed
  [width=3cm,height=3cm,strut=no]
  {\framed[width=2.5cm,height=2.5cm]{hello world}}
```

When both examples are placed close to one another we see the difference:



`strut=yes`



`strut=no`

A `\hairline` is normally draw over the complete width of a text (`\hsize`). Within a frame the line is drawn from the left to the right of framed box.

Consequently the code:

```
\framed[width=8cm,align=middle]
  {when you read between the lines \hairline
   you may see what effort it takes \hairline
   to write a macropackage}
```

produces the following output:

| |
|:-:|
| when you read between the lines |
| you may see what effort it takes |
| to write a macropackage |

When no width is specified only the vertical lines are displayed.

| | | |
|---|---|---|
| their opinions | differ | considerately |

Which was obtained with:

```
\framed
  {their opinions \hairline differ \hairline considerately}
```

The default setup of `\framed` can be changed with the command:

```
\setupframed [.1.] [..,.2.,..]
              OPTIONAL
1   IDENTIFIER

2   height              =  fit broad DIMENSION
    width               =  fit broad fixed local DIMENSION
    autowidth           =  yes no force
    offset              =  none overlay default DIMENSION
    location            =  depth hanging high lohi low top middle bottom keep
    option              =  none empty
    strut               =  yes no global local
    align               =  inner outer left right flushleft flushright middle center normal no
                           yes
    bottom              =  COMMAND
    top                 =  COMMAND
    frame               =  on off none overlay
    topframe            =  on off
    bottomframe         =  on off
    leftframe           =  on off
    rightframe          =  on off
    frameoffset         =  DIMENSION
    framedepth          =  DIMENSION
    framecorner         =  round rectangular
    frameradius         =  DIMENSION
    framecolor          =  IDENTIFIER
    background          =  screen color none foreground IDENTIFIER
    backgroundscreen    =  NUMBER
    backgroundcolor     =  IDENTIFIER
    backgroundoffset    =  frame DIMENSION
    backgrounddepth     =  DIMENSION
    backgroundcorner    =  round rectangular
    backgroundradius    =  DIMENSION
    depth               =  DIMENSION
    corner              =  round rectangular
    radius              =  DIMENSION
    empty               =  yes no
    foregroundcolor     =  IDENTIFIER
    foregroundstyle     =  normal bold slanted boldslanted type cap small... COMMAND
    rulethickness       =  DIMENSION
```

The command `\framed` is used within many other commands. The combined use of `offset` and `strut` may be very confusing. It really pays off to spend some time playing with these macros and parameters, since you will meet `\framed` in many other commands. Also, the parameters `width` and `height` are very important for the framing texts. For that reason we summarize the consequences of their settings in **table 14.1**, **14.2 and 14.3**.

| | | offset | | | |
|---|---|---|---|---|---|
| | | .25ex | 0pt | none | overlay |
| strut | yes | ▯ | \| | . | |
| | no | ▫ | . | . | |

**Table 14.1** The influence of `strut` and `offset` in `\framed` (1).

| | | offset | | | |
|---|---|---|---|---|---|
| | | `.25ex` | `0pt` | `none` | `overlay` |
| `strut` | `yes` | TEX | TEX | TEX | TEX |
| | `no` | TEX | TEX | TEX | TEX |

**Table 14.2** The influence of `strut` and `offset` in `\framed` (2).

| | | width | |
|---|---|---|---|
| | | `fit` | `broad (\hsize=4cm)` |
| `height` | `fit` | xxxx | xxxx |
| | `broad` | xxxx | xxxx |

**Table 14.3** The influence of `height` and `width` in `\framed`.

happy
birthday
to you

At first sight it is not so obvious that `\framed` can determine the width of a paragraph by itself. When we set the parameter `align` the paragraph is first typeset and then framed. This feature valuable when typesetting titlepages. In the example left of this text, linebreaks are forced by `\\`, but this is not mandatory. This example was coded as follows:

```
\placefigure
  [left,none]
  {}
  {\framed[align=middle]{happy\\birthday\\to you}}
```

The parameter `offset` needs some special attention. By default it is set at `.25ex`, based on the cureently selected font. The next examples will illustrate this:

```
\hbox{\bf \framed{test} \sl \framed{test} \tfa \framed{test}}
\hbox{\framed{\bf test} \framed{\sl test} \framed{\tfa test}}
```

The value of `1ex` outside `\framed` determines the offset. This suits our purpose well.

**test** *test* test
**test** *test* test

The differences are very subtle. The distance between the framed boxes depends on the actual font size, the dimensions of the frame, the offset, and the strut.

TEX can only draw straight lines. Curves are drawn with small line pieces and effects the size of dvi–files considerately and will cause long processing times. Curves in ConTEXt are implemented by means of PostScript. There are two parameters that affect curves: `corner` and `radius`. When `corner` is set at `round`, round curves are drawn.

Don't be to edgy.

It is also possible to draw circles by setting `radius` at half the width or height. But do not use this command for drawing, it is meant for framing text. Use MetaPost instead.

Technically speaking the background, the frame and the text are separate components of a framed text. First the background is set, then the text and at the last instance the frame. The curved corner of a frame belongs to the frame and is not influenced by the text. As long as the radius is smaller than the offset no problems will occur.

## 14.7 Framed texts

When you feel the urge to put a frame around or a backgroud behind a paragraph there is the command:

```
\startFRAMEDTEXT [.1.] [..,.2=.,..] ... \stopFRAMEDTEXT
                 OPTIONAL  OPTIONAL
1   left right middle none
2   inherits from \setupframedtexts
```

An application may look like this:

```
\startframedtext[left]
From an experiment that was conducted by C. van Noort (1993) it was
shown that the use of intermezzos as an attention enhancer is not very
effective.
\stopframedtext
```

From an experiment that was conducted by C. van
Noort (1993) it was shown that the use of intermezzos
as an attention enhancer is not very effective.

This can be set up with:

```
\setupframedtexts [.1.] [..,.2=.,..]
                  OPTIONAL
1   IDENTIFIER
2   bodyfont          =  5pt ... 12pt small big
    style             =  normal bold slanted boldslanted type cap small... COMMAND
    left              =  COMMAND
    right             =  COMMAND
    before            =  COMMAND
    after             =  COMMAND
    inner             =  COMMAND
    linecorrection    =  on off
    depthcorrection   =  on off
    margin            =  standard yes no
    location          =  left right middle none
    indenting         =  never none not no yes always first next small medium big normal odd
                         even DIMENSION
    inherits from \setupframed
```

Framed texts can be combined with the place block mechanism, as can be seen in **intermezzo 14.1**.

```
\placeintermezzo
  [here][int:demo 1]
  {An example of an intermezzo.}
  \startframedtext
    For millions of years mankind lived just like animals. Then
    something happened, which unleashed the power of our imagination.
    We learned to talk.
    \blank
    \rightaligned{--- The Division Bell / Pink Floyd}
  \stopframedtext
```

In this case the location of the framed text (between `[ ]`) is left out.

For millions of years mankind lived just like animals.
Then something happened, which unleashed
the power of our imagination. We learned to talk.

— The Division Bell / Pink Floyd

**Intermezzo 14.1** An example of an intermezzo.

You can also draw a partial frame. The following setup produces **intermezzo 14.2**.

```
\setupframedtexts[frame=off,topframe=on,leftframe=on]
```

Why are the world leaders not moved by songs
like *Wozu sind Kriege da?* by Udo Lindenberg. I
was, and now I wonder why wars go on and on.

**Intermezzo 14.2** An example of an intermezzo.

You can also use a background. When the background is active it looks better to omit the frame.

An intermezzo like this will draw more attention,
but the readability is far from optimal. However,
you read can it. This inermezzo was set up with :

```
\setupframedtexts[frame=off,background=screen]
```

**Intermezzo 14.3** An example of an intermezzo with background.

**Intermezzo 14.4** demonstrate how to use some color:

```
\setupframedtexts
  [background=screen,
      frame=off,
      rightframe=on,
      framecolor=darkgreen,
      rulethickness=3pt]
\placeintermezzo
  [here][int:color]
  {An example of an intermezzo with a trick.}
  \startframedtext
    The trick is really very simple. But the fun is gone when Tom, Dick
    and Harry would use it too.
  \stopframedtext
```

The trick is really very simple. But the fun is gone
when Tom, Dick and Harry would use it too.

**Intermezzo 14.4** An example of an intermezzo with a trick.

So, in order to get a partial frame, we have to set the whole `frame` to `off`. This is an example of a situation where we can get a bit more readable source when we say:

```
\startbuffer
\startframedtext ... \stopframedtext
\stopbuffer
\placeintermezzo
  [here][int:color]
  {An example of an intermezzo with a trick.}{\getbuffer}
```

You do not want to set up a framed text every time you need it, so there is the following command:

```
\defineframedtext [.1.] [..,.2.,..]
                             OPTIONAL
1   IDENTIFIER
2   inherits from \setupframedtexts
```

The definition:

```
\defineframedtext
  [musicfragment]
  [frame=off, rightframe=on, leftframe=on]
\placeintermezzo
  [here][]
  {An example of a predefined framed text.}
\startmusicfragment
Imagine that there are fragments of music in your interactive document.
You will not be able to read undisturbed.
```

```
\stopmusicfragment
```

results in:

Imagine that there are fragments of music in your interactive document. You will not be able to read undisturbed.

**Intermezzo 14.5** An example of a predefined framed text.

## 14.8 Margin rules

To add some sort of flags to paragraphs you can draw vertical lines in the margin. This can be used to indicate that the paragraph was altered since the last version. The commands are:

```
\startmarginrule [.*.] ... \stopmarginrule

*   NUMBER
```

```
\marginrule [.1.] {.2.}

1   NUMBER

2   CONTENT
```

The first command is used around paragraphs, the second within a paragraph.

By specifying a level you can suppress a margin rule. This is done by setting the ‘global’ level higher than the ‘local’ level.

```
\setupmarginrules [.*.]

*   level          =  NUMBER
    rulethickness  =  DIMENSION
```

In the example below we show an application of the use of margin rules.

```
\startmarginrule
The sound of a duck is a good demonstration of how different
people listen to a sound. Everywhere in Europe the sound is
equal. But in every country it is described differently:
kwaak||kwaak (Netherlands), couin||couin (French),
gick||gack (German), rap||rap (Danish) and mech||mech
(Spanish). If you speak these words aloud you will notice
that
\startmarginrule[4] in spite of the \stopmarginrule
consonants the sound is really very well described. And what
```

```
about a cow, does it say boe, mboe or mmmmmm?
\stopmarginrule
```

Or:[24]

The sound of a duck is a good demonstration of how different people listen to a sound. Everywhere in Europe the sound is equal. But in every country it is described differently: kwaak–kwaak (Netherlands), couin–couin (French), gick–gack (German), rap–rap (Danish) and mech–mech (Spanish). If you speak these words aloud you will notice that in spite of the consonants the sound is really very well described. And what about a cow, does it say boe, mboe or mmmmmm?

If we would have set `\setupmarginrules[level=2]` we would have obtained a margin rule in the middle of the paragraph. In this example we also see that the thickness of the line is adapted to the level. You can undo this feature with `\setupmarginrules[thickness=1]`.

## 14.9 Black rules

Little black boxes —we call them black rules— (■) can be drawn by `\blackrule`:

```
\blackrule [..,..=..,..]
              OPTIONAL
*   inherits from \setupblackrules
```

When the setup is left out, the default setup is used.

```
\setupblackrules [..,..=..,..]

*   width        =  DIMENSION max
    height       =  DIMENSION max
    depth        =  DIMENSION max
    alternative  =  a b
    distance     =  DIMENSION
    n            =  NUMBER
    color        =  IDENTIFIER
```

The height, depth and width of a black rule are in accordance with the usual height, depth and width of TEX. When we use the key `max` instead of a real value the dimensions of TEX's `\strutbox` are used. When we set all three dimensions to `max` we get: █.

■ Black rules may have different purposes. You can use them as identifiers of sections or subsections. This paragraph is tagged by a black rule with default dimensions: `\inleft{\blackrule}`.

A series of black rules can be typeset by `\blackrules`:

```
\blackrules [..,..=..,..]

*   inherits from \setupblackrules
```

[24] G.C. Molewijk, Spellingsverandering van zin naar onzin (1992).

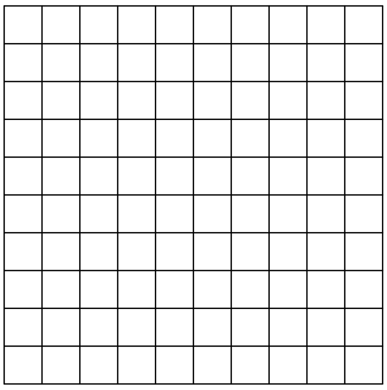
There are two versions. Version `a` sets `n` black rules next to each other with an equal specified width. Version `b` divides the specified width over the number of rules. This paragraph is tagged with `\inleft{\blackrules}`. The setup after `\blackrule` and `\blackrules` are optional.

## 14.10 Grids

We can make squared paper (a sort of grid) with the command:

```
                *
\grid [..,..=..,..]

*   x         =  NUMBER
    y         =  NUMBER
    nx        =  NUMBER
    ny        =  NUMBER
    dx        =  NUMBER
    dy        =  NUMBER
    xstep     =  NUMBER
    ystep     =  NUMBER
    offset    =  yes no
    factor    =  NUMBER
    scale     =  NUMBER
    unit      =  cm pt em mm ex es in
    location  =  left middle
```

The default setup produces:



It is used in the background when defining interactive areas in a figure. And for the sake of completeness it is described in this chapter.

# 15 Blocks

## 15.1 Introduction

A block in ConTEXt is defined as typographical unit that needs specific handling. We distinguish the following block types:

- **floats**

  Examples of floats are figures, tables, graphics, intermezzos etc. The locations of these blocks are determined by TEX and depends on the available space on a page.

- **textblocks**

  Examples of textblocks are questions and answers in a studybook, summaries, definitions or derivatives of formulas. The location of these kind of blocks in the final document cannot be determined beforehand. And the information may be used repeatedly in several settings.

- **opposite blocks**

  Opposite (or spread) blocks are typeset on the left–hand page when a single sided output is generated. The layout of the right–hand side page is influenced by the blocks on the left.

- **margin blocks**

  Margin blocks are more extensive than single margin words. Text and figures can be placed in the margin with this feature.

There are a number of commands that support the use of these block types. These are discussed in this chapter. Furthermore we will discuss other forms of text manipulation. Formulas can also be seen as blocks. Since formulas are covered in a separate chapter we don't go into details here.

This chapter is typeset with the option `\version [temporary]`. This does not refer to the content but to the typesetting. With this option, design information is placed in the margin.

## 15.2 Floats

Floats are composed of very specific commands. For example a table in ConTEXt is typeset using a shell around TABLE. Drawings and graphics are made with external packages, as TEX is only capable of reserving space for graphics.

Most floats are numbered and may have a caption. A float is defined with the command:

```
\definefloat [.1.] [.2.]

1  SINGULAR NAME
2  PLURAL NAME
```

In ConTEXt, figures, graphics, tables, and intermezzos are predefined with:

```
\definefloat [figure]     [figures]
```

```
\definefloat [table]      [tables]
\definefloat [graphic]    [graphics]
\definefloat [intermezzo] [intermezzos]
```

As a result of these definitions you can always use `\placefigure`, `\placetable`, `\placegraphic` and `\placeintermezzo`. Of course, you can define your own floats with `\definefloat`. You place your newly defined floats with the command:

When a float cannot be placed at a specific location on a page, ConTEXt will search for the most optimal alternative. ConTEXt provides a number of placement options for floats. These are listed in **table 15.1**.

| **preference** | **result** |
|---|---|
| `left` | left of text |
| `right` | right of text |
| `here` | preferably here |
| `top` | at top of page |
| `bottom` | at bottom of page |
| `inleft` | in left margin |
| `inright` | in right margin |
| `inmargin` | in the margin (left or right) |
| `margin` | in the margin (margin float) |
| `page` | on a new (empty) page |
| `opposite` | on the left page |
| `always` | precedence over stored floats |
| `force` | per se here |

**Table 15.1** Preferences for float placement.

The commands can be used without the left and right brackets. For example:

```
\place...{<<caption>>}{<<content>>}
```

When the caption is left out, the float number is generated anyway. When the number is not needed you type `none`, like in:

```
\placefigure[none]{}{.....}
```

It is mandatory to end this command by an empty line or a `\par`. You don't have to embed a table in braces, since the `\start` and `\stop` commands have them built in:

```
\placetable
  [here][tab:example]
  {A very simple example of a table.}
  \starttable[|c|c|]
  \HL
  \VL this \VL is    \VL\FR
  \VL a    \VL table \VL\LR
  \HL
  \stoptable
```

| | |
|---|---|
| this | is |
| a | table |

**Table 15.2** A very simple example of a table.

The vertical whitespace for a float can be reserved with:

This command can be used without the left and right bracket. An example of a reservation is:

```
\reservefigure
  [height=4cm,width=10cm,frame=on][here][fig:reservation]
  {An example of a reservation.}
```

Which results in **figure 15.1**.

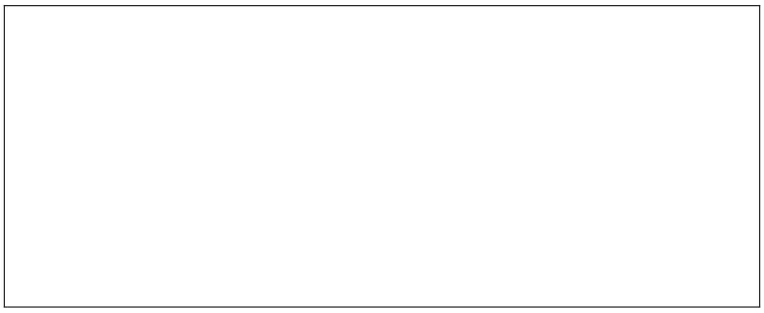


**Figure 15.1** An example of a reservation.

When the content of a float is not yet available, you can type `\empty...` instead of `\place....` In this way you can also reserve vertical whitespace. When no option is added, so `{}` is typed, the default empty float is used. However, whether the figure or table is available is not that important. You can always type:

```
\placefigure{This is a figure caption.}{}
```

As a first argument you can specify a key `left` or `right` that will cause ConTEXt to let the text flow around the float. The second optional parameter can be a cross reference, to be used later, like `\at{page} [fig:schematic process]`.

```
\placefigure[here][fig:demo]{This a figure caption.}{}
```

As we will later see, you can also use the next command:

Preferences are `left`, `right` or `middle`. Furthermore you can specify `offset` in case the text should align with the float. Both setups can be combined: `[left,offset]`.

A list of used floats is generated with the command:

For example, the command `\placelistoffigures` would typeset a list of figures. The list follows the numbering convention that is set with the command `\setupnumbering`, which was discussed at page ??.

The next command generates a list of floats on a separate page.

Pagebreaks that occur at unwanted locations can be enforced in the same way that is done with a table of contents (see **section 12.1**):

```
\completelistof<<floats>>[pageboundaries={8.2,20.4}]
```

As with tables of content the default local lists are generated. Recalling a list within a chapter produces a list for that specific chapter. So, if you want a list of all figures, you need to specify `criterium` as `all`.

The previous list was produced by saying:

```
\placelistoffigures[criterium=chapter]
```

The characteristics of a specific class of floats are specified with the command:

```
\setupfloat [.1.] [..,..2..,..]

1   IDENTIFIER

2   height               =  DIMENSION
    width                =  DIMENSION
    maxheight            =  DIMENSION
    maxwidth             =  DIMENSION
    minwidth             =  DIMENSION
    default              =  IDENTIFIER
    pageboundaries       =  LIST
    leftmargindistance   =  DIMENSION
    rightmargindistance  =  DIMENSION
    location             =  left middle right
    inherits from \setupframed
```

The (predefined) floats can also be set up with the more meaningful commands `\setupfigures`, `\setuptables` etc.

The height and width relate to the vertical whitespace that should be reserved for an empty float. All settings of `\framed` can be used, so when `frame` is set to `on`, we get a framed float.

The next two commands relate to *all* floats. The first command is used for setting the layout including the caption:

```
\setupfloats [..,..=..,..]
                  *

*   location            =  left right middle
    width               =  fit DIMENSION
    before              =  COMMAND
    after               =  COMMAND
    margin              =  DIMENSION
    spacebefore         =  small medium big none
    spaceafter          =  small medium big none
    sidespacebefore     =  small medium big none
    sidespaceafter      =  small medium big none
    indentnext          =  yes no
    ntop                =  NUMBER
    nbottom             =  NUMBER
    nlines              =  NUMBER
    default             =  IDENTIFIER
    tolerance           =  0 1 2
    leftmargindistance  =  DIMENSION
    rightmargindistance =  DIMENSION
    sidealign           =  normal line
    numbering           =  yes nocheck
    inherits from \setupframed
```

The second command is used for setting the enumerated captions of figures, tables, intermezzos, etc.

```
\setupcaptions [..,..=..,..]
                    *

*   location    =  top bottom none high low middle left middle right lefthanging
                   righthanging leftmargin rightmargin innermargin outermargin
    width       =  fit broad max DIMENSION
    minwidth    =  fit DIMENSION
    headstyle   =  normal bold slanted boldslanted type cap small... COMMAND
    style       =  normal bold slanted boldslanted type cap small... COMMAND
    number      =  yes no none
    inbetween   =  COMMAND
    align       =  inner outer left right flushleft flushright middle center normal no yes
    conversion  =  numbers characters Characters romannumerals Romannumerals
    way         =  bytext bycd:section
    separator   =  TEXT
    stopper     =  TEXT
    command     =  COMMAND
    distance    =  DIMENSION
```

You can also set up captions for a specific class of floats, like figures. The first argument of the next command is the name of that class of floats.

```
\setupcaption [...] [..,..=..,..]
                1          2

1   IDENTIFIER

2   inherits from \setupcaptions
```

The commands assigned to `before`, `after` are are executed before and after placing the float. The parameter `inbetween` is executed between the float and the caption. All three normally have a `\blank` command assigned.

The parameter `style` is used for numbering (**Figure x.y**) and `width` for the width of the caption label. The parameter `margin` specifies the margin space around a float when it is surrounded by text. The float macros optimize the width of the caption (at `top` or `bottom`) related to the width of the figure or table.

**Figure 15.2**

```
\setupcaptions[location=high]
\setupfloats[location=left]
```

With the three variables `ntop`, `nbottom` and `nlines` the float storage mechanism can be influenced. The first two variables specify the maximum number of floats that are saved per page at the top or the bottom of a page.

By default these variables have the values 2 and 0. Assume that ten figures, tables and/or other floats are stored, then by default two floats will be placed at each new page (if possible). For example, at a forced pagebreak or at the beginning of a new chapter, all stored floats are placed.

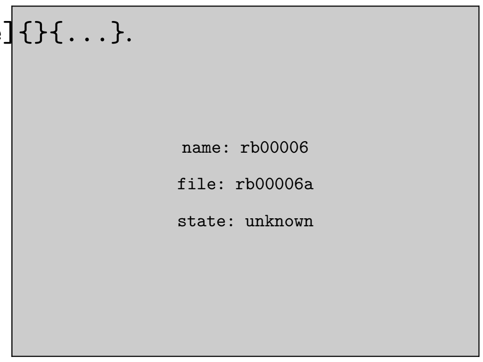
The parameter `nlines` has the default value 4. This means that never less than four lines will be typeset on the page where the floats are placed.

We continue with a few examples of floats (figures) placed next to the running text. This looks like:

```
\placefigure[right,none]{}{}

... here is where the text starts ....
```

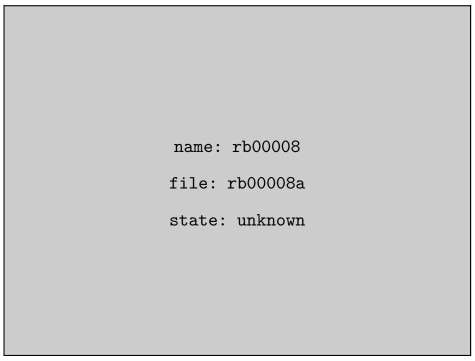
For illustrating the mechanism we do need some text. Therefore the examples are used to explain some issues on the float mechanism.

Floats are placed automatically. The order of appearance follows the order you have keyed in the source. This means that larger floats are placed somewhere else in your document. When `\version[temporary]` is set, you can get information on the float mechanism. By consulting that information you get some insight into the process.



Floats can be surrounded by text. The float at the right was set with `\placefigureright[right,none]{}{...}`. The float mechanism works automatically. Should it occur that pages are left blank as a result of poor float placement, you will need to make some adaptations manually. You can downsize your figure or table or alter your text. It is also a good practice to define your float some paragraphs up in your source. However, all of this should be done at the final production stage.

With the key `force` you can force a float to be placed at that exact location. Tables or figures that are preceded by text like: 'as we can see in the figure below' may be defined with this option.



In manuals and study books we encounter many illustrations. It is almost unavoidable to manually adapt these for optimal display. However, the float commands in ConTEXt are optimized in such a way that you can produce books with hundreds of floats effortlessly. The worst case is that some floats are stored and placed at the end of the chapter. But this can be influenced with the command `\startpostponing`. Postponing is done with the keys `always` which can be combined with the location, like `[left,always]` or `[here,always]`. Because the order of the floats is changed several parses are necessary for the document. These processes can be traced via messages on the terminal.

This brings us to a figure that is placed at the left side of a page. The side float mechanism in inspired and based on a mechanism of D. Comenetz. In the background three mechanisms are active. A mechanism to typeset a figure on top, inbetween, of under existing text. There is a mechanism to place figures on the right or left of a page. And there is a third mechanism to typeset text next to a figure.

We see an example of the last mechanism. The text is enclosed by the commands:

```
\startfiguretext
   [right]{}{\externalfigure[rb00015]}
....
\stopfiguretext
```

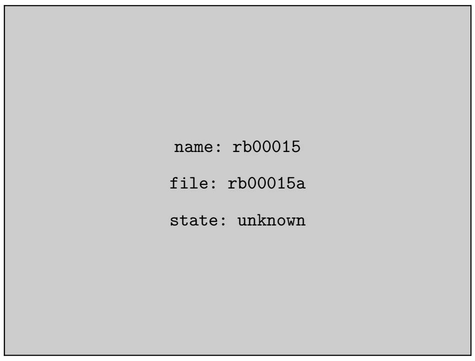


**Figure 15.3**

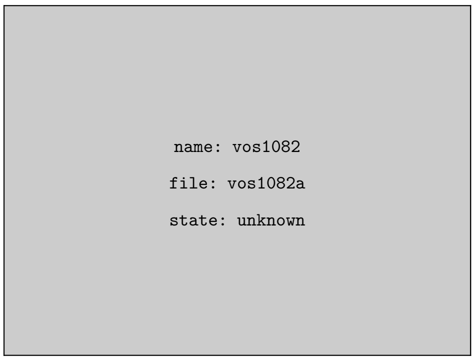
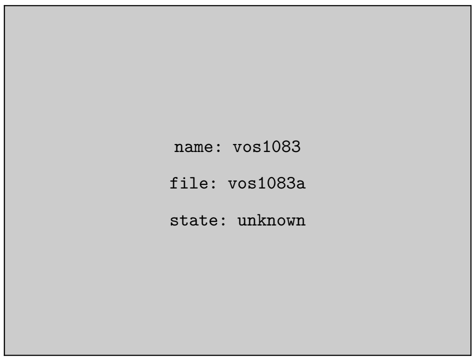


**Figure 15.4**

It is obvious that we can also place the figure at the left. With `\start...text` we can add the key `offset`. Here we used `[left,offset]`.
When the text is longer than expected, then it will *not* flow around the float. By default the floats are handled in the same order they are typed in the source file. This means that the stored figures are placed first. If this is not desired you can type the key `always`. The actual float will get priority.



There are more options. In this case the setup `[right,middle]` is given. In the same way we place text `high` and `low`.
When the key `long` is used the rest of the text is filled out with empty lines, as here.

**Figure 15.5**

When several figures are set under each other, making them the same width makes for a nice presentation on the page. This looks better.

## 15.3 Combining figures

For reasons of convenience we now discuss a command that enables us to combine floats into one.

