

ConT_EXt

reference manual

Hans Hagen, Taco Hoekwater

August 16, 2009

This document is typeset using Lua $\mathrm{T}_{\mathrm{E}}\mathrm{X}$.

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ and $\mathrm{AmS}_{\mathrm{E}}\mathrm{X}$ are trademarks of the American Mathematical Society; MetaFont is a trademark of Addison–Wesley Publishing Company; PostScript, Portable Document Format and Acrobat are trademarks of Adobe Systems Incorporated; all other product names are trademarks of their producers.

©1991–2008 Pragma ADE, Taco Hoekwater Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘GNU Free Documentation License’.

Content

Preface

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

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

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

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

For newcomers we advise (??), for the curious (??), and for the impatient (??). ConT_EXt users will not need this literature, unless one wants to program in T_EX, uses special characters, or has to typeset math. Again, we would advise (??).

You may ask yourself if T_EX is not 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, T_EX still does a pretty good job compared to others.

T_EX 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 T_EX

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

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

T_EX 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.

T_EX 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 T_EX, the system can also be qualified as an intentional typesetting system.

FIXME: Next paragraph is messy, and introduces ConT_EXt prematurely

In most documents one can stick to commands that define the structure and leave the typographic details to ConT_EXt. One can concentrate on the content, instead of on makeup; the author can concentrate on his reader and his intentions with the text. In this respect one can classify ConT_EXt as an intentional system. We prefer such a system over a page-oriented system, especially in situations where you have to process bulky documents of with regularly changing content. Furthermore an intentional typesetting system is rather flexible and makes it possible to change layout properties depending on its application. It can also cooperate quite well with other text-processing programs and tools.

1.2 ConT_EXt

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

The functionality of ConT_EXt 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, . . . interfaces are supported). Though ConT_EXt is as (un)stable as any other macro package there are still a great number of wishes. These will be implemented in the spirit of the existing ConT_EXt commands.

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

1.3 Commands

A ConT_EXt document is normally coded in utf or another plain text encoding like ISO Latin1. Inside such a file, the actual document text is interspersed with ConT_EXt commands. These commands tell the system how the text should be typeset. A ConT_EXt 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 it 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 T_EX 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.

A command is often followed by setups and / or argument text. Setups are placed between brackets ([], there may be more than one sets of those). 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).

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

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

ConT_EXt

T_EX

Figure 1.1

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

that's it

Setups in ConT_EXt 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
```

And often keywords or key–value pairs can be passed, that inform ConT_EXt of the users wishes like

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

or

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

The simplest ConT_EXt document is

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

1.4 Running ConT_EXt

TODO: Explain basic use of `texexec` and `context` here, maybe from a text editor or environment like `texworks`.

1.5 Advanced commands

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

```
\definesomething[name]
```

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]
```

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 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{.....}
.....
\beginofAnswer
.....
\endofAnswer
.....
```

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 ?? is typeset this way.

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

You may have noticed that a spacy layout of your ascii text is allowed. In our opinion, this increases readability considerably, but you may of course decide to format your document otherwise. When the ConT_EXt commands in this manual are discussed they are displayed in the following way:

```
\setupfootertexts [.1.] [.2.] [.3.]
                     OPTIONAL OPTIONAL OPTIONAL
1
2
3
```

The command `\setupfootertexts`, which we will discuss in detail in a later chapter, has three arguments of which the first is optional. The first argument defaults to `[text]`. Optional arguments are displayed with the word `OPTIONAL` below the brackets. 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 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

`TEX` is used with `ascii` source files. `ascii` is an international standardized computer alphabet. The `ascii` file with the prescribed extension `tex` is processed by `TEX`. During this process `TEX` produces a file with graphical commands. This file has the extension `dvi`. A machine-specific driver transforms this file into a format that is accepted by photosetters and printers. Usually, `PostScript` drivers are used to produce `PostScript` files.

`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.

A dvi file can be viewed on screen with a dedicated program. For electronic distribution PostScript files can be transformed (distilled) into Portable Document Format (pdf) files. pdf files are of high graphical quality and are also interactive (hyperlinked). ConT_EXt fully supports pdfT_EX, which means that you can generate pdf output directly.

1.8 Texts

1.8.1 Characters

A T_EX text contains ascii characters. Higher ascii values to produce characters like *ë*, *ô* and *ñ* can also be used in this version of T_EX. Some characters in T_EX have a special meaning. 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 T_EX 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.

1.8.2 Paragraphs

T_EX 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.

1.8.3 Boxes

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

When T_EX is processing a document many messages may occur on the screen. Some of these messages relate to overfull or underfull boxes. Horizontal and vertical boxes can be typeset by the T_EX 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 T_EX, because that way one can far more easily write his or her own alternatives to, for instance, chapter headers.

1.8.4 Fonts

T_EX is one of the few typesetting systems that does math typesetting right. To do so T_EX needs a complete fontfamily. This means not only the characters and numbers but also the 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.

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 ?? . For example the linespacing in this document is 14.44502pt.

dimension	meaning	equivalent
pt	point	72.27pt = 1in
pc	pica	1pc = 12pt
in	inch	1in = 2.54cm
bp	big point	72bp = 1in
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 T_EX.

We will often specify layout dimensions in points or centimeters or millimeters. A point is about .35mm. Most dimensions are rather American. The European Didot point is equivalent to $1/2660\text{m} = 3.759398496\text{mm}$.

Next to the mentioned dimension T_EX 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 $1/2\text{em}$, while a `—` (`---`) is one `em`.

1.8.6 Error messages

While processing a document, T_EX generates status messages (what is T_EX doing), warning messages (what could T_EX do better) and error messages (what considers T_EX wrong). An error message is always followed by a `halt` and processing will be stopped. A linenummer 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. T_EX 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

T_EX 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 T_EX version 3.141592 is being used. The final T_EX version number will be π , while MetaFont will become the Euler number e .

ConT_EXt can handle both ϵ -T_EX and pdfT_EX, which are extensions to T_EX. Both are still under development, so we suggest using the latest versions available. This manual is typeset using pdf ϵ T_EX, with ϵ -T_EX version 2.2 and pdfT_EX version 1409.

ConT_EXt is still under development. Macros are continually improved in terms of functionality and processing speed. Improvements are made within existing macros. For example the possibility to produce highly interactive pdf documents has altered some low-level functionality of ConT_EXt but did not alter the interface. We hope that in due time ConT_EXt will be a reasonable complete document processing system, and we hope this manual shows enough of its possibilities. This document was processed with version 2008.05.21 15:21.

1.10 Top ten

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

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

1.11 Warning

ConT_EXt users can define their own commands. These newly defined commands may conflict with plain T_EX or ConT_EXt commands. Therefore it is advisable to use capital characters in your own command definitions.

```
\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 $\text{Con}\text{\TeX}$ t 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 \backslash end .

It is advisable to type the document setups before the \backslash 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. ConT_EXt 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
```

```
*
```

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

```
*
```

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

```
*
```

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

```
*
```

Before a `\start`–`\stop`–pair commands can be added. When a file is not found on the directory ConT_EXt 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
<code>\project name</code>			★	★
<code>\environment name</code>	(★)	(★)	(★)	(★)
<code>\product name</code>	★			(★)
<code>\componentonderdeel name</code>			(★)	(★)

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 ?? 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
```

An example of a project file.


```
\startproduct teacher
\project documents
\component teacher1
\component teacher2
\stopproduct
```

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

```
\startcomponent teacher2
\project documents
\product teacher
... text ...
\stopcomponent
```

Here we see the component.

In most cases working with only `\starttext` and `\stoptext` in combination with `\input` or `\environment` 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 structure is defined properly.

In principal a project file contains only a list of products and environments. If you would process the project file all products will be placed in one document. This is seldom wanted. This manual for example has a project structure. Every part is a product and every chapter is a component. There are several environments that are loaded in the main project file.

Schematically the coherence between files could be displayed as illustrated in figures ??, ?? and ??.

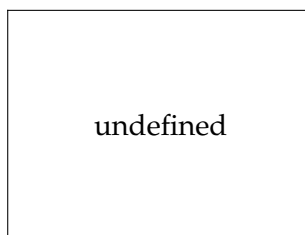


Figure 2.1 An example of project structure.

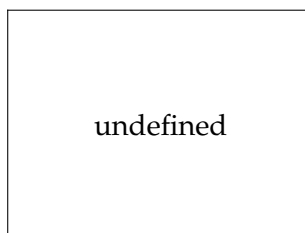


Figure 2.2 An example with only products.



Figure 2.3 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
```

file: `demos.tex`

This file is used to define the products and environments.

```
\startenvironment environ
\setupwhitespace[big]
\setupfootertexts[part][chapter]
\stopenvironment
```

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
```

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
\part{One}
\completecontent
\chapter{First}
..... text .....
\chapter{Second}
..... text .....
\completeindex
\stopcomponent
```

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
\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.

The files `first.tex`, `second.tex` and `example.tex` can be processed separately. As long as there is one product in this project you can also process `project.tex`. 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 $\text{Con}\text{\TeX}t$ 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 T_EX is processing in advance.

2.6 Modes

T_EX 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 T_EX 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 T_EXexec. This Perl script provides ConT_EXt with a command-line-interface. When we want pdf instead of dvi, we can launch T_EXexec 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 T_EXexec. 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 \TeX exec 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, Con \TeX t provide 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, ConT_EXt 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 post-processing 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-tex`

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 ConT_EXt 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 T_EX 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 `i12`) have an associated regime. You can get some insight in what a regime involves by showing it:

```
\showregime[i12]
```

In addition there are a couple of platform dependent ones:

regime	platform
<code>ibm</code>	the old standard msdos code page
<code>win</code>	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 T_EX makes use of the actual `\hsize` (width) and `\vsize` (height). As soon as `\vsize` is exceeded T_EX'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 ??.

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.

Other formats like B0–B9 and C0–C9 are also available. You could also use: `letter`, `legal`, `folio` and `executive`, `envelop 9–14`, `monarch`, `check`, `DL` and `CD`.

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 ConT_EXt 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.

ConT_EXt 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.

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\bff` 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 ??).

The header is located on top of the main text area, and the footer comes after it. Normally, in the header and footer page numbers and running titles are placed. The left and/or right margin are often used for structural components like marginal notes and/or chapter and section numbers.

left

The margins are located in the backspace. Their width has *no* influence on the location of the typesetting area on the page.

right

On the contrary, the height of the header and footer influence the height of the text area. When we talk about the height, we mean the sum of the header, text and footer areas. When one occasionally hides the header or footer, this guarantees a consistent 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).

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

It is possible to trace the setting by using the following commands:

```
\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 the `\dimen`-registers `\textwidth` and `\textheight`, `\makeupwidth` and `\makeupheight`.

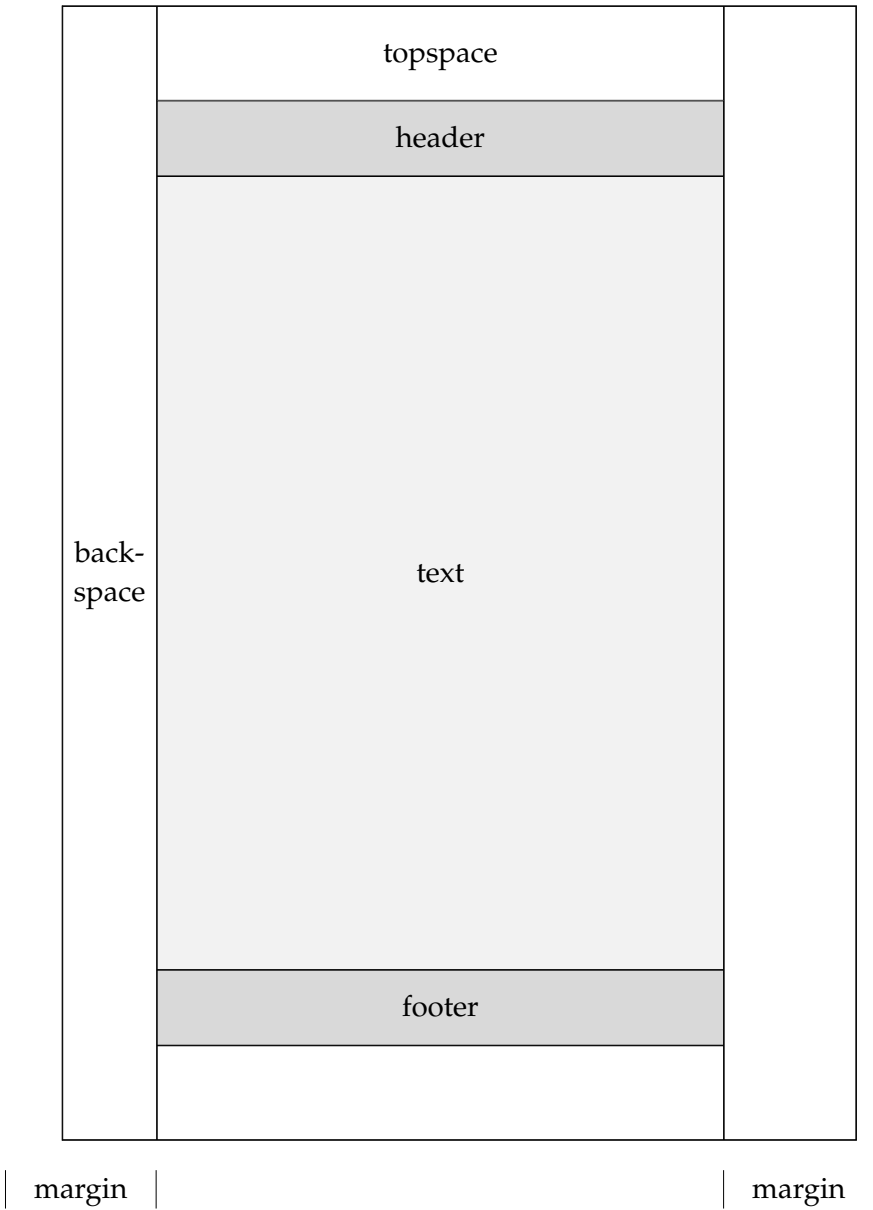


Figure 3.1 The A4 typesetting area and margins (height = header + text + footer).

```
\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           =
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
columnndistance = DIMENSION
grid            = yes no
bottomspace     = DIMENSION
cutspace        = DIMENSION
textdistance    = DIMENSION
textwidth       = NUMBER
textmargin      = DIMENSION
clipoffset      = DIMENSION
page            = IDENTIFIER
paper           = IDENTIFIER
```

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.

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`.

In principal 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 momentarily (just for that page) by typing:

```
\adaptlayout [...1,...] [...2,...]
                OPTIONAL
1  NUMBER
2  height = DIMENSION max
   lines  = NUMBER
```

The use of these commands should be avoided because if you alter your document the adjustment would 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 numbers in the output file.

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

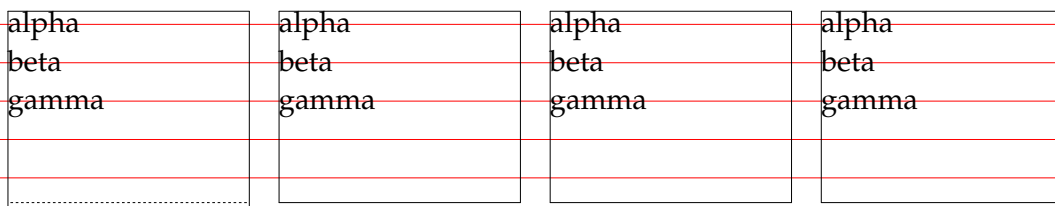
```
\setuplayout [reset]
```

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. Notice 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

alpha	alpha	alpha	alpha
beta	beta	beta	beta
gamma	gamma	gamma	gamma
delta	delta	gamma	

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

² Here! Another footnote.

In those situations we prefer to typeset on a grid.³ Finally, the last footnote!

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:

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. ConT_EXt has a

¹ Hey, watch this. A footnote!

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:

```
\moveongrid[both]
\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 TEXT

```
\moveongrid [...*,...]
                OPTIONAL
```

- * standard top both bottom -top -both -bottom TEXT high middle low page broad
fit depth line reset none DIMENSION

\showgrid [... ¹ ,...] {... ² .}		1
OPTIONAL		2
1	reset top bottom none all lines frame nonumber right left	3
2	TEXT	4
		5
		6
		7
		8
		9
		10
		11
		12
		13
		14
		15
		16
		17
		18
		19
		20
		21
		22
		23
		24
		25
		26
		27
		28
		29
		30
		31
		32
		33
		34
		35
		36
		37
		38
		39
		40
		41
		42

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 ?? and ?? 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 `papersize` 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 ConT_EXt will automatically move front and back side to the correct corner. In figure ?? we show both alternatives.

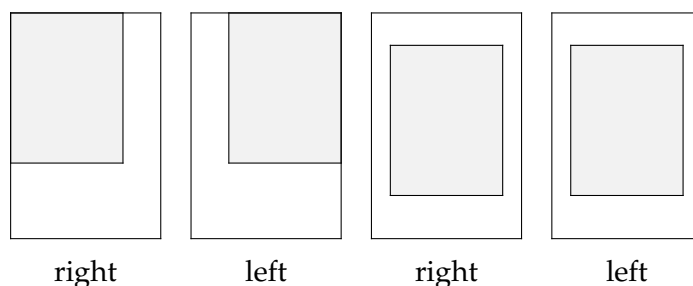


Figure 3.2 Positioning the page on paper for cutting.

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`.

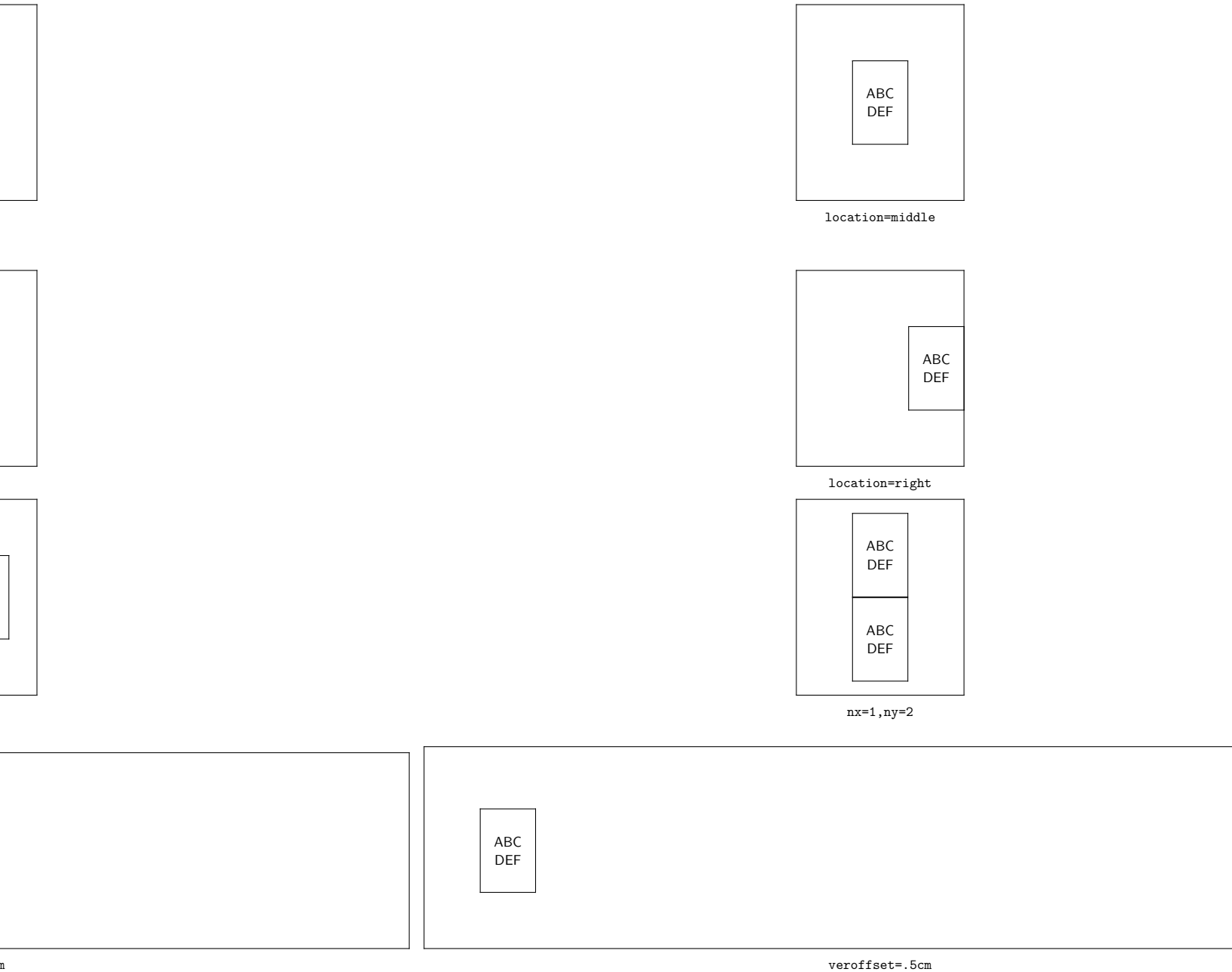


Figure 3.3 Manipulating the page composition with `\setuplayout`.

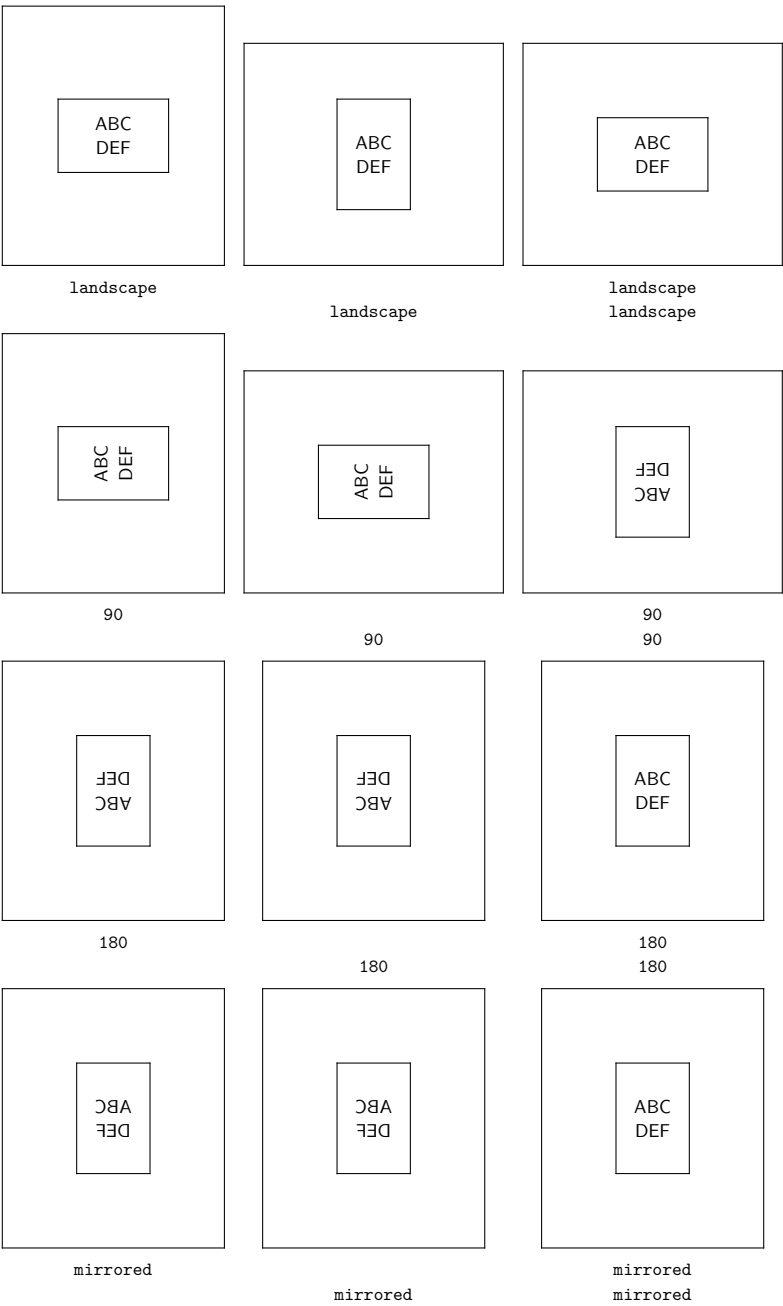


Figure 3.4 Manipulating the page composition with `\setuppapersize`.

	Page design	37
3.7	<pre>\showprint [...¹,...] [...²,...] [...,<u>³</u>,...]</pre> <pre>1 inherits from \setuppapersize</pre> <pre>2 inherits from \setuppapersize</pre> <pre>3 inherits from \setuplayout</pre> <p>You can use \showprint to get an idea of how your print will look. However, it is just a representation of the real page as is shown in the examples above.</p> <pre>\showprint[mirrored][90][location=middle]</pre> <h2>Arranging pages</h2> <p>By means of \setuplayout one can arrange pages on a sheet of paper. A special arrangement for example is that for booklets.</p> <pre>\setuparranging [...*,...]</pre> <pre>* disable 2*16 2*8 2*4 2*2 2**2 2*2*4 2*4*2 2UP 2DOWN 2SIDE 2TOP mirrored</pre> <pre> rotated doublesided negative background 90 180 270</pre> <p>We will show some page arrangements on the next pages. If you want to understand how it really works you should try this yourself one day.</p> <p>The next examples show the cooperation of the commands \setuppapersize, \setuplayout and \setuparranging. Notice how these tests were generated.</p>	
3.8	<h2>Logo types</h2> <p>It is possible to place for example company logos at the top or the bottom of a page. We show some examples on the next pages. It is advisable to define a command for typesetting a logo type.</p> <p>The location of a logo type is defined by:</p> <pre>\definelogo [...¹.] [...².] [...³.] [...,<u>⁴</u>.,...]</pre> <pre>1 IDENTIFIER</pre> <pre>2 top header footer bottom</pre> <pre>3 none page leftedge leftmargin left middle right rightmargin rightedge</pre> <pre>4 command = COMMAND TEXT</pre> <pre> state = start stop</pre>	
	Arranging pages	3

8	9	12	5	6	11	10	7
1	9	13	4	3	14	15	2

Figure 3.5 The 2*8 arrangement.

4	5	3	6
1	8	7	2

Figure 3.6 The 2*4 arrangement.

1	4	3	2
---	---	---	---

Figure 3.7 The 2*2 arrangement.

1	8	2	7	3	6	4	5
---	---	---	---	---	---	---	---

Figure 3.8 The 2UP arrangement.

8	7	6	5
1	2	3	4

Figure 3.9 The 2DOWN arrangement.

All logo types with state=start are automatically typeset on the page. A logo can also be recalled by:

```
\placelogos [...*...]
                OPTIONAL
* IDENTIFIER
```

In that case only the listed logos are typeset.

On this page a few potential locations of logos are shown. Temporarily headers and footers of this manual are suppressed. For example the left logo types are defined by means of:

```
\definologo
  [logo a] [bottom] [left]
  [command=left bottom]
\definologo
  [logo d] [top] [left]
  [command=left top]
\definologo
  [logo g] [footer] [left]
  [command=left footer]
\definologo
  [logo j] [header] [left]
  [command=left header]
\placelogos[logo a,logo b,logo c,logo d]
```

Instead of command we could have chosen text. We define the logo with command because it is evident that we will use the logo more than once. The example is discussed below.

First we define a command that generates a small logo.

```
\def\ContextLogo%
  {\externalfigure[mp-cont.502] [height=24pt,method=mps]}
```

If we want to set this logo at the bottom of every page we type:

```
\definologo
  [small logo] [bottom] [middle]
  [command=\ContextLogo,state=start]
```

This logo is placed at the bottom of every page. In letters however the logos are located on different positions on the paper. Again, we define the bigger logo including all address information. Watch the use of \framed.

```
\def\ContextLetterhead%
  {\hbox
    {\definefont[ContextFont][RegularBold sa 1.5]%
      \ContextFont \setupinterlinespace
      \setupframed
        [align=middle,top=\vfill,bottom=\vfill,
          height=10\bodyfontsize,offset=overlay,frame=off]%
      \framed
        {The\\Con\TeX t\\Chronicle}%
      \externalfigure
        [mp-cont.502] [height=10\bodyfontsize]%
```

The ConTeXt Chronicle

name: dummy
file: mp-cont
state: unknown
Ridderstraat 27\\8061GH Hasselt NL\\pragma@wxs.nl}}

Ridderstraat 27
8061GH Hasselt NL
pragma@wxs.nl

We also define the position on the paper:

```
\definologo  
[big logo] [header] [right]  
[command=\ContextLetterhead]
```

This letterhead logo should appear only on the first page. So we simply say:

```
\placelogos[big logo]
```

You will notice that the smaller logo is not placed at the bottom of the page because the command `\placelogos` typesets only the listed logos and suppresses all other logos.

The big logo needs some space on this page so the content of the letter should be moved to a somewhat lower location. We do this with the command:

```
\blank[force,8\bodyfontsize]
```

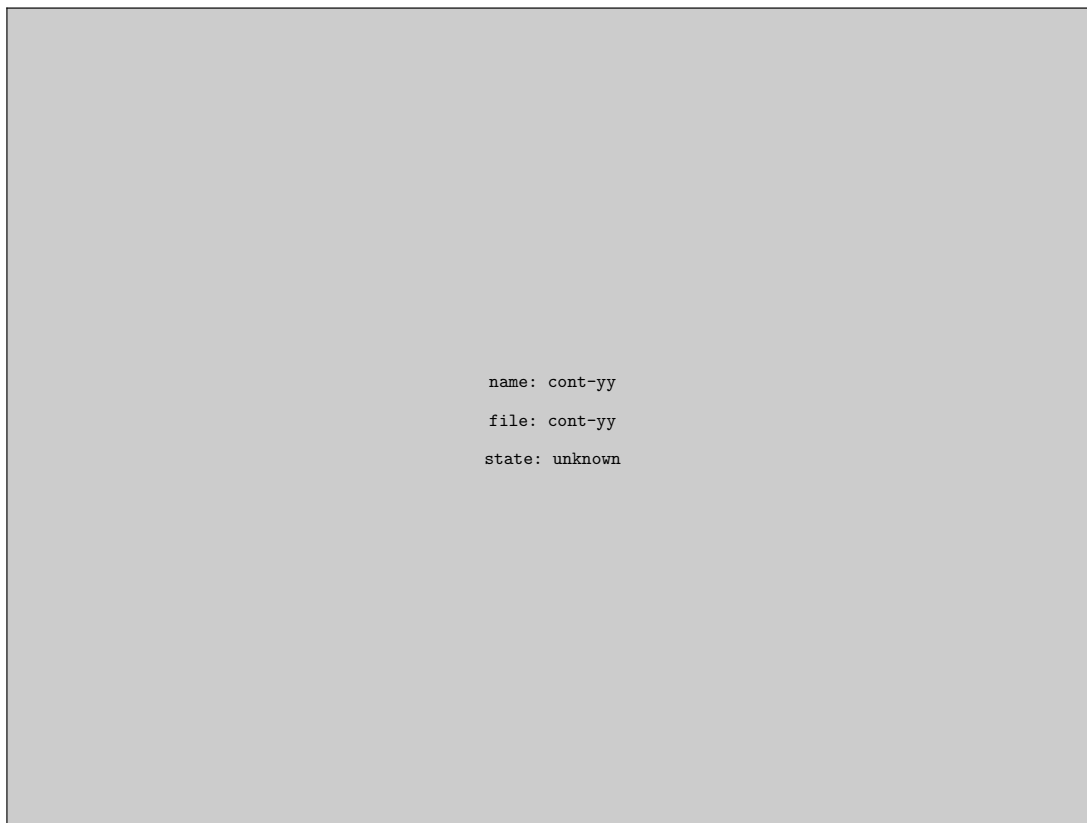


Figure 3.10 The location of header, footer, bottom and top logos on a page.

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:

0. an empty line
1. the \TeX -command `\par` or `\endgraf`
2. the $\text{Con}\text{\TeX}$ t-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 ??).

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 [...*...]  
*
```

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 T_EX in some cases *not* to indent. This is done by:

```
\noindenting
```

We can set up indenting by:

```
\indenting [...*...]  
*
```

The meaning of the setups is described in table ?? . Next to the commands described above we could use the T_EX-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,1left. 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 ConT_EXt takes care of most situations.

T_EX 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 ConT_EXt, so only use these commands when the typeset text looks really bad.

```
\startlinecorrection
```

For example:

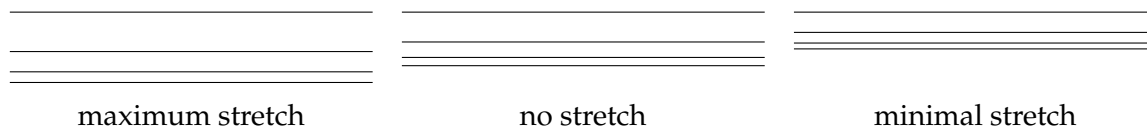
```
\startlinecorrection
\framed{To boxit or not, that's a delicate question!}
\stoplinecorrection
```

One can add vertical spacing with the T_EX 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 T_EX 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 11pt and is downscaled by a few percent. The setup that is used in this document is shown in table ?? . 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%.

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:

setup	value
small	2.70844pt plus 0.9028pt minus 0.9028pt
medium	5.41687pt plus 1.80562pt minus 1.80562pt
big	10.83376pt plus 3.61125pt minus 3.61125pt
line	14.44502pt

Table 4.2 The whitespace values to a 11pt bodyfont.

```
\defineblank [.] [.]
```

```
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
```

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 ConT_EXt.

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

```
\correctwhitespace {...}
```

```
*   TEXT
```

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, then 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 `\hbox{test}` (with strut) with `\hbox{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 ConT_EXt can be made visible with the command:

```
\showstruts
```

4.7 Text in the margin

Texts can be place in the margins with:

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

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 advertisement over here 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.

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  TEXT
```

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

There is also:

```
\inotherrmargin [.1.] [.2.] {.3.}
                  OPTIONAL OPTIONAL
1  + - low
2  REFERENCE
3  TEXT
```

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
to be And just as it is meant to be
that quote will never perish

or not

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:

```
\setupinmargin [...]1 [...]2 [...]
      OPTIONAL
1  left right NUMBER
2  location  = left right both
   style     =
   before    = COMMAND
   after     = COMMAND
   align     =
   line      = NUMBER
   distance  = DIMENSION
   separator = TEXT
   width     = DIMENSION
   distance  = DIMENSION
   stack     = yes no
   inherits from \setupframed
```

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.

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

left
middle
right
yes
no
inner
outer

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

that way we can
move quite some
text into
the margin

With `before` and `after` we can influence margin texts. By 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:

```
\setupinmargin
  [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  TEXT
```

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 `Con \TeX t` 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 ??).

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.

```
\setupmargin[1][align=right, line=1,style=slanted]
\setupmargin[2][align=middle,line=2,style=boldslanted]
\setupmargin[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 {...}
```

```
* TEXT
```

```
\low {...}
```

```
* TEXT
```

```
\lohi [.1.] {...} {.3.}
      OPTIONAL
```

```
1 low
```

```
2 TEXT
```

```
3 TEXT
```

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 ^{high} heels or _{low} heels but your height is still the same.

These commands relate to the `^` and `_` in math mode. In case of larger font sizes 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^{test} test test^{test} test
 test^{test} test test^{test} test
 test^{test} test test^{test} test

4.9 Columns

The T_EX programmer knows that it is not easy to put text in columns. Gratefully a ConT_EXt 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 T_EX has limited capabilities for typesetting

columns. For insiders: columns are produced with the primitives: `\output` and `\vsplit`.

The number of columns is unlimited, however T_EX's 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
    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 T_EX 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 $.75 \times \text{\lineheight}$. 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 T_EX 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 ?? typesetting on a grid is explained. This mechanism works quite well within columns.

<p>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</p>
--

<p>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.</p>
--

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

```
\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 =

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

unknown setup 'start«paragraph»'

The layout can be set up by:

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

1 IDENTIFIER

2 NUMBER each

3 style =

width = DIMENSION

height = DIMENSION

align =

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:

unknown setup '«paragraph»'

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	The student is able to write a detailed planning for the design and construction of a water purification plant.	<ul style="list-style-type: none"> ○ yes ○ no 	<ul style="list-style-type: none"> ○ self study ○ class room ○ 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 ConT_EXt 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 `[|1|p|]` and `[|1|1|1|]` are called a template. The set ups are similar to those of `\starttable` (see in ??).

The default template looks like this: `[|1|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 nohanging hyphenated nothyphenated lessshyphenation 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 {...*}

* TEXT
```

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.


```
\midaligned {...}
```

```
* TEXT
```

```
\rightaligned {...}
```

```
* TEXT
```

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 {...}
```

```
* TEXT
```

When typesetting a paragraph, T_EX tries several alternatives and decides which one to choose based on a system, of penalties. Normally T_EX is very strict, but we can instruct T_EX 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— T_EX 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.

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

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

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

```
\crlf
```

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

```
\startlines
```

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

```
\setuplines [...,*,...]
```

```
*   before      =  COMMAND
    after       =  COMMAND
    inbetween    =  COMMAND
    indenting    =
    space        =  yes  no
```

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

```
Come on, he said, give me a while,
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      =
  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 ?? we will explain how we can refer to a linenumber. The parameters `prefix` and `referencing` can be used to influence that process.

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	a but when fed to \TeX the program
2 random at first sight	b you will be surprised
a bunch of stupid tokens that	c thanks to macros your text too
4 looks less that awful right	d will look quite organized

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

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 ???. 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            =
    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

Table 4.4 setups to `\setuppagenumbering`.

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 ??).

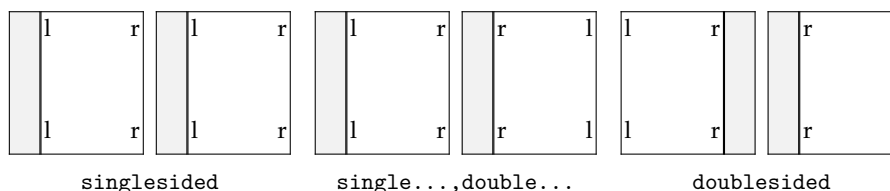


Figure 4.1 Three ways to mirror.

You can assign text to the parameters `left` and `right`. These texts will enclose the pagenum-
ber:

```
\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 com-
mand has an argument and will be executed every time a pagenum 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 pagenum must align with the
texts of the headers and footers.

With `textseparator` you can define a separator between the section and pagenum. Default
this is a –. When the pagenum 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 OPTIONAL OPTIONAL
1
2
3
```

```
\setupfootertexts [.1.] [.2.] [.3.]
                   OPTIONAL OPTIONAL OPTIONAL
1
2
3
```

A great number of arguments can be added. When the first argument is left out it is taken for granted that the footer and header should be placed under or over the pagebody (`text`). 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 key `date` generates a date and `pagenumber` generates the page number. 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. Section numbers 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      =
   leftstyle  =
   rightstyle =
   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

Table 4.5 Setups with `\setupheader` and `\setupfooter`.

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 OPTIONAL OPTIONAL
1
2
3
```

```
\setuptexttexts [.1.] [.2.] [.3.]
                OPTIONAL OPTIONAL OPTIONAL
```

```
1
2
3
```

```
\setupbottomtexts [.1.] [.2.] [.3.]
                  OPTIONAL OPTIONAL OPTIONAL
```

```
1
2
3
```

```
\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[pagenumbers]
\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 lose the pagenumber. A more adequate solution is:

```
\definertext[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.

```
\definertext [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  TEXT
```

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-music⁷, his later works, like 'The Desert Music', are compositions for full orchestra, where the orchestra is extended with a for Reich characteristic rhythm section⁸ 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 seen¹⁰ 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.

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.

```

\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        =
style           =
distance        = DIMENSION
columndistance  = DIMENSION
margindistance  = DIMENSION
n               = NUMBER
numbercommand   = \...#1
textcommand     = \...#1
split           = tolerant strict verystRICT NUMBER
textstyle       =
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.

```

\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.

Nota ¹	Bene ²
Bene ³	Nota ⁴

- ¹ Bene
² Nota
³ Nota
⁴ Bene

Table 4.6 A (latin) table.

```
\startlocalfootnotes
```

```
\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 $\text{T}_{\text{E}}\text{X}$ is using the dimensions of the footnotes to determine the page break. More information can be found in the source code of the $\text{ConT}_{\text{E}}\text{Xt}$ 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^{*}

note: be brief^{**}

note: no notes are even better^{***}

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

```
\setupfootnotedefinition [...,.*,...]
```

```
* inherits from \setupdescriptions
```

^{*} During the development of $\text{ConT}_{\text{E}}\text{Xt}$ the footnote mechanism was one of the first real challenges. And I'm challenged still since I just encountered documents with footnotes within footnotes.

^{**} Why? See note.

^{***} QED.

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:

two	
one three	four
	five

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.

aa	a	a	a	a	a	aa	aa
	a	a	a	a	a		

```
\hbox \vbox \vtop \lbox \cbox \rbox \sbox \tbox \bbox
```

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

dummy file: cow state: unknown <code>\hbox</code>	dummy file: cow state: unknown <code>\vbox</code>	dummy file: cow state: unknown <code>\sbox</code>	name: dummy file: cow state: unknown <code>\tbox</code>	name: dummy file: cow state: unknown <code>\bbox</code>
---	---	---	--	--

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.

The Final Cut

— *Pink Floyd*

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 ConT_EXt and T_EX.

The next example shows that we can use T_EX for more than only the straight forward type-setting. 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 `\switchto-bodyfont` 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.

<pre> name: dummy file: vew1091a state: unknown </pre>	<p>Drink geen water direct uit de kraan!</p> <p>Het drinkwater is tijdelijk niet betrouwbaar. Kook het water voor consumptie ten minste 2 minuten. Zodra het water weer betrouwbaar is, krijgt u bericht.</p> <p>Do not drink water directly from the tap!</p> <p>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.</p>
--	---

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
```

unknown setup 'start<name>makeup'

The first command generates a `\start...stop`-pair between which the new typesetting commands can be typed. By 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*.

T_EX 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.

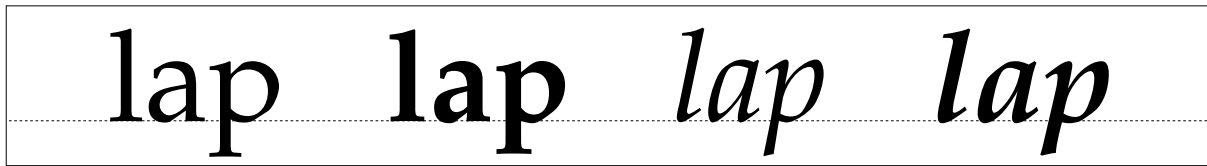


As you can see, quite some design variation is possible. It follows that when fonts from different sources (designers) are intermixed, the result is not always pleasing to look at. The term *font collection* refers to a set of fonts combined together in such a way that the overall appearance on a page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

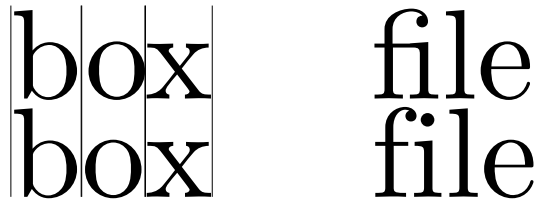
The next example shows an attempt at such a font collection: the fonts were picked such that the glyph sizes and the line thicknesses are roughly the same.



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*. In the example below there are a normal, a bold, an italic, and a bold italic *alternative* of a font.



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 T_EX 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.