```
\startcombination [.*.] ... \stopcombination

*   N*M
```

This command is used to place the figures under or next to each other.

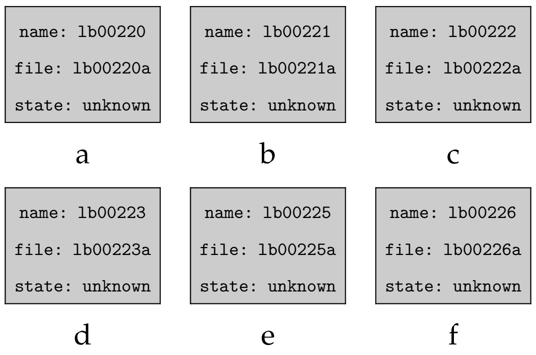


**Figure 15.6** An example of `\startcombination...`.

The example in **figure 15.6** is typeset with the commands:

```
\placefigure
  [here]
  [fig:combinations]
  {An example of \tex{startcombination...}.}
  {\startcombination[3*2]
     {\externalfigure[lb00220]} {a} {\externalfigure[lb00221]} {b}
     {\externalfigure[lb00222]} {c} {\externalfigure[lb00223]} {d}
```

```
    {\externalfigure[lb00225]} {e} {\externalfigure[lb00226]} {f}
  \stopcombination}
```

Between [ ] we specify how the combination is combined: [3*2], [4*2] etc. When we put two floats next to each other it is sufficient to specify [2], [4] etc.

The floats, mostly figures or tables, are specified within two arguments. The first content is placed over the second content: {xxx}{yyy}. The second argument can be empty: {xxx}{}. The general construct looks like this:

```
\startcombination[n*m]
  {text 1} {subcaption 1}
  {text 2} {subcaption 2}
  ........ ..............
\stopcombination
```

The combination can be set up with:

```
\setupcombinations [..,.*=.,..]

*   before     =  COMMAND
    inbetween  =  COMMAND
    after      =  COMMAND
    distance   =  DIMENSION
    height     =  DIMENSION fit
    width      =  DIMENSION fit
    location   =  top middle bottom left right
    align      =  inner outer left right flushleft flushright middle center normal no yes
    style      =  normal bold slanted boldslanted type cap small... COMMAND
    color      =  IDENTIFIER
```

With distance you specify the horizontal distance between objects. The parameters align relates to the subcaption. By default the text and objects are centered. The width is the total width of the combination.

The three parameters before, after and between are processed in the order of specification in figure 15.8. There are some examples in figure 15.7. We can see in figure 15.9 that when the title in the second argument is empty the spacing adapted.

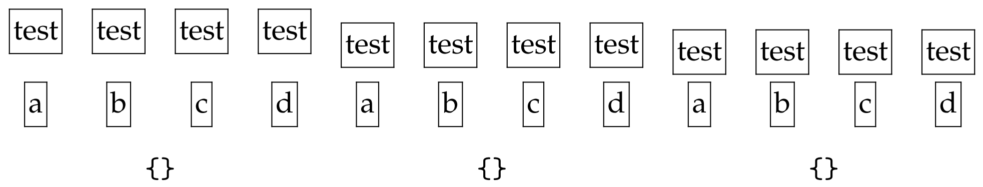


**Figure 15.7** The spacing within combinations (1).

Using combinations require figures that have the correct dimensions or equal proportions. Unequally proportioned figures are hard to combine.

The simple version of combining is this:

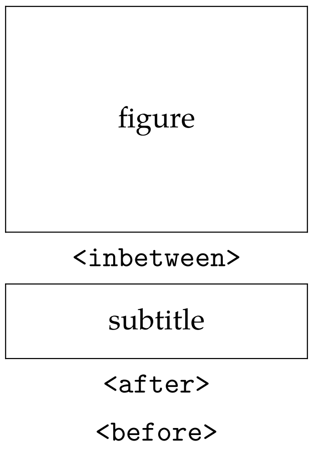


**Figure 15.8** The spacing within combinations (2).

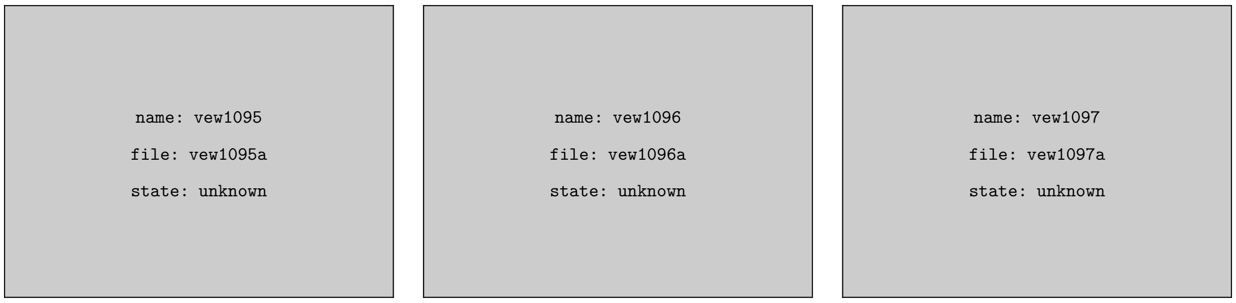


**Figure 15.9** Combinations without captions.

```
\placesidebyside {.1.} {.2.}

1   CONTENT
2   CONTENT
```

```
\placeontopofeachother {.1.} {.2.}

1   CONTENT
2   CONTENT
```

We use them in this way:

```
\placesidebyside       {\framed{\Logo[ADE]}} {\framed{\Logo[BUR]}}
\placeontopofeachother {\framed{\Logo[ADE]}} {\framed{\Logo[BUR]}}
```

## 15.4 Text blocks

For practical reasons we sometimes want to key text somewhere in the source that should be typeset at a completely different location in the typeset document. It is also useful to be able to use text more than once. The commands described below are among the eldest of ConTEXt. They were one of the reasons to start writing the macropackage.

You can mark text (a text block) and hide or move that block, but first you have to define it using:

```
\defineblock [.*.]

*   IDENTIFIER
```

If necessary you can pass several names in a comma–delimited list. After the definition you can mark text with:

```
\begin<<name>>
....................
....................
\end<<name>>
```

Between the `begin`– and `end` command you can use any command you want.

The commands below tell ConTEXt to hide or recall text blocks:

```
\hideblocks [...,1,...] [...,2,...]
                         OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
```

```
\useblocks [...,1,...] [...,2,...]
                        OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
```

```
\keepblocks [...,1,...] [...,2,...]
                         OPTIONAL
1   IDENTIFIER
2   all IDENTIFIER
```

```
\selectblocks [...,1,...] [...,2,...] [.3.]
                           OPTIONAL    OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   criterium  =  all SECTION
```

```
\processblocks [...,1...] [...,2...]
                                OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
```

These commands make it necessary to process your text at least twice. You can also recall more than one text block, for example `[question,answer]`.

In hidden and re–used blocks commands for numbering can be used. Assume that you use questions and answers in your document. By defining the questions as text blocks you can:

1. at that location typeset the questions
2. only use the questions and use the answers in a separate chapter
3. use questions and answers in a separate chapter
4. hide the answers
5. etc.

When we choose **option 2** the definitions look like this:

```
\defineenumeration[question][location=top,text=Question]
\defineenumeration[answer][location=top,text=Answer]

\defineblock[question,answer]

\hideblocks[answer]
```

A question and answer in the source look like this:

```
\beginquestion
\question Why do we use blocks? \par
\endquestion

\beginanswer
\answer I really don't know. \par
\endanswer
```

The questions are only used in the text. Questions and answers are both numbered. Answers are summoned by:

```
\chapter{Answers}

\reset[answer]
\useblocks[answer]
```

The command `\reset...` is necessary for resetting the numbering mechanism. When the answers are used in the same chapter you can use the following commands:

```
\section{Answers}

\reset[answer]
\selectblocks[answer][criterium=chapter]
```

You must be aware of the fact that it may be necessary to (temporarily) disable the reference mechanism also:

```
\setupreferencing[state=stop]
```

A more complex situation is this one. Assume that you have several mathematical formulas in your document, and that you want to recapitulate the more complex ones in a separate chapter at the end of the document. You have to specify an `[-]` at formulas you do not want repeated.

```
\defineblock[formula]

\beginformula
\placeformula[newton 1]$$f=ma$$
\endformula

This can also be written as:

\beginformula[-]
\placeformula[newton 2]$$m=f/a$$
\endformula
```

When you re–use the formulas only the first one is typeset. The rest of the formulas is processed, so the numbering will not falter.

The opposite is also possible. By default all local specifications are undone automatically. This means for example that the enumeration of text elements like questions, answers, definitions, etc. can be temporarily stopped. When numbering should continue you specify: `[+]`.[25]

Among the parameters of the number mechanism we (in some cases) use the parameter `blockwise`. This parameter relates to numbering within a set of blocks, for example per chapter.

You may have a document in which the questions and answers are collected in text blocks. The questions are typeset in the document and the answers in a separate appendix. Answers and question are put at the same location in the source file. When we number the questions and answers per chapter, then question 4.12 is the 12th question in chapter 4. The correct number is used in the appendix. In this example answer 4.12 refers to question 4.12 and not the appendix number.

In case we do want the appendix number to be the prefix of the blocknumber we set the parameter `blockwise` at `no`. This is a rather complex situation and will seldom occur.

Earlier we discussed the initializing and resetting of counters. For reasons of uniformity we also have:

```
\reset [...,...]
        *

*   IDENTIFIER
```

In future there will be an option to sort blocks. For that purpose a second set of optional `[ ]` in and `\selectblocks` is available. The first argument is used for 'tags'. These tags are logical labels that enable us to recall the blocks.

```
\beginremark[important]
This is an important message!
\endremark
```

Now we can recall the 'important' messages by:

[25] When you use enumerations within text blocks you can best use the `\start ...stop` alternative (see page ??).

```
\useblocks[remark][important]
```

or:

```
\selectblocks[remark][important][criterium=chapter]
```

Here, `criterium` has the same function as in lists (like tables of content) and registers: it limits the search. In this case, only the blocks belonging to this chapter will be typeset.

More than one 'tag' is allowed in a comma delimited list. Text blocks may be nested:

```
\beginpractice
\beginquestion
\question Is that clear? \par
\endquestion
\beginanswer
\answer Yes it is! \par
\endanswer
\endpractice
```

In this case we use three blocks. Such blocks are stored in a file. This file must be available when the blocks are re–used. This means that the document must be processed at least twice. When blocks are summoned at the end of your source file only one processing step is sufficient but then you have to type the command `nomoreblocks` before the blocks are recalled:

```
\nomoreblocks
```

After this command no blocks should be specified. In the future commands will be developed for local adaptations of the layout of text blocks. Until that moment the following command is all there is:

```
\setupblock [...,...] [..,..=..,..]
                1              2

1   IDENTIFIER

2   before  =  COMMAND
    after   =  COMMAND
    inner   =  COMMAND
    style   =  normal bold slanted boldslanted type cap small... COMMAND
    file    =  FILE
```

A block is being processed within a group, in other words: within `{}`. The setup of `before` and `after` are used outside this group, and the setup of `inner` is used within the group. For example if we mark a re–used text block in the margin we can use the following setup:

```
\defineblock[exampletext]

\beginexampletext
If you wonder why this mechanism was implemented consider an educational
document with hundreds of \quote {nice to know} and \quote {need to know}
text blocks at several ability levels.
\endexampletext
```

```
\setupblock[exampletext][inner=\margintitle{reused}]
\useblocks[exampletext]
```

The first text is set without an indicator in the margin and the second is. If we would have used `before` instead of `inner` some grouping problems had occurred.

**reused** If you wonder why this mechanism was implemented consider an educational document with hundreds of 'nice to know' and 'need to know' text blocks at several ability levels.

You can import text blocks from other source files. For example if you want to use text blocks from a manual for students in a manual for teachers, you can specify:

```
\setupblock
  [homework]
  [file=student,
   before=\startbackground,
   after=\stopbackground]
```

In that case the blocks are imported from the file `student.tex`. In this example these blocks are typeset differently, with a background. When the student material is specified with:

```
\beginhomework[meeting 1]
..........
\endhomework
```

we can summon the blocks in the teacher's manual with:

```
\useblocks[homework][meeting 1]
```

In extensive documents it will take some time to generate these products. But this mechanism garantees we use the same homework descriptions in the students and teachers manual. Furthermore it saves typing and prevents errors.

Questions and answers are good examples of text blocks that can be hidden and moved. The example below will illustrate this. Because commands like `\question` have a paragraph as an argument the `\par`'s and/or empty lines are essential.

In the setup we see that questions and answers are coupled. A coupling has a meaning in interactive documents.

```
\defineblock[question]
\defineblock[answer]

\defineenumeration[question][location=inmargin,coupling=answer]
\defineenumeration[answer][location=top,coupling=question]

\hideblocks[answer]

\starttext

\chapter{\CONTEXT}

\CONTEXT\ is a macropackage that is based on \TEX. \TEX\ is a typesetting
system and a programm. This unique combination is used extensively in
\CONTEXT.

\beginquestion
  \startquestion
```

```
    To date, the fact that \TEX\ is a programming language enables \CONTEXT\
    to do text manipulations that cannot be done with any other known package.

    Can you mention one or two features of \CONTEXT\ that are based on the
    fact that \TEX\ is programming language?
    \stopquestion
  \endquestion

  \beginanswer
    \answer You can think of features like floating blocks and text block
    manipulation. \par
  \endanswer

  \beginquestion
    \question Are there any limitations in \TEX ? \par
  \endquestion

  \beginanswer
    \answer Yes and no. The implementation of \TEXEXEC\ is done in
    \PERL\ rather than in \TEX.
  \endanswer

  \TEX\ is a very powerful tool, but much of its power is yet to be
  unleashed. \CONTEXT\ tries to make a contribution with its user||friendly
  interface and its support of many features, like interactivety.

  \chapter{Answers}

  \useblocks[question,answer]

  \stoptext
```

With `\processblocks` blocks are processed but not typeset. Assume that we have two types of questions:

```
\defineblock[easyquestion,hardquestion]
```

When both types of questions use the same numbering mechanism, we can recall the hard questions in their original order by hiding the easy questions.

```
\processblocks[easyquestion]
\useblocks[hardquestion]
```

## 15.5 Opposite blocks

In future versions of ConTEXt there will be support of spread based typesetting. For the moment the only command available is:

```
\startopposite ... \stopopposite
```

Everything between `start` and `stop` is typeset at the left page in such a way that it is aligned with the last paragraph that is typeset on the right page.

```
\setupoppositeplacing [.*=.]

*   state      =  start stop
    before     =  COMMAND
    inbetween  =  COMMAND
    after      =  COMMAND
```

## 15.6 Margin blocks

Within limits you can place text and figures in the margin. In this case the margin is handled as a separate (very narrow) page next to the actual page.

```
\startmarginblock ... \stopmarginblock
```

This can be setup with:

```
\setupmarginblocks [..,.*=.,..]

*   location   =  inmargin left middle right
    style      =  normal bold slanted boldslanted type cap small... COMMAND
    width      =  DIMENSION
    align      =  inner outer left right flushleft flushright middle center normal no yes
    top        =  COMMAND
    inbetween  =  COMMAND
    bottom     =  COMMAND
    left       =  COMMAND
    right      =  COMMAND
    before     =  COMMAND
    after      =  COMMAND
```

*The mechanism to place blocks is still under construction.*

## 15.7 Hiding text

It is possible to hide text (skip during processing) by:

```
\starthiding ... \stophiding
```

## 15.8 Postponing text

Text elements can be postponed (stored) and placed at the next empty page. This option is needed in case ConTEXt encounters large figures or tables. The postponed textelement is placed at the next page generated by TEX or forced by the user with a manual page break.

```
\startpostponing ... \stoppostponing
```

Several text blocks can be postponed and stored. This proces can be followed on screen during document generation.

```
\startpostponing
\placefigure{A rather large figure.}{...}
\stoppostponing
```

When a lot of text elements are postponed or when a figure uses a complete page we advise you to add \page after the postponing. Otherwise there is the possibility that a blank page is inserted. This is caused by the fact that the postponing mechanism and the float mechanism are completely independent.

```
\startpostponing
\placefigure{A very large figure.}{...}
\page
\stoppostponing
```

## 15.9 Buffers

Buffers simplify the moving of text blocks. They are stored in a file with the extension tmp and are used to bring readability to your source. Furthermore they can be recalled at any location without retyping them.

```
\startBUFFER [.*.] ... \stopBUFFER
              OPTIONAL
*   IDENTIFIER
```

```
\startbuffer [.*.] ... \stopbuffer
              OPTIONAL
*   IDENTIFIER
```

```
\getbuffer [.*.]
            OPTIONAL
*   IDENTIFIER
```

```
\typebuffer [.*.]

*   IDENTIFIER
```

The example below shows the use of these commands.

```
\startbuffer
We see that a {\em buffer} works something like a {\em block}.\par
```

```
\stopbuffer

\startlines
{\tf \getbuffer}
{\bf \getbuffer}
{\sl \getbuffer}
\stoplines
```

This results in:

We see that a *buffer* works something like a *block*.
**We see that a *buffer* works something like a *block*.**
*We see that a* buffer *works something like a* block.

The name is optional. A name makes sense only when several buffers are used. Most of the time the default buffer will do. Most examples in this manual are typed in buffers.

In chapter `??` we can see that the last argument of a `\place<<block>>` can be rather extensive. A buffer can be useful when such large tables are defined.

```
\startbuffer
... <</rm many lines>> ...
\stopbuffer

\placetable{A table.}{\getbuffer}
```

The buffer is set up with:

```
\setupbuffer [.1.] [..,.=2.,..]
               OPTIONAL
1   IDENTIFIER

2   paragraph  =  NUMBER
    before     =  COMMAND
    after      =  COMMAND
```

The first argument is optional and relates to the buffers you defined yourself. You can define your own buffer with:

```
\definebuffer [.*.]

*   IDENTIFIER
```

Be aware of possible conflicting names and use capital letters. After this command `/get<<buffer>>` and `/type<<buffer>>` are available where `<<buffer>>` is the name of the buffer.

# 16 Figures

## 16.1 Introduction

In this chapter we discuss how to place figures in your document. In **section 15.2** we introduced the float mechanism. In this chapter the placement of figures is discussed. Most of the time these figures are created with external applications.

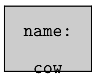
After processing a document the result is a dvi file or, when we use pdfTEX, a pdf file. The dvi document reserves space for the figure, but the figure itself will be put in the document during postprocessing of the dvi file. pdfTEX needs no postprocessing and the external figures are automatically included in the pdf file.

External figures may have different formats like the vector formats eps and pdf, or the bitmap formats tif, png and jpg. Note that we refer to figures but we could also refer to movies. ConTEXt has special mechsnisms to handle figures generated by MetaPost. We have to take care that fonts used in MetaPost figures are recognized by pdfTEX. Finally, we'll see that MetaPost code can be embedded in ConTEXt documents.

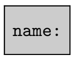
Normally, users need not concern themselves with the internal mechanisms used by ConTEXt for figure processing. However some insight may be useful.

## 16.2 Defining figures

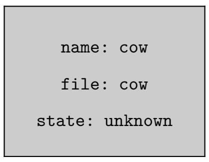
A figure is designed within specific dimensions. These dimensions may of may not be known by the document designer.



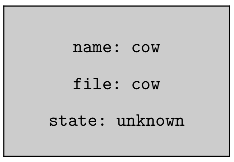
natural dimension



scaled to 25%



a height of 2 cm



a height of 2 cm and a width 3 cm

If the original dimensions are unknown, then scaling the figure to 40% can have some astonishing results. A figure with width and height of 1 cm becomes almost invisible, but a figure width width and height of 50 cm will still be very large when scaled to 40% of its original size. A better strategy is to perform the scaling based on the current bodyfont size, the width of text on the page, or to set absolute dimensions, such as 3 cm by 2 cm.

To give TEX the opportunity to scale the figure adequately the file format must be known. **Table 16.1** shows the file formats supported by dvips, dvipsone, and pdfTEX respectively. pdfTEX has the unique capability to determine the file format during processing.

When we use dvi, TEX can determine the dimensions of an eps illustration by searching for the so called *bounding box*. However, with other formats such as tif, the user is responsible for the determination of the figure dimensions.

| | eps | pdf | MetaPost | tif | png | jpg | mov |
|---|---|---|---|---|---|---|---|
| dvips | + | – | + | - | - | - | + |
| dvipsone | + | – | + | + | - | - | + |
| pdfTEX | - | + | + | + | + | + | + |

**Table 16.1** Some examples of supported file formats.

Now, let us assume that the dimensions of a figure are found. When we want to place the same figure many times, it would be obvious to search for these dimensions only once. That is exactly what happens. When a figure is found it is stored as an object. Such an object is re–used in TEX and in pdf but not in dvi, since reuse of information is not supported by the dvi format. To compensate for this shortcoming, when producing dvi output, ConTEXt will internally reuse figures, and put duplicates in the dvi file.

```
\useexternalfigure[some logo][logo][width=3cm]

\placeexternalfigure{first logo}{\externalfigure[some logo]}

\placeexternalfigure{second logo}{\externalfigure[some logo]}
```

So, when the second logo is placed, the information collected while placing the first one is used. In pdfTEX even the content is reused, if requested, at a different scale.

A number of characteristics of external figures are specified by:

```
\setupexternalfigures [.*=.]

*   scale       =  NUMBER
    yscale      =  NUMBER
    yscale      =  NUMBER
    factor      =  max fit broad
    wfactor     =  NUMBER max broad fit
    hfactor     =  NUMBER max broad fit
    width       =  DIMENSION
    height      =  DIMENSION
    frame       =  on off
    preset      =  yes no
    display     =  FILE
    preview     =  yes no
    repeat      =  yes no
    object      =  yes no
    type        =  eps mps pdf tif png jpg mov cd:tex
    method      =  eps mps pdf tif png jpg mov cd:tex
    option      =  frame empty test
    frames      =  on off
    ymax        =  NUMBER
    xmax        =  NUMBER
    directory   =  TEXT
    location    =  local global default none
    maxwidth    =  DIMENSION
    maxheight   =  DIMENSION
    conversion  =  TEXT
    prefix      =  TEXT
```

This command affect all figures that follow. Three options are available: `frame`, `empty` and `test`. With `empty` no figures are placed, but the necessary space is reserved. This can save you some time when 'testing' a document.[26] Furthermore the figure characteristics are printed in that space. When `frame` is set at `on` a frame is generated around the figure. The option `test` relates to testing hyperactive areas in figures.

When ConTEXt is not able to determine the dimensions of an external figure directly, it will fall back on a simple database that can be generated by the Perl script TEXutil. You can generate such a database by calling this script as follows:

```
texutil  --figures  *.tif
```

This will generate the `texutil.tuf` file, which contains the dimensions of the tif figures found. You need to repeat this procedure every time you change a graphic. Therefore, it can be more convenient to let ConTEXt communicate with TEXutil directly. You can enable that by adding `\runutilityfiletrue` to your local `cont-sys.tex` file.

When a figure itself is not available but it is listed in the `texutil.tuf` file then ConTEXt presumes that the figure does exist. This means that the graphics do not need to be physically present on the system.

Although ConTEXt very hard tries to locate a figure, it may fail due to missing or invalid figure, or invalid path specifications (more on that later). The actual search depends on the setup of directories and the formats supported. In most cases, it it best not to specify a suffix or type.

```
\externalfigure[hownice]
\externalfigure[hownice.pdf]
\externalfigure[hownice][type=pdf]
```

In the first case, ConTEXt will use the graphic that has the highest quality, while in both other cases, a pdf graphic will be used. In most cases, the next four calls are equivalent, given that `hownice` is available in MetaPost output format with a suffix `eps` or `mps`:

```
\externalfigure[hownice]
\externalfigure[hownice][type=eps]
\externalfigure[hownice][type=eps,method=mps]
\externalfigure[hownice][type=mps]
```

In most cases, a MetaPost graphic will have a number as suffix, so the next call makes the most sense:

```
\externalfigure[hownice.1]
```

Let us summarize the process. Depending on the formats supported by the currently selected driver (dvi, pdfTEX, etc.), ConTEXt tries to locate the graphics file, starting with the best quality. When found, ConTEXt first tries to determine the dimensions itself. If this is impossible, ConTEXt will look into `texutil.tuf`. The graphic as well as the file `texutil.tuf` are searched on the current directory (`local`) and/or dedicated graphics directories (`global`), as defined by `\setupexternalfugures`. By default the `location` is set at `{local,global}`, so both the local and global directories are searched. You can set up several directories for your search by providing a comma–delimited list:

```
\setupexternalfigures[directory={c:/fig/eps,c:/fig/pdf}]
```

[26] A similar effect can be obtained with the `--fast` switch in TEXexec.

Even if your operating uses a \ as separator, you should use a `/`. The figure directory may be system dependent and is either set in the file `cont-sys`, in the document preamble, or in a style.

An external figure is summoned by the command `\externalfigure`. The cow is recalled with:

```
\externalfigure[cow][width=2cm]
```

For reasons of maintenance it is better to specify all figures at the top of your source file or in a separate file. The figure definition is done with:

```
\useexternalfigure [.1.] [.2.] [.3.] [..,.4.,..]
                    OPTIONAL      OPTIONAL  OPTIONAL
1   IDENTIFIER
2   FILE
3   IDENTIFIER
4   inherits from \setupexternalfigures
```

Valid definitions are:

```
\useexternalfigure [cow]
\useexternalfigure [some cow] [cow230]
\useexternalfigure [big cow]  [cow230] [width=4cm]
```

In the first definition, the figure can be recalled as `cow` and the graphics file is also `cow`. In the second and third definition, the symbolic name is `some cow`, while the filename is `cow230`. The last example also specifies the dimensions.

The `scale` is given in percentages. A scale of `800` (80%) reduces the figure, while a value of `1200` (120%) enlarges the figure. Instead of using percentages you can also scale with a factor that is related to the actual bodyfont. A setup of `hfactor=20` supplies a figure with 2 times the height of the bodyfont size, and `hfactor=120` will result in a width of 12 times the bodyfont size (so 144pt when using a 12pt bodyfont size). When we want to place two figures next to one another we can set the height of both figures with `hfactor` at the same value:

```
\useexternalfigure[alfa][file0001][hfactor=50]
\useexternalfigure[beta][file0002][hfactor=50]

\placefigure
  {Two figures close to one another.}
  \startcombination[2]
    {\externalfigure[alfa]} {this is alfa}
    {\externalfigure[beta]} {this is beta}
  \stopcombination
```

We can see that `\externalfigure` is capable of using a predefined figure. The typographical consistency of a figure may be enhanced by consistently scaling the figures. Also, figures can inherit characteristics of previously defined figures:

```
\useexternalfigure [alfa]  [file0001] [hfactor=50]
\useexternalfigure [beta]  [file0002] [alfa]
\useexternalfigure [gamma] [file0003] [alfa]
\useexternalfigure [delta] [file0004] [alfa]
```

Normalizing a figure's width must also be advised when figures are placed with `\startfiguretext` below one another.

In most cases you will encounter isolated figures of which you want to specify width or height. In that case there is no relation with the bodyfont except when the units `em` or `ex` are used.

In **figure 16.1** we drew a pattern with squares of a factor 10.

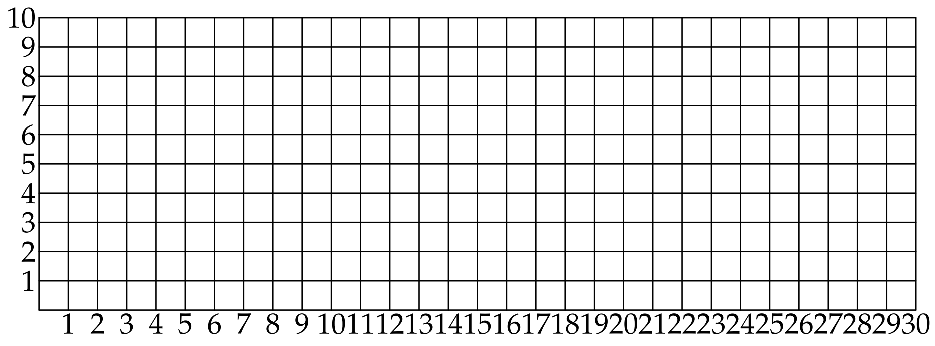


**Figure 16.1** Factors at the actual bodyfont.

## 16.3 Recalling figures

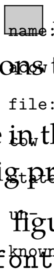
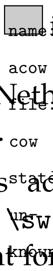
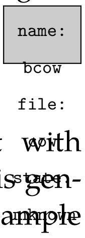
A figure is recalled with the command:

```
\externalfigure [.1.] [..,.2.,..]
                          OPTIONAL
1   FILE
2   inherits from \setupexternalfigures
```

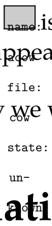
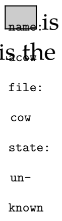
For reasons of downward compatibility a figure can also be recalled with a command that equals the figure name. In the example below we also could have used `\acow` and `\bcow`, unless they are already defined. Using `\externalfigure` instead is more safe, since it has its own namespace.

```
\useexternalfigure[acow][cow][factor=10]
\useexternalfigure[bcow][cow][factor=20]

\placefigure[left,none]{}{\externalfigure[bcow]}

The \hbox {\externalfigure[acow]} is a very well known animal in the Dutch
landscape. But for environmental reasons the \hbox {\externalfigure[acow]}
is slowly disappearing. In the near future the cow will fulfil a marginal
\inleft {\externalfigure[bcow]} role in the Netherlands. That is the
reason why we would like to write the word \hbox {\externalfigure[bcow]}
in big print.
```

Here we see how `acow` and `bcow` are reused. This code will result in:

The  is a very well known animal in the Dutch landscape. But for environmental reasons the  is slowly disappearing. In the near future the cow will fulfil a marginal role in the Netherlands. That is the reason why we would like to write the word  in big print.

Normalized figures adapt to the actual bodyfont at least when the font is set with `\setupbodyfont` or `\switchtobodyfont`. When a text is used for different media and is generated with different fontsizes the use of normalized figures is a good practice. The example above looks different in a smaller fontsize.

The  is a very well known animal in the Dutch landscape. But for environmental reasons the  is slowly disappearing. In the near future the cow will fulfil a marginal role in the Netherlands. That is the reason why we would like to write the word in big print.

## 16.4 Automatic scaling

In cases where you want the figure displayed as big as possible you can set the parameter `factor` at `max`, `fit` or `broad`. In most situations the value `broad` will suffice, because then the caption still fits on a page.

| **setup** | **result** |
|---|---|
| `max` | maximum width or height |
| `fit` | remaining width or height |
| `broad` | more remaining width or height |
| *`number`* | scaling factor (times 10) |

**Table 16.2** Normalized figures.

So, one can use `max` to scale a figure to the full page, or `fit` to let it take up all the remaining space. With `broad` some space is reserved for a caption.

Sometimes it is not clear whether the height or the width of a figure determines the optimal display. In that case you can set `factor` at `max`, so that the maximal dimensions are determined automatically.

```
\externalfigure[cow][factor=max]
```

This figure of a cow will scale to the width or height of the text, whichever fits best. Even combinations of settings are possible:

```
\externalfigure[cow][factor=max,height=.4\textheight]
```

In this case, the cow will scale to either the width o fthe text or 40% of the height of the text, depending on what fits best.

As already said, the figures and their characteristics are stored in the file `texutil.tuf` and can be displayed with:

```
\showexternalfigures [..,..=..,..]
                            OPTIONAL
*   alternative  =  a b c
```

There are two alternatives: `a`, `b` and `c`. The first alternative leaves room for figure corrections and annotations, the second alternative is somewhat more efficient and places more figures on one page. The third alternative puts each figure on its own page. Of course one needs to provide the file `texutil.tuf` by saying:

```
texutil --figures *.mps *.jpg *.png
```

Even more straightforward is running TEXexec, for instance:

```
texexec --figures=c --pdf *.mps *.jpg *.png
```

This will give you a pdf file of the figures requested, with one figure per page.

## 16.5 TEX–figures

Figures can be scaled. This mechanism can also be used for other text elements. These elements are then stored in separate files or in a buffer. The next example shows how a table is scaled to the pagewidth. The result is typeset in **figure 16.2**.

```
\startbuffer[table]
  \starttable[||||||]
    \HL
    \VL \bf factor          \VL \bf width            \VL
        \bf height          \VL \bf width and height \VL
        \bf nothing         \VL \SR
    \HL
    \VL \type{max}          \VL automatically        \VL
        automatically       \VL automatically        \VL
        width or height     \VL \FR
    \VL \type{fit}          \VL automatically        \VL
        automatically       \VL automatically        \VL
        width or height     \VL \MR
    \VL \type{broad}        \VL automatically        \VL
        automatically       \VL automatically        \VL
        width or height     \VL \MR
    \VL \type{...}          \VL width                \VL
        height              \VL isometric            \VL
        original dimensions \VL \LR
    \HL
  \stoptable
\stopbuffer
```

```
\placefigure
  [here][fig:table]
  {An example of a \TEX\ figure.}
  {\externalfigure[table.buffer][width=\textwidth]}
\placefigure
  {An example of a \TEX\ figure.}
  {\externalfigure[table][width=.5\textwidth, type=buffer]}
```

| factor | width | height | width and height | nothing |
|---|---|---|---|---|
| `max` | automatically | automatically | automatically | width or height |
| `fit` | automatically | automatically | automatically | width or height |
| `broad` | automatically | automatically | automatically | width or height |
| `...` | width | height | isometric | original dimensions |

**Figure 16.2** An example of a TEX figure.

| factor | width | height | width and height | nothing |
|---|---|---|---|---|
| `max` | automatically | automatically | automatically | width or height |
| `fit` | automatically | automatically | automatically | width or height |
| `broad` | automatically | automatically | automatically | width or height |
| `...` | width | height | isometric | original dimensions |

**Figure 16.3** An example of a TEX figure.

With `\typesetbuffer` you go a step further: not just one text element but a whole document can be typeset and inserted as figure. As an example, the second page of some text on A7-paper with landscape orientation is shown in **figure 16.4**.

```
\startbuffer[a7-buf]
\setuppapersize[A7, landscape][A7, landscape]
\showframe
\input tufte
\stopbuffer
```

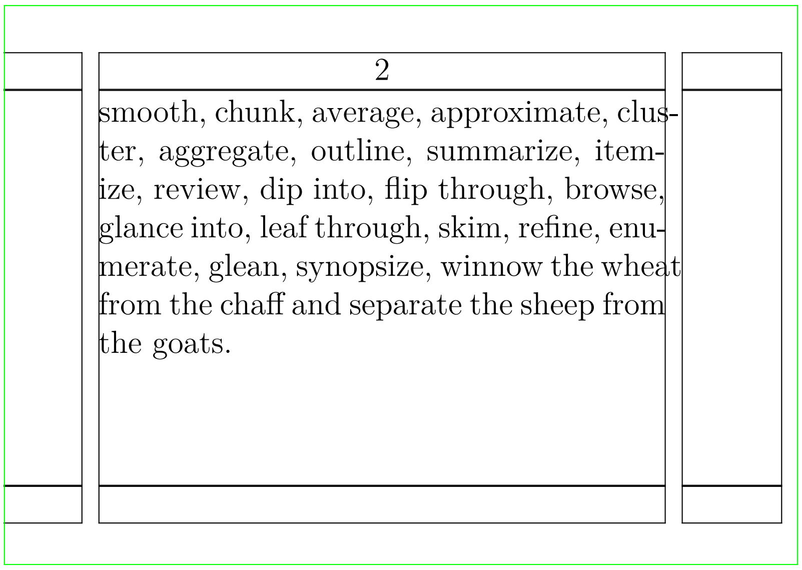
```
\placefigure[][fig:a7-landscape-layout]
  {An example for \tex{typesetbuffer}.}
  {\typesetbuffer[a7-buf][page=2]}
```

## 16.6 Extensions of figures

In the introduction we mentioned different figure formats like eps and png. In most situations the format does not have to be specified. On the contrary, format specification would mean that we would have to re–specify when we switch from dvi to pdf output. The figure format that ConTEXt will use depends on the special driver. First preference is an outline, second a bitmap.

MetaPost figures, that can have a number as suffix, are recognized automatically. ConTEXt will take care of the font management when it encounters MetaPost figures. When color is disabled, or rgb is to be converted to cmyk, ConTEXt will determine what color specifications have to be converted in the MetaPost file. If needed, colors are converted to weighted grey scales, that print acceptable on black and white printers. In the next step the fonts are smuggled into the file.[27] In case of pdf output the MetaPost code is converted into pdf by TEX.

[27] Fonts are a problem in MetaPost files, since it it up to the postprocessor to take care of them. In this respect, MetaPost output is not self contained.


2

smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

**Figure 16.4** An example for `\typesetbuffer`.

If necessary the code needed to insert the graphic is stored as a so called object for future re–use. This saves processing time, as well as bytes when producing pdf. You can prevent this by setting `object=no`.

When eps and mps (MetaPost) figures are processed ConTEXt searches for the high resolution bounding box. By default the PostScript bounding box may have a deviation of half a point, which is within the accuracy of our eyes. Especially when aligning graphics, such deviations will not go unnoticed.

ConTEXt determines the file format automatically, as is the case when you use:

```
\externalfigure[cow]
```

Sometimes however, as we already explained, the user may want to force the format for some reason. This can be done by:

```
\externalfigure[cow.eps]
\externalfigure[cow][type=eps]
```

In special cases you can specify in which way figure processing takes place. In the next example ConTEXt determines dimensions asif the file were in eps format, that is, it has a bounding box, but processes the files as if it were a MetaPost file. This kind of detailed specification is seldom needed.

```
\externalfigure[graphic.xyz][type=eps,method=mps]
```

The automatic searching for dimensions can be blocked by `preset=no`.

## 16.7 Movies

In ConTEXt moving images or 'movies' are handled just like figures. The file format type is not determined automatically yet. This means the user has to specify the file format.

```
\externalfigure[demo.mov][label=demo,width=4cm,height=4cm,preview=yes]
```

With this setup a preview is shown (the first image of the movie). If necessary an ordinary (static) figure can be layed over the first movie image with the overlay mechanism.

Movies can be controlled either by clicking on them, or by providing navigational tools, like:

```
... \goto {start me} [StartMovie{demo}] ...
```

A more detailed discussion on controlling widgets is beyond this chapter. Keep in mind that you need to distribute the movies along with your document, since they are not included. This makes sense, since movies can be pretty large.

## 16.8 Some remarks on figures

Figures, and photos in particular, have to be produced with consistent proportions. The proportions specified in figure 16.5 can be used as a guideline. Scaling of photos may cause quality loss.

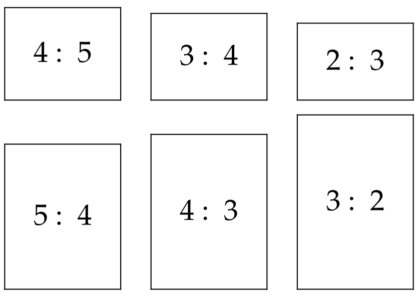


**Figure 16.5** Some preferred image proportions.

In the background of a figure you typeset a background (see figure ??). In this example the external figures get a background (for a black and white reader: a green screen).

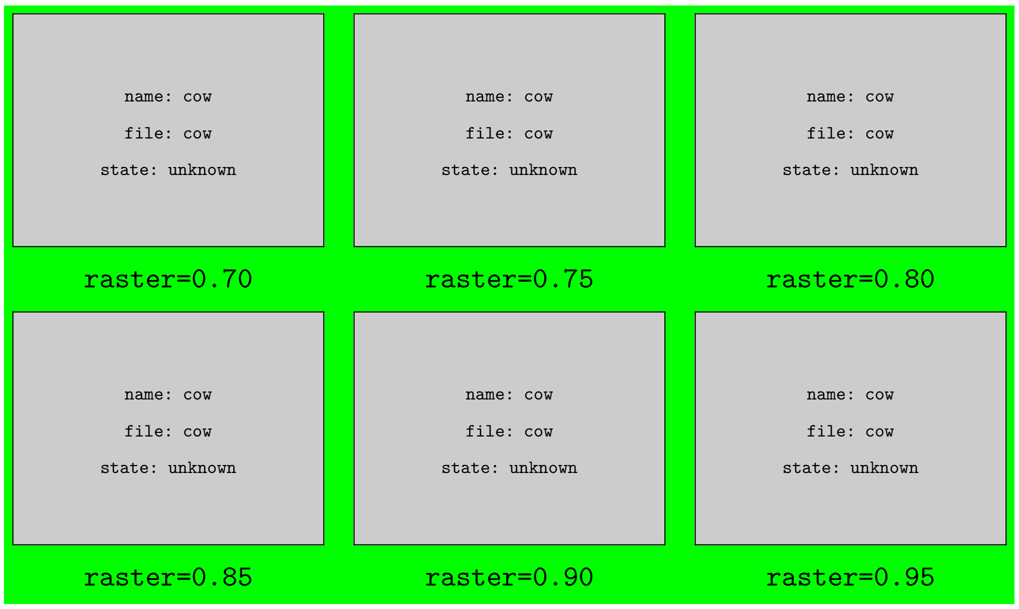


**Figure 16.6** Some examples of backgrounds in figures.

```
\setupfloats
  [background=color,
   backgroundcolor=green,
   backgroundoffset=3pt]
\useexternalfigure [cow]
  [hfactor=80,
   background=screen,
   backgroundscreen=0.75]
```

Note that we use only one float and that there are six external figures. The background of the float is used for the complete combination and the background of the external figure only for the figure itself.

# 17 Tabulation

The second mechanism for generating tabular information is tabulation. We will see that the specification of tabulations does not differ much from that of tables.

Tabular information can be found in the running text and the location of that information is fixed (i.e. it is not allowed to float like tables and figures).

The tabulation mechanism is meant for that tabular information in which cells may contain information with more that one paragraph. However the table and tabular mechanism can be used indifferently we advise you to use them consistently because the spacing within the both mechanisms differ.

The table commands form a layer around TABLE, but the tabulation commands are written for ConTEXt. The tabulation mechanism uses the same interface when possible. As we do in the table mechanism we use `\NC` as column separator and `\NR` as row separator.

```
\starttabulate[|l|c|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC low and high  \NC up and down    \NC back and forth \NC \NR
\stoptabulate
```

| | | |
|---|---|---|
| this and that | left and right | here and there |
| low and high | up and down | back and forth |

The three commands `l`, `c` and `r` stand for:

| | |
|---|---|
| `l` | left align |
| `c` | center |
| `r` | right align |

There are spacing commands. These relate to one–line as well as multi–line (paragraphs) cells.

| | |
|---|---|
| `i`*n* | spacing left |
| `j`*n* | spacing right |
| `k`*n* | spacing around |

The factor *n* is applied to the unit of spacing which is default set at .5em (see `\setuptabulate`).

```
\starttabulate[|l|k2c|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC low and high  \NC up and down    \NC back and forth \NC \NR
\stoptabulate
```

| | | |
|---|---|---|
| this and that | left and right | here and there |
| low and high | up and down | back and forth |

The width of a column is set with:

```
\starttabulate[|lw(4cm)|w(4cm)l|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC low and high  \NC up and down    \NC back and forth \NC \NR
\stoptabulate
```

| | | |
|---|---|---|
| this and that | left and right | here and there |
| low and high | up and down | back and forth |

The most important reason for developing the tabulation mechanism lies in the fast that we wanted to be able to type set multi paragraph columns. A prerequisite was that we should be able to use the full width of the text body. This option is supported by:

| | |
|---|---|
| `w(`*d*`)` | 1 line, fixed width |
| `p(`*d*`)` | paragraph, fixed width |
| `p` | paragraph, maximum width |

In the next example the first column has an unknown width. The second column contains a left aligned paragraph with a width of 4 cm. The third column has a width of 2 cm and consists of one line. The last column contains a paragraph that occupies the remaining width.

```
\starttabulate[|l|p(4cm)l|w(2cm)|p|]
...
\stoptabulate
```

A four column table with four paragraphs is specified with:

```
\starttabulate[|p|p|p|p|]
...
\stoptabulate
```

In stead of specifying a body font in each cell we can specify them per column. In the next tabulation the definition is `[|lT|p|]`.

| | |
|---|---|
| `B` | **boldface** |
| `I` | *italic* |
| `R` | *roman* |
| `S` | *slanted* |
| `T` | `teletype` |

Math is possible with:

| | |
|---|---|
| `m` | in–line math |
| `M` | display math |

With the letter `f` we can specify a body font, like `f\bs`. There are also the following commands:

| | |
|---|---|
| `f\command` | font specification |
| `b{..}` | place .. before the entry |
| `a{..}` | place .. after the entry |
| `h\command` | apply `\command` on the entry |

The `h`–command (hook) allows some tricks like:

```
\starttabulate[|w(2cm)h{\inframed}|b{(}a{)}|p|]
\HC {Uggly}     \NC isn't it?       \NC he says.     \NC \NR
\HC {Beautiful} \NC but meaningless \NC I would say. \NC \NR
\stoptabulate
```

Because we use `\inframed` the frame remains within the line. The command applies only to the cells that are preceded by `\HC`. The `{}` are important because `\inframed` expects these.

| | | |
|---|---|---|
| $\boxed{\text{Uggly}}$ | (isn't it?) | he says. |
| $\boxed{\text{Beautiful}}$ | (but meaningless) | I would say. |

We can use `h` for alternative situations, like:

em]

| **item** | **number** |
|---|---|
| figures | =5 |
| tables | =8 |
| formulas | =12 |

All three cells are adapted. Do not forget the `{}` in the column with the numbers!

```
\unexpanded\def\SmallDash#1{\blackrule[width=#1em]}
\starttabulate[|l|lh\SmallDash|]
\HL
\NC \bf item \NC \bf number \NC \NR
\HL
\NC figures  \HC  {5}       \NC \NR
\NC tables   \HC  {8}       \NC \NR
\NC formulas \HC {12}       \NC \NR
\HL
\stoptabulate
```

We used `\NC` as a column separator but an alternative is `\EQ` that places a specified character.

```
\starttabulate
\NC =||sign      \EQ a separator can be specified by altering the
                     variable \type {EQ} \NC \NR
\NC :||character \EQ default a colon is used but an equal sign
                     is a reasonable alternative \NC \NR
\stoptabulate
```

This results in:

| | | |
|---|---|---|
| =–sign | = | a separator can be specified by altering the variable `EQ` |
| :–character | = | default a colon is used but an equal sign is a reasonable alternative |

We saw `\NC` for normal cell entries, `\EQ` for entries separated by a character and `\HC` for entries that are influenced by a command. There is also `\HQ` for a cell entry with a separator and a command. When no formatting is needed there are the commands: `\RC` and `\RQ`.

| **separator** | **normal** | **raw** | **command** |
|---|---|---|---|
| **yes** | `\EQ` | `\RQ` | `\HQ` |
| **no** | `\NC` | `\RC` | `\HC` |

This small tabulation shows all three alternatives. Here we have a tabulation with four centered columns, **boldface** or `verbatim`, of which two cells have a different alignment. The table is coded as:

```
\starttabulate[|*{4}{cBh{\type}|}]
\NC     separator \NC normal \NC raw   \NC command \NC \NR
\RC \bf yes       \HC {\EQ}  \HC {\RQ} \HC {\HQ}   \NC \NR
```

```
\RC \bf no        \HC {\NC}  \HC {\RC} \HC {\HC}   \NC \NR
\stoptabulate
```

The equal sign or any other character can be forced with the `e` command in the definition.

|   |   |
|---|---|
| `e` | sets a symbol in front of the next column |

When several columns have an equal specification we can combine those specifications. Note that the number of `|` must be correct.

```
\starttabulate[|*{3}{k1pc|}]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC low and high  \NC up and down    \NC back and forth \NC \NR
\stoptabulate
```

Here we typed $1 + 3 \times 1 = 4$ times a `|`.

|   |   |   |
|---|---|---|
| this and that | left and right | here and there |
| low and high | up and down | back and forth |

A better example of the automatic cell width determination is the next one.

|   |   |
|---|---|
| tables | We use `\starttable` when we typeset tables but the exact location is not fixed and the information is allowed to float in the running text. |
| tabulation | The command `\starttabulate` is meant for tabular information that is part of the running text. The automatic calculation of the cell width is a feature in this mechanism. |

This tabulation was typed as:

```
\starttabulate[|l|p|]
\NC tables     \NC We use \type {\starttable} when we typeset tables
                   but the exact location is not fixed and the
                   information is allowed to float in the running
                   text. \NC \NR
\NC tabulation \NC The command \type {\starttabulate} is meant for
                   tabular information that is part of the running text.
                   The automatic calculation of the cell width
                   is a feature in this mechanism. \NC \NR
\stoptabulate
```

When no tabulation is specified it is assumed that `[|l|p|]` is wanted. To prevent typing the same specification all over again you can use the tabulation format definition command:

```
\definetabulate[Three][|lB|lS|p|]

\startThree
\NC one \NC two  \NC three four five six seven eight nine ten eleven
                      twelve thirteen fourteen fifteen and so on \NC \NR
\stopThree
```

|   |   |   |
|---|---|---|
| **one** | *two* | three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen and so on |

The tabulation commands can be summarized with:

```
\definetabulate [.1.] [.2.] [.3.]
                     OPTIONAL
1   IDENTIFIER
2   IDENTIFIER
3   TEXT
```

The first argument gives the tabulation a logical name. The second argument is optional and specifies the associated tabulations; later on we will give an example. The last argument specifies the cells.

Then we have:

```
\startTABULATE [.1.] [..,.2.,..] ... \stopTABULATE
               OPTIONAL  OPTIONAL
1   TEXT
2   inherits from \setupexternalfigures
```

In this command the first argument specifies the cells, the second and optional argument the set up.