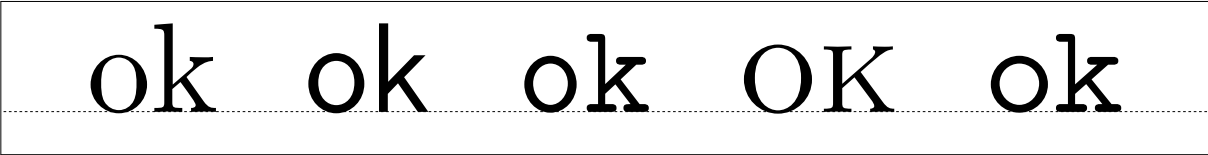
There are many different methods that can be used to classify fonts. There are classification systems based on the period in which the style was first developed; on the characteristics of the font; or the font application, like a newspaper or a book. Often, classification systems mix these characteristics 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 ‘serif’ (glyphs strokes have embellishments at the end), ‘sans serif’ (shapes end abruptly), or ‘mono-spaced’ (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 style. In ConT_EXt you will normally use a reimplement 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.



In this example you see five font styles of Latin Modern: the Roman, Sans, Typewriter, Small-caps and Variable Typewriter. 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 ?? 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 T_EX. In ConT_EXt 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 ConT_EXt. 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 ConT_EXt 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. ConT_EXt's terminology probably differs from what you are accustomed to, especially if you were previously a L^AT_EX 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 collection of fonts, like Lucida or Computer Modern Roman, a *family*. Within such a family, the members can be grouped according to characteristics. 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 alternative classifications, but they all refer to the presence of serifs and the glyphs having equal widths. 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 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.

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 ??.

<code>\rm</code>	serif, regular, roman, rm
<code>\ss</code>	sans, support, sansserif, ss
<code>\tt</code>	mono, type, teletype, tt
<code>\hw</code>	handwritten, hw
<code>\cg</code>	calligraphic, cg
–	mm

Table 5.2 Font style switching commands

The ‘handwritten’ and ‘calligraphic’ font styles are sometimes useful when dealing with very elaborate document layout definitions. In the ConT_EXt 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.

5.3.2 Font alternative switching

The alternatives within a style are given in table ??. 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, ConT_EXt 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.

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

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

Each of the ordered alphabetic suffixes a, b, ... select a somewhat larger actual font than the previous one. The x and xx suffixes select smaller and yet smaller versions.


```

\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 ConT_EXt that you prefer mediaeval or old-style numbers as in 139 over 139.

```

\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’ switches mentioned in table ?? 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 ConT_EXt 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 method 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 ??.

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, ConT_EXt 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 T_EX 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 ConT_EXt 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 ?? and ??, but a few predefined ones have not been mentioned yet. These are displayed in table ??, together with the commands they execute. As is normal in ConT_EXt, you can extend the list of accepted keywords by defining your own. This will be explained in section ?? in the next chapter.

<code>\tt</code>	type, mono
<code>\ttx</code>	smalltype
<code>\ss</code>	sans, sansserif
<code>\ss \bf</code>	sansbold
<code>\setsmallbodyfont</code>	smallbodyfont
<code>\setbigbodyfont</code>	bigbodyfont
<code>\smallcapped</code>	cap, capital
<code>\WORD</code>	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 ConT_EXt `\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 ?? for more details:

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

5.5 Line spacing

In T_EX linespacing is determined by a number of variable dimensions like `\topskip`, `\parskip` and `\baselineskip`. However, in ConT_EXt 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:

$$\blacksquare + \blacksquare = \blacksquare$$

This sum is in ConT_EXt 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 than the sum of the height and depth of each separate line. This distance is called the baseline distance (`\baselineskip`) and is in this document 14.44502pt. 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
```

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.

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

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

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). Con \TeX t provides the following commands for changing both upper- and lowercase characters into capitals.

```
\cap {...}
```

```
* TEXT
```

```
\Cap {...}
```

```
* TEXT
```

```
\CAP {...}
```

```
* TEXT
```

```
\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 {...}
```

```
* TEXT
```

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 ?? (“??”).

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 ConT_EXt 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 font sizes 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 ?? 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 ?? 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’.

ConT_EXt 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 [...] [...] [...]
                  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 pdfT_EX 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 ???. For now, please take for granted the fact that pdfT_EX 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 X_YT_EX 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 X_YT_EX 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 ??

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 pdf \TeX	Loaded by \XeTeX	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`.

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.

The predefined typescripts, the typefaces they define, the files in which they are contained in the ConT_EXt distribution, and the encodings they support in MkII mode are listed in table ??.

In the following section there is a table (??) 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]
\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]

\definetypescriptface[palatino] [rm] [serif][palatino] [default]
\definetypescriptface[palatino] [ss] [sans] [modern] [default] [rscale=1.075]
\definetypescriptface[palatino] [tt] [mono] [modern] [default] [rscale=1.075]
\definetypescriptface[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`.

```
\definetypface [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 `\definetypface` are pointers to already declared font sets; these are defined elsewhere. Table ?? 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 `\definetypface`).

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 `\definetypface`, 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 ConT_EXt; these are all listed in the four tables ??, ??, ??, and ??.

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 `\definetypface` 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.

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 bfm ^{math}
lucidaboldmath	lucida	lucida	lucida	lucida bold ^{math}
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.

There are four possible keys in the sixth argument:

key	default value	explanation
r ^{scale}	1	a scaling factor for this typescript relative to the selected body font size

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

Selecting body fonts

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
			iwona-cond	type-otf	sans
ams	type-otf	math	iwona-light-cond	type-otf	sans
pagella	type-otf	serif	iwona-medium-cond	type-otf	sans
termes	type-otf	serif	iwona-heavy-cond	type-otf	sans
bonum	type-otf	serif		type-otf	sans, math
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-otf	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-otf	serif
optima-nova	type-hgz	sans	palatino	type-otf	serif
optima-nova-os	type-hgz	sans	helvetica	type-otf	sans
optima-nova-light	type-hgz	sans	courier	type-otf	mono
optima-nova-medium	type-hgz	sans	hoefler	type-otf	serif
palatino-nova	type-hgz	serif	lucida	type-otf	sans
palatino-nova-os	type-hgz	serif	optima	type-otf	sans
palatino-nova-light	type-hgz	serif	gillsans	type-otf	sans
palatino-nova-medium	type-hgz	serif	gillsanslt	type-otf	sans
palatino-sans	type-hgz	sans	zapfino	type-otf	handwriting, serif
palatino-informal	type-hgz	sans	applechancery	type-otf	calligraphy, serif
melior	type-hgz	serif	timesnewroman	type-otf	serif
– all four-variant fonts –	type-otf	Xserif	arial	type-otf	sans
– all four-variant fonts –	type-otf	Xsans	lucida	type-otf	serif, sans, mono, handwriting, fax, calligraphy
– all four-variant fonts –	type-otf	Xmono			

Table 5.12 The predefined body font identifiers for commercial OpenType fonts

<code>encoding</code>	<code>\defaultencoding</code>	the encoding for the typeface, normally inherited from the typescript automatically
<code>features</code>		this applies a predefined font feature set (see section ??)
<code>text</code>		sets up the forced math text style

If you look closely, in table ?? you will notice three very special items: `Xserif`, `Xsans` and `Xmono`. These belong to a special \TeX -only trick called ‘wildcard typescripts’.

\TeX 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 \TeX 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.] [...,.3.,...]
                        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            =
```

The first argument is optional, and specifies 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 ConT_EXt 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 \tfc (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 T_EX 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 X_YT_EX and LuaT_EX, 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 (X_YTeX 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=dfnt,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.

<code>compose</code>	Use fallback composition in MkIV (experimental, undocumented)
<code>protrusion</code>	Character protrusion in MkIV (see section ??)
<code>expansion</code>	Character expansion in MkIV (see section ??)
<code>script</code>	An OpenType script identifier
<code>language</code>	An OpenType script language identifier
<code>tlig</code>	A virtual feature for legacy (TeX-style) automatic ligatures (for compatibility, there is an alias for this key called <code>texligatures</code>)
<code>trep</code>	A virtual feature for legacy (TeX-style) automatic ligatures (for compatibility, there is an alias for this key called <code>texquotes</code>) (only works in MkIV)
<code>mode</code>	Processing mode for MkIV. <code>node</code> and <code>base</code> allowed, <code>base</code> is default
<code><tag></code>	Any OpenType feature tag is acceptable, but in MkIV only a ‘known’ subset actually has any effect, and then only in <code>node</code> mode. This list is given in table ??. In X _Y TeX, processing depends on the internal subengine that is used by X _Y TeX, 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 X_YTeX (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 IDENTIFIERS 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]`

[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:

aalt	Access All Alternates	ital	Italics	size	Optical Size
abvf	Above-Base Forms	jalt	Justification Alternatives	smcp	Small Capitals
abvm	Above-Base Mark Positioning	jp04	JIS2004 Forms	smp1	Simplified Forms
		jp78	JIS78 Forms	ss01	Stylistic Set 1
abvs	Above-Base Substitutions	jp83	JIS83 Forms	ss02	Stylistic Set 2
afrc	Alternative Fractions	jp90	JIS90 Forms	ss03	Stylistic Set 3
akhn	Akhands	kern	Kerning	ss04	Stylistic Set 4
blwf	Below-Base Forms	lfbd	Left Bounds	ss05	Stylistic Set 5
blwm	Below-Base Mark Positioning	liga	Standard Ligatures	ss06	Stylistic Set 6
		ljmo	Leading Jamo Forms	ss07	Stylistic Set 7
blws	Below-Base Substitutions	lnum	Lining Figures	ss08	Stylistic Set 8
c2pc	Petite Capitals From Capitals	loc1	Localized Forms	ss09	Stylistic Set 9
		mark	Mark Positioning	ss10	Stylistic Set 10
c2sc	Small Capitals From Capitals	medi	Medial Forms	ss11	Stylistic Set 11
		med2	Medial Forms #2	ss12	Stylistic Set 12
calt	Contextual Alternates	mgrk	Mathematical Greek	ss13	Stylistic Set 13
case	Case-Sensitive Forms	mkmk	Mark to Mark Positioning	ss14	Stylistic Set 14
ccmp	Glyph Composition/Decomposition	mset	Mark Positioning via Substitution	ss15	Stylistic Set 15
cjct	Conjunct Forms			ss16	Stylistic Set 16
clig	Contextual Ligatures	nalt	Alternate Annotation Forms	ss17	Stylistic Set 17
cpsp	Capital Spacing	nlck	NLC Kanji Forms	ss18	Stylistic Set 18
cswh	Contextual Swash	nukt	Nukta Forms	ss19	Stylistic Set 19
curs	Cursive Positioning	numr	Numerators	ss20	Stylistic Set 20
dflt	Default Processing	onum	Old Style Figures	subs	Subscript
dist	Distances	opbd	Optical Bounds	supr	Superscript
dlig	Discretionary Ligatures	ordn	Ordinals	swsh	Swash
dnom	Denominators	ornm	Ornaments	titl	Titling
expt	Expert Forms	palt	Proportional Alternate Width	tjmo	Trailing Jamo Forms
falt	Final glyph Alternates	pcap	Petite Capitals	tnam	Traditional Name Forms
fin1	Terminal Forms	pnum	Proportional Figures	tnum	Tabular Figures
fin2	Terminal Forms #2	pref	Pre-base Forms	trad	Traditional Forms
fin3	Terminal Forms #3	pres	Pre-base Substitutions	twid	Third Widths
frac	Fractions	pstf	Post-base Forms	unic	Unicase
fwid	Full Width	psts	Post-base Substitutions	valt	Alternate Vertical Metrics
half	Half Forms	pwid	Proportional Widths	vatu	Vattu Variants
haln	Halant Forms	qwid	Quarter Widths	vert	Vertical Writing
halt	Alternate Half Width	rand	Randomize	vhal	Alternate Vertical Half Metrics
hist	Historical Forms	rkrf	Rakar Forms		
hkna	Horizontal Kana Alternates	rlig	Required Ligatures	vjmo	Vowel Jamo Forms
hlig	Historical Ligatures	rphf	Reph Form	vkna	Vertical Kana Alternates
hngl	Hangul	rtbd	Right Bounds	vkern	Vertical Kerning
hojo	Hojo Kanji Forms	rtla	Right-To-Left Alternates	vpal	Proportional Alternate Vertical Metrics
hwid	Half Width	ruby	Ruby Notation Forms		
init	Initial Forms	salt	Stylistic Alternates	vrt2	Vertical Rotation
isol	Isolated Forms	sinf	Scientific Inferiors	zero	Slashed Zero

Table 5.13 The OpenType features that are understood by MkIV in mode=node processing mode

[palatino] [12pt]							
text	script	scriptscript	x	xx	small	big	interlinespace
20.7pt	14.4pt	12pt	17.3pt	14.4pt	17.3pt	20.7pt	not set
17.3pt	12.1pt	8.6pt	13.8pt	10.3pt	13.8pt	20.7pt	not set
14.4pt	10pt	7.2pt	11.5pt	8.6pt	11.5pt	17.2pt	not set
12pt	8.3pt	6pt	9.6pt	7.2pt	9.6pt	14.3pt	not set
11pt	7.6pt	5.5pt	8.8pt	6.6pt	8.8pt	13.1pt	not set
10pt	6.9pt	5pt	8pt	6pt	8pt	11.9pt	not set
9pt	6.2pt	4.5pt	7.2pt	5.4pt	7.2pt	10.7pt	not set
8pt	5.5pt	4pt	6.4pt	4.8pt	6.4pt	9.5pt	not set
7pt	4.8pt	3.5pt	5.6pt	4.2pt	5.6pt	8.3pt	not set
6pt	4.1pt	3pt	4.8pt	3.6pt	4.8pt	7.1pt	not set
5pt	3.4pt	2.5pt	4pt	3pt	4pt	5.9pt	not set
4pt	2.7pt	2pt	3.2pt	2.4pt	3.2pt	4.7pt	not set

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.

```

 $\text{\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\rm 6x^2}$ 
 $\text{\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\tf 6x^2}$ 
 $\text{\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\bf 6x^2}$ 
 $\text{\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\sl 6x^2}$ 

```

When this is typeset you see this:

$$\begin{aligned}
 x^2 + \mathbf{x}^2 + x^2 + x^2 + \mathbf{x}^2 + \mathbf{x}^2 &= 6x^2 \\
 x^2 + \mathbf{x}^2 + x^2 + x^2 + \mathbf{x}^2 + \mathbf{x}^2 &= 6x^2 \\
 x^2 + \mathbf{x}^2 + x^2 + x^2 + \mathbf{x}^2 + \mathbf{x}^2 &= \mathbf{6x^2} \\
 x^2 + \mathbf{x}^2 + x^2 + x^2 + \mathbf{x}^2 + \mathbf{x}^2 &= \mathit{6x^2}
 \end{aligned}$$

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.

```
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2
x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^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 ?? 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	12	12	12	12	12
M	M	M	M	M	M
H	H	H	H	H	H

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 ?? 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 T_EX do so because of the quality of the output it produces. pdfT_EX (and through inheritance LuaT_EX as well) contains a few extensions to the typesetting engine that make the output even better than the results achieved by Knuth’s original T_EX. Although the extensions are made available by pdfT_EX, 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 microtypography and the hz-program" in Electronic Publishing, vol 6 (3), 1993.

[file zapf does not exist]

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 T_EX considers to be a space. This example also demonstrates that T_EX does not have spaces, but stretches the white area between words to suit its demands. When breaking lines, T_EX'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 receive their knowledge and information about the rules of typography from books, from computer magazines or the in-

struction 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.

This time we have enabled pdfT_EX'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 pdfT_EX as font adjusting, we will use the short qualification hz since this is how it is called in the pdfT_EX community.

First, here is again the same example text that was used in the previous section, typeset using normal T_EX-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), ConT_EXt 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 ConT_EXt as the user-level interface. The ConT_EXt 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:

<code>hanging</code>	turns on character protrusion
<code>nohanging</code>	turns off character protrusion
<code>hz</code>	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 ConT_EXt 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]
\definetypface [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 ConT_EXt 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.

<code>type</code>	the type of this font handling feature, for use by <code>\setupalign</code>
<code>right</code>	used by <code>type=hanging</code> , default 1
<code>left</code>	used by <code>type=hanging</code> , default 1
<code>factor</code>	used by <code>type=spacing</code> , default 1
<code>min</code>	used by <code>type=hz</code> , default 20
<code>max</code>	used by <code>type=hz</code> , default 20
<code>step</code>	used by <code>type=hz</code> , 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 pdf \TeX are not present in Lua \TeX , and there is no replacement yet. The font handling `no_ligs` 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 pdf \TeX . If you are exclusively using X \TeX 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 preferred 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 T _E X font encoding	?
t5	an encoding dedicated to vietnamese (many (double) accents)	okay
texnansi	a combination of T _E X and Adobe standard encoding	okay

These encodings are font related as is demonstrated in figure ??, ??, ??, and ??. Here we used the `\showfont` command.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
000	00001	01	002	02	003	03	004	04	005	05	006	06	007	07	010	08
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
020	10	021	11	022	12	023	13	024	14	025	15	026	16	027	17	030
32	33	34	35	36	37	38	39	40	41	42	43	44	hyph	45	46	47
040	20	041	21	042	22	043	23	044	24	045	25	046	26	047	27	050
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
060	30	061	31	062	32	063	33	064	34	065	35	066	36	067	37	070
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
100	40	101	41	102	42	103	43	104	44	105	45	106	46	107	47	110
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
120	50	121	51	122	52	123	53	124	54	125	55	126	56	127	57	130
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
140	60	141	61	142	62	143	63	144	64	145	65	146	66	147	67	150
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
160	70	161	71	162	72	163	73	164	74	165	75	166	76	167	77	170
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
200	80	201	81	202	82	203	83	204	84	205	85	206	86	207	87	210
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
220	90	221	91	222	92	223	93	224	94	225	95	226	96	227	97	230
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
240	a0	241	a1	242	a2	243	a3	244	a4	245	a5	246	a6	247	a7	250
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
260	b0	261	b1	262	b2	263	b3	264	b4	265	b5	266	b6	267	b7	270
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
300	c0	301	c1	302	c2	303	c3	304	c4	305	c5	306	c6	307	c7	310
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
320	d0	321	d1	322	d2	323	d3	324	d4	325	d5	326	d6	327	d7	330
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
340	e0	341	e1	342	e2	343	e3	344	e4	345	e5	346	e6	347	e7	350
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
360	f0	361	f1	362	f2	363	f3	364	f4	365	f5	366	f6	367	f7	370
371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387

name: ec-lmr10 at 11.Opt encoding: ec mapping: ec handling: default

Figure 5.1 The Latin Modern Roman font in ec encoding.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	0001	01002	02003	03004	04005	05006	06007	07010	08011	09012	0a013	0b014	0c015	0d016	0e017	0f018
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
020	10021	11022	12023	13024	14025	15026	16027	17030	18031	19032	1a033	1b034	1c035	1d036	1e037	1f038
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
040	20041	21042	22043	23044	24045	25046	26047	27050	28051	29052	2a053	2b054	2c055	2d056	2e057	2f058
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
060	30061	31062	32063	33064	34065	35066	36067	37070	38071	39072	3a073	3b074	3c075	3d076	3e077	3f078
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
100	40101	41102	42103	43104	44105	45106	46107	47110	48111	49112	4a113	4b114	4c115	4d116	4e117	4f118
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
120	50121	51122	52123	53124	54125	55126	56127	57130	58131	59132	5a133	5b134	5c135	5d136	5e137	5f138
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
140	60141	61142	62143	63144	64145	65146	66147	67150	68151	69152	6a153	6b154	6c155	6d156	6e157	6f158
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
160	70161	71162	72163	73164	74165	75166	76167	77170	78171	79172	7a173	7b174	7c175	7d176	7e177	7f178
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
200	80201	81202	82203	83204	84205	85206	86207	87210	88211	89212	8a213	8b214	8c215	8d216	8e217	8f218
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
220	90221	91222	92223	93224	94225	95226	96227	97230	98231	99232	9a233	9b234	9c235	9d236	9e237	9f238
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
240	a0241	a1242	a2243	a3244	a4245	a5246	a6247	a7250	a8251	a9252	aa253	ab254	ac255	ad256	ae257	af258
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
260	b0261	b1262	b2263	b3264	b4265	b5266	b6267	b7270	b8271	b9272	ba273	bb274	bc275	bd276	be277	bf278
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
300	c0301	c1302	c2303	c3304	c4305	c5306	c6307	c7310	c8311	c9312	ca313	cb314	cc315	cd316	ce317	cf318
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
320	d0321	d1322	d2323	d3324	d4325	d5326	d6327	d7330	d8331	d9332	da333	db334	dc335	dd336	de337	df338
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
340	e0341	e1342	e2343	e3344	e4345	e5346	e6347	e7350	e8351	e9352	ea353	eb354	ec355	ed356	ee357	ef358
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
360	f0361	f1362	f2363	f3364	f4365	f5366	f6367	f7370	f8371	f9372	fa373	fb374	fc375	fd376	fe377	ff378

name: texnansi-lmr10 at 11.0pt encoding: texnansi mapping: texnansi handling: default

Figure 5.2 The Latin Modern Roman font in texnansi 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.

```
name: qx-lmr10 at 11.0pt encoding: qx mapping: qx handling: default
```

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 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 Con \TeX t handles this for you automatically.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	0001	01002	02003	03004	04005	05006	06007	07010	08011	09012	0a013	0b014	0c015	0d016	0e017	0f018
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
020	10021	11022	12023	13024	14025	15026	16027	17030	18031	19032	1a033	1b034	1c035	1d036	1e037	1f038
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
040	20041	21042	22043	23044	24045	25046	26047	27050	28051	29052	2a053	2b054	2c055	2d056	2e057	2f058
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
060	30061	31062	32063	33064	34065	35066	36067	37070	38071	39072	3a073	3b074	3c075	3d076	3e077	3f078
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
100	40101	41102	42103	43104	44105	45106	46107	47110	48111	49112	4a113	4b114	4c115	4d116	4e117	4f118
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
120	50121	51122	52123	53124	54125	55126	56127	57130	58131	59132	5a133	5b134	5c135	5d136	5e137	5f138
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
140	60141	61142	62143	63144	64145	65146	66147	67150	68151	69152	6a153	6b154	6c155	6d156	6e157	6f158
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
160	70161	71162	72163	73164	74165	75166	76167	77170	78171	79172	7a173	7b174	7c175	7d176	7e177	7f178
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
200	80201	81202	82203	83204	84205	85206	86207	87210	88211	89212	8a213	8b214	8c215	8d216	8e217	8f218
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
220	90221	91222	92223	93224	94225	95226	96227	97230	98231	99232	9a233	9b234	9c235	9d236	9e237	9f238
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
240	a0241	a1242	a2243	a3244	a4245	a5246	a6247	a7250	a8251	a9252	aa253	ab254	ac255	ad256	ae257	af258
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
260	b0261	b1262	b2263	b3264	b4265	b5266	b6267	b7270	b8271	b9272	ba273	bb274	bc275	bd276	be277	bf278
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
300	c0301	c1302	c2303	c3304	c4305	c5306	c6307	c7310	c8311	c9312	ca313	cb314	cc315	cd316	ce317	cf318
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
320	d0321	d1322	d2323	d3324	d4325	d5326	d6327	d7330	d8331	d9332	da333	db334	dc335	dd336	de337	df338
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
340	e0341	e1342	e2343	e3344	e4345	e5346	e6347	e7350	e8351	e9352	ea353	eb354	ec355	ed356	ee357	ef358
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256
360	f0361	f1362	f2363	f3364	f4365	f5366	f6367	f7370	f8371	f9372	fa373	fb374	fc375	fd376	fe377	ff378

name: t5-lmr10 at 11.0pt encoding: t5 mapping: t5 handling: default

Figure 5.4 The Latin Modern Roman font in t5 encoding.