```
\setuptabulate [.1.] [..,.2.,..]
               OPTIONAL
1   IDENTIFIER

2   unit           =  DIMENSION
    indenting      =  never none not no yes always first next small medium big normal odd
                      even DIMENSION
    before         =  COMMAND
    after          =  COMMAND
    inner          =  COMMAND
    EQ             =  TEXT
    rulecolor      =  IDENTIFIER
    align          =  inner outer left right flushleft flushright middle center normal no
                      yes
    rulethickness  =  DIMENSION
    distance       =  blank grid depth DIMENSION small medium big none
    bodyfont       =  5pt ... 12pt small big
    rule           =  normal line
    split          =  yes no
```

The optional argument specifies the associated tabulations. When the parameter `indenting` is set at `yes`, the width of the tabulations will adapt to the actual indent. In case of a `\start ... \stopnarrower` environment the left and right indent are taken into account. The parameter `unit` is used for the spacing commands `i`, `j` and `k`. The commands specified after the parameter `inner` are applied just in front of the first row and are effective in the whole tabulation.

The possibilities for framing tabulations are limited. You can add horizontal lines with `\HL`. This command takes care of the vertical spacing as the next example illustrates:

```
\starttabulate[|l|p|]
\HL
```

```
\NC small  \NC They say, small is beautiful.        \NC \NR
\HL
\NC medium \NC It seems that medium is the message. \NC \NR
\HL
\NC large  \NC Large T||shirts are always sold out. \NC \NR
\HL
\stoptabulate
```

When a pagebreak occurs in the middle of a tabulation the horizontal line is repeated automatically. Vertical spacing can be set by `\FL`, `\ML` and `\LL`. These commands stand for *first*, *middle* and *last line*.

| | |
|---|---|
| small | They say, small is beautiful. |
| medium | It seems that medium is the message. |
| large | Large T–shirts are always sold out. |

The spacing around the lines is related to the depth of a line.

```
\setuptabulate[distance={depth,medium}]
```

There are different ways to adapt this set up, like:

```
\setuptabulate[distance=none]
\setuptabulate[distance=big]
\setuptabulate[distance={blank,small}]
\setuptabulate[distance={1ex,medium}]
\setuptabulate[distance=1cm]
```

Tabulation is meant for the running text but it can also be used in a floating block. In that case the spacing around tabulation is suppressed. In the running text the actual whitespace and textwidth are taken into account.

- This means that a tabulation within an itemization is adapted to the indent.

  | | |
  |---|---|
  | You see? | As we can expect the width of a paragraph is adapted to the width of the text. And you can even put an itemize in such a cell.<br>– like this<br>– or that |

- This little table was defined like this:

  ```
  \starttabulate
  \NC You see? \NC As we can expect the width of a paragraph is adapted
                   to the width of the text. And you can even put an
                   itemize in such a cell.
                   \startitemize[packed]
                   \item like this
                   \item or that
                   \stopitemize \NC \NR
  \stoptabulate
  ```

We can use and abuse tabulations to obtain some special effects. Vice versa common effects can be combined quite well with tabulations. The next, somewhat strange example will illustrate that.

| | | | | | |
|---|---|---|---|---|---|
| 1. | first | ● ● ● ● ● | this or that | $\alpha$. | alpha |
| 2. | second | ● ● ● ● ● | so and so | $\beta$. | beta |
| 3. | third | ● ● ● ● ● | here or there | $\gamma$. | gamma |

In these kind of situations we should set the itemization with the key `packed`.

```
\starttabulate[|p(2cm)|p(4cm)|p|]
\NC \startitemize[n,packed]
    \item first \item second \item third
    \stopitemize
\NC \startitemize[packed][items=5,width=4em,distance=.5em]
    \its this or that \its so and so \its here or there
    \stopitemize
\NC \startitemize[g,packed,broad]
    \item alpha \item beta \item gamma
    \stopitemize
\NC\NR
\stoptabulate
```

The content of a tabulation has some limitations, because TEX first reads the complete table. These limitations relate to the macros that use `\catcode` adaptations. In normal situations you will not notice these limitations, only when you have typeset TEX input with TEX.

While discussing tables we already saw a financial table. These kind of tables can best be set with the tabulation commands.

| | |
|---|---|
| not so much | 1.220 |
| somewhat more | 5.186 |
| together | $\overline{6.406}$ |

This tabulation was typed like this:

```
\starttabulate[|l|r|]
\NC not so much   \NC          1.220  \NC \NR
\NC somewhat more \NC          5.186  \NC \NR
\NC together      \NC \overbar{6.406} \NC \NR
\stoptabulate
```

As soon as we work with numbers there are several ways of alignment. Like in tables we can make use of ~, but we have to indicate the meaning of ~ explicitly. This is caused by the fact that we still want to use the ~ within paragraphs as an non–hyphenatable space.

```
\starttabulate[|l|~c|]
\NC this is less \NC ~12 \NC \NR
\NC than that    \NC 185 \NC \NR
\stoptabulate
```

We return to the defining of categories of tabulations. An application of this option can be found in the commands that make up a legend with a formula.

```
\definetabulate [legend]       [|emj1|i1|mR|]
\definetabulate [legend] [two] [|emj1|emk1|i1|mR|]
\setuptabulate  [legend]       [unit=.75em,EQ={=}]
```

After these definitions that are default in ConTEXt we can type:

```
\startlegend
\NC w \NC the width of a box  \NC pt \NR
\NC h \NC the height of a box \NC pt \NR
\NC d \NC the depth of a box  \NC pt \NR
\stoplegend
```

This very simple legend becomes this:

| | | | |
|---|---|---|---|
| *w* | = | the width of a box | *pt* |
| *h* | = | the height of a box | *pt* |
| *d* | = | the depth of a box | *pt* |

An extra entry is possible when we add the key `two`:

```
\startlegend[two]
\NC w \NC width  \NC the width of a box  \NC pt \NR
\NC h \NC height \NC the height of a box \NC pt \NR
\NC d \NC depth  \NC de depth of a box   \NC pt \NR
\stoplegend
```

This related tabulation inherits the set up of the original. We also could have defined `\startlegendtwo`, but the mentioned definition origins from the older functionality that was part of earlier ConTEXt versions.

| | | | | | |
|---|---|---|---|---|---|
| *w* | = | *width* | = | the width of a box | *pt* |
| *h* | = | *height* | = | the height of a box | *pt* |
| *d* | = | *depth* | = | de depth of a box | *pt* |

In a similar way the commands for typesetting facts are defined.

```
\definetabulate [fact] [|R|ecmj1|i1mR|]
\setuptabulate  [fact] [unit=.75em,EQ={=}]
```

The first column is set in roman and the next column is separated by an equal sign. That second column is centered and is set in math mode. That column also has some more whitespace. The last column is also set in math mode but the characters are set in roman. Some whitespace is added.

```
\startfact
\NC width  \NC w \NC 48pt \NR
\NC height \NC h \NC  9pt \NR
\NC depth  \NC d \NC  3pt \NR
\stopfact
```

This results in:

| | | | |
|---|---|---|---|
| width | $w$ | = | $48pt$ |
| height | $h$ | = | $9pt$ |
| depth | $d$ | = | $3pt$ |

In reality we also give a value to `inner` and then specifications as below are possible:

```
\startfact
\\ width  \\ w \\ 48pt \\
\\ height \\ h \\  9pt \\
\\ depth  \\ d \\  3pt \\
\stopfact
```

We want to conclude with an example of an automatic calculation of the width of a paragraph. This command shows —and we already saw that in other examples— that the last `\NC` is redundant.

```
\starttabulate[|Bl|p|Bl|]
\NC Read Me \NC \input tufte \NC Edward Tufte \NR
\stoptabulate
```

| | | |
|---|---|---|
| **Read Me** | We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats. | **Edward Tufte** |

As was said earlier ConTEXt takes care of adequate page breaking in the middle of a tabulation. When we set `\tracetabulatetrue` red lines are drawn in positions where breaking is not allowed.

```
\starttabulate[|c|p|p|]
\NC \bf Alpha \NC \bf Beta    \NC \bf Gamma            \NC\NR
\NC 1         \NC right indeed    \NC definitely wrong   \NC\NR
\NC 2         \NC \thinrules[n=3] \NC \thinrules[n=3]    \NC\NR
\NC 3         \NC oh yes          \NC simply no          \NC\NR
\NC 4         \NC very true       \NC as false as can be \NC\NR
\NC 5         \NC \thinrules[n=5] \NC \thinrules[n=5]    \NC\NR
\NC 6         \NC \thinrules[n=3] \NC \thinrules[n=4]    \NC\NR
\stoptabulate
```

| **Alpha** | **Beta** | **Gamma** |
|---|---|---|
| 1 | right indeed | definitely wrong |
| 2 | | |
| 3 | oh yes | simply no |
| 4 | very true | as false as can be |
| 5 | | |

| | | |
|---|---|---|
| 6 | | |

```
\starttabulate[|c|p|p|]
\NC \bf Alpha \NC \bf Beta        \NC \bf Gamma          \NC\NR
\NC 1         \NC right indeed    \NC definitely wrong   \NC\NR
\NC 2         \NC oh yes          \NC simply no          \NC\NR
\NC 3         \NC very true       \NC as false as can be \NC\NR
\NC 4         \NC the whole truth \NC but the truth      \NC\NR
\stoptabulate
```

| **Alpha** | **Beta** | **Gamma** |
|---|---|---|
| 1 | right indeed | definitely wrong |
| 2 | oh yes | simply no |
| 3 | very true | as false as can be |
| 4 | the whole truth | but the truth |

# 18 Formulas

## 18.1 Introduction

For what reason do we need a complete chapter on formulas? The reason is obvious: a considerable part of the functionality of TEX relates to math typesetting since the main reason for developing TEX was the need for typesetting math.

In ConTEXt math typesetting is not really an issue. ConTEXt was developed for typesetting educational materials and not necessarily math. Therefore more attention was paid to chemical formulas and consistent use of units than to math. Math was available anyhow.

In ConTEXt the functionality is more oriented towards the educational disciplines and these can be found in specific modules. A module will not supply basic functionality because it can be found in the core.

There are modules for chemical stuff, units and flow–charts, which all have their own manual. The same goes for the math module. This module contains the same functionality as the macros developed by the *American Mathematical Society*. Those macros are well–known in the TEX community. Most extensions concern the interface and consistent spacing. In this chapter we pay attention to the standard functionality in ConTEXt.

## 18.2 Basic commands

Typesetting formulas is one of the strong points of TEX. Special commands are available for typesetting math. These commands are enclosed by single or double dollar signs.

In the running text we use single dollar signs: `$a=b^2+1/c$` becomes $a = b^2+1/c$. In conjunction with in–line–math there is display–math, or rather formulas surrounded by whitespace. Those formulas are frequently numbered. The location and way of numbering can be set with:

```
\setupformulas [..,.*=.,..]

*   location     =  left right
    left         =  TEXT
    right        =  TEXT
    align        =  inner outer left right flushleft flushright middle center normal no yes
    option       =  middle
    strut        =  yes no
    distance     =  DIMENSION
    margin       =  DIMENSION standard yes no
    align        =  flushleft flushright middle center
    leftmargin   =  DIMENSION
    rightmargin  =  DIMENSION
    indentnext   =  yes no
    alternative  =  IDENTIFIER
    spacebefore  =  DIMENSION
    after        =  DIMENSION
    separator    =  TEXT
    conversion   =  numbers characters Characters romannumerals Romannumerals TEXT
```

With `left` and `right` characters on the left or right side of the formula number are set up. Default these are `(` and `)`.

A (numbered) formula is defined with the commands:

```
\placeformula [...,...] {.2.} $$.3.$$
                 1
              OPTIONAL   OPTIONAL
1   REFERENCE
2   CONTENT
3   DISPLAY MATH
```

```
\placesubformula [...,...] {.2.} $$.3.$$
                    1
                 OPTIONAL   OPTIONAL
1   REFERENCE
2   CONTENT
3   DISPLAY MATH
```

The reference and subnumber are optional. Below we give some examples of formulas. In the margin we display the references. Typing the formula number manually is necessary when we make use of tables, matrices and TEX–commando's like `\displaylines`. In the examples we use `$$` to save some space; however we advise you to use the command `\startformula`.

**FIXME:** Mark IV doesn't have a `\formulanumber` command at the moment, the rest of this paragraph is suppressed

When we want *no* numbers we have to indicate that explicitly by means of `[-]`:

```
\placeformula[-]
  $$\displaylines
      {ab=ba\hfill\cr
       ac+bc=(a+b)c\hfill\cr}$$
```

This results in:

$ab = ba$

$ac + bc = (a + b)c$

We also could have used here `\startformula...\stopformula`:

```
\placeformula[-]
  \startformula
  \displaylines{ab=ba\hfill\cr ac+bc=(a+b)c\hfill\cr}
  \stopformula
```

The use of the `\start...\stop`–pair has the advantage that we can test symmetry in some wordprocessors. The disadvantage is we can not see immediately that we work in math mode.

```
\startFORMULA ... \stopFORMULA
```

The next examples does use numbers. In this example `[that's it]` is a logical name, a label, for future referencing.

```
\placeformula
  \startformula
    \displaylines
        {a\times b=b\times a\hfill\formulanumber\cr
         a+b=b+a\hfill\subformulanumber\cr
         ac+bc=(a+b)c\hfill\formulanumber[that's it]{x}\cr}
  \stopformula
```

This becomes:

**FIXME:** getbuffer suppressed

## 18.3 Legends

In case of physics formulas you may want to explain the meaning of the used symbols. There are two commands to do that:

```
\startlegend [.1.] .2. .3. .4. ... \stoplegend
              OPTIONAL
1   two
2   NOTHING
3   NOTHING
4   NOTHING
```

```
\startfact .1. .2. .3. ... \stopfact
1   NOTHING
2   NOTHING
3   NOTHING
```

A legend and facts are coded as follows:

```
\placeformula[for:force]$$F = m a$$

\startlegend
\leg F \\ force        \\ N       \\
\leg m \\ mass         \\ kg      \\
\leg a \\ acceleration \\ m/{s^2} \\
\stoplegend
```

```
Determine by means of formula~\in[for:force] the acceleration~$a$
when given is that:

\startfact
\fact mass  \\ m \\ 10~kg  \\
\fact force \\ F \\ 1500~N \\
\stopfact
```

This results in:

$$F = ma \tag{18.1}$$

| | | | |
|---|---|---|---|
| $F$ | = | force | $N$ |
| $m$ | = | mass | $kg$ |
| $a$ | = | acceleration | $m/s^2$ |

Determine by means of formula **18.1** the acceleration $a$ when given is that:

| | | | |
|---|---|---|---|
| mass | $m$ | = | 10 $kg$ |
| force | $F$ | = | 1500 $N$ |

A combination is also possible:

| | | | | | |
|---|---|---|---|---|---|
| $F$ | = | | = | force | $N$ |
| $m$ | = | 10 | = | mass | $kg$ |
| $a$ | = | 1500 | = | acceleration | $m/s^2$ |

This was specified in this way:

```
\startlegend[two]
\leg F \\      \\ force        \\ N       \\
\leg m \\   10 \\ mass         \\ kg      \\
\leg a \\ 1500 \\ acceleration \\ m/{s^2} \\
\stoplegend
```

## 18.4 Units

A unit can be typeset with:

```
10~$\rm m^3$
```

For the purpose of consistent typesetting the command `\unit` is available. This is an example of the use of synonyms as described in **section 12.2**.

```
\unit {strange} {m^3\!/s^2} {a strange unit}
```

In this case the `\!` takes care of backskipping the / in such a way that in stead of $m^3/s^2$ we get $m^3\!/s^2$. In fact we can do without these kind of cryptic typing, because the unit module offers a better alternative. The module is loaded in the set up area of your source file with:

```
\usemodule[unit]
```

After that you can type the recall unit by typing them. For example:

```
... 10 \Meter \Per \Second\ ...
... 33 \Kilo \Gram \Per \Square \Meter\ ...
```

At this point we advise you to read the manual that comes with this module for more examples.

When we use math commands there may occur problems as soon as we use $ in a nested way. When we are in math mode and we use a $ for the purpose of switching to math mode we just end math mode like this:

```
$a $\times$ b$
```

TEX will produce an error because `\times` is typed outside math mode. In this example we saw what goes wrong but the problem is less obvious in the next example:

```
\def\multiply{$\times$}
$a \multiply b$
```

This seems correct but with `\multiply` we leave math mode. We can prevent errors by defining `\multiply` as follows:

```
\def\multiply{\ifmmode \times \else $\times$ \fi}
```

The next commands does just that:

```
\mathematics {.*.}

*   CONTENT
```

We can use this command in nested situations:

```
\mathematics{a\mathematics{b\mathematics{c\mathematics{d\mathematics{e}}}}}
```

and it will result in a correct output:

*abcde*

so do not use this:

*a*bc*d*e

which we would have obtained by typing:

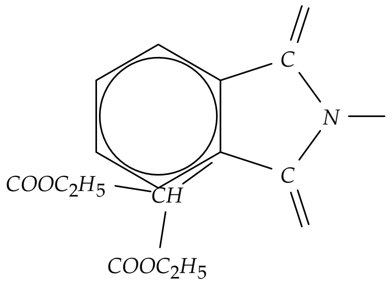
```
$a$b$c$d$e$
```

## 18.5 Chemicals

Earlier we stated that in this chapter we also describe the module for chemical typesetting. This module is loaded with:

```
\usemodule[chemic]
```

The first version of this module used PICTEX for positioning text and drawing the chemical structures, the current version uses MetaPost for drawing the graphics. The results are better and the files are more compact.



This chemical structure was typed as follows:

```
\startchemical[with=fit,height=fit]
  \chemical
    [SIX,B,C,ADJ1,
     FIVE,ROT3,SB34,+SB2,-SB5,Z345,DR35,SR4,CRZ35,SUB1,
     ONE,OFF1,SB258,Z0,Z28]
    [C,N,C,O,O,
     CH,COOC_2H_5,COOC_2H_5]
\stopchemical
```

The interface (syntax) looks rather cryptic but after some practice its compactness is an asset. There is an extensive manual and a collection of examples available.

One characteristic of chemical typesetting is the fact that all super– respectively subscripts are at the same height. This is not the case in math typesetting where the location of the super– and subscripts depend on the available vertical space. The command `\chemical` takes this into account. When you want to put a chemical formula in a math formula —for example when you want to display an expression for a chemical equilibrium— there is the command `\ch`. This command has one argument and adapts automatically to its context: `$\frac{\ch{N}}{\ch{O}}$`

## 18.6 Math

We limit ourselves only to those commands that are available by default. In addition to the commands mentioned here, the math module implements many more:

```
\usemodule[math]
```

The extra commands are described in a separate manual.

Like in plain TEX we offer the next commands for switching to some specialized fonts:

| | | |
|---|---|---|
| `\frak` | fraktur | ABC |
| `\cal` | calligraphic | ABC |

Alternatively one can use the commands `\fraktur`, `\gothic` and `\calligraphic` which each take one argument, like in `\fraktur{TEXT}`.

These are typical fonts meant for math typesetting and special characters.

Fractions can occur quite often so we also added the command `\frac` on request: `$\frac{a}{b}$` results as expected $\frac{a}{b}$. This command adapts to its surroundings as good as possible.

For instructional purposes a frame or a background can be useful to indicate the specific math symbol. There is a special version of `\framed`: `\maframed`. We give some examples:

```
\startformula
  y + \maframed{y} + y^{2} + y^{\maframed{2}}
\stopformula

\nonknuthmode % todo: one day this can be removed

\startformula
  x \times \maframed{y} \times y^{\maframed{z}_{\maframed{z}}}
\stopformula
```

$$y + \boxed{y} + y^2 + y^{\boxed{2}}$$

$$x \times \boxed{y} \times y^{\boxed{z}_{\boxed{z}}}$$

To obtain a good spacing in framed math texts the `offset` equals `overlay`. The offset is produced by giving `frameoffset` an adequate value. Other setups are also possible:

```
\startformula
  x \times y^{\maframed[framecolor=red]{z}_z}
\stopformula
```

$$x \times y^{\boxed{z}_z}$$

For in–line math the command `\inmaframed` is available.

It is possible to typeset fractions without switching to math mode with the command:

```
\fraction {.1.} {.2.}

1   CONTENT
2   CONTENT
```

The braces are essential in the next example.

```
If \fraction{123}{456} equals \fraction{x}{y}, then \fraction{y}{x} equals
\fraction{456}{123}.
```

results in:

If $\frac{123}{456}$ equals $\frac{x}{y}$, then $\frac{y}{x}$ equals $\frac{456}{123}$.

## 18.7 Math collection

Math is a complicated matter and therefore we will not spend that many words on the gory details. For the user it is enough to know that you can mix different math fonts in a comfortable way and that ConTEXt will take care of the proper mapping on specific math fonts.

Because the wide range of math symbols can come from different fonts, math characters are organized into so called math collections. Normally such a collection is chosen automatically when you load a font definition, just as with font encodings. The ams math fonts extend the default math collection, which gives you a comfortable fall back. More information can be found in the documentation of the math module.

You can generate a list of the current math character set with the command `\showmathcharacters`.

command does not exist in mkiv

# 19 MetaPost

In a ConTEXt document we can use MetaPost code directly. For example:

```
\startMPgraphic
  fill unitsquare scaled 100 withcolor (.2,.3,.4) ;
\stopMPgraphic
```

A direct relation with the ConTEXt color mechanism is obvious:

```
\startMPgraphic
  fill unitsquare scaled 100 withcolor \MPcolor{mark} ;
\stopMPgraphic
```

MetaPost support is very extensive. You can store definitions and re–use them at random. If possible processed MetaPost pictures are re–used.

A detailed discussion on embedding MetaPost graphics is beyond this manual, and therefore will be covered elsewhere. For the moment it is enough to know the basics of putting for instance graphics in the background. In the next example, a graphic is calculated each time it is refered to:

```
\startuseMPgraphic{test a}
  fill unitsquare xscaled \overlaywidth yscaled \overlayheight ;
\stopuseMPgraphic

\defineoverlay[A Nice Rectangle][\useMPgraphic{test a}]

\setupbackgrounds[page][background=A Nice Rectangle]
```

When the graphic does not change, we can best reuse it, like:

```
\startreusableMPgraphic{test b}
  fill unitsquare xscaled \overlaywidth yscaled \overlayheight ;
\stopreusableMPgraphic

\defineoverlay[A Nice Rectangle][\reuseMPgraphic{test b}]

\setupbackgrounds[page][background=A Nice Rectangle]
```

When using the ConTEXt command line interface TEXexec, graphics are processed automatically. Unless one calls MetaPost at runtime, a second pass is needed to get the graphics in their final state.

# 20 Layers

**TODO:** All about layers

# 21 Interactive documents

**TODO:** This should explain the various interaction menus and the use of widgets / ECMAscript

# 22 Modules

**TODO:** What modules are and how to write them

# A Definitions

**FIXME:** No definitions can be placed

# B Index

The pagenumbers refer to the chapter or paragraph that describes the topic.

# C Commands

The pagenumbers refer to the chapter or paragraph that describes the command.

# D Distributed ConTEXt files

## D.1 Files in `tex/context/base`

| **filename(s)** | title | subtitle |
|---|---|---|
| `anch-bar.mkii` (mkiv) | ConTEXt Anchoring Macros | Margin Bars and alike |
| `anch-bck.mkvi` | ConTEXt Anchoring Macros | Backgrounds |
| `anch-pgr.lua` (mkii,mkiv) | ConTEXt Anchoring Macros | Positioning Graphics |
| `anch-pos.lua` (mkii,mkiv) | ConTEXt Anchoring Macros | Positioning Support |
| `anch-snc.mkii` (mkiv) | ConTEXt Anchoring Macros | Synchronization |
| `anch-tab.mkiv` | ConTEXt Anchoring Macros | Table Extensions |
| `attr-col.lua` (mkiv) | ConTEXt Attribute Macros | Color |
| `attr-eff.lua` (mkiv) | ConTEXt Attribute Macros | Effects |
| `attr-ini.lua` (mkiv) | ConTEXt Attribute Macros | Initialization |
| `attr-lay.lua` (mkiv) | ConTEXt Attribute Macros | Viewerlayers |
| `attr-mkr.lua` (mkiv) | ConTEXt Attribute Macros | Markers |
| `attr-neg.lua` (mkiv) | ConTEXt Attribute Macros | Negation |
| `back-exp.lua` (mkiv) | ConTEXt Backend Macros | XML export |
| `back-ini.lua` (mkiv) | ConTEXt Backend Macros | Initialization |
| `back-pdf.lua` (mkiv) | ConTEXt Backend Macros | pdf |
| `back-swf.mkiv` | ConTEXt Backend Macros | Shockwave Experiment |
| `back-u3d.mkiv` | ConTEXt Backend Macros | U3D Experiment |
| `bibl-bib.lua` (mkiv) | ConTEXt Bibliography Support | Initialization |
| `bibl-tra.lua` (mkii,mkiv) | ConTEXt Publication Module | Publications |
| `bibl-tst.lua` | | |
| `blob-ini.lua` (mkiv) | ConTEXt Lua Typesetting | Initialization |
| `buff-imp-default.lua` (mkiv) | ConTEXt Visualizer Macros | Default |
| `buff-imp-escaped.lua` (mkiv) | ConTEXt Visualizer Macros | Escaped |
| `buff-imp-lua.lua` (mkiv) | ConTEXt Visualizer Macros | Lua |
| `buff-imp-mp.lua` (mkiv) | ConTEXt Visualizer Macros | MetaPost |
| `buff-imp-nested.lua` (mkiv) | ConTEXt Visualizer Macros | Nested |
| `buff-imp-parsed-xml.lua` (mkiv) | ConTEXt Visualizer Macros | Parsed xml |
| `buff-imp-tex.lua` (mkiv) | ConTEXt Visualizer Macros | TEX |
| `buff-imp-xml.lua` (mkiv) | ConTEXt Visualizer Macros | xml |
| `buff-ini.lua` (mkii,mkiv) | ConTEXt Buffer Macros | Buffers |
| `buff-par.lua` (mkvi) | ConTEXt Buffer Macros | Parallel |
| `buff-ver.lua` (mkii,mkiv) | ConTEXt Buffer Macros | Verbatim |
| `bxml-apa.mkiv` | APA bibliography style | Publications |
| `catc-act.mkii` (mkiv) | ConTEXt Catcode Macros | Default Catcode Tables |
| `catc-ctx.mkii` (mkiv) | ConTEXt Catcode Macros | Extra Tables |
| `catc-def.mkii` (mkiv) | ConTEXt Catcode Macros | Default Tables |
| `catc-ini.lua` (mkii,mkiv) | ConTEXt System Macros | Catcode Handling |
| `catc-sym.mkii` (mkiv) | ConTEXt Catcode Macros | Some Handy Constants |
| `catc-xml.mkii` (mkiv) | ConTEXt Catcode Macros | xml Catcode Tables |
| `char-act.mkiv` | ConTEXt Character Support | Active |
| `char-def.lua` | companion to char-ini.mkiv | |
| `char-enc.lua` (mkiv) | ConTEXt Character Support | Encodings |
| `char-ent.lua` | companion to math-ini.mkiv | |
| `char-ini.lua` (mkiv) | ConTEXt Character Support | Initialization |
| `char-map.lua` | companion to char-ini.mkiv | |
| `char-tex.lua` | companion to char-ini.mkiv | |
| `char-utf.lua` (mkiv) | ConTEXt Character Support | Unicode UTF |
| `chem-ini.lua` (mkiv) | companion to chem-ini.mkiv | Chemistry |
| `chem-str.lua` (mkiv) | companion to chem-str.mkiv | Chemistry |

| | | |
|---|---|---|
| `cldf-bas.lua` (mkiv) | ConTEXt Lua Document Functions | Basics |
| `cldf-com.lua` (mkiv) | ConTEXt Lua Document Functions | Initialization |
| `cldf-ini.lua` (mkiv) | ConTEXt Lua Document Functions | Initialization |
| `cldf-int.lua` (mkiv) | ConTEXt Multilingual Macros | Initialization |
| `cldf-prs.lua` | companion to cldf-ini.mkiv | |
| `cldf-ver.lua` (mkiv) | ConTEXt Lua Document Functions | Verbatim |
| `colo-ema.mkii` | ConTEXt Color Macros | Emacs Colors |
| `colo-ext.mkii` (mkiv) | ConTEXt Color Macros | Extras |
| `colo-grp.mkiv` | ConTEXt Color Macros | Groups |
| `colo-hex.mkii` | ConTEXt Color Macros | Hex Colors |
| `colo-icc.lua` | companion to colo-ini.mkiv | |
| `colo-imp-dem.mkiv` | ConTEXt Color Macros | Demo Palets and Groups |
| `colo-imp-ema.mkiv` | ConTEXt Color Macros | Emacs Colors |
| `colo-imp-rgb.mkiv` | ConTEXt Color Macros | RGB |
| `colo-imp-x11.mkiv` | ConTEXt Color Macros | X11 |
| `colo-imp-xwi.mkiv` | ConTEXt Color Macros | X Windows |
| `colo-ini.lua` (mkii,mkiv) | ConTEXt Color Macros | Initialization |
| `colo-rgb.mkii` | ConTEXt Color Macros | RGB |
| `colo-run.lua` (mkii,mkiv) | ConTEXt Color Macros | Runtime loaded commands |
| `colo-x11.mkii` | ConTEXt Color Macros | X11 |
| `colo-xwi.mkii` | ConTEXt Color Macros | X Windows |
| `cont-cs.mkii` (mkiv) | ConTEXt | ConTEXt Czech Format Generation |
| `cont-de.mkii` (mkiv) | ConTEXt | ConTEXt German Format Generation |
| `cont-en.mkii` (mkiv) | ConTEXt | ConTEXt English Format Generation |
| `cont-err.mkii` | ConTEXt System Files | Just A warning |
| `cont-fil.mkii` (mkiv) | ConTEXt Miscellaneous Macros | File Synonyms |
| `cont-fr.mkii` (mkiv) | ConTEXt | ConTEXt French Format Generation |
| `cont-gb.mkii` (mkiv) | ConTEXt | ConTEXt English Format Generation |
| `cont-it.mkii` (mkiv) | ConTEXt | ConTEXt Italian Format Generation |
| `cont-log.mkii` (mkiv) | ConTEXt Miscellaneous Macros | TEX Logos |
| `cont-new.mkii` (mkiv) | ConTEXt Miscellaneous Macros | New Macros |
| `cont-nl.mkii` (mkiv) | ConTEXt | ConTEXt Dutch Format Generation |
| `cont-nop.mkiv` | ConTEXt Miscellaneous Macros | Startup Dummy |
| `cont-pe.mkiv` | ConTEXt | ConTEXt English Format Generation |
| `cont-ro.mkii` (mkiv) | ConTEXt | ConTEXt Romanian Format Generation |
| `cont-sys.ori` | ConTEXt Miscellaneous Macros | System Specific Setups |
| `cont-yes.mkiv` | ConTEXt Miscellaneous Macros | Startup Stub |
| `context.css` (lus,mkii,mkiv,rme) | ConTEXt | ConTEXt Format Generation |
| `context-base.lmx` | companion to mtx-server-ctx-startup.tex | |
| `context-characters.lmx` | companion to context.tex | |
| `context-debug.lmx` | companion to context.tex | |
| `context-error.lmx` | companion to context.tex | |
| `context-fonttest.lmx` | companion to mtx-server-ctx-fonttest.tex | |
| `context-help.lmx` | companion to comm-xml.tex | |
| `context-timing.lmx` | companion to mtx-timing.tex | |
| `context-version.pdf` (png) | | |
| `core-con.lua` (mkii,mkiv) | ConTEXt Core Macros | Conversion |
| `core-ctx.ctx` (lua,mkii,mkiv) | ConTEXt Core Macros | Job Control |
| `core-dat.lua` (mkiv) | ConTEXt Core Macros | Multipass Datasets |
| `core-def.mkii` (mkiv) | ConTEXt Core Macros | Defaults |
| `core-env.lua` (mkii,mkiv) | ConTEXt Core Macros | New ones |
| `core-fil.mkii` | ConTEXt Core Macros | File Support |
| `core-fnt.mkii` | ConTEXt Core Macros | Fonts |
| `core-gen.mkii` | ConTEXt Core Macros | General |
| `core-ini.mkii` (mkiv) | ConTEXt Core Macros | Additional Initialization |

| | | |
|---|---|---|
| core-job.mkii | ConTEXt Core Macros | Job Handling |
| core-mis.mkii | ConTEXt Core Macros | Miscelaneous |
| core-par.mkii | ConTEXt Core Macros | Paragraph Tricks |
| core-stg.mkii | ConTEXt Core Macros | Strategies |
| core-sys.lua (mkii,mkiv) | ConTEXt Core Macros | System |
| core-two.lua (mkii,mkiv) | ConTEXt Core Macros | Two Pass Data |
| core-uti.lua (mkii,mkiv) | ConTEXt Core Macros | Utility File Handling |
| core-var.mkii | ConTEXt Core Macros | Variables |
| data-aux.lua | companion to luat-lib.mkiv | |
| data-bin.lua | companion to luat-lib.mkiv | |
| data-con.lua | companion to luat-lib.mkiv | |
| data-crl.lua | companion to luat-lib.mkiv | |
| data-ctx.lua | companion to luat-lib.mkiv | |
| data-env.lua | companion to luat-lib.mkiv | |
| data-exp.lua | companion to luat-lib.mkiv | |
| data-fil.lua | companion to luat-lib.mkiv | |
| data-gen.lua | companion to luat-lib.mkiv | |
| data-ini.lua | companion to luat-lib.mkiv | |
| data-inp.lua | companion to luat-lib.mkiv | |
| data-lst.lua | companion to luat-lib.mkiv | |
| data-lua.lua | companion to luat-lib.mkiv | |
| data-met.lua | companion to luat-lib.mkiv | |
| data-out.lua | companion to luat-lib.mkiv | |
| data-pre.lua | companion to luat-lib.mkiv | |
| data-res.lua | companion to luat-lib.mkiv | |
| data-sch.lua | companion to luat-lib.mkiv | |
| data-tex.lua | companion to luat-lib.mkiv | |
| data-tmf.lua | companion to luat-lib.mkiv | |
| data-tmp.lua | companion to luat-lib.mkiv | |
| data-tre.lua | companion to luat-lib.mkiv | |
| data-use.lua | companion to luat-lib.mkiv | |
| data-vir.lua | companion to luat-lib.mkiv | |
| data-zip.lua | companion to luat-lib.mkiv | |
| enco-032.mkii | ConTEXt Encoding Macros | Unicode Goodies |
| enco-037.mkii | ConTEXt Unicode Macros | Encoding for vector 37 |
| enco-acc.mkii | ConTEXt Encoding Macros | Composed Characters Commands |
| enco-agr.mkii | ConTEXt Unicode Macros | Ancient Greek |
| enco-ans.mkii | ConTEXt Encoding Macros | y&y texnansi Encoding |
| enco-cas.mkii | ConTEXt Encoding Macros | Named Glyph Case Mapping |
| enco-chi.mkii | ConTEXt Encoding Macros | Traditional and Simplified Chinese |
| enco-com.mkii | ConTEXt Encoding Macros | Composed Characters Commands |
| enco-cyr.mkii | ConTEXt Encoding Macros | Cyrillic |
| enco-def.mkii | ConTEXt Encoding Macros | Default Character Definitions |
| enco-ec.mkii | ConTEXt Encoding Macros | LaTEX EC Encoding |
| enco-ecm.mkii | ConTEXt Encoding Macros | Glyphs that may not be present in EC |
| enco-el.mkii | ConTEXt Encoding Macros | EuroLetter |
| enco-fde.mkii | ConTEXt Encoding Macros | German Input Filter |
| enco-ffr.mkii | ConTEXt Encoding Macros | French Input Filter |
| enco-fpl.mkii | ConTEXt Encoding Macros | Polish Input Filter |
| enco-fro.mkii | ConTEXt Encoding Macros | Romanian Input Filter |
| enco-fsl.mkii | ConTEXt Encoding Macros | Slovenian Specialities |
| enco-grk.mkii | ConTEXt Encoding Macros | Greek |
| enco-heb.mkii | ConTEXt Encoding Macros | Hebrew |
| enco-ibm.mkii | | |
| enco-il2.mkii | ConTEXt Encoding Macros | Czech and Slovak ISO Latin 2 Encoding |
| enco-ini.mkii (mkiv) | ConTEXt Encoding Macros | Initialization |
| enco-l7x.mkii | ConTEXt Encoding Macros | LaTEX L7x Encoding |

| | | |
|---|---|---|
| `enco-lat.mkii` | | |
| `enco-mis.mkii` | ConTEXt Encoding Macros | Missing Glyphs |
| `enco-pdf.mkii` | ConTEXt Encoding Macros | y&y texnansi Encoding |
| `enco-pfr.mkii` | ConTEXt Encoding Macros | PDF Resources |
| `enco-pol.mkii` | ConTEXt Encoding Macros | Polish Mixed Encoding |
| `enco-qx.mkii` | ConTEXt Encoding Macros | Polish QX Encoding |
| `enco-raw.mkii` | | |
| `enco-run.mkii` | ConTEXt Encoding Macros | Runtime Macros |
| `enco-t5.mkii` | ConTEXt Encoding Macros | New Vietnamese Encoding |
| `enco-tbo.mkii` | ConTEXt Encoding Macros | TeXBaseOne Encoding |
| `enco-uc.mkii` | ConTEXt Encoding Macros | Unicode (backwards mapping) |
| `enco-vis.mkii` | | |
| `enco-vna.mkii` | ConTEXt Encoding Macros | Vietnamese Accents |
| `enco-win.mkii` | | |
| `enco-x5.mkii` | ConTEXt Encoding Macros | Vietnamese Encoding |
| `export-example.tex (css,rng)` | companion to context.mkiv | |
| `file-ini.lua (mkvi)` | ConTEXt File Macros | Helpers |
| `file-job.lua (mkvi)` | ConTEXt Core Macros | Job Handling |
| `file-lib.lua (mkvi)` | ConTEXt File Macros | Module Support |
| `file-mod.lua (mkvi)` | ConTEXt File Macros | Module Support |
| `file-res.lua (mkvi)` | ConTEXt File Macros | Resolvers |
| `file-syn.lua (mkvi)` | ConTEXt File Macros | Module Support |
| `filt-bas.mkii` | ConTEXt Filter Macros | A Base Collection |
| `filt-ini.mkii` | ConTEXt Filter Macros | Initialization |
| `font-afk.lua` | companion to font-afm.lua | |
| `font-afm.lua` | companion to font-ini.mkiv | |
| `font-age.lua` | companion to luatex-fonts.lua | |
| `font-agl.lua` | companion to font-ini.mkiv | |
| `font-arb.mkii` | | |
| `font-aux.lua (mkvi)` | ConTEXt Font Support | Helpers |
| `font-bfm.mkii` | ConTEXt Font Macros | Mixed Normal and Bold Math |
| `font-chi.mkii` | ConTEXt Font Macros | Chinese |
| `font-chk.lua (mkiv)` | ConTEXt Font Macros | Checking |
| `font-cid.lua` | companion to font-otf.lua (cidmaps) | |
| `font-col.lua (mkvi)` | ConTEXt Font Macros | Fallbacks (collections) |
| `font-con.lua` | companion to font-ini.mkiv | |
| `font-ctx.lua` | companion to font-ini.mkiv | |
| `font-def.lua` | companion to font-ini.mkiv | |
| `font-emp.mkvi` | ConTEXt Font Macros | Emphasis |
| `font-enc.lua` | companion to font-ini.mkiv | |
| `font-enh.lua` | companion to font-ini.mkiv | |
| `font-ext.lua` | companion to font-ini.mkiv and hand-ini.mkiv | |
| `font-fbk.lua` | companion to font-ini.mkiv | |
| `font-fea.mkvi` | ConTEXt Font Macros | features |
| `font-fil.mkvi` | ConTEXt Font Macros | Classes and Files |
| `font-gds.lua (mkvi)` | ConTEXt Font Support | Colorschemes |
| `font-heb.mkii` | | |
| `font-hsh.lua` | companion to font-ini.mkiv | |
| `font-ini.lua (mkii,mkvi)` | ConTEXt Font Macros | Initialization |
| `font-jap.mkii` | ConTEXt Font Macros | Japanese |
| `font-ldr.lua` | companion to font-ini.mkiv | |
| `font-lib.mkvi` | ConTEXt Font Macros | Libraries |
| `font-log.lua` | companion to font-ini.mkiv | |
| `font-lua.lua` | companion to font-ini.mkiv | |
| `font-map.lua` | companion to font-ini.mkiv | |

| | | |
|---|---|---|
| `font-mat.mkvi` | ConTEXt Font Macros | Math |
| `font-mis.lua` | companion to mtx-fonts | |
| `font-nod.lua` | companion to font-ini.mkiv | |
| `font-odk.lua` | | |
| `font-odv.lua` | companion to font-ini.mkiv | |
| `font-ota.lua` | companion to font-otf.lua (analysing) | |
| `font-otb.lua` | companion to font-ini.mkiv | |
| `font-otc.lua` | companion to font-otf.lua (context) | |
| `font-otd.lua` | companion to font-ini.mkiv | |
| `font-otf.lua` | companion to font-ini.mkiv | |
| `font-oth.lua` | companion to font-oth.lua (helpers) | |
| `font-oti.lua` | companion to font-ini.mkiv | |
| `font-otn.lua` | companion to font-ini.mkiv | |
| `font-otp.lua` | companion to font-otf.lua (packing) | |
| `font-ott.lua` | companion to font-otf.lua (tables) | |
| `font-otx.lua` | companion to font-otf.lua (analysing) | |
| `font-pat.lua` | companion to font-ini.mkiv | |
| `font-pre.mkiv` | ConTEXt Font Macros | Predefined |
| `font-run.mkii` (mkiv) | ConTEXt Font Macros | Runtime Macros |
| `font-set.mkvi` | ConTEXt Font Macros | Initial Loading |
| `font-sol.lua` (mkvi) | ConTEXt Font Macros | Solutions |
| `font-sty.mkvi` | ConTEXt Font Macros | Styles |
| `font-sym.mkvi` | ConTEXt Font Macros | Symbolic Access |
| `font-syn.lua` | companion to font-ini.mkiv | |
| `font-tfm.lua` | companion to font-ini.mkiv | |
| `font-tra.mkiv` | ConTEXt Font Macros | Tracing |
| `font-trt.lua` | companion to font-ini.mkiv | |
| `font-uni.mkii` (mkiv) | ConTEXt Font Macros | Unicode |
| `font-unk.mkii` (mkiv) | ConTEXt Font Macros | Unknown Defaults |
| `font-var.mkvi` | ConTEXt Font Macros | Common Variables |
| `font-vf.lua` | companion to font-ini.mkiv | |
| `font-xtx.mkii` | ConTEXt Font Macros | XƎTEX Hacks |
| `grph-epd.lua` (mkiv) | ConTEXt Graphic Macros | Merging Goodies |
| `grph-fig.mkii` (mkiv) | ConTEXt Graphic Macros | Figure Inclusion |
| `grph-fil.lua` | companion to grph-fig.mkiv | |
| `grph-inc.lua` (mkii,mkiv) | ConTEXt Graphic Macros | Figure Inclusion |
| `grph-raw.lua` (mkiv) | ConTEXt Graphic Macros | Raw Bitmaps |
| `grph-swf.lua` | companion to grph-inc.mkiv | |
| `grph-trf.mkii` (mkiv) | ConTEXt Graphic Macros | Transformations |
| `grph-u3d.lua` | companion to grph-inc.mkiv | |
| `grph-wnd.lua` | companion to grph-inc.mkiv | |
| `hand-def.mkii` | ConTEXt Handling Macros | Default Protruding Factors |
| `hand-ini.mkii` (mkiv) | ConTEXt Handling Macros | Initialization |
| `java-ans.mkii` | ConTEXt JavaScript Macros | Answer Analization |
| `java-exa.mkii` | ConTEXt JavaScript Macros | Example Support |
| `java-fil.mkii` | ConTEXt JavaScript Macros | Filing and Printing |
| `java-fld.mkii` | ConTEXt JavaScript Macros | Field Support |
| `java-imp-exa.mkiv` | ConTEXt JavaScript Macros | Example Support |
| `java-imp-fil.mkiv` | ConTEXt JavaScript Macros | Filing and Printing |
| `java-imp-fld.mkiv` | ConTEXt JavaScript Macros | Field Support |
| `java-imp-rhh.mkiv` | ConTEXt JavaScript Macros | Runtime Highlight Hack |
| `java-imp-stp.mkiv` | ConTEXt JavaScript Macros | Stepping |
| `java-ini.lua` (mkii,mkiv) | ConTEXt JavaScript Macros | Initialization |

| | | |
|---|---|---|
| `java-stp.mkii` | ConTEXt JavaScript Macros | Stepping |
| `l-boolean.lua` | companion to luat-lib.mkiv | |
| `l-dir.lua` | companion to luat-lib.mkiv | |
| `l-file.lua` | companion to luat-lib.mkiv | |
| `l-function.lua` | companion to luat-lib.mkiv | |
| `l-gzip.lua` | | |
| `l-io.lua` | companion to luat-lib.mkiv | |
| `l-lpeg.lua` | companion to luat-lib.mkiv | |
| `l-lua.lua` | companion to luat-lib.mkiv | |
| `l-math.lua` | companion to luat-lib.mkiv | |
| `l-md5.lua` | | |
| `l-number.lua` | companion to luat-lib.mkiv | |
| `l-os.lua` | companion to luat-lib.mkiv | |
| `l-package.lua` | companion to luat-lib.mkiv | |
| `l-pdfview.lua` | companion to mtx-context.lua | |
| `l-set.lua` | companion to luat-lib.mkiv | |
| `l-string.lua` | companion to luat-lib.mkiv | |
| `l-table.lua` | companion to luat-lib.mkiv | |
| `l-unicode.lua` | companion to luat-lib.mkiv | |
| `l-url.lua` | companion to luat-lib.mkiv | |
| `l-xml.lua` | this module is replaced by the lxml-* ones | |
| `lang-all.xml` | | |
| `lang-alt.mkii` | ConTEXt Language Macros | Altaic Languages |
| `lang-ana.mkii` | ConTEXt Language Macros | Anatolian Languages |
| `lang-art.mkii` | ConTEXt Language Macros | Artificial Languages |
| `lang-bal.mkii` | ConTEXt Language Macros | Baltic Languages |
| `lang-cel.mkii` | ConTEXt Language Macros | Celtic Languages |
| `lang-chi.mkii` | ConTEXt Language Macros | Chinese |
| `lang-ctx.mkii` | ConTEXt Language Macros | Generic Patterns |
| `lang-cyr.mkii` | ConTEXt Language Macros | Cyrillic Languages |
| `lang-def.lua` (mkiv) | ConTEXt Language Macros | Languages Definitions |
| `lang-dis.mkii` | ConTEXt Language Macros | Distribution Patterns |
| `lang-frd.mkii` (mkiv) | ConTEXt Language Macros | Language Frequency Table Data |
| `lang-frq.mkii` (mkiv) | ConTEXt Language Macros | Frequency Tables |
| `lang-frq-de.lua` | | |
| `lang-frq-en.lua` | | |
| `lang-frq-nl.lua` | | |
| `lang-ger.mkii` | ConTEXt Language Macros | Germanic Languages |
| `lang-grk.mkii` | ConTEXt Language Macros | Uralic Languages |
| `lang-ind.mkii` | ConTEXt Language Macros | Indo Iranian Languages |
| `lang-ini.lua` (mkii,mkiv) | ConTEXt Language Macros | Initialization |
| `lang-ita.mkii` | ConTEXt Language Macros | Italic Languages |
| `lang-jap.mkii` | ConTEXt Language Macros | Japanese |
| `lang-lab.lua` (mkii,mkiv) | ConTEXt Language Macros | Labels |
| `lang-mis.mkii` (mkiv) | ConTEXt Language Macros | Compounds |
| `lang-rep.lua` | companion to lang-rep.mkiv | |
| `lang-run.mkii` | ConTEXt Language Macros | Runtime Macros |
| `lang-sla.mkii` | ConTEXt Language Macros | Slavic Languages |
| `lang-spa.mkii` (mkiv) | ConTEXt Language Macros | Spacing |
| `lang-spe.mkii` | ConTEXt Language Macros | Specifics |
| `lang-txt.lua` | companion to lang-lab.mkiv | |
| `lang-ura.mkii` | ConTEXt Language Macros | Uralic Languages |
| `lang-url.lua` (mkii,mkiv) | ConTEXt Language Macros | Language Options |
| `lang-vn.mkii` | ConTEXt Language Macros | Vietnamese |
| `lang-wrd.lua` (mkiv) | ConTEXt Language Macros | Checking |
| `layo-ini.lua` (mkiv) | ConTEXt Layout Macros | Initialization |

| | | |
|---|---|---|
| lpdf-ano.lua | companion to lpdf-ini.mkiv | |
| lpdf-col.lua | companion to lpdf-ini.mkiv | |
| lpdf-enc.lua | companion to lpdf-ini.mkiv | |
| lpdf-epa.lua | companion to lpdf-epa.mkiv | |
| lpdf-epd.lua | companion to lpdf-epa.mkiv | |
| lpdf-fld.lua | companion to lpdf-ini.mkiv | |
| lpdf-fmt.lua | companion to lpdf-ini.mkiv | |
| lpdf-grp.lua | companion to lpdf-ini.mkiv | |
| lpdf-ini.lua | companion to lpdf-ini.mkiv | |
| lpdf-mis.lua | companion to lpdf-ini.mkiv | |
| lpdf-mov.lua | companion to lpdf-ini.mkiv | |
| lpdf-nod.lua | companion to lpdf-ini.mkiv | |
| lpdf-pda.xml | | |
| lpdf-pdx.xml | | |
| lpdf-ren.lua | companion to lpdf-ini.mkiv | |
| lpdf-swf.lua | companion to lpdf-ini.mkiv | |
| lpdf-tag.lua | companion to lpdf-tag.mkiv | |
| lpdf-u3d.lua | companion to lpdf-ini.mkiv | |
| lpdf-wid.lua | companion to lpdf-ini.mkiv | |
| lpdf-xmp.lua | with help from Peter Rolf | |
| luat-bas.mkiv | ConTEXt Lua Macros | Basic Lua Libraries |
| luat-bwc.lua | companion to luat-lib.mkiv | |
| luat-cbk.lua | companion to luat-lib.mkiv | |
| luat-cnf.lua | companion to luat-lib.mkiv | |
| luat-cod.lua (mkiv) | ConTEXt Lua Macros | Code |
| luat-env.lua | companion to luat-lib.mkiv | |
| luat-exe.lua | companion to luat-lib.mkiv | |
| luat-fio.lua | companion to luat-lib.mkiv | |
| luat-fmt.lua | companion to mtxrun | |
| luat-ini.lua (mkiv) | ConTEXt Lua Macros | Initialization |
| luat-iop.lua | companion to luat-lib.mkiv | |
| luat-lib.mkiv | ConTEXt Lua Macros | Libraries |
| luat-lua.lua | companion to luat-lib.mkiv | |
| luat-mac.lua | companion to luat-lib.mkiv | |
| luat-run.lua | companion to luat-lib.mkiv | |
| luat-soc.lua | | |
| luat-sta.lua | | |
| luat-sto.lua | companion to luat-lib.mkiv | |
| lxml-aux.lua | this module is the basis for the lxml-* ones | |
| lxml-css.lua (mkiv) | ConTEXt Modules | Css Helpers |
| lxml-ctx.lua (mkiv) | ConTEXt xml Support | Initialization |
| lxml-dir.lua | this module is the basis for the lxml-* ones | |
| lxml-ent.lua | this module is the basis for the lxml-* ones | |
| lxml-inf.lua | this module is the basis for the lxml-* ones | |
| lxml-ini.mkiv | ConTEXt xml Support | Initialization |
| lxml-lpt.lua | this module is the basis for the lxml-* ones | |
| lxml-mis.lua | this module is the basis for the lxml-* ones | |
| lxml-sor.lua (mkiv) | ConTEXt xml Support | Sorting |
| lxml-tab.lua | this module is the basis for the lxml-* ones | |
| lxml-tex.lua | companion to lxml-ini.mkiv | |

| | | |
|---|---|---|
| `lxml-xml.lua` | this module is the basis for the lxml-* ones | |
| `m-arabtex.mkii` | ConTEXt Modules | Arabic |
| `m-barcodes.mkiv` | ConTEXt Extra Modules | Barcodes |
| `m-chart.lua` (mkii,mkvi) | ConTEXt Modules | Flow Charts |
| `m-chemic.mkii` (mkiv) | ConTEXt Extra Modules | ppchTEX (Plain Pictex Context cHemie TEX) |
| `m-cweb.tex` | ConTEXt Extra Modules | cweb Pretty Printing Macros |
| `m-database.lua` (mkii,mkiv) | ConTEXt Modules | Database Thingies |
| `m-datastrc.tex` | ConTEXt Modules | Database Support |
| `m-directives.mkiv` | | |
| `m-dratex.mkii` | ConTEXt Extra Modules | DraTEX Loading Macros |
| `m-edtsnc.mkii` | ConTEXt Modules | Editor Synchronization |
| `m-educat.tex` | ConTEXt Extra Modules | Educational Extras |