[illegible]

Figure 5.5 Output of `\showaccents` for the current (palatino) font in pdfTeX

With `\showcharacters`, you get a list of named characters (and glyphs) as known to the system. Note: the following table will look different in each of the three typesetting engines.

ec pplr8t at 11.0pt: composed bottom char raw			
,	textcomma	™	trademark
.	textperiod	§	sectionmark
´	textacute	¶	paragraphmark
.	textbottomdot	¼	onequarter
˘	textbreve	½	onehalf
ˇ	textcaron	¾	threequarter
ˆ	textcedilla	¹	onesuperior
^	textcircumflex	²	twosuperior
¨	textdiaeresis	³	threesuperior
˙	textdotaccent	¢	textcent
`	textgrave	■	textcurrency
¨	texthungarumlaut	\$	textdollar
-	textmacron	€	texteuro
ˆ	textogonek	f	textflorin
°	textring	£	textsterling
~	texttilde	¥	textyen
,	textbottomcomma	ª	ordfeminine
ı	dotlessi	º	ordmasculine
ı	dotlessj	%	percent
I	dotlessI	‰	perthousand
J	dotlessJ	-	softhyphen
-	endash	·	periodcentered
—	emdash		compoundwordmark
æ	aeligature	^	textasciicircum
Æ	AEligature	~	textasciitilde
ij	ijligature	/	textslash
IJ	IJligature	\	textbackslash
œ	oeligature	{	textbraceleft
Œ	OEligature	}	textbraceright
ß	ssharp	_	textunderscore
Š	Ssharp	·	textvisiblespace
þ	thorn		textbrokenbar
Þ	Thorn	•	textbullet
ð	eth	†	textdag
Ð	Eth	‡	textddag
ı	exclamdown	°	textdegree
¿	questiondown	÷	textdiv
©	copyright	…	textellipsis
®	registered	/	textfraction
¬	textlognot		
-	textminus		
μ	textmu		
×	textmultiply		
±	textpm		
"	quotedbl		
„	quotedblbase		
“	quotedblleft		
”	quotedblright		
‘	quotesingle		
,	quotesinglebase		
’	quotelleft		
’	quoteright		
<	guilsingleleft		
>	guilsingleright		
«	leftguillemot		
»	rightguillemot		
Â	Acircumflex		
â	acircumflex		
Ĉ	Ccircumflex		
ĉ	ccircumflex		
Ê	Ecircumflex		
ê	ecircumflex		
Ĝ	Gcircumflex		
ĝ	gcircumflex		
Ĥ	Hcircumflex		
ĥ	hcircumflex		
Î	Icircumflex		
î	icircumflex		
Ĵ	Jcircumflex		
ĵ	jcircumflex		
Ô	Ocircumflex		
ô	ocircumflex		
Ŝ	Scircumflex		
ŝ	scircumflex		
Û	Ucircumflex		
û	ucircumflex		
Ŵ	Wcircumflex		
ŵ	wcircumflex		

Ŷ	Ycircumflex	É	Eacute	Ō	Omacron
ŷ	ycircumflex	é	eacute	ō	omacron
À	Agrave	Í	Iacute	Ū	Umacron
à	agrave	í	iacute	ū	umacron
È	Egrave	Ĺ	Lacute	Ç	Ccedilla
è	egrave	ĺ	lacute	ç	ccedilla
Ì	Igrave	Ń	Nacute	Ꞥ	Kcedilla
ì	igrave	ń	napute	ꞥ	kcedilla
Ò	Ograve	Ó	Oacute	Ꞧ	Lcedilla
ò	ograve	ó	oacute	ꞧ	lcedilla
Ù	Ugrave	Ř	Racute	Ꞩ	Ncedilla
ù	ugrave	ř	racute	ꞩ	ncedilla
Ÿ	Ygrave	Ś	Sacute	Ɦ	Rcedilla
Ź	ygrave	ś	sacute	Ɜ	rcedilla
Ā	Atilde	Ú	Uacute	Ş	Scedilla
ā	atilde	ú	uacute	ş	scedilla
Ĩ	Itilde	Ý	Yacute	Ț	Tcedilla
ĩ	itilde	ý	yacute	ț	tcedilla
Ñ	Ntilde	Ž	Zacute	Ő	Ohungarumlaut
ñ	ntilde	ž	zacute	ő	ohungarumlaut
Õ	Otilde	đ	dstroke	Ű	Uhungarumlaut
õ	otilde	Đ	Dstroke	ű	uhungarumlaut
Ŭ	Utilde	H	Hstroke	Ą	Aogonek
ŭ	utilde	h	hstroke	ą	aogonek
Ỹ	Ytilde	T	Tstroke	Ę	Eogonek
ỹ	ytilde	t	tstroke	ę	eogonek
Ä	Adiaeresis	Ć	Cdotaccent	Į	Iogonek
ä	adiaeresis	ć	cdotaccent	į	iogonek
Ė	Ediaeresis	Ė	Edotaccent	Ų	Uogonek
ė	ediaeresis	ė	edotaccent	u	uogonek
Į	Idiaeresis	Ġ	Gdotaccent	Å	Aring
į	idiaeresis	ġ	gdotaccent	å	aring
Ö	Odiaeresis	İ	Idotaccent	Ů	Uring
ö	odiaeresis	ı	idotaccent	ů	uring
Ü	Udiaeresis	Ž	Zdotaccent	Ă	Abreve
ü	udiaeresis	ž	zdotaccent	ă	abreve
Ỳ	Ydiaeresis	Ā	Amacron	Ě	Ebreve
ỳ	ydiaeresis	ā	amacron	ě	ebreve
Á	Aacute	Ē	Emacron	Ĝ	Gbreve
á	aacute	ē	emacron	ğ	gbreve
Ć	Cacute	Ī	Imacron	İ	Ibreve
ć	cacute	ī	imacron	ı	ibreve

Œ	Obreve	l	lcommaaccent	Ǻ	Abrevetilde
ø	obreve	Ł	Lcommaaccent	ǻ	Abrevehook
Ů	Ubreve	Ě	Etilde	ǣ	abrevegrave
ů	ubreve	ě	etilde	ǵ	abreveacute
Č	Ccaron	Ǻ	Ahook	Ƕ	abrevetilde
č	ccaron	ǻ	ahook	Ƿ	abrevehook
Ǻ	Dcaron	Ǽ	Ehook	Ǻ	Adotbelow
ǻ	dcaron	ǽ	ehook	ǻ	adotbelow
Ǽ	Ecaron	ǿ	Ihook	Ǽ	Edotbelow
ǽ	ecaron	ǿ	ihook	ǽ	edotbelow
Ǻ	Lcaron	Ǿ	Ohook	ǿ	Idotbelow
ǻ	lcaron	ǿ	ohook	ǿ	idotbelow
Ǻ	Ncaron	Ǿ	Uhook	Ǿ	Odotbelow
ǻ	ncaron	ǿ	uhook	ǿ	odotbelow
Ǻ	Rcaron	ǿ	Yhook	ǿ	Udotbelow
ǻ	rcaron	ǿ	yhook	ǿ	udotbelow
Ǻ	Scaron	Ǻ	Acircumflexgrave	ǿ	Ydotbelow
ǻ	scaron	Ǻ	Acircumflexacute	ǿ	ydotbelow
Ǽ	Tcaron	Ǻ	Acircumflextilde	Ǿ	Ohorndotbelow
ǽ	tcaron	Ǻ	Acircumflexhook	ǿ	ohorndotbelow
Ǻ	Ycaron	Ǻ	acircumflexgrave	Ǿ	Uhorndotbelow
ǻ	ycaron	Ǻ	acircumflexacute	ǿ	uhorndotbelow
Ǻ	Zcaron	Ǻ	acircumflextilde	Ǻ	Acircumflexdotbelow
ǻ	zcaron	Ǻ	acircumflexhook	Ǻ	acircumflexdotbelow
Ǻ	Lstroke	Ǻ	Ecircumflexgrave	Ǻ	Ecircumflexdotbelow
ǻ	lstroke	Ǻ	Ecircumflexacute	Ǻ	ecircumflexdotbelow
Ǻ	Ostroke	Ǻ	Ecircumflextilde	Ǻ	Ocircumflexdotbelow
ǻ	ostroke	Ǻ	Ecircumflexhook	Ǻ	ocircumflexdotbelow
Ǻ	aumlaut	Ǻ	ecircumflexgrave	Ǻ	Abrevedotbelow
ǻ	eumlaut	Ǻ	ecircumflexacute	Ǻ	abrevedotbelow
Ǽ	iumlaut	Ǻ	ecircumflextilde	Ǻ	Ohorn
ǽ	oumlaut	Ǻ	ecircumflexhook	Ǻ	Ohorngrave
Ǻ	uumlaut	Ǻ	Ô	Ǻ	Ohornacute
Ǻ	Aumlaut	Ǻ	Ô	Ǻ	Ohorntilde
Ǻ	Eumlaut	Ǻ	Ô	Ǻ	Ohornhook
Ǻ	Iumlaut	Ǻ	Ô	Ǻ	ohorn
Ǻ	Oumlaut	Ǻ	ô	Ǻ	ohorngrave
Ǻ	Uumlaut	Ǻ	ô	Ǻ	ohornacute
Ǻ	scommaaccent	Ǻ	ô	Ǻ	ohorntilde
Ǻ	Scommaaccent	Ǻ	ô	Ǻ	ohornhook
Ǻ	tcommaaccent	Ǻ	Ǻ	Ǻ	Uhorn
Ǻ	Tcommaaccent	Ǻ	Ǻ	Ǻ	Uhorngrave

Ú	Uhornacute	u	uhorn	ũ	uhorntilde
Ũ	Uhorntilde	ù	uhorngrave	u	uhornhook
U	Uhornhook	ú	uhornacute		

6 Colors

6.1 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 ConT_EXt color support. We will also pay attention to backgrounds and overlays because these are related to the color mechanism.

6.2 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 ConT_EXt facilities on this matter and the way ConT_EXt 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 facilities 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 TEXT
```

```
\startcolor [.*.] ... \stopcolor
```

```
* IDENTIFIER
```

A complete palette of colors is generated with:

```
\showcolor [.*.]
```

```
* IDENTIFIER
```

Figure ?? shows the colors that are standard available (see `colo-rgb.tex`).



Figure 6.1 Some examples of colors.

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 still be set in the correct colors. However, the mechanism is not robust.

In this sentence we use colors within colors. Aesthetically this is bad.

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.

6.3 Grayscales

When we print a document on a black and white printer we observe that the differences between some colors are gone. Figure ?? illustrates this effect.



Figure 6.2 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:

$$\text{gray} = .30 \times \text{red} + .59 \times \text{green} + .11 \times \text{blue}$$

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]
```

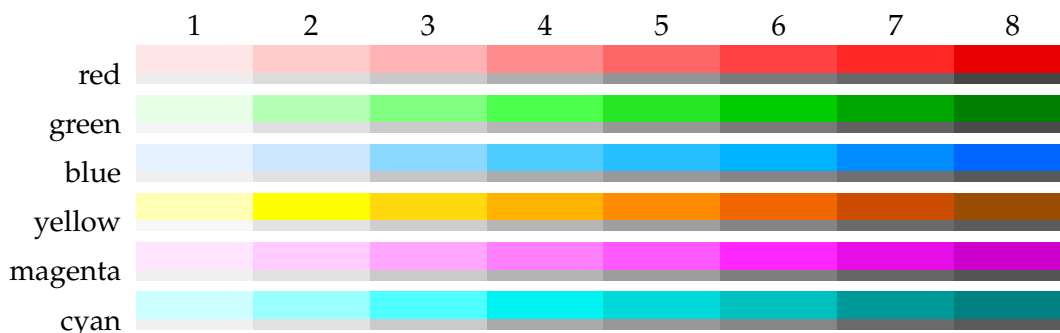
6.4 Colorgroups and palettes

T_EX itself has hardly any built-in graphical features. However the ConT_EXt 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 picture may look fine, but in black and white the results may be disappointing.

In T_EX we can approach 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.

The different gradations within a colorgroup are represented by a number. A colorgroup is defined with:



```
\definecolorgroup [.1.] [.2.] [x:y:z=,...]
                        OPTIONAL
```

```
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:



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
```

ConT_EXt 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.

	red	green	blue	yellow	magenta	cyan
1						
2						
3						
4						
5						
6						
7						
8						

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 $\text{Con}\text{\TeX}$ t: 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
```

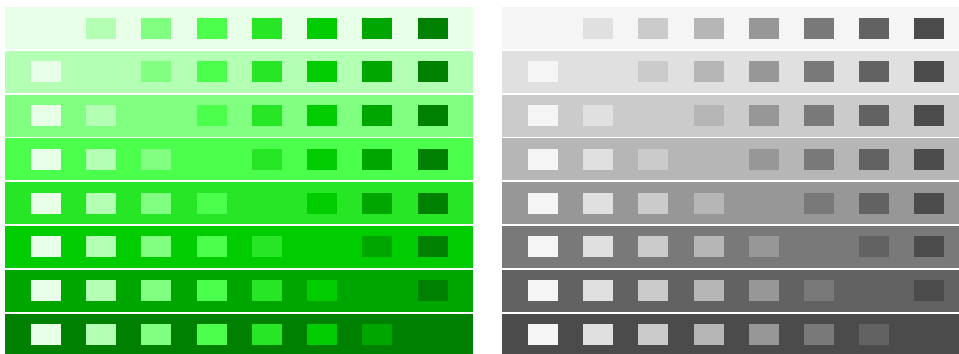
```
\comparecolorgroup [...]
```

```
* IDENTIFIER
```

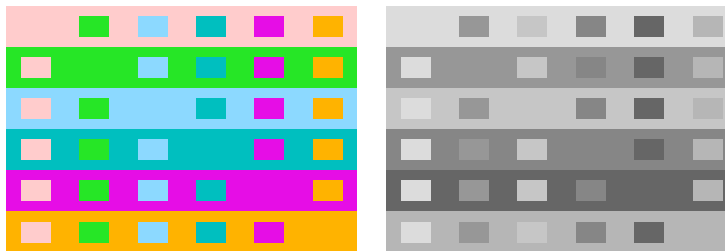
```
\comparepalet [...]
```

```
* 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.



This overview is made with `\comparecolorgroup[green]` and the one below with `\comparepalet[gamma]`.



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.

<pre>name: vew1179 file: vew1179a state: unknown</pre>	<pre>name: vew1182 file: vew1182a state: unknown</pre>	<pre>name: vew1218 file: vew1218a state: unknown</pre>	<pre>name: spin016 file: spin016a state: unknown</pre>
--	--	--	--

Figure 6.3 Some examples of the use of color.

7 Verbatim text

Text can be displayed in verbatim (typed) form. The text is typed between the commands:

unknown setup 'starttyping'

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 {...}
```

```
* TEXT
```

A complete file can be added to the text with the command:

```
\typefile [...]1 {...}2
          OPTIONAL
```

```
1 IDENTIFIER
```

```
2 TEXT
```

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 halflines line
   escape     =
   space      = on off
   tab        = NUMBER yes no
   page       = yes no
   indentnext =
   style      =
   color      = IDENTIFIER
   palet      = IDENTIFIER
   lines      = yes no hyphenated
   empty      = yes all no
   numbering  = line file no
   bodyfont   =

```

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 structure of words

```

A very special case is:

```

\definotyping
  [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   =  
    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: *aabbcc*.

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 : this command has no effect  
:  
\sihtees : 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:

```
\definotyping [MP] [option=MP]
\definotyping [PL] [option=PL]
\definotyping [JS] [option=JS]
\definotyping [TEX] [option=TEX]
```

Here we see that it is possible to define your own verbatim environment. For that purpose we use the command:

```
\definotyping [..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 T_EX 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 {...}

*   TEXT
```

When you are thinking of producing a manual on T_EX you have two commands that may serve you well:

```
\tex {...}
```

```
* TEXT
```

unknown setup 'arg'

The first command places a \ in front of typed text and the second command encloses the text with arg.

8 Backgrounds and Overlays

8.1 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

*   TEXT
```

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 arg by itself and no extra braces are necessary.

```
\background {...}
```

```
* TEXT
```

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, ConT_EXt does not keep track of screens across page boundaries. This makes sense, because screens nearly always are used as simple backgrounds.

8.2 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 ??).

	leftedge	leftmargin	text	rightmargin	rightedge
top					
header					
text					
footer					
bottom					

Figure 8.1 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 parameter `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 ConT_EXt will calculate a new background.

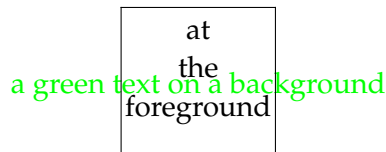
8.3 Overlays

T_EX has only limited possibilities to enhance the layout with specific features. In ConT_EXt 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 parameters `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}
```

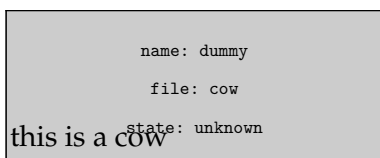
This would look like this:



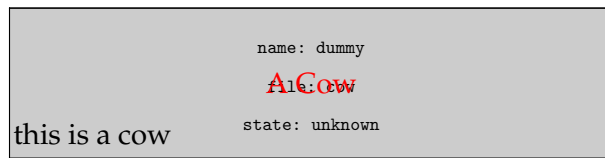
An overlay can be anything:

```
\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 T_EX 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}
```

9 Language specific issues

9.1 Introduction

One of the more complicated corners of ConT_EXt 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 ConT_EXt user interface, we also support existing input schemes.

In the early days T_EX was very American oriented, but since T_EX 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 T_EX 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.

9.2 Automatic hyphenating

Each language has its own hyphenation rules. As soon as you switch to another language, ConT_EXt will activate the appropriate set of hyphenation patterns for that language. Languages are identified by their official two character identifiers, like: Dutch (`n1`), English (`en`), German (`de`) and French (`fr`). A language is chosen with the following command:¹⁶

```
\language [...]
```

```
*
```

Some short cut commands are also available. They can be used enclosed in braces:

```
\n1 \en \de \fr \sp \uk \pl \cz ...
```

The command `\language[n1]` can be compared with `\n1`. The first command is more transparent. The two character commands may conflict with existing commands. Take, for

¹⁶ In case of any doubt please check if the hyphenation patterns are included in the `fmt-file`.

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.

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 'Qua-	tuor pour la fin du temps' during the World War II in a	concentration camp. This may well be one of the most mov-	ing 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 T_EX-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.

9.3 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 transparent for the user. We show the process to give some insight in the mechanism. An example:¹⁷

```
\installlanguage
[en]
[spacing=broad,
leftsentence=---,
rightsentence=---,
leftsubsentence=---,
rightsubsentence=---,
leftquote=\upperleftsinglesixquote,
rightquote=\upperrightsingleninequote,
leftquotation=\upperlefthdoublesixquote,
rightquotation=\upperrightdoubleninequote,
date={month,\ ,day,{,\ },year},
```

¹⁷ The somewhat strange name \upperleftsinglesixquote is at least telling us what the quote will look like.

```

    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. 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 character set, 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
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:

```
August 16, 2009
—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.

9.4 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 16 August 2009. We use \currentdate[weekday,month,day,{,},year] to obtain Sunday August 16,2009.

A short cut looks like this: [dd,mm,yy] and will result in 160809. Something like [d,m,y] would result in 16August2009 and with [referral] you will get a 20090816. Combinations are also possible. Characters can also be added to the date pattern. The date 16–08–09 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

9.5 Labels and heads

In some cases ConT_EXt 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  
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 be recalled with:

```
\labeltext {...}  
  
* TEXT
```

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  
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 {...}  
  
* TEXT
```

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 [.*.]
*
```

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.

9.6 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 `Con \TeX t`. 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\midwordddiscretionary}
  \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 `Con \TeX t` the / is used instead of a ".

9.7 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.

9.8 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 9.1 Hyphenation of composed words.

The main reason behind this mechanism is that T_EX doesn't really know how to hyphenate composed words and how to handle subsentences. T_EX know a lot about math, but far less about normal texts. Using this command not only serves consistency, but also makes sure that

T_EX can break compound words at the right places. It also keeps boundary characters at the right place when a breakpoint is inserted.