| `m-fields.mkiv` | ConTEXt Extra Modules | Fields |
| `m-format.tex` | ConTEXt Modules | Ancient Formatting Code |
| `m-graph.mkii` (mkiv) | ConTEXt Extra Modules | MetaPost graph module support |
| `m-hemistich.mkiv` | ConTEXt Extra Modules | Hemistiches |
| `m-ipsum.mkiv` | ConTEXt Extra Modules | Ipsum |
| `m-json.mkiv` | ConTEXt Modules | Json |
| `m-layout.tex` | ConTEXt Modules | Additional Layouts |
| `m-level.mkii` | ConTEXt Extra Modules | Catching Nesting Errors |
| `m-logcategories.mkiv` | | |
| `m-markdown.lua` (mkiv) | ConTEXt Modules | Processing MarkDown |
| `m-mathcrap.mkiv` | ConTEXt Modules | Math Crap |
| `m-mkii.mkiv` | | |
| `m-mkivhacks.mkiv` | ConTEXt Modules | Temporary Compatilibility Hacks |
| `m-morse.mkvi` | ConTEXt Extra Modules | Morse |
| `m-narrowtt.tex` | ConTEXt Modules | Narrow Verbatim |
| `m-newmat.tex` | ConTEXt Math Module | AMS-like math extensions |
| `m-nodechart.lua` (mkvi) | ConTEXt Modules | Node Visualization |
| `m-ntb-to-xtb.mkiv` | | |
| `m-obsolete.mkii` (mkiv) | | |
| `m-oldfun.mkiv` | ConTEXt Support Macros | Fun Stuff |
| `m-oldnum.mkiv` | ConTEXt Support Macros | Numbers |
| `m-pdfsnc.mkii` | ConTEXt Modules | Editor Synchronization |
| `m-pictex.tex` | ConTEXt Extra Modules | PICTEX Loading Macros |
| `m-pstricks.lua` (mkii,mkiv) | ConTEXt Extra Modules | pstricks Connections |
| `m-punk.mkiv` | ConTEXt Modules | Punk Support |
| `m-r.mkii` | ConTEXt Modules | R Support |
| `m-spreadsheet.lua` (mkiv) | ConTEXt Extra Modules | Spreadsheets |
| `m-sql.mkiv` | ConTEXt Extra Modules | SQL |
| `m-steps.lua` (mkii,mkvi) | ConTEXt Modules | Step Charts & Tables |
| `m-streams.tex` | ConTEXt Modules | Streams |
| `m-subsub.tex` | ConTEXt Private Modules | More Section Levels |
| `m-tex4ht.mkii` | | |
| `m-timing.mkiv` | ConTEXt Modules | Timing |
| `m-trackers.mkiv` | | |
| `m-translate.mkiv` | ConTEXt Modules | Translations |
| `m-units.mkii` (mkiv) | ConTEXt Extra Modules | Scientific Units |
| `m-visual.mkii` (mkiv) | ConTEXt Extra Modules | Visualization and Faking |
| `m-zint.mkiv` | ConTEXt Extra Modules | Zint Barcode Generator |
| `math-acc.mkvi` | ConTEXt Math Macros | Accents |
| `math-act.lua` | companion to math-ini.mkiv | |
| `math-ali.mkiv` | ConTEXt Math Macros | Math Alignments |
| `math-ams.mkii` | ConTEXt Math Macros | AMS Specials |
| `math-arr.mkii` (mkiv) | ConTEXt Math Macros | Arrows |

| | | |
|---|---|---|
| math-cow.mkii | ConTEXt Math Macros | Cow Math |
| math-def.mkiv | ConTEXt Math Macros | Definitions |
| math-del.mkiv | ConTEXt Math Macros | Delimiters |
| math-dim.lua | companion to math-ini.mkiv | |
| math-dir.lua | companion to typo-dir.mkiv | |
| math-dis.mkiv | ConTEXt Math Macros | Display |
| math-eul.mkii | ConTEXt Math Macros | Virtual Euler Specials |
| math-ext.lua | companion to math-ini.mkiv | |
| math-fbk.lua | companion to math-ini.mkiv | |
| math-fen.mkiv | ConTEXt Math Macros | Fences |
| math-for.mkiv | ConTEXt Structure Macros | Math Numbering |
| math-fou.mkii | ConTEXt Math Macros | Fourier Specials |
| math-frc.lua (mkii,mkiv) | ConTEXt Math Macros | Fractions |
| math-ini.lua (mkii,mkiv) | ConTEXt Math Macros | Initializations |
| math-inl.mkiv | ConTEXt Math Macros | Inline |
| math-int.mkiv | ConTEXt Math Macros | Scripts |
| math-lbr.mkii | ConTEXt Math Macros | Lucida Specials |
| math-map.lua | companion to math-ini.mkiv | |
| math-mis.mkiv | ConTEXt Math Macros | Miscellaneous |
| math-noa.lua | companion to math-ini.mkiv | |
| math-pln.mkii (mkiv) | ConTEXt Math Macros | Plain Helpers |
| math-rad.mkvi | ConTEXt Math Macros | Radicals |
| math-ren.lua | companion to math-ren.mkiv | |
| math-run.mkii | ConTEXt Math Macros | Runtime Macros |
| math-scr.mkiv | ConTEXt Math Macros | Scripts |
| math-stc.mkvi | ConTEXt Math Macros | Stackers |
| math-tag.lua | companion to math-ini.mkiv | |
| math-tex.mkii | | Plain Specials |
| math-tim.mkii | ConTEXt Math Macros | Mathtime Specials |
| math-ttv.lua | traditional tex vectors, companion to math-vfu.lua | |
| math-uni.mkii | ConTEXt Math Macros | unicode support |
| math-vfu.lua | companion to math-ini.mkiv | |
| meta-clp.mkii | MetaPost Graphics | Clipping |
| meta-dum.mkii | MetaPost Graphics | Dummy (External) Graphics |
| meta-fig.mkii (mkiv) | MetaPost Graphics | Stand Alone Graphics |
| meta-fnt.lua (mkiv) | MetaPost Graphics | Fonts |
| meta-fun.lua (mkiv) | MetaPost Graphics | Goodies |
| meta-grd.mkiv | MetaPost Graphics | grids |
| meta-imp-clp.mkiv | MetaPost Graphics | Clipping |
| meta-imp-dum.mkiv | MetaPost Graphics | Dummy (External) Graphics |
| meta-imp-fen.mkiv | MetaPost Graphics | Fences |
| meta-imp-mis.mkiv | MetaPost Graphics | Misc Test Graphics |
| meta-imp-nav.mkiv | MetaPost Graphics | Navigational Graphics |
| meta-imp-pre.mkiv | MetaPost Graphics | Predefined Goodies |
| meta-imp-txt.mkiv | MetaPost Graphics | Text Tricks |
| meta-ini.lua (mkii,mkiv) | MetaPost Graphics | Initialization |
| meta-mis.mkii | MetaPost Graphics | Misc Test Graphics |
| meta-nav.mkii | MetaPost Graphics | Navigational Graphics |
| meta-pag.mkii (mkiv) | MetaPost Graphics | Initialization |
| meta-pdf.lua (mkii,mkiv) | MetaPost Graphics | Conversion to pdf |
| meta-pdh.lua (mkiv) | MetaPost Graphics | Conversion to pdf |
| meta-pre.mkii | MetaPost Graphics | Predefined Goodies |
| meta-tex.lua (mkii,mkiv) | ConTEXt Support Macros | MetaPost fast text insertion |
| meta-txt.mkii | MetaPost Graphics | Text Tricks |
| meta-xml.mkii (mkiv) | MetaPost Graphics | XML Hacks |
| metatex.tex (lus) | MetaTEX | MetaTEX Format Generation |

| | | |
|---|---|---|
| mlib-ctx.lua (mkiv) | MetaPost Integrated Graphics | Basics |
| mlib-pdf.lua (mkiv) | MetaPost Integrated Graphics | Conversion to PDF |
| mlib-pps.lua (mkiv) | MetaPost Integrated Graphics | Basics |
| mlib-run.lua | companion to mlib-ctx.mkiv | |
| mtx-context-arrange.tex | ConTEXt Extra Trickry | Arrange Files |
| mtx-context-combine.tex | ConTEXt Extra Trickry | Combine Files |
| mtx-context-common.tex | ConTEXt Extra Trickry | Common Stuff |
| mtx-context-copy.tex | ConTEXt Extra Trickry | Copying Files |
| mtx-context-ideas.tex | ConTEXt Extra Trickry | Placeholder File |
| mtx-context-listing.tex | ConTEXt Extra Trickry | Listing Files |
| mtx-context-markdown.tex | ConTEXt Extra Trickry | Rendering Markdown Files |
| mtx-context-select.tex | ConTEXt Extra Trickry | Selecting Files |
| mtx-context-sql.tex | ConTEXt Extra Trickry | SQL Tables |
| mtx-context-timing.tex | ConTEXt Extra Trickry | Timing Runs |
| mtx-context-xml.tex | ConTEXt Extra Trickry | Analyzing XML files |
| mult-aux.lua (mkii,mkiv) | ConTEXt Multilingual Macros | helpers |
| mult-chk.lua (mkii,mkiv) | ConTEXt Multilingual Macros | Checking |
| mult-com.mkii | ConTEXt Multilingual Macros | Commands |
| mult-con.mkii | ConTEXt Multilingual Macros | Constants |
| mult-de.mkii | | |
| mult-def.lua (mkii,mkiv) | ConTEXt Multilingual Macros | Definitions |
| mult-dim.mkvi | ConTEXt Core Macros | General |
| mult-en.mkii | | |
| mult-fr.mkii | | |
| mult-fst.mkii | ConTEXt Multilingual Macros | Speed Up |
| mult-fun.lua | | |
| mult-ini.lua (mkii,mkiv) | ConTEXt Multilingual Macros | Initialization |
| mult-it.mkii | | |
| mult-low.lua | companion to mult-ini.mkiv | |
| mult-mcs.mkii | | |
| mult-mde.mkii | | |
| mult-men.mkii | | |
| mult-mes.lua | companion to mult-ini.mkiv | |
| mult-mfr.mkii | | |
| mult-mit.mkii | | |
| mult-mnl.mkii | | |
| mult-mno.mkii | | |
| mult-mpe.mkii | | |
| mult-mps.lua | | |
| mult-mro.mkii | | |
| mult-nl.mkii | | |
| mult-pe.mkii | | |
| mult-prm.lua (mkiv) | ConTEXt Multilingual Macros | Primitives |
| mult-ro.mkii | | |
| mult-sys.mkii (mkiv) | ConTEXt Multilingual Macros | System |
| node-acc.lua | companion to node-ini.mkiv | |
| node-aux.lua | companion to node-ini.mkiv | |
| node-bck.lua (mkiv) | ConTEXt Node Macros | Backgrounds |
| node-dir.lua | companion to node-ini.mkiv | |
| node-ext.lua | companion to node-ini.mkiv | |
| node-fin.lua (mkiv) | ConTEXt Node Macros | Finalizing |
| node-fnt.lua | companion to font-ini.mkiv | |
| node-ini.lua (mkiv) | ConTEXt Node Macros | Initialization |
| node-inj.lua | companion to node-ini.mkiv | |
| node-ltp.lua | a translation of the built in parbuilder, initial convertsin by Taco Hoekwater | |

| | | |
|---|---|---|
| `node-met.lua` | companion to node-ini.mkiv | |
| `node-mig.lua (mkiv)` | ConTEXt Node Macros | Inserts |
| `node-pag.lua (mkiv)` | ConTEXt Node Macros | Page Building |
| `node-pro.lua` | companion to node-ini.mkiv | |
| `node-ref.lua` | companion to node-ref.mkiv | |
| `node-res.lua` | companion to node-ini.mkiv | |
| `node-rul.lua (mkiv)` | ConTEXt Core Macros | Bars |
| `node-ser.lua` | companion to node-ini.mkiv | |
| `node-shp.lua` | companion to node-ini.mkiv | |
| `node-snp.lua` | companion to node-ini.mkiv | |
| `node-tex.lua` | companion to node-ini.mkiv | |
| `node-tra.lua` | companion to node-ini.mkiv | |
| `node-tsk.lua` | companion to node-ini.mkiv | |
| `node-tst.lua` | companion to node-ini.mkiv | |
| `node-typ.lua` | companion to node-ini.mkiv | |
| `norm-alo.mkii` | ConTEXt Norm Macros | Aleph and Omega |
| `norm-ctx.mkii (mkiv)` | ConTEXt Norm Macros | Aleph and Omega |
| `norm-etx.mkii` | ConTEXt Norm Macros | ε-TEX |
| `norm-ltx.mkii` | ConTEXt Norm Macros | LuaTEX |
| `norm-ptx.mkii` | ConTEXt Norm Macros | pdfTEX |
| `norm-tex.mkii` | ConTEXt Norm Macros | TEX |
| `norm-xtx.mkii` | ConTEXt Norm Macros | XƎTEX |
| `pack-bar.mkiv` | ConTEXt Packaging Macros | Bars |
| `pack-bck.mkvi` | ConTEXt Packaging Macros | Simple Backgrounds |
| `pack-box.mkii (mkiv)` | ConTEXt Packaging Macros | Boxes |
| `pack-com.mkiv` | ConTEXt Packing Macros | Combinations |
| `pack-cut.mkiv` | ConTEXt Packaging Macros | Cut boxes |
| `pack-fen.mkiv` | ConTEXt Packaging Macros | Fences for Ruled Content |
| `pack-lyr.mkii (mkiv)` | ConTEXt Packaging Macros | Layers |
| `pack-mis.mkvi` | ConTEXt Core Macros | Miscelaneous |
| `pack-mrl.mkiv` | ConTEXt Packaging Macros | More Rules |
| `pack-obj.lua (mkii,mkiv)` | ConTEXt Packaging Macros | Objects |
| `pack-pos.mkiv` | ConTEXt Packaging Macros | Positioning |
| `pack-rul.lua (mkii,mkiv)` | ConTEXt Packaging Macros | Ruled Content |
| `page-app.mkii (mkiv)` | ConTEXt Page Macros | Independent page building |
| `page-bck.mkii (mkiv)` | ConTEXt Page Macros | Backgrounds |
| `page-box.mkvi` | ConTEXt Page Macros | Page Boxing |
| `page-brk.mkiv` | ConTEXt Page Macros | Breaks |
| `page-col.mkiv` | ConTEXt Page Macros | Column Helpers |
| `page-com.mkiv` | ConTEXt Page Macros | Page Comments |
| `page-fac.mkiv` | ConTEXt Page Macros | Facing Pages |
| `page-flt.lua (mkiv)` | ConTEXt Page Macros | Float Management |
| `page-flw.mkii (mkiv)` | ConTEXt Page Macros | Text Flows |
| `page-grd.mkiv` | ConTEXt Page Macros | Grids |
| `page-imp.mkii (mkiv)` | ConTEXt Page Macros | Pagebody Building (Imposition) |
| `page-inf.mkiv` | ConTEXt Page Macros | Tracing Info |
| `page-ini.mkii (mkiv)` | ConTEXt Page Macros | Initializations |
| `page-inj.lua (mkvi)` | ConTEXt Page Module | Injections |
| `page-ins.lua (mkii,mkiv)` | ConTEXt Insertion Macros | Insertions |
| `page-lay.mkii (mkiv)` | ConTEXt Page Macros | Layout Specification |
| `page-lin.lua (mkii,mkiv)` | ConTEXt Core Macros | Line Numbering |
| `page-log.mkii` | ConTEXt Page Macros | Logos |
| `page-mak.mkii (mkvi)` | ConTEXt Page Macros | Simple MakeUp |
| `page-mar.mkii` | ConTEXt Page Macros | Marginal Things |
| `page-mbk.mkvi` | ConTEXt Page Macros | Margin Floats |
| `page-mis.mkii` | ConTEXt Page Macros | Misc Float Things |
| `page-mix.lua (mkiv)` | ConTEXt Page Macros | Mixed Columns |

| | | |
|---|---|---|
| `page-mrk.mkiv` | ConTEXt Page Macros | Cutmarks and Colorbars |
| `page-mul.mkii (mkiv)` | ConTEXt Page Macros | Multi Column Output |
| `page-not.mkii (mkiv)` | ConTEXt Page Macros | Footnotes |
| `page-one.mkii (mkiv)` | ConTEXt Page Macros | Default Routine |
| `page-otr.mkvi` | ConTEXt Page Macros | Output Routines |
| `page-par.mkii (mkiv)` | ConTEXt Page Macros | Line Numbering |
| `page-plg.mkii (mkiv)` | ConTEXt Page Macros | Page Setup |
| `page-pst.lua (mkiv)` | ConTEXt Page Macros | Postponing |
| `page-run.mkii (mkiv)` | ConTEXt Page Macros | Runtime Macros |
| `page-sel.mkvi` | ConTEXt Page Macros | Page Selection |
| `page-set.mkii (mkiv)` | ConTEXt Page Macros | Column Sets |
| `page-sid.mkii (mkiv)` | ConTEXt Page Macros | Side Floats |
| `page-spr.mkii (mkiv)` | ConTEXt Page Macros | Spreading |
| `page-str.lua (mkii,mkiv)` | ConTEXt Page Macros | Page Streams |
| `page-txt.mkii (mkvi)` | ConTEXt Page Macros | Texts |
| `page-var.mkiv` | ConTEXt Page Macros | Variables |
| `pdfr-def.mkii` | | |
| `pdfr-ec.mkii` | ConTEXt PDF Font Resources | EC encoding |
| `pdfr-il2.mkii` | ConTEXt PDF Font Resources | ISO Latin 2 |
| `phys-dim.lua (mkiv)` | ConTEXt Physics | Digits and Units |
| `ppchtex.mkii (mkiv)` | ConTEXt Extra Modules | ppchTEX (Plain Pictex Context cHemie TEX) |
| `prop-ini.mkii (mkiv)` | ConTEXt Property Macros | Initialization |
| `prop-lay.mkii` | ConTEXt Property Macros | Layers |
| `prop-mis.mkii` | ConTEXt Property Macros | Miscelaneous |
| `regi-8859-1.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-1 (West European) |
| `regi-8859-10.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-10 (Nordic) |
| `regi-8859-11.lua` | companion to regi-ini.mkiv | |
| `regi-8859-13.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-13 (Baltic) |
| `regi-8859-14.lua` | companion to regi-ini.mkiv | |
| `regi-8859-15.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-15 (West European) |
| `regi-8859-16.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-16 (Romanian) |
| `regi-8859-2.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-2 (East European) |
| `regi-8859-3.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-3 (South European) |
| `regi-8859-4.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-4 (North European) |
| `regi-8859-5.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-5 (Cyrillic) |
| `regi-8859-6.lua` | companion to regi-ini.mkiv | |
| `regi-8859-7.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-7 (Greek) |
| `regi-8859-8.lua` | companion to regi-ini.mkiv | |
| `regi-8859-9.lua (mkii)` | ConTEXt Encoding Macros | iso-8859-9 (Turkish) |
| `regi-cp1250.lua (mkii)` | ConTEXt Encoding Macros | cp1250 (East European) |
| `regi-cp1251.lua (mkii)` | ConTEXt Encoding Macros | cp1251 (Cyrillic) |
| `regi-cp1252.lua (mkii)` | ConTEXt Encoding Macros | cp1252 (West European) |
| `regi-cp1253.lua (mkii)` | ConTEXt Encoding Macros | cp1253 (Greek) |
| `regi-cp1254.lua (mkii)` | ConTEXt Encoding Macros | cp1254 (Turkish) |
| `regi-cp1255.lua` | companion to regi-ini.mkiv | |
| `regi-cp1256.lua` | companion to regi-ini.mkiv | |
| `regi-cp1257.lua (mkii)` | ConTEXt Encoding Macros | cp1257 (Windows Baltic) |
| `regi-cp1258.lua` | companion to regi-ini.mkiv | |
| `regi-cyp.mkii` | ConTEXt Encoding Macros | Cyrillic Plus |
| `regi-cyr.mkii` | ConTEXt Encoding Macros | Cyrillic |
| `regi-def.mkii` | ConTEXt Regime Macros | Default Character Definitions |
| `regi-demo.lua` | companion to regi-ini.mkiv | |
| `regi-ibm.mkii` | ConTEXt Encoding Macros | The Good Old MSDOS IBM codepage |
| `regi-ini.lua (mkii,mkiv)` | ConTEXt Regime Macros | Initialization |
| `regi-mac.mkii` | ConTEXt Encoding Macros | Mac Encoding |
| `regi-syn.mkii` | ConTEXt Regime Macros | Synonyms |

| | | |
|---|---|---|
| regi-uni.mkii | ConTEXt Encoding Macros | Unicode |
| regi-utf.mkii | ConTEXt Encoding Macros | UTF-8 |
| regi-vis.mkii | ConTEXt Encoding Macros | viscii |
| rlxcache.rlx | | |
| rlxtools.rlx | | |
| s-abr-01.tex | ConTEXt Style File | General Abbreviations 1 |
| s-abr-02.tex | ConTEXt Style File | General Abbreviations 2 |
| s-abr-03.tex | ConTEXt Style File | General Abbreviations 3 |
| s-abr-04.tex | ConTEXt Style File | General Abbreviations 2 |
| s-art-01.mkiv | | |
| s-cdr-01.tex | ConTEXt Style File | CDROM Cover |
| s-chi-00.mkii | ConTEXt Style File | Basic Chinese Style |
| s-def-01.mkiv | | |
| s-faq-00.tex | ConTEXt Style File | FAQ Common Macros |
| s-faq-01.tex | ConTEXt Style File | FAQ Interactive Version |
| s-faq-02.tex | ConTEXt Style File | FAQ Paper Version |
| s-faq-03.tex | ConTEXt Style File | FAQ General Framework |
| s-fnt-01.mkii | ConTEXt Style File | Font Environment 1 |
| s-fnt-02.mkii | | |
| s-fnt-10.mkiv | ConTEXt Style File | Listing Glyphs in Large Fonts |
| s-fnt-20.mkiv | ConTEXt Style File | Tracing Feature Application (1) |
| s-fnt-21.mkiv | ConTEXt Style File | Tracing Feature Application (2) |
| s-fnt-24.mkiv | ConTEXt Style File | CJK Glyph Combination Testing |
| s-fonts-coverage.lua (mkiv) | ConTEXt Style File | Show Fonts Coverage |
| s-fonts-features.lua (mkiv) | ConTEXt Style File | Features |
| s-fonts-goodies.lua (mkiv) | ConTEXt Style File | Goodies Tables |
| s-fonts-missing.lua (mkiv) | ConTEXt Style File | Some Missing Character Info |
| s-fonts-shapes.lua (mkiv) | ConTEXt Style File | Tracing Shapes |
| s-fonts-system.lua (mkiv) | ConTEXt Style File | Listing Installed Fonts |
| s-fonts-tables.lua (mkiv) | ConTEXt Style File | Basic Font Data Tables |
| s-fonts-vectors.lua (mkiv) | ConTEXt Style File | Protrusion and Expansion |
| s-grk-00.mkii | ConTEXt Style File | CB Greek Support |
| s-inf-01.mkvi | ConTEXt Style File | Information 1 (MkII/MkIV usage) |
| s-inf-02.mkiv | ConTEXt Style File | Information 2 (filenames) |
| s-inf-03.mkiv (pdf) | | |
| s-inf-04.mkiv | | |
| s-jap-00.mkii | ConTEXt Style File | Basic Japanese Style |
| s-languages-counters.lua (mkiv) | ConTEXt Style File | Language Counters |
| s-languages-frequencies.lua (mkiv) | ConTEXt Style File | Language Frequencies |
| s-languages-hyphenation.lua (mkiv) | ConTEXt Style File | Language Hyphenation |
| s-languages-sorting.lua (mkiv) | ConTEXt Style File | Language Sorting |
| s-languages-system.lua (mkiv) | ConTEXt Style File | Installed Languages |
| s-mag-01.tex | ConTEXt Style File | ConTEXt Magazine Base Style |
| s-map-10.mkii (mkiv) | ConTEXt Style File | Maps journal style |
| s-math-characters.lua (mkiv) | ConTEXt Style File | Math Glyph Checking |
| s-math-coverage.lua (mkiv) | ConTEXt Style File | Show Math Coverage |
| s-math-extensibles.mkiv | ConTEXt Style File | Math Stackers Checking |
| s-math-parameters.lua (mkiv) | ConTEXt Style File | Show Math Parameters |
| s-math-repertoire.mkiv | ConTEXt Style File | Show Math Character Repertoire |
| s-mod.ctx | | |
| s-mod-00.mkii (mkiv) | ConTEXt Style File | Documentation Base Environment |
| s-mod-01.mkii (mkiv) | ConTEXt Style File | Module Documentation |
| s-mod-02.mkii (mkiv) | ConTEXt Style File | Documentation Screen Environment |
| s-pages-statistics.mkiv | ConTEXt Style File | Page Statistics |

| | | |
|---|---|---|
| `s-physics-units.mkiv` | ConTEXt Modules | Physics Units |
| `s-pre-00.tex` | ConTEXt Style File | Presentation Environment 0 |
| `s-pre-01.tex` | ConTEXt Style File | Presentation Environment 1 |
| `s-pre-02.tex` | ConTEXt Style File | Presentation Environment 2 |
| `s-pre-03.tex` | ConTEXt Style File | Presentation Environment 3 |
| `s-pre-04.tex` | ConTEXt Style File | Presentation Environment 4 |
| `s-pre-05.tex` | ConTEXt Style File | Presentation Environment 5 |
| `s-pre-06.tex` | ConTEXt Style File | Presentation Environment 6 |
| `s-pre-07.tex` | ConTEXt Style File | Presentation Environment 7 |
| `s-pre-08.tex` | ConTEXt Style File | Presentation Environment 8 |
| `s-pre-09.tex` | ConTEXt Style File | Presentation Environment 9 |
| `s-pre-10.tex` | ConTEXt Style File | Presentation Environment 10 |
| `s-pre-11.tex` | ConTEXt Style File | Presentation Environment 11 |
| `s-pre-12.tex` | ConTEXt Style File | Presentation Environment 12 |
| `s-pre-13.tex` | ConTEXt Style File | Presentation Environment 13 |
| `s-pre-14.tex` | ConTEXt Style File | Presentation Environment 14 |
| `s-pre-15.tex` | ConTEXt Style File | Presentation Environment 15 |
| `s-pre-16.tex` | ConTEXt Style File | Presentation Environment 16 |
| `s-pre-17.mkii (mkiv)` | ConTEXt Style File | Presentation Environment 17 |
| `s-pre-18.tex` | ConTEXt Style File | Presentation Environment 18 |
| `s-pre-19.tex` | ConTEXt Style File | Presentation Environment 19 |
| `s-pre-22.tex` | ConTEXt Style File | Presentation Environment 22 |
| `s-pre-23.tex` | ConTEXt Style File | Presentation Environment 20 |
| `s-pre-26.tex` | ConTEXt Style File | Presentation Environment 26 |
| `s-pre-27.tex` | ConTEXt Style File | Presentation Environment 27 |
| `s-pre-30.mkii (mkiv)` | ConTEXt Style File | Presentation Environment 30 |
| `s-pre-50.tex` | ConTEXt Style File | Presentation Environment 50 |
| `s-pre-60.mkii (mkiv)` | ConTEXt Style File | Presentation Environment 60 |
| `s-pre-61.tex` | ConTEXt Style File | Presentation Environment 61 |
| `s-pre-62.tex` | ConTEXt Style File | Presentation Environment 62 |
| `s-pre-63.tex` | ConTEXt Style File | Presentation Environment 63 |
| `s-pre-64.tex` | ConTEXt Style File | Presentation Environment 64 |
| `s-pre-66.tex` | ConTEXt Style File | Presentation Environment 66 |
| `s-pre-67.tex` | | |
| `s-pre-68.tex` | ConTEXt Style File | Presentation Environment 68 |
| `s-pre-69.mkiv` | ConTEXt Style File | Presentation Environment 69 |
| `s-pre-70.mkiv` | ConTEXt Style File | Presentation Environment 70 |
| `s-pre-71.lua (mkii,mkiv)` | ConTEXt Style File | Presentation Environment 71 |
| `s-pre-93.tex` | ConTEXt Style File | Presentation Environment 20 |
| `s-pre-96.tex` | ConTEXt Style File | Presentation Environment 26 |
| `s-present-tiles.mkiv` | ConTEXt Style File | Presentation Environment Tiles |
| `s-ptj-01.tex` | ConTEXt Style File | PracTeX Journal Style |
| `s-reg-01.mkiv` | ConTEXt Style File | Extra Regime Support |
| `s-set-31.mkiv` | | |
| `s-sql-tables.lua (mkiv)` | ConTEXt Extra Modules | SQL |
| `s-syn-01.tex` | ConTEXt Style File | Preliminary Syntax Stuff |
| `scrn-bar.mkvi` | ConTEXt Core Macros | Progress Bars |
| `scrn-but.lua (mkvi)` | ConTEXt Core Macros | Interaction |
| `scrn-fld.lua (mkii,mkvi)` | ConTEXt Screen Macros | Fields |
| `scrn-hlp.lua (mkii,mkvi)` | ConTEXt Screen Macros | Help (Experimental) |
| `scrn-ini.lua (mkvi)` | ConTEXt Interaction Macros | Initialization |
| `scrn-int.mkii` | ConTEXt Core Macros | Interaction |
| `scrn-nav.mkii` | ConTEXt Core Macros | Navigation |
| `scrn-pag.lua (mkvi)` | ConTEXt Screen Macros | Pages |
| `scrn-ref.lua (mkvi)` | ConTEXt Screen Macros | References |
| `scrn-wid.lua (mkvi)` | ConTEXt Core Macros | Widgets |
| `scrp-cjk.lua` | companion to scrp-ini.mkiv | |

| | | |
|---|---|---|
| scrp-eth.lua | companion to scrp-ini.mkiv | |
| scrp-ini.lua (mkiv) | ConTEXt Script Macros | Initialization |
| scrp-tha.lua | companion to scrp-ini.mkiv | |
| sort-def.mkii | ConTEXt Sort Macros | Defaults |
| sort-ini.lua (mkii,mkiv) | ConTEXt Sorting Macros | Initialization |
| sort-lan.lua (mkii) | ConTEXt Sort Macros | Language Definitions |
| spac-adj.lua (mkiv) | ConTEXt Spacing Macros | Paragraphs |
| spac-ali.lua (mkiv) | ConTEXt Spacing Macros | Alignments |
| spac-cha.mkiv | ConTEXt Spacing Macros | Character Alignment |
| spac-chr.lua (mkiv) | ConTEXt Spacing Macros | Characters |
| spac-def.mkiv | ConTEXt Spacing Macros | Definitions |
| spac-flr.mkiv | ConTEXt Spacing Macros | Fillers |
| spac-gen.mkii | ConTEXt Core Macros | Spacing |