9.9 Pattern files manual

TODO: A large part of this section is obsolete

T_EX 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 T_EX that a word should be hyphenated in a certain way. The second command can only be issued when T_EX 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 T_EX 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 ConT_EXt development it came with its own pattern files. Their names started with `lang-` and their suffixes were `pat` and `hyp`.

However, when ConT_EXt went public, I was convinced to drop those files and use the files already available in distributions. This was achieved by using the ConT_EXt 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 ConT_EXt. Even worse, their names are not consistent and the names of some files as well as locations in the tree keep changing. The price ConT_EXt 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 noticeably their encoding.

After the need to adapt the name mapping once again, I decided to get back to providing ConT_EXt specific pattern files. Pattern cooking is a special craft and T_EX users may call themselves lucky that it's taken care of. So, let's start with thanking all those T_EX 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 ConT_EXt 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 T_EX 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 ConT_EXt way of naming glyphs and the names and reverse mappings are taken from the encoding files in the ConT_EXt distribution, so you need to have ConT_EXt installed.

The ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ε -T_EX 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 T_EX specific one anyway.

9.10 Installing languages

Installing a language in ConT_EXt 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 T_EX distribution unless `lang-nl.pat` is found, in which case ConT_EXt assumes that you prefer the ConT_EXt patterns. In that case, run

```
ctxtools --patterns all
```

You need to move the files to the ConT_EXt 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 X_YT_EX 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 ConT_EXt versions of the 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 ConT_EXt 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>nehyp96.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 afbreektteken)</comment>
  </copyright>
</description>
```

This file is 'work in process': more details will be added and comments will be enriched.

9.11 Commands

You can at any moment add additional hyphenation exceptions to the language specific dictionaries. For instance:

```
\language[nl] \hyphenation{pa-ti  n-ten}
```

Switching to another language is done with the `\language` command. The document language is set with `\mainlanguage`.

If you want to let T_EX 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-tiĀĳn-ten</hyphenation>
  <hyphenation>pa-tiĀĳn-ten-or-ga-ni-sa-tie</hyphenation>
  <hyphenation>pa-tiĀĳn-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 patiĀĳnten<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`.

9.12 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 T_EX 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 ConT_EXt that is used to process this file.

FIXME: `\showpatterns` doesn't exist anymore

In the (near) future the somewhat arcane `p10` and `i12` 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.

9.13 Hyphenation

If you want to know what patterns are used, you can try to hyphenate a word with `\showhyphenations`.

```
language : en (internal code:1)
font      : pplr8t at 11.0pt
encoding  : ec
mapping   : not set
handling  : not set

sample    : abra-cadabra
```

While hyphenating, 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.¹⁸

fi fl ffi ffl

The most common ligatures.

Any direct use of a ligature (as accessed by `\char` or through active characters) is wrong and will create faulty hyphenation. 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:

¹⁸ Thomas Schmitz is responsible for the associated third party module.

```
[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.

10 Text elements

10.1 Introduction

The core of ConT_EXt 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:

- | | | |
|-------------------------|--------------------------|-----------------|
| 0. French title | 5. . . . | 10. notes |
| 1. title | 6. list of illustrations | 11. literature |
| 2. history & copyright | 7. acknowledgement | 12. register(s) |
| 3. mission | 8. errata | 13. colofon |
| 4. preface/introduction | 9. the content | |

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:

- | | | |
|---------------------|-----------------------------|----------------------------|
| 0. French title | 7. list of illustrations | 14. literature |
| 1. titlepage | 8. introduction/preface (2) | 15. used words |
| 2. colofon | 9. . . . | 16. addenda |
| 3. copyright | 10. epilogue | 17. register |
| 4. mission | 11. appendices | 18. acknowledgement photos |
| 5. preface (1) | 12. summaries | |
| 6. table of content | 13. notes | 19. (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:

- | | | |
|-----------------|---------------------|---------------------------|
| 0. French title | 6. preface | 12. list of illustrations |
| 1. series title | 7. table of content | 13. used words |
| 2. title | 8. introduction | 14. bibliography |
| 3. copyright | 9. . . . | 15. colofon |
| 4. mission | 10. appendices | 16. register |
| 5. blank | 11. notes | |

Since there seems to be no standardized way of setting up a document, ConT_EXt will only provide general mechanisms. These are designed in such a way that they meet the following specifications:

0. 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.
1. Not every level needs a number but in the background every level is numbered to be able to refer to these unnumbered structuring elements.
2. The names given to the structuring elements must be a logical ones and must relate to their purpose.
3. It is possible to generate tables of contents and registers at every level of the document and they must support complex interactivity.
4. A document will be divided in functional components like introductions and appendices with their respective (typographical) characteristics.
5. The hyphenation of titles must be handled correctly.
6. 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.
7. A ConT_EXt 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 ConT_EXt. The commands that are described in the following sections are an example of an implementation. We will also show examples of self designed titles.

10.2 Subdividing the text

A text is divided in chapters, sections, etc. with the commands:

```
\part [...1...] {..2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\chapter [...1...] {..2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\section [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\subsection [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\subsubsection [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

and

```
\title [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\subject [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\subsubject [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

```
\subsubsubject [...1...] {.2.}
```

OPTIONAL

1 REFERENCE

2 TEXT

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.

level	numbered title	unnumbered title
1	<code>\part</code>	
2	<code>\chapter</code>	<code>\title</code>
3	<code>\section</code>	<code>\subject</code>
4	<code>\subsection</code>	<code>\subsubject</code>
5	<code>\subsubsection</code>	<code>\subsubsubject</code>

Table 10.1 The structuring elements.

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.

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 ?? this mechanism is described in full. A reference is optional and can be left out.

```
\section{Subdividing a text}
```

ConT_EXt 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[section] [ownnumber=yes]
\subsection{399}{The old number}
\subsection[someref]{400}{Another number}
```

In this example an additional argument appears. In the background ConT_EXt 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.

399 The old number

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 {...}

* TEXT
```

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 ?? . 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 ConT_EXt 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 T_EX or ConT_EXt commands. It is of good practice to use capitals for your own definitions. For example:

```
\definehead[WorkInstruction][section]
```

10.3 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	=
	textstyle	=
	numberstyle	=
	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
	margin text	= yes no
	<i>inherits from \setupheads</i>	

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          =
  aligntitle     = yes float no
  tolerance      = verystRICT strict tolerant verytolerant stretch
  indentnext     =
  command        = \...#1#2
  margin         = DIMENSION
```

The number of a title can be set up with:

```
\setupheadnumber [..] [..]
```

```
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 : 3
\headnumber [chapter] : 10
\headnumber [section] : 10.3
```

When you want to use the titlenumber in calculations you must use the command `\current-headnumber`. 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=no`. Figure ?? shows the difference between these two alternatives.

```
\setuphead
  [section]
  [page=yes,continue=no,
   after=\blank]
```

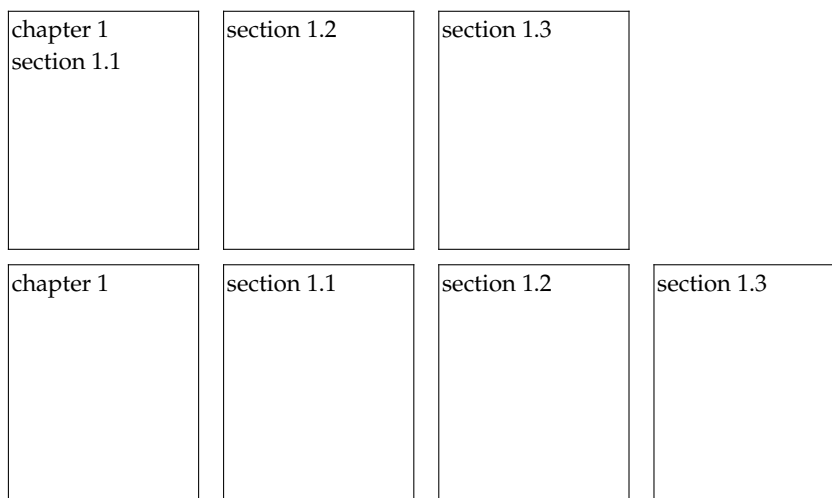


Figure 10.1 Two alternatives for the first section.

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 ??).

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.

10.3.1 Title alternative equals normal

10.3.2 Title alternative equals inmargin

Title alternative equals middle

These titles were generated 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.

```
\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]
```

10.3.3	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.

10.3.4 | Another title

This alternative looks somewhat better. The first definition is slightly altered. This example also shows the features of the command `\framed`.

```

\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 ConT_EXt encounters a long title. Here is the answer.

10.3.5 | 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.

```

\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`).

10.3.6 | A considerably longer title

When the title design becomes more complex you have to know more of T_EX. 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 T_EX than we may expect beforehand.* The answer lies in the fact that ConT_EXt 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` identifies 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.

10.4 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}      in content, no number
\stopfrontmatter

\startbodymatter
  \chapter{First}            number 1, in content
  \section{Alfa}             number 1.1, in content
  \section{Beta}            number 1.2, in content
```

```

\chapter{Second}           number 2, in content
  \subject{Blabla}         no number, not in content
\stopbodymatter

\startappendices
  \chapter{Index}           letter A, in content
  \chapter{Abbreviations}   letter B, in content
\stopappendices

\startbackmatter
  \chapter{Acknowledgement} no number, in content
  \title{Colofon}           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.

10.5 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 ConT_EXt 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 ConT_EXt 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> 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`. ConT_EXt has set up the lower level section headings to inherit their settings from the higher level. The default setups for ConT_EXt 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 ConT_EXt 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 arg 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.

11 References

11.1 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:

```
\definelist [1.] [2.] [...,3.,...]  
                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:

$$\backslash\mathrm{setuplist}[firstlevel][width=2em]$$

Lists can be set up simultaneously, for example:

$$\backslash\mathrm{setuplist}[firstlevel,secondlevel][width=2em]$$

To generate a list you type:

```

\setuplist [...,1...] [...,2,...]

1 IDENTIFIER
2 state          = start stop
  alternative    = a b c ... none command
  coupling       = on off
  criterium      = SECTION local previous current all
  pageboundaries = LIST
  style          =
  numberstyle    =
  textstyle      =
  pagestyle      =
  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

```

```

\placelist [...,1...] [...,2,...]
                                OPTIONAL
1 IDENTIFIER
2 inherits from \setuplist

```

The layout of a list is determined by the values of alternative (see table ??), margin, width and distance. The alternatives a, b and c are line oriented. A line has the following construct:

margin	width	distance	
	headnumber		head and pagenumber

In a paper document it is sufficient to set up width. In an interactive document however the width determines the clickable area.¹⁹

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:

```
\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:

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.

¹⁹ This also depends on the value assigned to interaction.

```
\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:

unknown setup 'complete«combinedlist»'

unknown setup 'place«combinedlist»'

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 11.1 Alternatives in combined lists.

Possible alternatives are summed up in table ???. 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 or 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. Page-breaking 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 TEXT
3 TEXT
```

```
\writebetweenlist [.1.] {.2.}
```

```
1 SECTION IDENTIFIER
2 TEXT
```

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 {...}

*   TEXT
```

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 [...1,...] [...OPTIONAL2,...]
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]

\setuplabeltext[en] [section={ugh }]
\placelist
  [section]
  [criterium=chapter,alternative=a,
   label=yes,width=2cm]

\placelist
  [section]
  [criterium=chapter,alternative=b]

\placelist
  [section]
  [criterium=chapter,alternative=b,
   pagenumber=no,width=fit,distance=1em]

\placelist
  [section]
  [criterium=chapter,alternative=c,
   chapternumber=yes,margin=1.5cm]
```

```
\placelist % note the spaces on each side of the colon
[section]
[criterium=chapter,alternative=c,
chapternumber=yes,separator={ : },width=fit]
```

```
\placelist
[section]
[criterium=chapter,alternative=d]
```

```
\placelist
[section]
[criterium=chapter,alternative=d,
distance=2cm]
```

```
\placelist
[section]
[criterium=chapter,alternative=d,
left={ ( },right={ ) }]
```

```
\placelist
[section]
[criterium=chapter,alternative=e]
```

```
\placelist
[section]
[criterium=chapter,alternative=e,
width=\textwidth,background=screen]
```

```
\placelist
[section]
[criterium=chapter,alternative=e,
width=4cm]
```

```
\placelist
[section]
[criterium=chapter,alternative=f]
```

```
\placelist
[section]
[criterium=chapter,alternative=g]
```

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]
```

```
\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.

```
\def\ListCommand#1#2#3%
{At p~#3 we discuss {\em #2}; }

\placelist
[section]
[criterium=chapter,alternative=command,
command=\ListCommand]
```

As an alternative for none, we can use `horizontal` and `vertical`. Both commands have their spacing tuned for typesetting lists in for instance menus.

11.2 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      =
   synonymstyle   =
   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 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:

```
unknown setup '«synonym»'
```

A list of synonyms is generated by:

```
unknown setup 'placelistof«synonyms»'
```

The next command generates a list with a title (`\chapter`):

```
unknown setup 'completelistof«synonyms»'
```

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 ??).

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:

```
unknown setup 'load«synonyms»'
```

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 ??.

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}
```

Then we can recall number and name by \FunctionName (Lithographer and Typesetter) and \FunctionNumber (0001a and 0002x), so:

The \FunctionName{0001} has functionnumber \FunctionNumber{0001}.

11.3 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	=
	expansion	= yes no command

After the definition the next command is available. The *sort* indicates the name for the list you defined.

unknown setup ‘«sort»’

In accordance to lists there are two other commands available:

unknown setup ‘placelistof«sorts»’

The title can be set up with \setuphead:

unknown setup ‘completelistof«sorts»’

An example of sorting is:

```
\definesorting[city][cities]
\setupsorting[city][criterium=all]
```

```

\city {London}
\city {Berlin}
\city {New York}
\city {Paris}
\city {Hasselt}

\placelistofcities

```

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.

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 `[]` ConT_EXt 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 ConT_EXt and/or T_EX 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:

unknown setup 'load«sorts»'

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 ??).

11.4 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 TEXT
```

and recalled by:

```
\getmarking [.1.] [.2.]
```

```
1 IDENTIFIER
```

```
2 first last previous both all current
```

In analogy with the T_EX-command `\mark`, we keep record of three other marks per mark (see table ??).

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 11.2 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 ??.

page	previous	first	last
1	—	first	second
2	second	third	fourth
3	fourth	fifth	fifth

Table 11.3 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 ??) or pose limits to the length. In ConT_EXt 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 ConT_EXt works.

11.5 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 TEXT
2 TEXT
3 REFERENCE
```

```
\at {.1.} {.2.} [.3.]
```

```
1 TEXT
2 TEXT
3 REFERENCE
```

```
\about {.1.} [.2.]
```

```
1 TEXT
2 REFERENCE
```

A cross reference to a page, text (number) or both can be made with:

```
\pagereference [...]
```

```
* REFERENCE
```

```
\textreference [.1.] {.2.}
```

```
1 REFERENCE
2 TEXT
```

```
\reference [.1.] {.2.}
```

```
1 REFERENCE
2 TEXT
```

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 ??, titled “??”, we describe how a cross reference can be defined. This section starts at page ?? and is part of chapter ??.

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/after and discuss somewhere previous/later the descriptions.

```
\somewhere {.1.} {.2.} [.3.]
```

```
1  TEXT
2  TEXT
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 ??).

```
\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 `\sectionnumber` 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:

```
\userreferences [...,*,...]

*   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`:

```
\definereference[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. \somedline[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 `\somedline` you will get the first line number. In this example we see that we can either let ConT_EXt generate a label automatically, or provide our own text between braces.

- 1 A special case of referencing is that of referring to linenumbers. Different line numbering
- 2 mechanism can be used interchangeably. This leads to confusing input. Doesn't it? A cross ref-
- 3 erence to a line can result in one line number or a range of lines. **■ ■ ■ ■ ■ ■ ■ ■ ■ ■**
- 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: line ??, line ??, line ?? and as the last reference
- 6 ??.

```
\startlines
```

```
\somedline [...]
```

```
* REFERENCE
```

```
\inline [...]
```

```
* REFERENCE
```

11.6 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.] [...,\u2.,...]

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:

```
??
some text ??
some text ?? and more text ??
?? ??
```

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).

11.7 Registers

A book without a register is not likely to be taken seriously. Therefore we can define and generate one or more registers in ConT_EXt. The index entries are written to a separate file. The Perl script T_EXutil converts this file into a format T_EX 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:

unknown setup ‘«register»’

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 ??), 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:

unknown setup ‘see«register»’

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:

unknown setup ‘place«register»’

The next command results in register with title:

unknown setup ‘complete«register»’

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           =
    style           =
    pagestyle       =
    textstyle       =
    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 to 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:

unknown setup 'writeto«register»'

There the expansion mechanism can be activated. Default expansion is inactive (see page ??).

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):²⁰

```
\placeregister [macro]
[criterium=chapter,n=2,before=,after=]
```

and we will obtain:

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 11.4 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}
```

²⁰ Of course, `\placemacro` and `\completemacros` are also available.

```

\index          {form::entry}
\index[key]     {form::entry}
\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:]{\sqrteroot}
\index[hm:root]{\sqrt{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:

unknown setup '«start»register'

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 uncomfortable 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:

unknown setup 'coupled«register»'

For example `\coupleindex{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.

12 Descriptions

12.1 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 T_EX and therefore of ConT_EXt 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.

12.2 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.

unknown setup '«description»'

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 width 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 serifed format, the parameter `margin` is of importance. When set to `standard` or `ja`, the margin 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      =
   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 =
   titlecolor = IDENTIFIER
   align      =
   margin     = standard yes no DIMENSION
   location   = left right top serifed inmargin inleft inright hanging
   headstyle  =
   headcolor  = IDENTIFIER
   headcommand = COMMAND
   hang       = fit broad NUMBER
   before     = COMMAND
   inbetween  = COMMAND
   after      = COMMAND
   indentnext =
   indenting  =
   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:

```
unknown setup 'start«description»'
```

```
\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.

12.3 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 ConT_EXt 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`
`\subname`
`\subsubname`
`\subsubsubname`

Where *name* stands for any chosen name.

unknown setup '«enumeration»'

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 precedes every generated number. A number can be set and reset with the command:

`\setenumeration{value}`
`\resetenumeration`

You can use the *start* parameter in the *setup* command to explicitly 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 explicitly increased with the commands:

`\nextenumeration`
`\nextsubenumeration`
`\nextsubsubenumeration`

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...] [...2...]
                        OPTIONAL
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.

unknown setup ‘start«enumeration»’

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 ???. 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,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,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.

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. **second 2**

The numbered elements third and fourth are both unique and are placed in right margin. **third 1**

Both are defined in one command but they do have own counters that are in no way coupled. **fourth 1**

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]
```

12.4 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`, `\subname` and `\subsubname` are available.

unknown setup '«indentation»'

The parameters can be set up with the command:

```
\setupindentations [...1,...] [...2.,...]
                        OPTIONAL

1 IDENTIFIER
2 style      =
  headstyle =
  width     = fit DIMENSION
  text      = TEXT
  sample    = TEXT
  before    = COMMAND
  after     = COMMAND
  distance  = DIMENSION
  separator = TEXT
```

It is possible to change the setup of `\indentation` with the command:

```
\setupindentations[name][setups]
```

An example of how you can use the indentation mechanism is given below:

```

\setupindentations
  [sample={rime m},
   separator={:},
   distance=.5em]

\defineindenting[ra][text=rime a]
\defineindenting[rb][text=rime b]
\defineindenting[rc][text=rime c]

\startpacked
\ra pretty litte girl \par
\ra pretty litte girl in a blue dress \par
\rb pretty little girl in a blue dress \par
\rc playing in the sand \par
\rb make my day \par
\rc smile for me \par
\stoppacked

```

This results in:

```

rime a : pretty litte girl
rime a : pretty litte girl in a blue dress
rime b : pretty little girl in a blue dress
rime c : playing in the sand
rime b : make my day
rime c : smile for me

```

A series of indenting commands can be enclosed with the commands:

```

\startindentation
\stopindentation

```

12.5 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.] [...,\underline{.},...]
```

```
1 IDENTIFIER
2 text      = TEXT
  location  = inmargin intext
  way       = bytext bycd:section
  blockway  = yes no
  headstyle =
  headcolor = IDENTIFIER
  before    = COMMAND
  after     = COMMAND
```

Where the parameter `location` is set at `intext` and `inmargin`. After this definition the following commands are available:

```
\resetname
\incrementname
\nextname
\currentname[reference]
```

The `[reference]` after `currentname` is optional. After

```
\definelabel[video][text=video,location=inmargin]
```

This defines `\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: **video 12.1**

unknown setup '«labeling»'

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).

12.6 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 ??). The layout can also be influenced. Itemization has a maximum of four levels.

The command to itemize is:

```
\startitemize[setups]
\item .....
\item .....
\stopitemize
```

So you can do things like this:

Which of these theses are true?

setup	result	setup	result
n	1, 2, 3, 4	1	dot (●)
a	a, b, c, d	2	dash (–)
A	A, B, C, D	3	star (★)
KA	A, B, C, D	4	triangle (▷)
r	i, ii, iii, iv	5	circle (○)
R	I, II, III, IV	6	big circle (◯)
KR	I, II, III, IV	7	bigger circle (⊝)
m	1, 2, 3, 4	8	square (□)
g	α, β, γ		
G	A, B, Γ		

Table 12.1 Item separator identifications in itemizations.

```
\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?

- . The difference between a village and a city is the existence of a townhall.
- A. 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:

- . a g for Greek characters
- α . 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:

unknown setup 'setupitemize'

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?

- : 2000 is a leap-year
- a: 2001 is a leap-year
- b: 2002 is a leap-year
- c: 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?

0. 2000 is a leap-year
1. 2001 is a leap-year
2. 2002 is a leap-year
3. 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 ??.

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
<code>n*serried</code>	little horizontal white space after symbol
<code>n*broad</code>	extra horizontal white space after symbol
<code>inmargin</code>	item separator in margin
<code>atmargin</code>	item separator at the margin
<code>stopper</code>	punctuation after item separator
<code>intro</code>	no pagebreak
<code>columns</code>	two columns

Table 12.2 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 ?? 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:

0. 2005 is a leap-year

Another example illustrates that `continue` even works at other levels of itemizations:

- **supported image formats in pdfTeX**
 - . png
 - a. eps
 - b. pdf
- **non supported image formats in pdfTeX**
 - . jpg
 - a. gif
 - b. 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.

0. png	3. eps	6. pic	9. jpe
1. tif	4. pdf	7. bmp	
2. jpg	5. gif	8. 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:

. png	III. eps	VI. pic	IX. jpe
I. tif	IV. pdf	VII. bmp	
II. jpg	V. gif	VIII. 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:

0. 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 an itemization consists of only one item we can leave out the commands `\startitemize` and `\stopitemize` and the level 1 symbol is used.

`\item` The itemization commands force the user into a consistent layout of the itemizations. `\par`

Instead of the `\par` you could have used an empty line. In each case, we get the following output:

- The itemization commands force the user into a consistent layout of the itemizations.

Only the text directly following the command and ended by an empty line or `\par` is indented.

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.

0. This itemization is preceded by `\startitemize[n,packed]`.

+0. This item is preceded by `\sub`, the other items by `\item`.

1. The itemization is ended by `\stopitemize`.

The most important commands are:

unknown setup 'startitemize'

```
\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 {...}
*   TEXT
```

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   TEXT
```

Some at first sight rather strange alternatives are:

```
\its [...,*...]
      OPTIONAL
*   REFERENCE
```

```
\ran {...}
*   TEXT
```

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
o o o o o   I can not do without TEX.
o o o o o   I will use TEX forever.
o o o o o   I expect an alternative to TEX in the next few years.
o o o o o   I use TEX and other packages.
o o o o o   I hardly use TEX.
o o o o o   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**

- 0. of all the items
 - he had to learn at school
- ++ 1. because the marginal explanation
 - +1. 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]
```

12.7 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      =
  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 [...,\u1,...] {... \u2 ...}
      OPTIONAL
1  inherits from \setupitems
2  TEXT
```

12.8 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 {...}
```

```
* TEXT
```

```
\quotation {...}
```

```
* TEXT
```

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  =
  color  = IDENTIFIER
  location = TEXT margin
```

The location of a period, inside or outside a citation is somewhat arbitrary. The opinions on this issue differ considerably.

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=\upperlefthdoublesixquote]
```


For consistency, 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:

<code>\lowerleftsingleninequote</code>	,	<code>\lowerrightsingleninequote</code>	,
<code>\lowerlefthdoubleninequote</code>	”	<code>\lowerrighthdoubleninequote</code>	”
<code>\upperleftsingleninequote</code>	’	<code>\upperrightsingleninequote</code>	’
<code>\upperlefthdoubleninequote</code>	”	<code>\upperrighthdoubleninequote</code>	”
<code>\upperleftsinglesixquote</code>	’	<code>\upperrightsinglesixquote</code>	’
<code>\upperlefthdoublesixquote</code>	”	<code>\upperrighthdoublesixquote</code>	”

13 Lines and frames

13.1 Introduction

T_EX 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 T_EX by combining T_EX 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.

13.2 Single lines

The simplest way to draw a line in ConT_EXt 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 ?? 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:

```
\v1 [...]
```

```
*   NUMBER
```

```
\h1 [...]
```

```
*   NUMBER
```

The argument is optional. To `\v1 (|)` you may pass a factor that relates to the actual height of a line and to `\h1 (—)` a width that relates to the width of an em. So `\v1[2]` produces a rule with a height of two lines.

13.3 Fill in rules

On behalf of questionnaires there is the command:

```
\fillinline [...,1.,...] 2.
                        OPTIONAL
1   inherits from \setupfillinlines
2   EMPTY
```

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 [...,.*,...] {.2.} {.3.}
                        OPTIONAL      OPTIONAL
```

```
1 inherits from \setupfillinrules
2 TEXT
3 TEXT
```

```
\setupfillinrules [...,.*,...]
```

```
* width          = fit broad DIMENSION
  distance       = DIMENSION
  before         = COMMAND
  after          = COMMAND
  style          =
  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

13.4 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 TEXT

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   =
    color      = IDENTIFIER
    style      =
    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   TEXT
```

13.5 Underline

Underlining text is not such an ideal method to banner your text. Nevertheless we introduced this feature in ConT_EXt. Here is how it works. We use:

```
\underbar {...}
```

```
*   TEXT
```

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 {...}
```

```
* TEXT
```

```
\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      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 {...}
```

```
*   TEXT
```

This command supports no nesting. Single words are ~~striked~~ with:

```
\overstrikes {...*...}
```

```
*   WORD
```

13.6 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 [...,*...]{...}
          OPTIONAL
1   inherits from \setupframed
2   TEXT
```

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 ??, ?? and ??). 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`:

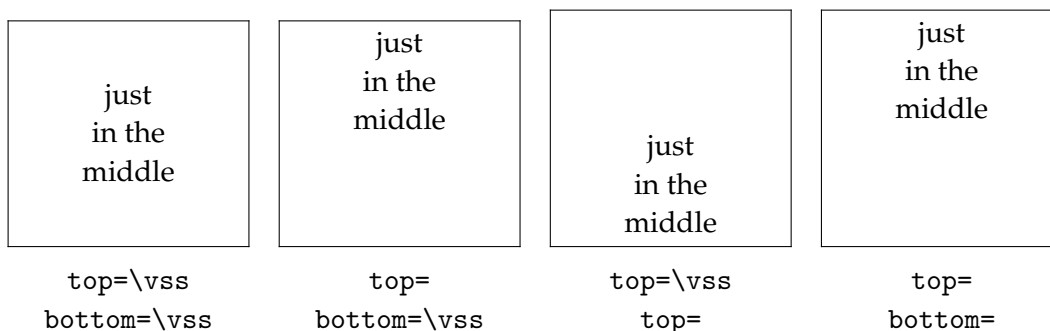
left of the middle	just in the middle	right of the middle
<code>\lbox</code>	<code>\cbox</code>	<code>\rbox</code>

The second text is typed as follows:

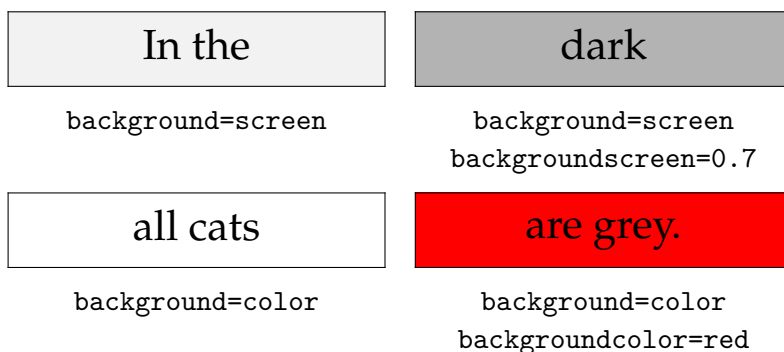
```
\framed
  [width=.2\hsize,height=3cm]
  {\cbox to 2.5cm{\hsize2.5cm just\\in the\\middle}}
```

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
```



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]`).



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 ??.

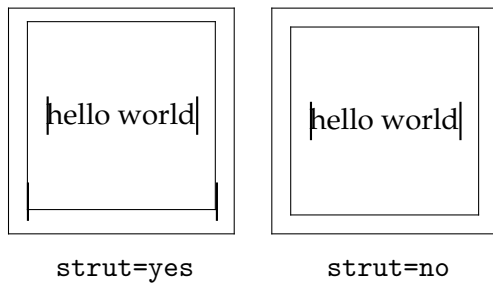
The command `\framed` itself can be an argument of `\framed`. We will obtain a framed frame.

```
\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:



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              =
   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     =
   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 ??, ?? and ??.




		offset			
		.25ex	0pt	none	overlay
strut	yes			.	
	no		.	.	

Table 13.1 The influence of strut and offset in `\framed` (1).









		offset			
		.25ex	0pt	none	overlay
strut	yes				
	no				

Table 13.2 The influence of strut and offset in `\framed` (2).


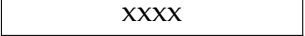

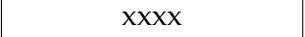
		width	
		fit	broad (<code>\hsize=4cm</code>)
height	fit		
	broad		

Table 13.3 The influence of height and width in `\framed`.

happy
birthday
to you

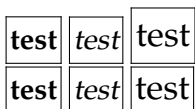
At first sight it is not so obvious that `\framed` can determine the width of a paragraph by itself. When we set the parameter `align` the paragraph is first typeset and then framed. This feature valuable when typesetting titlepages. In the example left of this text, linebreaks are forced by `\\`, but 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 currently 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.



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 considerably 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.

13.7 Framed texts

When you feel the urge to put a frame around or a background behind a paragraph there is the command:

unknown setup 'startframedtext'

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            =
   style              =
   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          =
   inherits from      \setupframed

```

Framed texts can be combined with the place block mechanism, as can be seen in intermezzo ??.

```

\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 13.1 An example of an intermezzo.

You can also draw a partial frame. The following setup produces intermezzo ??.

```

\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 13.2 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 innermezzo was set up with :
`\setupframedtexts[frame=off,background=screen]`

Intermezzo 13.3 An example of an intermezzo with background.

Intermezzo ?? 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 13.4 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 13.5 An example of a predefined framed text.

13.8 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   TEXT
```

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 \marginrule[4]{in spite of the} consonants the sound is really very well described. And what about a cow, does it say boe, mboe or mmmmmm?

```
\stopmarginrule
```

Or:²¹

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].

²¹ G.C. Molewijk, *Spellingsverandering van zin naar onzin* (1992).

13.9 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
```

- 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.

13.10 Grids

We can make squared paper (a sort of grid) with the command:

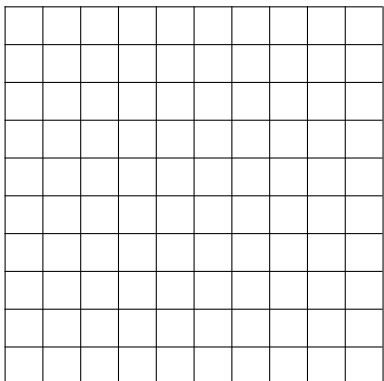
```

\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.

```

> i figures+placing
> i tables+placing
> i figures+numbering
> i tables+numbering
> i figures+listing
> i tables+listing
> i placing+figures
> i placing+tables
> i numbering+figures
> i numbering+tables
> i listing+figures
> i listing+tables
> t definefloat
> t place<float>
> t placelistof<floats>
> t completelistof<<f..
> t reserve<float>
> t setup<floats>
> t start<float>text
> t setupfloats
> t setupcaptions

```

14 Blocks

14.1 Introduction

A block in ConT_EXt 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 T_EX 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.

14.2 Floats

Floats are composed of very specific commands. For example a table in ConT_EXt is typeset using a shell around T_AB_LE. Drawings and graphics are made with external packages, as T_EX 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 ConT_EXt, 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:

unknown setup 'place«float»'

When a float cannot be placed at a specific location on a page, ConT_EXt will search for the most optimal alternative. ConT_EXt provides a number of placement options for floats. These are listed in table ??.

tab:floats->

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

84

Table 14.1 Preferences for float placement.

The commands can be used without the left and right brackets. For example:

```
\place...{caption}{content}
```


< - placed
< - placed

References

> tab:example
< fig:reservation
> fig:reservation

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

85

Table 14.2 A very simple example of a table.

The vertical whitespace for a float can be reserved with:

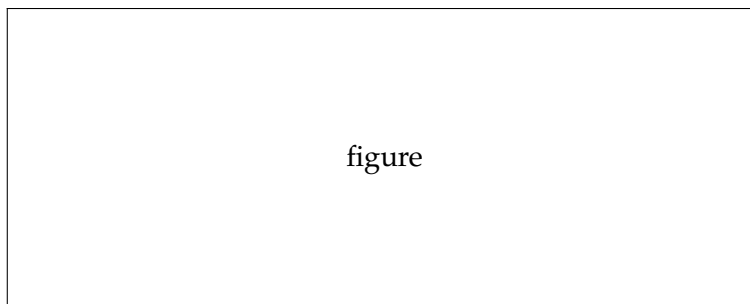
unknown setup 'reserve«float»'

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 ??.

fig:reservation->



86

Figure 14.1 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 `arg` 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 ConT_EXt 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:

unknown setup 'start«float»text'

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:

unknown setup 'placelistof«floats»'

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 ??.

numbering->

The next command generates a list of floats on a separate page.

unknown setup 'completelistof«floats»'

Pagebreaks that occur at unwanted locations can be enforced in the same way that is done with a table of contents (see section ??):

toc->

```
\completelistoffloats[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`.

```
figure[-- not found/processed]

```

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 =
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      =
   style          =
   number         = yes no none
   inbetween      = COMMAND
   align          =
   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 14.2

87

```
\setup
\set
```

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.

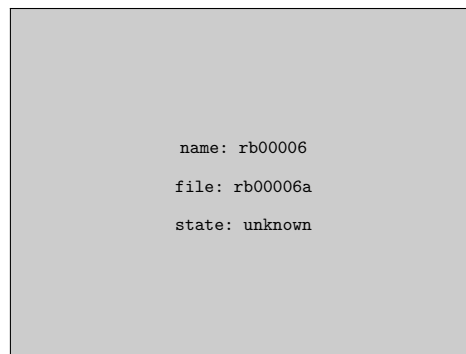
We continue with a few examples of floats (figures) placed next to the running text. This looks like:

```
\placefigure[right]{none}{}
... here is where the text starts ....
```

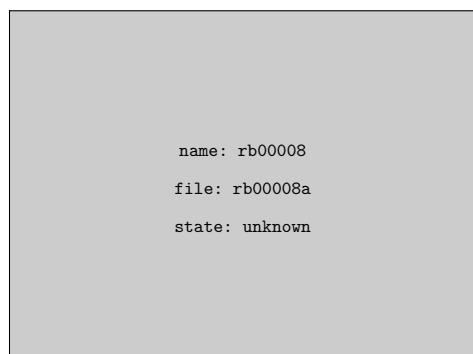
For illustrating the mechanism we do need some text. Therefore the examples are used to explain some issues on the float mechanism.

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 ConT_EXt 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.

```
< - placed
< - placed
< - placed
```

This brings us to a figure that is placed at the left side of a page. The side float mechanism is 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]{none}{\externalfigure[rb00015]}
....
\stopfiguretext
```

90

```
name: rb00015
file: rb00015a
state: unknown
```

91

```
name: vos1082
file: vos1082a
state: unknown
```

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.

92

```
name: vos1083
file: vos1083a
state: unknown
```

When several figures are set under each other, making them the same width makes for a nice presentation on the page. This looks better.

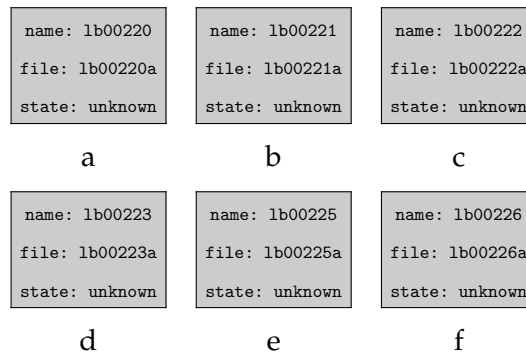
14.3 Combining figures

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.



93

Figure 14.3 An example of `\startcombination....`

fig:combinations-> The example in figure ?? is typeset with the commands:

```
\placefigure
[here]
[fig:combinations]
{An example of \type{\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
```

< - placed
< - placed
> - saved

The combination can be set up with:

References

```
\setupcombinations [...,.*,...]

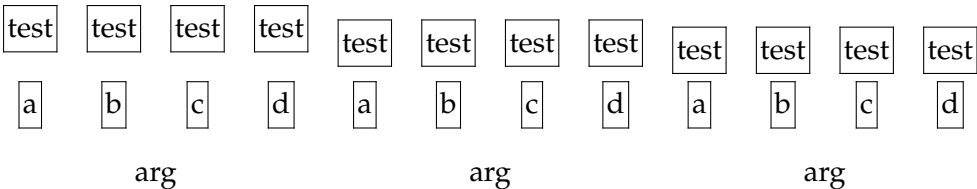
* before      = COMMAND
  inbetween   = COMMAND
  after       = COMMAND
  distance    = DIMENSION
  height      = DIMENSION fit
  width       = DIMENSION fit
  location    = top middle bottom left right
  align       =
  style       =
  color       = IDENTIFIER
```

< fig:order of combin..
< fig:spacing in comb..
< fig:no subcaptions
> fig:spacing in comb..
> fig:order of combin..
> fig:no subcaptions

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.

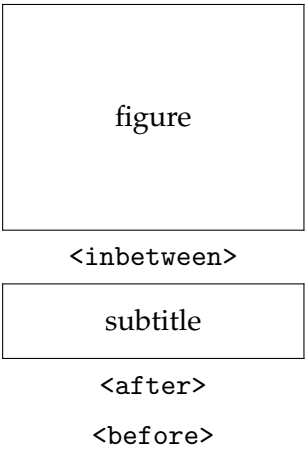
The three parameters before, after and between are processed in the order of specification in figure ??. There are some examples in figure ??. We can see in figure ?? that when the title in the second argument is empty the spacing adapted.

fig:spdehtgafpnooms->



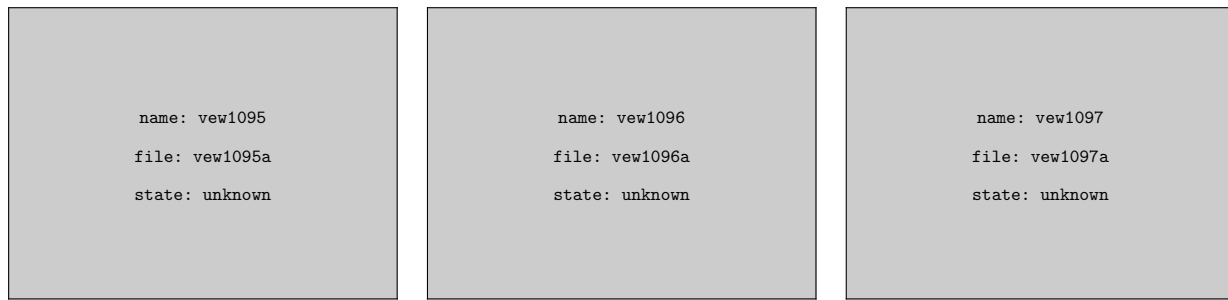
94

Figure 14.4 The spacing within combinations (1).



95

Figure 14.5 The spacing within combinations (2).



96

Figure 14.6 Combinations without captions.

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:

```
\placesidebyside {.1.} {.2.}
```

```
1 TEXT
```

```
2 TEXT
```

```
\placeontopofeachother {.1.} {.2.}
```

```
1 TEXT
```

```
2 TEXT
```

We use them in this way:

```
\placesidebyside {\framed{\Logo[ADE]}} {\framed{\Logo[BUR]}}
```

```
\placeontopofeachother {\framed{\Logo[ADE]}} {\framed{\Logo[BUR]}}
```

14.4 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 ConT_EXt. 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:

```
\beginname
.....
.....
\endname
```

Between the begin- and end command you can use any command you want.

The commands below tell ConT_EXt 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:

0. at that location typeset the questions
1. only use the questions and use the answers in a separate chapter
2. use questions and answers in a separate chapter
3. hide the answers
4. etc.

When we choose option ?? 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: [+].²²

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
```

²² When you use enumerations within text blocks you can best use the `\start...stop` alternative (see page ??).

Now we can recall the ‘important’ messages by:

```
\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  =
  file   = FILE
```

A block is being processed within a group, in other words: within `arg`. 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.

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.

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 guarantees 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 interactivity.

\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]
```

14.5 Opposite blocks

In future versions of ConT_EXt there will be support of spread based typesetting. For the moment the only command available is:

```
\startopposite
```

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
```

14.6 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
```

This can be setup with:

```
\setupmarginblocks [...,*.,...]
```

```
* location   = inmargin left middle right
  style      =
  width      = DIMENSION
  align      =
  top        = COMMAND
  inbetween  = COMMAND
  bottom     = COMMAND
  left       = COMMAND
  right      = COMMAND
  before     = COMMAND
  after      = COMMAND
```


The mechanism to place blocks is still under construction.

14.7 Hiding text

It is possible to hide text (skip during processing) by:

```
\starthiding
```

14.8 Postponing text

Text elements can be postponed (stored) and placed at the next empty page. This option is needed in case ConT_EXt encounters large figures or tables. The postponed textelement is placed at the next page generated by T_EX or forced by the user with a manual page break.

```
\startpostponing
```

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
```

14.9 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
```

```
\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 `\placeblock` can be rather extensive. A buffer can be useful when such large tables are defined.

```
\startbuffer
... 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 `/getbuffer` and `/typebuffer` are available where *buffer* is the name of the buffer.

15 Figures

15.1 Introduction

In this chapter we discuss how to place figures in your document. In section ?? 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.

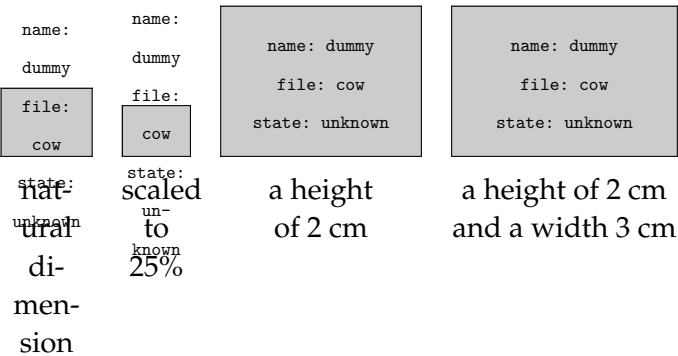
After processing a document the result is a dvi file or, when we use pdfTeX, a pdf file. The dvi document reserves space for the figure, but the figure itself will be put in the document during postprocessing of the dvi file. pdfTeX needs no postprocessing and the external figures are automatically included 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 mechanisms 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.

15.2 Defining figures

A figure is designed within specific dimensions. These dimensions may or may not be known by the document designer.



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 with 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 ?? shows the file formats supported by dvips, dvipsone, and pdf \TeX respectively. pdf \TeX 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	+	-	+	+	-	-	+
pdf \TeX	-	+	+	+	+	+	+

Table 15.1 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, Con \TeX t 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 pdf \TeX even the content is reused, if requested, at a different scale.

A number of characteristics of external figures are specified by:

```

\setuexternalfigures [.*.]