| spac-grd.mkii (mkiv) | ConTEXt Spacing Macros | Grid Snapping |
| spac-hor.lua (mkiv) | ConTEXt Spacing Macros | Horizontal |
| spac-lin.mkiv | ConTEXt Spacing Macros | Vertical |
| spac-pag.mkiv | ConTEXt Spacing Macros | Pages |
| spac-par.mkiv | ConTEXt Spacing Macros | Paragraphs |
| spac-ver.lua (mkiv) | ConTEXt Spacing Macros | Vertical |
| spec-def.mkii | ConTEXt Special Macros | Definitions |
| spec-dpm.mkii | ConTEXt Special Macros | DVIPDFM support |
| spec-dpx.mkii | ConTEXt Special Macros | DVIPDFMx support |
| spec-dvi.mkii | ConTEXt Special Macros | Generic TEX Solutions |
| spec-fdf.mkii | ConTEXt pdf Macros | Support Macros |
| spec-ini.mkii | ConTEXt Special Macros | Initialization |
| spec-mis.mkii | ConTEXt Special Macros | Miscellaneous Macros |
| spec-pdf.mkii | ConTEXt Special Macros | Adobe Acrobat version 2.1 |
| spec-ps.mkii | ConTEXt Special Macros | Adobe PostScript |
| spec-tpd.mkii | ConTEXt Special Macros | pdfTEX |
| spec-tr.mkii | ConTEXt Special Macros | Thomas Rokicki's dvips |
| spec-tst.mkii | ConTEXt pdf Macros | Special Test Macro |
| spec-var.mkii | ConTEXt Special Macros | Variables |
| spec-win.mkii | ConTEXt Special Macros | y&y's dviwindo |
| spec-xet.mkii | ConTEXt Special Macros | XeTEX support |
| spec-xtx.mkii | ConTEXt Special Macros | XeTEX support |
| spec-yy.mkii | ConTEXt Special Macros | y&y's dvipsone and dviwindo |
| status-files.pdf | | |
| status-lua.log | | |
| status-mkiv.tex (lua) | stub file for context | |
| strc-bkm.lua (mkiv) | ConTEXt Structure Macros | Bookmarks |
| strc-blk.lua (mkii,mkiv) | ConTEXt Structure Macros | Blockmoves |
| strc-con.lua (mkvi) | ConTEXt Structure Macros | Constructions |
| strc-def.mkiv | ConTEXt Structure Macros | Definitions |
| strc-des.mkii (mkvi) | ConTEXt Structure Macros | Descriptions |
| strc-doc.lua (mkiv) | ConTEXt Structure Macros | Document Structure |
| strc-enu.mkvi | ConTEXt Structure Macros | Enumerations |
| strc-flt.lua (mkii,mkvi) | ConTEXt Structure Macros | Float Numbering |
| strc-ind.mkiv | ConTEXt Structure Macros | Indented Text |
| strc-ini.lua (mkvi) | ConTEXt Structure Macros | Initialization & Helpers |
| strc-itm.lua (mkii,mkvi) | ConTEXt Structure Macros | itemgroups |
| strc-lab.mkiv | ConTEXt Structure Macros | Labels |
| strc-lev.lua (mkvi) | ConTEXt Structure Macros | Automatic Levels |
| strc-lnt.mkii (mkvi) | ConTEXt Structure Macros | Line Notes |
| strc-lst.lua (mkii,mkvi) | ConTEXt Structure Macros | Lists |
| strc-mar.lua (mkii,mkiv) | ConTEXt Structure Macros | Markings |
| strc-mat.lua (mkii,mkiv) | ConTEXt Structure Macros | Math Numbering |
| strc-not.lua (mkii,mkvi) | ConTEXt Structure Macros | Note Handling |

| | | |
|---|---|---|
| `strc-num.lua` (mkii,mkiv) | ConTEXt Structure Macros | Basic Numbering |
| `strc-pag.lua` (mkii,mkiv) | ConTEXt Structure Macros | Numbering |
| `strc-ref.lua` (mkii,mkvi) | ConTEXt Structure Macros | Cross Referencing |
| `strc-reg.lua` (mkii,mkiv) | ConTEXt Structure Macros | Register Management |
| `strc-ren.mkiv` | ConTEXt Structure Macros | Section Rendering |
| `strc-rsc.lua` | companion to strc-ref.mkiv | |
| `strc-sbe.mkiv` | ConTEXt Structure Macros | Section Block Environments |
| `strc-sec.mkii` (mkiv) | ConTEXt Structure Macros | Sectioning |
| `strc-swd.mkii` | ConTEXt Structure Macros | Section Worlds |
| `strc-syn.lua` (mkii,mkiv) | ConTEXt Structure Macros | Synonyms and Sorting |
| `strc-tag.lua` (mkiv) | ConTEXt Structure Macros | Tags |
| `strc-xml.mkiv` | ConTEXt Structure Macros | XML Processing |
| `supp-ali.mkii` | ConTEXt Support Macros | Alignment |
| `supp-box.lua` (mkii,mkiv) | ConTEXt Support Macros | Boxes |
| `supp-dir.mkii` (mkiv) | ConTEXt Support Macros | Directional Things |
| `supp-emp.mkii` | ConTEXt Support Macros | emTEX specials to pdf conversion |
| `supp-eps.mkii` | ConTEXt Support Macros | eps tools |
| `supp-fil.mkii` | ConTEXt Support Macros | Files |
| `supp-fun.mkii` | ConTEXt Support Macros | Fun Stuff |
| `supp-lat.mkii` | ConTEXt System Macros | General |
| `supp-mat.mkii` (mkiv) | ConTEXt Support Macros | Math |
| `supp-mis.tex` (mkii) | ConTEXt Support Macros | Missing (For Generic Use) |
| `supp-mpe.tex` (mkii) | ConTEXt Support Macros | METAPOST Special Extensions |
| `supp-mps.mkii` | ConTEXt Support Macros | MetaPost Inclusion |
| `supp-mrk.mkii` | ConTEXt Support Macros | Marks |
| `supp-num.mkii` | ConTEXt Support Macros | Numbers |
| `supp-pat.mkii` | ConTEXt Support Macros | Patterns |
| `supp-pdf.tex` (mkii) | ConTEXt Support Macros | MetaPost to pdf conversion |
| `supp-ran.lua` (mkii,mkiv) | ConTEXt Support Macros | Random Number Generation |
| `supp-spe.mkii` | ConTEXt Support Macros | Specials |
| `supp-tpi.mkii` | ConTEXt Support Macros | tpic Conversion |
| `supp-vis.mkii` (mkiv) | ConTEXt Support Macros | Visualization |
| `symb-cow.mkii` | ConTEXt Symbol Libraries | Cow Symbols |
| `symb-eur.mkii` | ConTEXt Symbol Libraries | Adobe Euro Symbols |
| `symb-glm.mkii` | ConTEXt Symbol Libraries | Guillemots |
| `symb-imp-cc.mkiv` | ConTEXt Symbol Libraries | Creative Commons |
| `symb-imp-cow.mkiv` | ConTEXt Symbol Libraries | Cow Symbols |
| `symb-imp-eur.mkiv` | ConTEXt Symbol Libraries | Adobe Euro Symbols |
| `symb-imp-jmn.mkiv` | ConTEXt Symbol Libraries | Special Navigational Symbols |
| `symb-imp-mis.mkiv` | ConTEXt Symbol Libraries | Miscelaneous |
| `symb-imp-mvs.mkiv` | ConTEXt Symbol Libraries | Martin Vogels Symbole |
| `symb-imp-nav.mkiv` | ConTEXt Symbol Libraries | Navigational Symbols |
| `symb-ini.lua` (mkii,mkiv) | ConTEXt Symbol Libraries | Basic Symbols Commands |
| `symb-jmn.mkii` | ConTEXt Symbol Libraries | Special Navigational Symbols |
| `symb-mis.mkii` | ConTEXt Symbol Libraries | Miscelaneous |
| `symb-mvs.mkii` | ConTEXt Symbol Libraries | Martin Vogels Symbole |
| `symb-nav.mkii` | ConTEXt Symbol Libraries | Navigational Symbols |
| `symb-run.mkii` (mkiv) | ConTEXt Symbol Libraries | Runtime Macros |
| `symb-uni.mkii` | ConTEXt Symbol Libraries | Unicode Symbols |
| `symb-was.mkii` | ConTEXt Symbol Libraries | Roland Waldi's Symbols (wasy-2) |
| `syst-aux.lua` (mkiv) | ConTEXt System Macros | General |
| `syst-con.lua` (mkii,mkiv) | ConTEXt System Macros | Conversions |
| `syst-ext.mkii` | ConTEXt System Macros | Extras |
| `syst-fnt.mkii` (mkiv) | ConTEXt System Macros | Font Things |
| `syst-gen.mkii` | ConTEXt System Macros | General |
| `syst-ini.mkii` (mkiv) | ConTEXt System Macros | Bootstrapping TEX |
| `syst-lua.lua` (mkiv) | ConTEXt System Macros | Helper macros based on Lua |

| | | |
|---|---|---|
| `syst-mes.mkiv` | ConTEXt System Macros | Messages |
| `syst-new.mkii` | ConTEXt Support Macros | New Ones |
| `syst-pln.mkii (mkiv)` | ConTEXt System Macros | Efficient Plain TEX loading |
| `syst-rtp.mkii (mkiv)` | ConTEXt Core Macros | Run Time Processes |
| `syst-str.mkii` | ConTEXt System Macros | String Processing |
| `syst-tex.mkii` | ConTEXt System Macros | Efficient Plain TEX loading |
| `tabl-com.mkii (mkiv)` | ConTEXt Table Macros | Common Code |
| `tabl-ltb.mkii (mkiv)` | ConTEXt Table Macros | Line Tables |
| `tabl-mis.mkiv` | ConTEXt Table Macros | Miscellaneous |
| `tabl-ntb.mkii (mkiv)` | ConTEXt Table Macros | Natural Tables |
| `tabl-nte.mkii (mkiv)` | ConTEXt Table Macros | Natural Tables Extensions |
| `tabl-pln.mkii (mkiv)` | | |
| `tabl-tab.mkii (mkiv)` | ConTEXt Table Macros | TABLE Embedding |
| `tabl-tbl.lua (mkii,mkiv)` | ConTEXt Table Macros | Text Flow Tabulation |
| `tabl-tsp.mkii (mkiv)` | ConTEXt Table Macros | Splitting |
| `tabl-xnt.mkvi` | ConTEXt Table Macros | Natural to Xtreme Tables |
| `tabl-xtb.lua (mkvi)` | ConTEXt Table Macros | Xtreme |
| `task-ini.lua (mkiv)` | ConTEXt Task Handler | Initialization |
| `thrd-pic.mkii` | | |
| `thrd-ran.mkii` | | |
| `thrd-tab.mkii` | | |
| `thrd-trg.mkii` | | |
| `toks-ini.lua (mkiv)` | ConTEXt Token Support | Initialization |
| `trac-ctx.lua (mkiv)` | ConTEXt Tracing Macros | TeX Trackers |
| `trac-deb.lua (mkiv)` | ConTEXt Tracing Macros | Debugger |
| `trac-exp.lua` | companion to trac-log.mkiv | |
| `trac-fil.lua` | for the moment for myself | |
| `trac-inf.lua` | companion to trac-inf.mkiv | |
| `trac-jus.lua (mkiv)` | ConTEXt Tracing Macros | Justification |
| `trac-lmx.lua` | companion to trac-lmx.mkiv | |
| `trac-log.lua` | companion to trac-log.mkiv | |
| `trac-pro.lua` | companion to luat-lib.mkiv | |
| `trac-set.lua` | companion to luat-lib.mkiv | |
| `trac-tex.lua (mkiv)` | ConTEXt Tracking Macros | TEX |
| `trac-tim.lua` | companion to m-timing.tex | |
| `trac-vis.lua (mkii,mkiv)` | ConTEXt Tracing Macros | Visualization |
| `trac-xml.lua` | companion to trac-log.mkiv | |
| `type-buy.mkii` | ConTEXt Typescript Macros | A Few Commercial Fonts |
| `type-cbg.mkii` | ConTEXt Typescript Macros | CB Greek |
| `type-cow.mkii` | ConTEXt Typescript Macros | Cow Fonts |
| `type-def.mkii (mkiv)` | ConTEXt Typescript Macros | Default Definitions |
| `type-exp.mkii` | ConTEXt Typescript Macros | Experimental Definitions |
| `type-fbk.mkiv` | ConTEXt Typescript Macros | Fallbacks |
| `type-fsf.mkii` | ConTEXt Page Macros | Fontsite 500 |
| `type-ghz.mkii` | ConTEXt Typescript Macros | Hermann Zapf's Fonts |
| `type-hgz.mkii` | | |
| `type-imp-antykwa.mkiv` | ConTEXt Typescript Macros | Antykwa Torunska |
| `type-imp-antykwapoltawskiego.mkiv` | ConTEXt Typescript Macros | Antykwa Poltawskiego |
| `type-imp-asana.mkiv` | ConTEXt Typescript Macros | Asana |
| `type-imp-averia.mkiv` | ConTEXt Typescript Macros | Averia fonts (iotic.com) |
| `type-imp-buy.mkiv` | ConTEXt Typescript Macros | A Few Commercial Fonts |
| `type-imp-cambria.mkiv` | ConTEXt Typescript Macros | Microsoft Cambria |
| `type-imp-charter.mkiv` | ConTEXt Typescript Macros | Charter |
| `type-imp-cleartype.mkiv` | ConTEXt Typescript Macros | Microsoft Cleartype |
| `type-imp-computer-modern-unicode.mkiv` | ConTEXt Typescript Macros | Computer Modern Unicode |

| | | |
|---|---|---|
| `type-imp-cow.mkiv` | ConTEXt Typescript Macros | Cow Fonts |
| `type-imp-dejavu.mkiv` | ConTEXt Typescript Macros | Dejavu fonts (dejavu-fonts.org) |
| `type-imp-euler.mkiv` | ConTEXt Typescript Macros | Euler |
| `type-imp-ghz.mkiv` | ConTEXt Typescript Macros | Hermann Zapf's Fonts |
| `type-imp-hgz.mkiv` | | |
| `type-imp-husayni.mkiv` | ConTEXt Typescript Macros | Husayni |
| `type-imp-hvmath.mkiv` | ConTEXt Typescript Macros | HV Math Fonts |
| `type-imp-inconsolata.mkiv` | ConTEXt Typescript Macros | Inconsolata |
| `type-imp-informal.mkiv` | ConTEXt Typescript Macros | Informal by M. Vulis |
| `type-imp-iwona.mkiv` | ConTEXt Typescript Macros | Iwona |
| `type-imp-kurier.mkiv` | ConTEXt Typescript Macros | Kurier by JMN |
| `type-imp-latinmodern.mkiv` | ConTEXt Typescript Macros | Latin Modern |
| `type-imp-liberation.mkiv` | ConTEXt Typescript Macros | Liberation fonts |
| `type-imp-libertine.mkiv` | ConTEXt Typescript Macros | Libertine fonts |
| `type-imp-lmnames.mkiv` | ConTEXt Typescript Macros | Opentype Definitions |
| `type-imp-lucida-opentype.mkiv` | ConTEXt Typescript Macros | Lucida Nova Opentype |
| `type-imp-lucida-typeone.mkiv` | ConTEXt Typescript Macros | Lucida |
| `type-imp-mathdesign.mkiv` | ConTEXt Typescript Macros | Mathdesign |
| `type-imp-mathdigits.mkiv` | ConTEXt Typescript Macros | Xits |
| `type-imp-mathtimes.mkiv` | ConTEXt Typescript Macros | Math Times |
| `type-imp-mscore.mkiv` | ConTEXt Typescript Macros | Microsoft Core Fonts |
| `type-imp-opendyslexic.mkiv` | ConTEXt Typescript Macros | Opendyslexic Fonts |
| `type-imp-osx.mkiv` | ConTEXt Typescript Macros | Mac OS X Definitions |
| `type-imp-postscript.mkiv` | ConTEXt Typescript Macros | Basic Font Set |
| `type-imp-punknova.mkiv` | ConTEXt Typescript Macros | Punk Nova |
| `type-imp-texgyre.mkiv` | ConTEXt Typescript Macros | TEXGyre Fonts |
| `type-imp-unfonts.mkiv` | ConTEXt Typescript Macros | UnFonts |
| `type-imp-xits.mkiv` | ConTEXt Typescript Macros | Xits |
| `type-imp-xitsbidi.mkiv` | ConTEXt Typescript Macros | Xits |
| `type-ini.lua` (mkii,mkvi) | ConTEXt Typescript Macros | Initialization |
| `type-lua.mkiv` | ConTEXt Typescript Macros | MkIV goodies |
| `type-mac.mkii` | ConTEXt Typescript Macros | Mac OS X Definitions |
| `type-msw.mkii` | | |
| `type-one.mkii` (mkiv) | ConTEXt Typescript Macros | Type One Definitions |
| `type-otf.mkii` (mkiv) | ConTEXt Typescript Macros | Opentype Definitions |
| `type-pre.mkii` | ConTEXt Typescript Macros | Compatibility scripts |
| `type-run.mkii` (mkiv) | ConTEXt Typescript Macros | Runtime Macros |
| `type-set.mkii` (mkiv) | ConTEXt Typescript Macros | Default Settings |
| `type-siz.mkii` (mkiv) | ConTEXt Typescript Macros | Sizing scripts |
| `type-tmf.mkii` (mkiv) | ConTEXt Typescript Macros | Core TEX Fonts |
| `type-win.mkii` | ConTEXt Typescript Macros | Microsoft Windows Fonts |
| `type-xtx.mkii` | ConTEXt Typescript Macros | XƎTEX's font treasures |
| `typo-bld.lua` (mkiv) | ConTEXt Typesetting Macros | Paragraph Building |
| `typo-brk.lua` (mkiv) | ConTEXt Typesetting Macros | Breakpoints |
| `typo-cap.lua` (mkiv) | ConTEXt Typesetting Macros | Capping |
| `typo-cln.lua` (mkiv) | ConTEXt Typesetting Macros | Cleaning |
| `typo-del.mkiv` | ConTEXt Typesetting Macros | Delimited Content |
| `typo-dha.lua` | companion to typo-dir.mkiv | |
| `typo-dig.lua` (mkiv) | ConTEXt Typesetting Macros | Digits |
| `typo-dir.lua` (mkiv) | ConTEXt Typesetting Macros | Directions |
| `typo-drp.lua` (mkiv) | ConTEXt Typesetting Macros | Initials |
| `typo-dua.lua` | Unicode bidi (sort of) variant a | |
| `typo-dub.lua` | Unicode bidi (sort of) variant b | |
| `typo-fln.lua` (mkiv) | ConTEXt Typesetting Macros | First Lines |
| `typo-ini.lua` (mkii,mkiv) | ConTEXt Typographic Macros | Initialization |
| `typo-itc.lua` (mkvi) | ConTEXt Typesetting Macros | Italic Correction |

| | | |
|---|---|---|
| typo-itm.mkiv | ConTEXt Typesetting Macros | Item Lists |
| typo-krn.lua (mkiv) | ConTEXt Typesetting Macros | Spacing |
| typo-lan.lua (mkiv) | ConTEXt Typesetting Macros | Language Goodies |
| typo-mar.lua (mkiv) | ConTEXt Typesetting Macros | Margindata |
| typo-pag.lua (mkiv) | ConTEXt Typesetting Macros | Pages |
| typo-prc.lua (mkvi) | ConTEXt Structure Macros | Processors |
| typo-rep.lua (mkiv) | ConTEXt Typesetting Macros | Stripping |
| typo-scr.mkiv | ConTEXt Typesetting Macros | Scripts |
| typo-spa.lua (mkiv) | ConTEXt Typesetting Macros | Spacing |
| typo-txt.mkvi | ConTEXt Typesetting Macros | Text Hacks |
| unic-000.mkii | ConTEXt Unicode Macros | Vector 0 |
| unic-001.mkii | ConTEXt Unicode Macros | Vector 1 |
| unic-002.mkii | ConTEXt Unicode Macros | Vector 2 |
| unic-003.mkii | ConTEXt Unicode Macros | Vector 3 |
| unic-004.mkii | ConTEXt Unicode Macros | Vector 4 |
| unic-005.mkii | ConTEXt Unicode Macros | Vector 5 |
| unic-030.mkii | ConTEXt Unicode Macros | Vector 30 |
| unic-031.mkii | ConTEXt Unicode Macros | Vector 31 |
| unic-032.mkii | ConTEXt Unicode Macros | Vector 32 |
| unic-033.mkii | ConTEXt Unicode Macros | Vector 33 |
| unic-034.mkii | ConTEXt Unicode Macros | Vector 34 |
| unic-035.mkii | ConTEXt Unicode Macros | Vector 35 |
| unic-037.mkii | ConTEXt Unicode Macros | Vector 37 |
| unic-039.mkii | ConTEXt Unicode Macros | Vector 39 |
| unic-251.mkii | ConTEXt Unicode Macros | Vector 251 |
| unic-cjk.mkii | ConTEXt Unicode Macros | CJK Vectors |
| unic-exp.mkii | ConTEXt Unicode Support | Expansion |
| unic-ini.lua (mkii,mkiv) | ConTEXt Unicode Support | Unicode & UTF-8 support |
| unic-run.mkii | ConTEXt Unicode Support | Goodies |
| util-deb.lua | companion to luat-lib.mkiv | |
| util-dim.lua | support for dimensions | |
| util-env.lua | companion to luat-lib.mkiv | |
| util-fmt.lua | companion to luat-lib.mkiv | |
| util-jsn.lua | companion to m-json.mkiv | |
| util-lib.lua | companion to luat-lib.mkiv | |
| util-lua.lua | the strip code is written by Peter Cawley | |
| util-mrg.lua | companion to luat-lib.mkiv | |
| util-pck.lua | companion to luat-lib.mkiv | |
| util-prs.lua | companion to luat-lib.mkiv | |
| util-ran.lua | companion to luat-lib.mkiv | |
| util-seq.lua | companion to luat-lib.mkiv | |
| util-soc.lua | support for sockets / protocols | |
| util-sql.lua | companion to m-sql.mkiv | |
| util-sql-imp-client.lua | companion to util-sql.lua | |
| util-sql-imp-library.lua | companion to util-sql.lua | |
| util-sql-imp-swiglib.lua | companion to util-sql.lua | |
| util-sql-loggers.lua | companion to lmx-* | |
| util-sql-sessions.lua | companion to lmx-* | |
| util-sql-tickets.lua | companion to lmx-* | |
| util-sql-tracers.lua | companion to m-sql.mkiv | |
| util-sql-users.lua | companion to lmx-* | |
| util-sta.lua | companion to util-ini.mkiv | |
| util-sto.lua | companion to luat-lib.mkiv | |
| util-str.lua | companion to luat-lib.mkiv | |
| util-tab.lua | companion to luat-lib.mkiv | |
| util-tpl.lua | companion to luat-lib.mkiv | |

| | | |
|---|---|---|
| verb-c.mkii | ConTEXt Verbatim Macros | Pretty C Verbatim |
| verb-eif.mkii | ConTEXt Verbatim Macros | Pretty Eiffel Verbatim |
| verb-ini.mkii | ConTEXt Verbatim Macros | Initialization |
| verb-js.mkii | ConTEXt Verbatim Macros | Pretty JavaScript Verbatim |
| verb-jv.mkii | ConTEXt Verbatim Macros | Pretty Java Verbatim |
| verb-mp.mkii | ConTEXt Verbatim Macros | Pretty MetaPost Verbatim |
| verb-pas.mkii | ConTEXt Verbatim Macros | Pretty Pascal and Modula Verbatim |
| verb-pl.mkii | ConTEXt Verbatim Macros | Pretty Perl Verbatim |
| verb-raw.mkii | | |
| verb-sql.mkii | ConTEXt Verbatim Macros | Pretty sql Verbatim |
| verb-tex.mkii | ConTEXt Verbatim Macros | Pretty TEX verbatim |
| verb-xml.mkii | ConTEXt Verbatim Macros | Pretty XML verbatim |
| x-asciimath.lua (mkiv) | ConTEXt Modules | AsciiMath |
| x-calcmath.lua (mkii,mkiv) | ConTEXt Modules | Calculator Math |
| x-cals.lua (mkiv) | ConTEXt XML Modules | Cals table renderer |
| x-chemml.lua (mkii,mkiv,xsd) | ConTEXt XML Modules | MkIV ChemML renderer |
| x-contml.mkii (xsd) | ConTEXt XML Support | Basic ConTEXt commands |
| x-corres.mkii (rng) | ConTEXt XML Modules | Handling Correspondence Base |
| x-ct.lua (mkiv) | ConTEXt XML Modules | ConTEXt Structures |
| x-dir-01.tex | ConTEXt Directory Handling | Overview (1) |
| x-dir-05.mkii (mkiv) | ConTEXt Directory Handling | Access |
| x-entities.mkiv | ConTEXt XML Modules | html entities |
| x-fdf-00.mkii | | |
| x-fe.mkii | foXet | Simple Extensions |
| x-fig-00.dtd (mkii,xsd) | ConTEXt Style File | Figure Base Loading |
| x-fig-01.mkii | ConTEXt Style File | Figure Base Generation |
| x-fig-02.mkii | ConTEXt Style File | Figure Base Inclusion (I) |
| x-fig-03.mkii | ConTEXt Style File | Figure Base Inclusion (II) |
| x-fo.mkii | foXet | Formatting Objects |
| x-foxet.mkii (mkiv) | foXet | Formatting Objects |
| x-ldx.ctx (lua,mkiv) | ConTEXt Modules | Lua Source Pretty Printing |
| x-mathml.lua (mkii,mkiv,xsd) | ConTEXt XML Modules | MathML |
| x-newcml.mkii | ConTEXt XML Macros | ChemML |
| x-newmme.mkii | ConTEXt XML Macros | MathML Entities |
| x-newmml.mkii (mkiv) | ConTEXt XML Macros | MathML |
| x-newmmo.mkii | ConTEXt XML Macros | MathML Renderer/Open Math Extensions |
| x-newpml.mkii | ConTEXt XML Support | Units |
| x-om2cml.xsl | | |
| x-openmath.mkii (xsl) | | |
| x-pfs-01.mkiv | | |
| x-pfsense.ctx | | |
| x-physml.mkii (mkiv,xsd) | ConTEXt XML Modules | Loading PHYSML Filters |
| x-res-00.mkii | ConTEXt Style File | Resource Libraries |
| x-res-01.mkii (mkiv) | ConTEXt Style File | Figure Base Generation |
| x-res-02.mkii | ConTEXt Style File | Figure Base Inclusion (I) |
| x-res-03.mkii | ConTEXt Style File | Figure Base Inclusion (II) |
| x-res-04.mkii | ConTEXt Style File | Figure Base Loading |
| x-res-08.mkii | ConTEXt Style File | Resource Reporting |
| x-res-09.mkii | ConTEXt Style File | Resource Reporting (2) |
| x-res-10.mkii | ConTEXt Style File | Resource Dummy Generation |
| x-res-11.mkii | ConTEXt Style File | Resource Reporting (3) |
| x-res-12.mkii | ConTEXt Style File | Resource Checking |
| x-res-20.mkii | ConTEXt Style File | Figure Lists |
| x-res-50.mkii (mkiv) | ConTEXt Style File | Multimedia Presentation |
| x-sch-00.mkii | ConTEXt Style File | XML Schema Basics |
| x-sch-01.mkii | ConTEXt Style File | XML Schema Presentation |

| | | |
|---|---|---|
| x-set-01.mkii | ConTEXt Setup Mappings | Macro Definitions |
| x-set-02.mkii | ConTEXt Setup Mappings | Macro Definitions |
| x-set-11.mkii (mkiv) | ConTEXt Setup Definitions | Macro Definitions |
| x-set-12.mkii (mkiv) | ConTEXt Setup Definitions | Macro Definitions |
| x-sm2om.xsl | | |
| x-steps.mkii (mkiv) | ConTEXt Modules | Step Charts & Tables |
| x-udhr.mkiv | ConTEXt Modules | Unicode Language Test Files |
| x-xfdf.mkiv | ConTEXt XML Modules | xfdf |
| x-xml-01.mkii | ConTEXt XML Style File | Formatting X?? files |
| x-xml-02.mkii | ConTEXt XML Style File | Pretty Printing |
| x-xml-11.mkii | ConTEXt XML Style File | Formatting X?? files |
| x-xtag.mkiv | ConTEXt Modules | xml stream handler |
| xetx-chr.mkii | | |
| xetx-cls.mkii | | |
| xetx-ini.mkii | ConTEXt System Macros | XeTEX Initializations |
| xetx-utf.mkii | | |
| xtag-cml.mkii | | |
| xtag-ent.mkii | ConTEXt XML Macros | A bunch of Entities |
| xtag-exp.mkii | ConTEXt XML Macros | Expansion |
| xtag-ext.mkii | ConTEXt XML Macros | Extra Macros |
| xtag-hyp.mkii | ConTEXt XML MAcros | Hyphenation |
| xtag-ini.mkii | ConTEXt XML Macros | Initialization |
| xtag-map.mkii | ConTEXt XML Macros | Remapping |
| xtag-mea.mkii | | |
| xtag-meb.mkii | | |
| xtag-mec.mkii | | |
| xtag-meh.mkii | | |
| xtag-men.mkii | | |
| xtag-meo.mkii | | |
| xtag-mer.mkii | | |
| xtag-mmc.mkii | ConTEXt XML Macros | Content MathML |
| xtag-mml.mkii | ConTEXt XML Macros | MathML |
| xtag-mmp.mkii | ConTEXt XML Macros | Presentation MathML |
| xtag-mxa.mkii | | |
| xtag-mxb.mkii | | |
| xtag-mxc.mkii | | |
| xtag-mxh.mkii | | |
| xtag-mxn.mkii | | |
| xtag-mxo.mkii | | |
| xtag-mxr.mkii | | |
| xtag-pml.mkii | ConTEXt XML Support | Physics ML |
| xtag-pmu.mkii | ConTEXt XML Macros | Units |
| xtag-pre.mkii | ConTEXt XML Macros | Predefined Things |
| xtag-prs.mkii | ConTEXt XML Macros | Parsing |
| xtag-raw.mkii | ConTEXt XML Macros | Raw Specials |
| xtag-rng.mkii | ConTEXt XML Macros | Relax NG |
| xtag-run.mkii | ConTEXt XML Macros | Visualization |
| xtag-stk.mkii | ConTEXt XML Macros | Stacking Data |
| xtag-utf.mkii | ConTEXt XML Macros | UTF |
| xtag-xsd.mkii | ConTEXt XML Support | Schemas |
| xtag-xsl.mkii | ConTEXt XML Support | XSLT processing |

# E texmfstart manual

## Introduction

This manual is about a small (Ruby) script that can be used to run a script or open a document which is located somewhere in the `texmf` tree. This scripts evolved out of earlier experiments and is related to scripts and programs like `runperl`, `runruby` and `irun`.

One of the main reasons for `texmfstart` to exist is that it enables us to be downward compatible when using a TEX based environment. TEX itself is pretty stable, but this is not true for the whole collection of files that comes with a distribution and the way they are organized. We will see some other reasons for using this script as well.

We can also use this script for lanching applications that need access to resources in the TEX tree but that lack the features to locate them.

The script has a few dependencies on libraries. This means that relocating the script to a bin path may give problems. One can make a self–contained version by saying:

```
texmfstart --selfmerge
```

One can undo this with the `--selfclean` option. Normally users don't have to worry about this because in the ConTEXt distribution the merged version is shipped. A MS Windows (pseudo) binary can be made with `exerb` or one can simply associate the `.rb` suffix with the Ruby program.

```
FTYPE RubyScript=c:\data\system\ruby\bin\ruby.exe %%1 %%*
```

```
ASSOC .rb=RubyScript
ASSOC .rbw=RubyScript
```

On Unix one can make a copy without suffix:

```
cp texmfstart.rb /path/to/bin/texmfstart
chmod +x texmfstart
```

Alternative approaches have been discussed on the ConTEXt and TEXLive mailing lists and can be found in their archives.

## Launching programs

The primary usage of `texmfstart` is to launch programs and scripts. We can start the `texexec` Perl script with:

```
texmfstart texexec.pl --pdf somefile
```

We can also start the `pstopdf` Ruby script:

```
texmfstart pstopdf.rb --method=3 cow.eps
```

However, we can omit the suffix:

```
texmfstart texexec --pdf somefile
texmfstart pstopdf --method=3 cow.eps
```

The suffixless method is slower unless the scripts are known. For familiar ConTEXt scripts it's best not to use the suffix since this permits us to change the scripting language. ConTEXt related scripts are known. Because in the meantime `texexec` has become a Ruby script, users who use the suffixless method automatically will get the right version.

You can also say:

```
texmfstart --file=pstopdf --method=3 cow.eps
```

When locating a file to run, several methods are applied, one being `kpsewhich`. You can control the path searching by providing a program space, which by default happens to be `context`.

```
texmfstart --program=context --file=pstopdf --method=3 cow.eps
```

The general pattern is:

```
texmfstart switches filename arguments
```

Here `switches` control `texmfstart`'s behaviour, and `arguments` are passed to the program identified by `filename`.

Sometimes the operating system will spoil our little game of passing arguments. In the following case we want the output of `texexec` to be written to a log file. By using quotes, we can pass the redirection without problems.

```
texmfstart texexec "somefile.tex > whatever.log"
```

## Generating stubs

One of the reasons for writing `texmfstart` is that it permits us to write upward compatible scripts (batch files), so instead of

```
texexec --pdf somefile
texexec --pdf anotherfile
```

We prefer to use:

```
texmfstart texexec --pdf somefile
texmfstart texexec --pdf anotherfile
```

Instead of using `texmfstart` directly you can also use it in a stub file. For MS Windows such a file looks like:

```
@echo off
texmfstart texexec %*
```

In this case, the file itself is named `texexec.cmd`. Now, given that no new functionality of `texmfstart` itself is needed, one will automatically use the version of `texexec` that is present in the (latest) installed ConTEXt tree.

It is possible to generate stubs automatically. You can provide a path where the stub will be written. This permits tricks like the following. Say that on a cdrom we have the following structure:

```
tex/texmf-mswin/bin/texexec.bat
tex/texmf-linux/bin/texexec
tex/texmf-local/scripts/context/ruby/texexec.rb
```

If we are on the main tex path, we can run texmfstart as follows:

```
texmfstart --make --windows --stubpath=tex/texmf-mswin/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
texmfstart --make --unix    --stubpath=tex/texmf-linux/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
```

This will generate start up scripts that point directly to the Perl script. Such a link may fail when files get relocated. In that case you can use the --indirect directive, which will force the texmfstart into the stub file.

```
texmfstart --make --windows --indirect --stubpath=tex/texmf-mswin/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
texmfstart --make --unix    --indirect --stubpath=tex/texmf-linux/bin \
    ../../texmf-local/scripts/context/ruby/texexec.rb
```

However, the prefered way and most simple way to generate the stubs for the scripts that come with ConTEXt is:

```
texmfstart --make all
```

This will generate stubs suitable for the current operating system in the current path.

## Documents

You can use texmfstart to open a document.

```
texmfstart showcase.pdf
```

This will open the document showcase.pdf, when found. The chance is minimal that such a document can be located by kpsewhich. In that case, texmfstart will search the tree itself.

Given that it is supported on your platform, you can also open a pdf file on a given page.

```
texmfstart --page=2 showcase.pdf
```

On MS Windows the following command will open the pdf file in a web browser. This is needed when you want support for form submission.

```
texmfstart --browser examplap.pdf
```

## Search strategy

In a first attempt, kpsewhich will be used to locate a file. When kpsewhich cannot locate the file, the following environment variables will be used:

| | |
|---|---|
| RUBYINPUTS | ruby scripts with suffix rb |
| PERLINPUTS | perl scripts with suffix pl |
| PYTHONINPUTS | python scripts with suffix py |
| JAVAINPUTS | java archives with suffix jar |
| PDFINPUTS | pdf documents with suffix pdf |

It using them fails as well, the whole tree is searched, which will take some time.

When a file found, its location is remembered and passed on to nested runs. So, in general, a nested run will start faster.

## Directives

The script accepts a few directives. Some are rather general:

| | |
|---|---|
| `--verbose` | report some status and progress information |
| `--arguments` | an alternative for providing the arguments to be passed |
| `--clear` | don't pass info about locations to child processes |

Directives that concern starting an application are:

| | |
|---|---|
| `--program=str` | the program space where `kpsewhich` will search |
| `--locate` | report the call as it should happen (no newline) |
| `--report` | report the call as it should happen (simulated) |
| `--browser` | start the document in a web browser |
| `--file` | an alternative for providing the file |
| `--direct` | run a program without searching for it's location |
| `--execute` | use Ruby's 'exec' instead of 'system' |
| `--batch` | *not yet implemented* |

You can create startup scripts by providing one of the following switches in combination with a filename.

| | |
|---|---|
| `--make` | create a start script or batch file for the given program |
| `--windows` | when making a startup file, create a windows batch file |
| `--linux` | when making a startup file, create a unix script |
| `--stubpath` | destination of the startup file |
| `--indirect` | always use `texmfstart` in a stub file |

Some directives can be accompanied by specifications, like:

| | |
|---|---|
| `--page=n` | open the document at this page |
| `--path=str` | change from the current path to the given path |
| `--before=str` | *not yet implemented* |
| `--after=str` | *not yet implemented* |
| `--tree=str` | use the given TEX tree |
| `--autotree` | automatically determine the TEX tree to use |
| `--environment=str` | use the given tmf environment file |

Conditional directives are:

| | |
|---|---|
| `--iftouched=str,str` | only run when the given files have different time stamps |
| `--ifchanged=str` | only run when the given file has changed (md5 check) |

Special features:

| | |
|---|---|
| `--showenv` | show the environment variables known at runtime |
| `--edit` | open the given file in an editor |

In addition, there are prefixes for filenames:

| | |
|---|---|
| `bin:filename` | expanded name, based on `PATH` environment variable |
| `kpse:filename` | expanded name, based on `kpsewhich` result |
| `rel:filename` | expanded name, backtracking on current path (`. .. ../..`) |
| `env:name` | expanded name, based on environment variable `name` |
| `path:filename` | pathpart of `filename` as located by `kpsewhich` |

## Performance

The performance of the indirect call is of course less than a direct call. You can gain some time by setting the environment variables or by using a small TEX tree.

The script tries to be clever. First it tries to honor a given path, and if that fails it will strip the path part and look on the current path. When this fails, it will consult the environment variables. Then it will use `kpsewhich` and when that fails as well, it will start searching the TEX trees. This may take a while, especially when you have a complete tree, like the one on TEX Live.[XXVIII]

If you want, you can use the built in `kpsewhich` functionality (written in Ruby) by setting the environment variable `KPSEFAST` to `yes`. The built in handler is a bit faster and maintains its own file database. Such a database is generated with:

```
tmftools --reload
```

## Using prefixes

You can also use `texmfstart` to launch other programs that need files in one of the TEX trees:

```
texmfstart --direct xsltproc kpse:somescript.xsl somefile.xml
```

or shorter:

```
texmfstart bin:xsltproc kpse:somescript.xsl somefile.xml
```

In both cases `somescript.xsl` will be resolved and in the second case `bin:` will be stripped. The `--direct` switch and `bin:` prefix tell `texmfstart` not to search for the program, but to assume that it is a binary. The `kpse:` prefix also works for previously mentioned usage.

A convenient way to edit your local context system setup file is the following; we don't need to go to the path where the file resides.

```
texmfstart bin:scite kpse:cont-sys.tex
```

Because editing is happening a lot, you can also say:

```
texmfstart --edit kpse:cont-sys.tex
```

You can set the environment variable `TEXMFSTART_EDITOR` to your favourite editor.

## Conditional processing

A bit obscure feature is triggered with `--iftouched`, for instance:

```
texmfstart --iftouched=normal.pdf,lowres.pdf \
    downsample.rb --verylow normal.pdf lowres.pdf
```

Here, `downsample.rb` is only executed when `normal.pdf` and `lowres.pdf` have a different modification time. After execution, the times are synchronized. This feature is rather handy when you want to minimize runtime. We use it in the resource library tools.

```
texmfstart --iftouched=foo.bar,bar.foo convert_foo_to_bar.rb
```

[XXVIII] On my computer I use multiple trees parallel to the latest TEX Live tree. This results in a not that intuitively and predictable search process. The cover of this manual reflects state of those trees.

A similar option is `ifchanged`:

```
texmfstart --ifchanged=whatever.mp texexec --mpgraphic whatever.mp
```

This time we look at the MD5 checksum, when the sum is changed, `texexec` will be run, otherwise we continue.

## TEX trees

There are a few more handy features built in. The reason for putting those into this launching program is that the sooner they are executed, the less runtime is needed later in the process.

Imagine that you have installed your tree on a network attached storage device. In that case you can say:

```
texmfstart --tree=//nas-1/tex texexec --pdf yourfile
```

There should be a file `setuptex.tmf` in the root of the tree. An example of such a file is part of the ConTEXt distribution (minimal trees). This feature permits you to have several trees alongside and run specific ones. You can also specify additional environments, using `--environment`.

Such an environment file is platform independent and looks as follows. The `%VAR%` variables will be replaced by their meaning, while the `$VAR` variables are left untouched. The = sets a value, while > and < prepend and append the given value to the current value.

```
# author: Hans Hagen - PRAGMA ADE - Hasselt NL - www.pragma-ade.com
#
# usage: texmfstart --tree=f:/minimal/tex ...
#
# this assumes that calling script sets TEXPATH without a trailing
# slash; %VARNAME% expands to the environment variable, $VARNAME
# is left untouched; we also assume that TEXOS is set.

TEXMFMAIN    = %TEXPATH%/texmf
TEXMFLOCAL   = %TEXPATH%/texmf-local
TEXMFFONTS   = %TEXPATH%/texmf-fonts
TEXMFEXTRA   = %TEXPATH%/texmf-extra
TEXMFPROJECT = %TEXPATH%/texmf-project
VARTEXMF     = %TMP%/texmf-var
HOMETEXMF    =

TEXMFOS      = %TEXPATH%/%TEXOS%
# OSFONTDIR  = %SYSTEMROOT%/fonts

TEXMFCNF     = %TEXPATH%/texmf{-local,}/web2c
TEXMF        = {$TEXMFOS,$TEXMFPROJECT,$TEXMFFONTS,
                   $TEXMFLOCAL,$TEXMFEXTRA,!!$TEXMFMAIN}
TEXMFDBS     = $TEXMF

TEXFORMATS   = %TEXMFOS%/web2c/{$engine,}
MPMEMS       = %TEXFORMATS%
TEXPOOL      = %TEXFORMATS%
MPPOOL       = %TEXPOOL%
```

```
PATH        > %TEXMFOS%/bin
PATH        > %TEXMFLOCAL%/scripts/perl/context
PATH        > %TEXMFLOCAL%/scripts/ruby/context

RUBYLIB     > %TEXMFLOCAL%/scripts/ruby/context

TEXINPUTS   =
MPINPUTS    =
MFINPUTS    =
```

When you only want to set a variable that has no value yet, you can use an `?`. These symbols have alternatives as well:

| | | |
|---|---|---|
| `=` | `<<` | assign a value to the variable |
| `?` | `??` | only assign a valuehen the variable is unset |
| `<` | `+=` | append a value to the current value of the variable |
| `>` | `=+` | prepend a value to the current value of the variable |

# F GNU Free Documentation License

The ``**Invariant Sections**'' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ``**Cover Texts**'' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ``**Transparent**'' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ``Transparent'' is called ``**Opaque**''.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ``**Title Page**'' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ``Title Page'' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section ``**Entitled XYZ**'' means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ``**Acknowledgements**'', ``**Dedications**'', ``**Endorsements**'', or ``**History**''.) To ``**Preserve the Title**'' of such a section when you modify the Document means that it remains a section ``Entitled XYZ'' according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this

if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled ``Acknowledgements'', ``Dedications'', or ``History'', the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ``or any later version'' applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.