*   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.²³ 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 ConT_EXt 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 T_EXutil. 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 ConT_EXt communicate with T_EXutil 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 ConT_EXt presumes that the figure does exist. This means that the graphics do not need to be physically present on the system.

²³ A similar effect can be obtained with the `--fast` switch in T_EXexec.

Although ConT_EXt 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 is best not to specify a suffix or type.

```
\externalfigure[hownice]
\externalfigure[hownice.pdf]
\externalfigure[hownice][type=pdf]
```

In the first case, ConT_EXt 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, pdfT_EX, etc.), ConT_EXt tries to locate the graphics file, starting with the best quality. When found, ConT_EXt first tries to determine the dimensions itself. If this is impossible, ConT_EXt 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 \setupexternalfigures. 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}]
```

Even if your operating system 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      OPTIONAL

1  IDENTIFIER
2  FILE
3  IDENTIFIER
4  inherits from \setuexternalfigures

```

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 figure ?? we drew a pattern with squares of a factor 10.

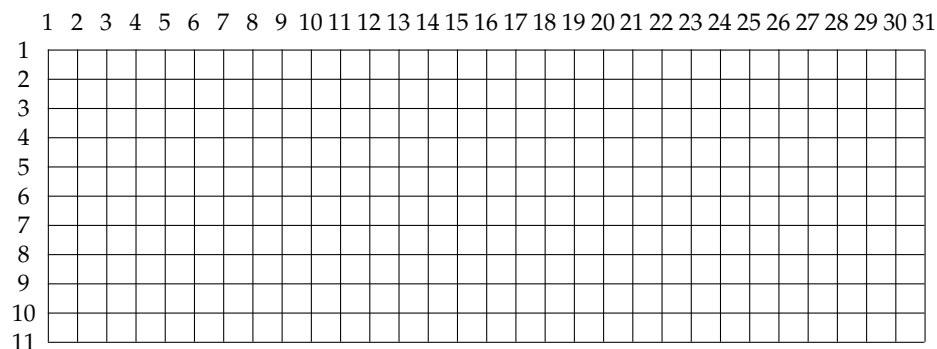


Figure 15.1 Factors at the actual bodyfont.

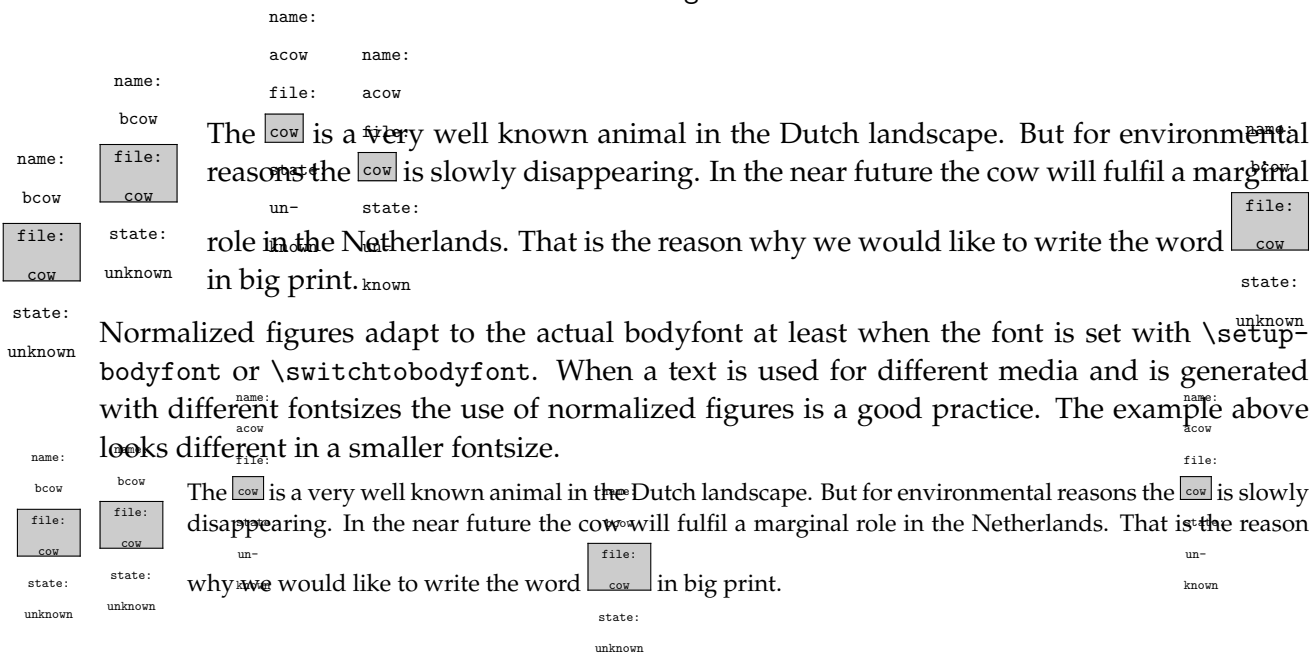
15.3 Recalling figures

```
\externalfigure [.1.] [...2.,...]
                                OPTIONAL
1   FILE
2   inherits from \setupexternalfigures
```

```
\useexternalfigure[acow][cow][factor=10]
\useexternalfigure[bcow][cow][factor=20]

\placefigure[left]{none}{\externalfigure[bcow]}
```

Here we see how `acow` and `bcow` are reused. This code will result in:



15.4 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 15.2 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 of the 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.

15.5 T_EX-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 ??.

```
\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.tmp][width=\textwidth]}
\placefigure
{An example of a \TEX\ figure.}
{\externalfigure[table.tmp][width=.5\textwidth]}

```

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 15.2 An example of a T_EX 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 15.3 An example of a T_EX figure.

Buffers are written to a file with the extension `tmp`, so we recall the table with `table.tmp`. Other types of figures are searched on the directories automatically. With T_EX figures this is not the case. This might lead to conflicting situations when an `eps` figure is meant and not found, but a T_EX file of that name is.

15.6 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 ConT_EXt 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. ConT_EXt will take care of the font management when it encounters MetaPost figures. When color is disabled, or `rgb` is to be converted to `cmyk`, ConT_EXt 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.²⁴ In case of pdf output the MetaPost code is converted into pdf by T_EX.

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 ConT_EXt 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.

ConT_EXt 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 ConT_EXt determines dimensions as if 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`.

15.7 Movies

In ConT_EXt 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}] ...
```

²⁴ Fonts are a problem in MetaPost files, since it is up to the postprocessor to take care of them. In this respect, MetaPost output is not self contained.

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.

15.8 Some remarks on figures

Figures, and photos in particular, have to be produced with consistent proportions. The proportions specified in figure ?? can be used as a guideline. Scaling of photos may cause quality loss.

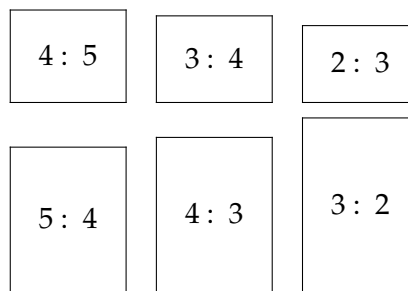


Figure 15.4 Some preferred image proportions.

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 15.5 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.

16 Tables

16.1 Introduction

In ConT_EXt there are two methods of making tables. The method chosen depends on the fact if the table is a component of the running text or the complexity of the table.

Originally ConT_EXt had only one table alternative based on the flexible, robust and complete macropackage T_AB_LE. We added some functionality to that macropackages on behalf of spacing, color and table splitting. This alternative is very powerful with table cells with little text.

As soon as you want to typeset complete paragraphs in a table cell the alternative is inadequate because the width of the cells have to be set manually. Therefore we added the tabulation alternative. Tabulation is not that advanced yet but it is able to automatically divide the available width of the cells. Cells can be filled with complete paragraphs and the tabulation environment has no problems with pagebreaks.

Both the table and the tabulation environment are discussed in this chapter. Because the definition of tables and tabulations differ not much they can be converted easily into each other. Many examples apply to both environments.

16.2 Tables

In (plain) T_EX there are a few options to construct tables, for example:

```
\settabs 5\columns  
\+ The & quick & brown & fox & jumps \cr  
\+ over & the & lazy & dog & ! \cr
```

or:

The	quick	brown	fox	jumps
over	the	lazy	dog	!

These kind of commands are based on the \halign mechanism. In most macropackages tables are developed around this mechanism. In ConT_EXt we use the functionality of T_AB_LE by M. Wichura. Although this macropackage is complete we decided to write a shell around this package. The most important reason was that we wanted to force users to work with a consistent spacing within tables without loss of the T_AB_LE functionality.²⁵

In due time we will develop a more flexible mechanism that is compatible with T_AB_LE and in which MetaPost functionality is integrated.

²⁵ T_AB_LE commands are described in a seperate manual.

year	population
1500	0.90 à 1.00 million
1550	1.20 à 1.30 million
1600	1.40 à 1.60 million
1650	1.85 à 1.90 million
1670	0.95 million
1700	1.85 à 1.95 million
1730	2.12 million
1750	1.90 à 1.95 million
1770	2.11 million
1800	2.08 million
1820	2.20 million

Table 16.1 Growth of population.

We begin with an example of a very simple table in normal `TABLE` coding. The table identified as `als table ??` was defined as follows:

```
\BeginTable
\BeginFormat | 1 | 1 | \EndFormat
\_
| \bf year | \bf population | \\\+22
\_
| 1500      | 0.90 \'a 1.00 million | \\\+20
| 1550      | 1.20 \'a 1.30 million | \\\+00
. ....
| 1800      | 2.08 million          | \\\+00
| 1820      | 2.20 million          | \\\+02
\_
\EndTable
```

We don't consider this as a transparent way of coding a table. Furthermore consistent spacing is far off since the user can alter every interline space at will. As most commands in `ConTEXt` we enclose a table with `\start-``\stop-`commands. These commands are used in stead of the `TABLE` commands `\BeginTable`, `\EndTable`, `\BeginFormat` and `\EndFormat`.

```
\starttable [...] ... \stoptable

* TEXT IDENTIFIER
```

Between [] the column format is defined. Because [] may be used within the table definition [] other variations are possible:

```
\starttable[|1|1|]
\starttable[{|1|1|}]
\starttable{|1|1|}
```

In section ?? we saw that the command `\placetable` gets a table as a last argument. When the command is used the way it is defined above, you will not need the curly braces around the table.

Spacing deserves somewhat more attention. Below the definition of the table ?? is given.²⁶ More advanced examples can be found in the `TABLE` manual.

```
\placetable
  [here] [tab:population]
  {Growth of population.}
  \starttable[|1|1|]
    \HL
    \VL \bf year \VL \bf population \VL\SR
    \HL
    \VL 1500      \VL 0.90 \‘a 1.00 million \VL\FR
    \VL 1550      \VL 1.20 \‘a 1.30 million \VL\MR
    ... ..
    \VL 1800      \VL 2.08 million          \VL\MR
    \VL 1820      \VL 2.20 million          \VL\LR
    \HL
  \stoptable
```

In the `ConTEXt` interface for table definition `\HL` and `\VL` stand for Horizontal Line and Vertical Line. The other commands stand for Separate Row, First Row, Mid Row and Last Row.

These commands can be compared with the active `TABLE` characters `|` and `"`, the spacing command `\` and the line command `\-`.

When keys like `s0` are used in the table specification we have to use spaces in the specification. For example:

```
\starttable[s0 |1|1|]
```

In that case we use a space to end the number. We give some more examples below. When a table is not framed by (`\VL`), we may have to suppress undesired whitespace. The background of the tables illustrates the effect of the different location of `s0`.

²⁶ Source: Delta 2, Nederlands verleden in vogelvlucht, de nieuwe tijd: 1500 tot 1813, S. Groenveld and G.J. Schutte, Martinus Nijhoff Uitgevers, Leiden, 1992.

first	second
alfa	beta
one	two

```

\starttable[|l|l|]
\NC first \NC second \NC \SR
\HL
\NC alfa \NC beta \NC \FR
\NC one \NC two \NC \LR
\stoptable

```

first	second
alfa	beta
one	two

```

\starttable[s0 | c s1 | c s0 |]
\NC first \NC second \NC \SR
\HL
\NC alfa \NC beta \NC \FR
\NC one \NC two \NC \LR
\stoptable

```

first	second
alfa	beta
one	two

```

\starttable[o0 | c | c o0 |]
\NC first \NC second \NC \SR
\HL
\NC alfa \NC beta \NC \FR
\NC one \NC two \NC \LR
\stoptable

```

We may want to typeset tables like these:

first weight (full can)	151,2 g
second weight (empty can)	35,6 g
filled weighed t	115,6 g

This table is defined by:

```

\starttable[ s0 | l w(6cm) | r s0 | l | ]
\NC first weight (full can) \NC 151,2 g \NC \NC\FR
\NC second weight (empty can) \NC 35,6 g \NC \NC\LR
\NC \NC \NC \- \NC~\Smash{--} \NC\NR
\NC filled weighed t \NC 115,6 g \NC \NC\SR
\stoptable

```

One or more table characteristics can be set up with:

```
\setuptables [...,.*,...]
```

```
* distance      = small medium big
bodyfont        =
HL              = NUMBER small medium big none
VL              = NUMBER small medium big none
depth           = NUMBER strut
height          = NUMBER strut
rulethickness    = DIMENSION
rulecolor       = IDENTIFIER
align           =
commands        = COMMAND
align           =
background       = screen color none
backgroundscreen = NUMBER
backgroundcolor  = IDENTIFIER
bodyfont        =
```

The keys HL and VL refer to the line thickness. The tables in this manual are set with VL=none en HL=medium.

first	second	first	second	first	second
first	second	first	second	first	second
first	second	first	second	first	second

HL=small HL=medium HL=big

Figure 16.1 The available line thicknesses in tables.

By means of commands specific parameters of `TABLE` itself can be defined, for example:

```
\setuptables[commands=\Expand]
```

This results in a table with a width `\hsize`.

The parameters `distance` and `bodyfont` relate to the commands mentioned earlier `\SR`, `\FR`, `\MR` and `\LR`.

10pt - big		10pt - medium		10pt - small		10pt - none	
T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X
T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X
T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X	T _E X

9pt - big	9pt - medium	9pt - small	9pt - none
TeX	TeX	TeX	TeX
TeX	TeX	TeX	TeX
TeX	TeX	TeX	TeX
8pt - big	8pt - medium	8pt - small	8pt - none
TeX	TeX	TeX	TeX
TeX	TeX	TeX	TeX
TeX	TeX	TeX	TeX

Next to these commands we have `\DL` that stands for Division Line, `\DC` stands for Division Column and `\DR` stands for Division Row, while `\NC` is used to jump to the Next Column. We also use the command `\NR` that stands for Next Row.

alfa	beta	gamma
beta	gamma	alfa
gamma	alfa	beta

```

\starttable[|c|c|c|]
\HL
\VL alfa \NC beta \NC gamma \VL\SR
\DC \DL \DC \DR
\VL beta \VL gamma \VL alfa \VL\SR
\DC \DL \DC \DR
\VL gamma \NC alfa \NC beta \VL\SR
\HL
\stoptable

```

The command `\DC` applies default to only one column and is equal to the `\TABLE` command `\=`. When we want to draw a multi column line we define that in the way that is shown in the example below.

low	n/m
	n m
alfa	1 a
alfa	2 b
alfa	3 c

```

\starttable[|c|c|c|]
\HL
\VL \LOW{low} \VL \TWO{n/m} \VL\SR
\DC \DL[2] \DR
\VL \VL n \VL m \VL\SR
\HL
\VL alfa \VL 1 \VL a \VL\FR
\VL alfa \VL 2 \VL b \VL\MR
\VL alfa \VL 3 \VL c \VL\LR
\HL
\stoptable

```

In stead of `\TWO` we also could have used the `\TABLE`-command `\use2`. Because of our criterium of consistent spacing we can not replace `\LOW` by `\lower` in every situation.

We show here another example of the use of \DL:

	Mickey	Goofy	\starttable[l c c]			
			\DC	\DL[2]		\DR
Donald	★		\NC	\VL Mickey	\VL Goofy	\VL\SR \HL
Pluto		★	\VL Donald	\VL \$★	\VL	\VL\SR \HL
			\VL Pluto	\VL	\VL \$★	\VL\SR \HL
			\stoptable			

In table ?? the commands are summarized.

command	meaning	command	meaning	command	meaning
\FR	First Row	\NC	Next Column	\HL	Horizontal Line
\LR	Last Row	\FC	First Column	\VL	Vertical Line
\MR	Mid Row	\MC	Mid Column		
\SR	Separate Row	\LC	Last Column	\DL	Division Line
\NR	No Row			\DL[n]	Division Line n
		\LOWargtext	Low Text	\DC	Division Column
\AR	Auto Row			\DR	Division Row

Table 16.2 Table-commands.

You could ask yourself whether determining inter line spacing within a table should be done manually or automatically. We would say automatically. In the first place this reduces the chance for errors. And in the second place T_EX can do the job for us. The commands \SR, \FR etc. are a consequence of T_AB_LE mechanism that was not strict enough for us. Instead of providing these spacing commands yourself, you can use \AR and rely on ConT_EXt to sort out the spacing as good as possible.

In table ?? we used \AR. The definition of this table looks likes this.

```
\starttable[|l|c|c|]
\DC          \DL[2]          \DR
\NC          \VL Mickey    \VL Goofy    \VL\SR \HL
\VL Donald  \VL $★        \VL          \VL\SR \HL
\VL Pluto   \VL          \VL $★        \VL\SR \HL
\stoptable
```

This mechanism works with rules too, as is demonstrated in table ??.

```
\starttable[|l|l|l|l|l|l|l|B|]
\HL
\NC\bf8\NC\bf7\NC\bf6\NC\bf5\NC\bf4\NC\bf3\NC\bf2\NC\bf1\NC    \NC\AR
\HL
```

name: dummy
file: registers-buffer
state: unknown

Table 16.3 Registration (valves) at the base-side of the Schnitke organ in Zwolle.

```

\NC      \NC  8 \NC      \NC  2 \NC  4 \NC  3 \NC  8 \NC      \NC A \NC\AR
\NC  8 \NC  1+\NC  2 \NC  4 \NC  4 \NC  6 \NC  8 \NC HZ \NC B \NC\AR
\NC      \NC  2 \NC  8 \NC 32 \NC  2 \NC  8 \NC 16 \NC HB \NC C \NC\AR
\NC  8 \NC 16 \NC      \NC  2 \NC  4 \NC  8 \NC 16 \NC HR \NC D \NC\AR
\NC      \NC  8 \NC      \NC      \NC  4 \NC  4 \NC  8 \NC PH \NC E \NC\AR
\HL
\stoptable

```

At the moment the commands `\FC`, `\MC` and `\LC` are equivalent to `\NC`, but in the future they may show additional functionality.

16.3 Color in tables

The macros that work in the background are rather complicated but the mechanism for color in tables is this:

8	7	6	5	4	3	2	1	
	8		2	4	3	8		A
8	1+	2	4	4	6	8	HZ	B
	2	8	32	2	8	16	HB	C
8	16		2	4	8	16	HR	D
	8			4	4	8	PH	E

Table 16.4 The pitch of the registrations (valves) at the base-side of the Schnitke organ in Zwolle.

- 0. draw a thick horizontal line
- 1. jump back vertically
- 2. set the textline

The height and depth of the drawn line have to be equal to that of the textline which accurately defined. We do not have to bother ourselves with the width because T_EX does that for us. The necessary commands fall back on the command \noalign.

The commands to make the cell backgrounds grey or colored look like the commands to draw Division Lines (see table ??).

command	meaning
\BL	Background Line
\BL[n,type,specification]	Background Line n
\BC	Background Column
\BR	Background Row
\CL[specification]	Color Line
\RL[specification]	Raster Line

Table 16.5 The table-commands to color cells.

The examples below illustrate the use of these commands. The lines where the backgrounds are specified *precede* those with the text. Note that just as with \DL the command automatically goes to the next column. So do not use more \BC's than necessary.

With \BR you recall the last specification. This command is followed by commands like \SR and \FR that give the height of the (yet to follow) textline.

Optional backgrounds are color and screen. When no column is specified the commands are in effect over only one column. We start with some simple examples.

test	test	\starttable[c c]
		\BC \BL \SR
test	test	\HL
test	test	\VL test \VL test \VL\SR
test	test	\HL
		\VL test \VL test \VL\FR
		\VL test \VL test \VL\MR
		\VL test \VL test \VL\LR
		\HL
		\stoptable

With \BC we go to column 1. With \BL we go to column 2 where a background is intended. At last we specify by \SR that the background should be used in a (still to follow) Separate Row. The space between the table columns is taken into account during the background generation. The reverse alternative is defined below. Keep in mind that we use \BL for skipping to the next column.

test	test	\starttable[c c]
		\BL \SR
test	test	\HL
test	test	\VL test \VL test \VL\SR
test	test	\HL
		\VL test \VL test \VL\FR
		\VL test \VL test \VL\MR
		\VL test \VL test \VL\LR
		\HL
		\stoptable

Two or more adjoining cells get a background when the number of columns is specified:

test	test	\starttable[c c]
		\BL[2] \SR
test	test	\HL
test	test	\VL test \VL test \VL\SR
test	test	\HL
		\VL test \VL test \VL\FR
		\VL test \VL test \VL\MR
		\VL test \VL test \VL\LR
		\HL
		\stoptable

And here is another example.

test	test	test	\starttable[c c c]	
test	test	test	\BL[3]	\SR
test	test	test	\HL	
test	test	test	\VL test \VL test \VL test \VL	\SR
test	test	test	\HL	
			\VL test \VL test \VL test \VL	\FR
			\VL test \VL test \VL test \VL	\MR
			\VL test \VL test \VL test \VL	\LR
			\HL	
			\stoptable	

test	test	test	\starttable[c c c]	
test	test	test	\BC \BL[2]	\SR
test	test	test	\HL	
test	test	test	\VL test \VL test \VL test \VL	\SR
test	test	test	\HL	
			\VL test \VL test \VL test \VL	\FR
			\VL test \VL test \VL test \VL	\MR
			\VL test \VL test \VL test \VL	\LR
			\HL	
			\stoptable	

test	test	test	\starttable[c c c]	
test	test	test	\BC \BC \BL	\SR
test	test	test	\HL	
test	test	test	\VL test \VL test \VL test \VL	\SR
test	test	test	\HL	
			\VL test \VL test \VL test \VL	\FR
			\VL test \VL test \VL test \VL	\MR
			\VL test \VL test \VL test \VL	\LR
			\HL	
			\stoptable	

test	test	test	\starttable[c c c]
test	test	test	\BC \BL \SR
test	test	test	\HL
test	test	test	\VL test \VL test \VL test \VL\SR
test	test	test	\HL
			\VL test \VL test \VL test \VL\FR
			\VL test \VL test \VL test \VL\MR
			\VL test \VL test \VL test \VL\LR
			\HL
			\stoptable

In the example below there seems to be a missing \BC. Note that there is \BL to jump to the next column.

test	test	test	\starttable[c c c]
test	test	test	\BL \BL \SR
test	test	test	\HL
test	test	test	\VL test \VL test \VL test \VL\SR
test	test	test	\HL
			\VL test \VL test \VL test \VL\FR
			\VL test \VL test \VL test \VL\MR
			\VL test \VL test \VL test \VL\LR
			\HL
			\stoptable

test	test	test	\starttable[c c c]
test	test	test	\BC \BL \SR
test	test	test	\HL
test	test	test	\VL test \VL test \VL test \VL\SR
test	test	test	\HL
			\BR\FR
			\VL test \VL test \VL test \VL\FR
			\BR\MR
			\VL test \VL test \VL test \VL\MR
			\BR\LR
			\VL test \VL test \VL test \VL\LR
			\HL
			\stoptable

Because \SR closes a line we do not have to specify \BC's in the other columns.

We complete these series of examples with a few wider tables.

aa	bb	cc	dd	<code>\starttable[c c c c]</code>
aa	bb	cc	dd	<code>\BC \BL[r,0.7] \BL[r,0.9] \SR</code>
aa	bb	cc	dd	<code>\HL</code>
aa	bb	cc	dd	<code>\VL aa \VL bb \VL cc \VL dd \VL\SR</code>
aa	bb	cc	dd	<code>\HL</code>
				<code>\BR\FR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\FR</code>
				<code>\BR\MR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\MR</code>
				<code>\BR\LR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\LR</code>
				<code>\HL</code>
				<code>\stoptable</code>

With `\BR` we recall the most recent specification so we do not have to specify each background separately. The spacing however should be specified.

The first line in the example above could have been shorter:

```
\BC \RL[0.7] \RL[0.9] \SR
```

You can use screens and colors in one table. Screens may also be specified in terms of colors. A screen with a greyscale $Gr = .9$ can be compared with a color with RGB-values $r = g = b = .9$. Most of the time you can use color and some greyscale.

TODO: above table doesnt compile

aa	bb	cc	dd	<code>\starttable[c c c c]</code>
aa	bb	cc	dd	<code>%\BC \BL[r,0.8] \BL[c,red] \SR</code>
aa	bb	cc	dd	<code>\HL</code>
aa	bb	cc	dd	<code>\VL aa \VL bb \VL cc \VL dd \VL\SR</code>
aa	bb	cc	dd	<code>\HL</code>
				<code>\BR\FR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\FR</code>
				<code>\BR\MR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\MR</code>
				<code>\BR\LR</code>
				<code>\VL aa \VL bb \VL cc \VL dd \VL\LR</code>
				<code>\HL</code>
				<code>\stoptable</code>

We can see that with `\BL` another background specification `r` or `c` is used. The same result is obtained with screen or color. Colors and screens can be used interchangeably.

A row can be typeset with a background by means of `\RL` and `\CL`, without adding rownumbers.

aa	bb	cc	dd	\starttable[c c c c]
aa	bb	cc	dd	\RL\FR
aa	bb	cc	dd	\VL aa \VL bb \VL cc \VL dd \VL\FR
aa	bb	cc	dd	\VL aa \VL bb \VL cc \VL dd \VL\MR
aa	bb	cc	dd	\RL\MR
				\VL aa \VL bb \VL cc \VL dd \VL\MR
				\VL aa \VL bb \VL cc \VL dd \VL\LR
				\stoptable

aa	bb	cc	dd	\starttable[c c c c]
aa	bb	cc	dd	%\CL[green]\FR
aa	bb	cc	dd	\VL aa \VL bb \VL cc \VL dd \VL\FR
aa	bb	cc	dd	\VL aa \VL bb \VL cc \VL dd \VL\MR
aa	bb	cc	dd	\VL aa \VL bb \VL cc \VL dd \VL\MR
				\VL aa \VL bb \VL cc \VL dd \VL\LR
				\stoptable

The next (specifications of) commands are equivalent:

```
\BL[c,...] \BL[color,...] \COLOR[...]
\BL[r,...] \BL[screen,...] \SCREEN[...]
```

The reader will have noticed that cells get a background even when no background is specified. These default backgrounds can be set up with:

```
\setuptables
[backgroundcolor=,
 backgroundscreen=,
 background=]
```

The key background can get the value `color` or `screen`. The default value is `screen`. As a `backgroundcolor` you can specify the name of a color or as a `backgroundscreen` with a number between 0 and 1.

Unfortunately the line mechanism is not that accurate. Whether the cause lies with ConT_EXt, T_EX or with the DVI-drivers is unclear stil. It is important that the Horizontal Rules (`\HL`) are placed *after* the background is set otherwise the background becomes foreground and part of the line will disappear. The earlier examples show how to specify correct; the example below shows how it is *not* done.

aa	bb	cc	dd

```

\starttable[|c|c|c|c|]
%\BC      \BL[c,green] \BL[c,red] \SR
\HL
\VL aa \VL bb \VL cc \VL dd \VL\SR
\BR\FR
\HL
\VL aa \VL bb \VL cc \VL dd \VL\FR
\BR\MR
\VL aa \VL bb \VL cc \VL dd \VL\MR
\BR\LR
\VL aa \VL bb \VL cc \VL dd \VL\LR
\HL
\stoptable

```

In none of the examples thusfar we see two adjacent colored columns. The reason is that this is not possible without complex constructions. One solution is using dummy-columns:

aa	bb	cc

```

\starttable[|c||c||c|]
%\BL[c,green] \BL[c,red] \FR
\NC aa \NC\NC bb \NC\NC cc \NC\FR
\BR\MR
\NC aa \NC\NC bb \NC\NC cc \NC\MR
\BR\LR
\NC aa \NC\NC bb \NC\NC cc \NC\LR
\stoptable

```

We see that the distance between columns is somewhat too big. We solve that by adapting the `TABLE`-variables `\InterColumn...`. The alternative of using `\-` in stead of `\=` is rejected, because the results are rather poor.

You are free to experiment on this issue. The example shows that we can fool the mechanism. In this case all textlines must end with `\SR`.

aa	bb	cc
aa	bb	cc

```

\starttable[|c|c|c|]
%\BL[c,green] \BL[c,red] \MR
\NC aa \NC bb \NC cc \NC\SR
\BR\MR
\NC aa \NC bb \NC cc \NC\SR
\BR\MR
\NC aa \NC bb \NC cc \NC\SR
\stoptable

```

We also see some extra text in this table. We can avoid extra spaces with the command `\trac-
etablesfalse`. Default interline inconsistencies are reported during document generation.

Next to the Format Keys from `\tblr` the Format Key `K` is available that results in typesetting the text in that column in capitals (`\kap`). In addition to `n` and `N` there are `q` and `Q`. This command is meant for aligning numbers and it works with commas in stead of dots.

```
\starttable[{| l k | q[3,4] | Q[2,1] | c |}]
.....
\stoptable
```

In this situation we use an extra set of `arg` to prevent any problems. The use of those are explained in the `\tblr` manual.

16.4 Tables with identical layouts

Integrating hundreds of tables is for \TeX hardly a problem. Definition of these tables is a formidable job. When tables have comparable formats its obvious to specify common elements only once. The next example will show how it works. First we define the table layout:

```
\definetabletemplate[demo][|r|l|]
```

The template, with the name `demo`, can be used in the table definition:

left	right	<code>\starttable[demo]</code>
over	under	<code>\VL left \VL right \VL\AR</code>
		<code>\VL over \VL under \VL\AR</code>
		<code>\stoptable</code>

and

left	right	<code>\starttable[demo]</code>
over	under	<code>\VL left \VL right \VL\AR</code>
		<code>\VL over \VL under \VL\AR</code>
		<code>\stoptable</code>

We can redefine such a table layout. Next to the layout we can specify the table head and tail.

```
\definetabletemplate[demo][|r|l|][demo head][demo tail]
```

The head and tail are defined separately:

```
\starttablehead[demo head]
\HL \VL this \VL that \VL\AR \HL
\stoptablehead

\starttabletail[demo tail]
\HL
\stoptabletail
```

The table we defined earlier looks like this:

this	that	<code>\starttable[demo]</code>
left	right	<code>\VL left \VL right \VL\AR</code>
over	under	<code>\VL over \VL under \VL\AR</code>
		<code>\stoptable</code>

The core of this mechanism is the command:

```
\definetabletemplate [...1.] [...2.] [...3.] [...4.]
                                OPTIONAL OPTIONAL
1  IDENTIFIER
2  TEXT
3  TEXT
4  TEXT
```

16.5 Splitting tables

Like the title of this section says: tables can be split. Results of splitting tables is satisfactory but the mechanism is not 100% waterproof. A table will be split at a pagebreak when you use `\starttables` in stead of `\starttable`

```
\starttables [...*.] ... \stoptables
*  TEXT  IDENTIFIER
```

The table head and tail are defined in the way described in the last section.

```
\starttablehead
...
\stoptablehead
\starttabletail
...
\stoptabletail
```

Off course you can also use head and tail definitions that are defined globally in combination with a specified table layout. It may be necessary to number a split table. The next command will do that.

```
\splitfloat [...,1,...] {...2.} {...3.}
                                OPTIONAL
1  inherits from \setupfloatsplitting
2  TEXT
3  TEXT
```


We may have specified the next splittable table:

```
\startbuffer
\starttablehead
\HL \VL Greec \VL Latin \VL\AR \HL
\stoptablehead
\starttabletail
\HL
\stoptabletail

\starttables[|mc|c|]
\VL \alpha \VL a \VL\AR
\VL \beta \VL b \VL\AR
...
\VL \zeta \VL z \VL\AR
\stoptables
\stopbuffer
```

Because we stored the table in a buffer we specify the table in the following way:

```
\splitfloat
  {\placetable[here][tab:demo]{A demo table.}}
  {\getbuffer}
```

And the result will be:

there is nothing to split

Set ups can be added as a first optional argument. One of the parameters is `lines`, with which you reserve space for the caption and vertical spacing. Default the value is 3.

```
\setupfloatsplitting [...,.*,...]
```

```
* conversion = numbers characters Characters romannumerals Romannumerals
  lines      = NUMBER
```

The parameter `conversion` is used for the subnumbering of the subtables. Every subtable automatically gets a subnumber. The parameter `conversion` takes care of its representation (1, 2, 3 – a, b, c – I, II, III – etc.).

16.6 Buffers and scaling

Very big tables combined with the floating block mechanism can be a confusing sight in the source file. The following alternative is recommended.

```

\startbuffer
... table ...
\stopbuffer
\placetable{A title.}{\getbuffer}

```

In this way we can keep track of what happens. Another advantage is the fact that we can manipulate tables in this way (see section ??). A table that is too wide for the page width can be downscaled to that width. Here is an example:

```

\placetable
{Fits exactly.}
{\externalfigure[buffer] [width=\textwidth]}

```

Default a figure buffer is defined as the standard buffer. Table ?? is type set in that way.

16.7 Remarks

Within `TABLE` the bar has a special meaning just like the double quote. Within `ConTEXt` we use `||` for combined words and other word related tricks. Furthermore German users want to use `"` as an umlaut trigger. Conflicts within the `TABLE` mechanism are at hand and therefore both characters keep their original meaning. If you can not live without the `|` and `"` you can use the next command to make them behave the way that was meant in `TABLE`. We do not recommend that.

```

\ObeyTableBarAndQuote
(unfinished)

```

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 than 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.

```
in   spacing left
jn   spacing right
kn   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 fact 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
barg.. place .. before the entry
aarg.. 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 arg are important because `\inframed` expects these.

Uggly	(isn't it?)	he says.
Beautiful	(but meaningless)	I would say.

We can use `h` for alternative situations, like:

item	number
figures	████████
tables	████████████████
formulas	████████████████████

All three cells are adapted. Do not forget the arg in the column with the numbers!

```
\unexpanded\def\SmallDash#1{\blackrule[width=#1em]}
\starttabulate[|1|1h\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	<code>\EQ</code>	<code>\RQ</code>	<code>\HQ</code>
no	<code>\NC</code>	<code>\RC</code>	<code>\HC</code>

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 <code>\starttable</code> 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 <code>\starttabulate</code> 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[|1|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:

`\definetaabulate[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:

```
\definetaabulate [.1.] [.2.] [.3.]
                  OPTIONAL 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:

unknown setup 'starttabulate'

In this command the first argument specifies the cells, the second and optional argument the set up.

```

\setuptabulate [.1.] [...,\u2.,...]
                        OPTIONAL

1  IDENTIFIER
2  unit                = DIMENSION
   indenting           =
   before              = COMMAND
   after               = COMMAND
   inner               = COMMAND
   EQ                  = TEXT
   rulecolor           = IDENTIFIER
   align               =
   rulethickness       = DIMENSION
   distance            = blank grid depth DIMENSION small medium big none
   bodyfont            =
   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.

- like this
- 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.

0. first	•••••	this or that	-. alpha
1. second	•••••	so and so	α . beta
2. third	•••••	here or there	β . 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	<u>5.186</u>
together	<u>6.406</u>

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 \sim , but we have to indicate the meaning of \sim explicitly. This is caused by the fact that we still want to use the \sim within paragraphs as a 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 $\text{Con}\text{\TeX}t$ 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 the 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 originates from the older functionality that was part of earlier ConT_EXt versions.

```
w = width = the width of a box pt
h = height = the height of a box pt
d = depth = the depth of a box pt
```

In a similar way the commands for typesetting facts are defined.

```
\definetaabulate [fact] [|R|ecmj1|i1mR|]
\setuptaabulate [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[|B|p|B|]
\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, synthesize, winnow the wheat from the chaff and separate the sheep from the goats.	Edward Tufte
----------------	--	---------------------

As was said earlier ConT_EXt 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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ relates to math typesetting since the main reason for developing $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ was the need for typesetting math.

In $\mathrm{ConT}_{\mathrm{E}}\mathrm{Xt}$ math typesetting is not really an issue. $\mathrm{ConT}_{\mathrm{E}}\mathrm{Xt}$ 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 $\mathrm{ConT}_{\mathrm{E}}\mathrm{Xt}$ 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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ community. Most extensions concern the interface and consistent spacing. In this chapter we pay attention to the standard functionality in $\mathrm{ConT}_{\mathrm{E}}\mathrm{Xt}$.

18.2 Basic commands

Typesetting formulas is one of the strong points of $\mathrm{T}_{\mathrm{E}}\mathrm{X}$. 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:

unknown setup 'setupformulas'

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 [...1...] {..2.} $$..3.$$
                OPTIONAL  OPTIONAL
1  REFERENCE
2  TEXT
3
```

`\placesubformula [...1,...] {..2} $$..3$$`

1 REFERENCE

2 TEXT

3

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 T_EX-commando's like `\displaylines`. In the examples we use `$$` to save some space; however we advise you to use the command `\startformula`.

$$a + b = c \tag{18.1}$$
$$a + b = c \tag{18.2}$$
$$a + b = c \tag{18.2a}$$
$$a + b = c \tag{18.2b}$$
$$a + b = c \tag{18.3}$$
$$a + b = c \tag{18.4}$$
$$a + b = c \tag{18.4a}$$
$$a + b = c \tag{18.4b}$$
$$a + b = c \quad (18.5) \quad (18.6)$$

	<code>\placeformula</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.7)$		(18.8)
	<code>\placesubformula {a}</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.8a)$		(18.8a)
	<code>\placesubformula {b}</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.8b)$		(18.8b)
e: ??	<code>\placeformula [e]</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.9)$		(18.10)
f: ??	<code>\placeformula [f]</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.11)$		(18.12)
g: ??	<code>\placesubformula[g]{a}</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.12a)$		(18.12a)
h: ??	<code>\placesubformula[h]{b}</code>	<code>\$\$ a + b = c \quad \formulanumber</code>	<code>\$\$</code>	
		$a + b = c \quad (18.12b)$		(18.12b)
	<code>\placesubformula</code>	<code>\$\$ a + b = c \quad \formulanumber {a} \$\$</code>		
		$a + b = c \quad (18.12a)$		(18.12)
	<code>\placesubformula</code>	<code>\$\$ a + b = c \quad \formulanumber {b} \$\$</code>		
		$a + b = c \quad (18.12b)$		(18.12)
i: ??	<code>\placeformula</code>	<code>\$\$ a + b = c \quad \formulanumber[i]</code>	<code>\$\$</code>	
		$a + b = c \quad (18.13)$		(18.14)
j: ??	<code>\placeformula</code>	<code>\$\$ a + b = c \quad \formulanumber[j]</code>	<code>\$\$</code>	
		$a + b = c \quad (18.15)$		(18.16)
k: ??	<code>\placesubformula</code>	<code>\$\$ a + b = c \quad \formulanumber[k]{a} \$\$</code>		
		$a + b = c \quad (18.16a)$		(18.16)

l: ?? \placesubformula	\$\$ a + b = c \quad \text{\formulanumber[l]{b}} \$\$	
	$a + b = c \quad (18.16b)$	(18.16)
m: ?? \placeformula [m]	\$\$ a + b = c \quad \text{\formulanumber {a}} \$\$	
	$a + b = c \quad (18.17a)$	(18.18)
n: ?? \placeformula [n]	\$\$ a + b = c \quad \text{\formulanumber {b}} \$\$	
	$a + b = c \quad (18.19b)$	(18.20)
o: ?? \placesubformula[o]	\$\$ a + b = c \quad \text{\formulanumber {a}} \$\$	
	$a + b = c \quad (18.20a)$	(18.20)
p: ?? \placesubformula[p]	\$\$ a + b = c \quad \text{\formulanumber {b}} \$\$	
	$a + b = c \quad (18.20b)$	(18.20)
q: ?? \placeformula [q]	\$\$ a + b = c \quad \text{\formulanumber[r]{a}} \$\$	
r: ??	$a + b = c \quad (18.21a)$	(18.22)
s: ?? \placeformula [s]	\$\$ a + b = c \quad \text{\formulanumber[t]{b}} \$\$	
t: ??	$a + b = c \quad (18.23b)$	(18.24)
u: ?? \placesubformula[u]	\$\$ a + b = c \quad \text{\formulanumber[v]{a}} \$\$	
v: ??	$a + b = c \quad (18.24a)$	(18.24)
w: ?? \placesubformula[w]	\$\$ a + b = c \quad \text{\formulanumber[x]{b}} \$\$	
x: ??	$a + b = c \quad (18.24b)$	(18.24)

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.

unknown setup 'startformula'

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:

$$a \times b = b \times a \quad (18.25)$$

$$a + b = b + a \quad (18.25)$$

$$ac + bc = (a + b)c \quad (18.26x)$$

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  EMPTY
3  EMPTY
4  EMPTY
```

```
\startfact .1. .2. .3. ... \stopfact

1  EMPTY
2  EMPTY
3  EMPTY
```

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:

$$F = ma \quad (18.27)$$

```
F = force      N
m = mass       kg
a = acceleration m/s^2
```

Determine by means of formula ?? 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 ??.

```
\unit {strange} {m^3\!/s^2} {a strange unit}
```

In this case the `\!` takes care of backskipping the `/` in such a way that instead 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 {...}

* TEXT
```

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:

abcde

which we would have obtained by typing:

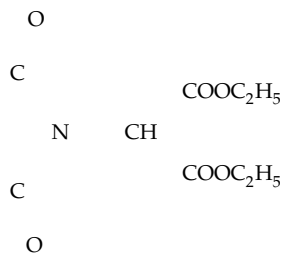
```
$a$b$c$d$e$
```

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 P_TCT_EX 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,0,0,
 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:

$$\frac{\text{\ch{N}}}{\text{\ch{O}}}$$

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 T_EX we offer the next commands for switching to some specialized fonts:

```
\frak  fraktur      \mathfrak{ABC}
\goth  gothic       \mathgoth{abc}
\cal   calligraphic \mathcal{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

\startformula
  x \times \maframed{y} \times y^{\maframed{z}_{\maframed{z}}}
\stopformula
```

$$y + \boxed{y} + y^2 + y^{\boxed{2}}$$

$$x \times \boxed{y} \times y^{\boxed{z}_{\boxed{z}}}$$

In this example we can see that the superscript 2 is rather big. This can be prevented by using the commands `\super` and `\suber` in stead of `^` and `_`.

```
\startformula
  x \times \maframed{y} \times y\super{\maframed{z}\suber{\maframed{z}}}\
\stopformula
```

$$x \times \boxed{y} \times y^{\boxed{z}}$$

If you want to use `^` and `_` anyhow, than you can use `\enablesupersub` to reach the same effect. 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\super{\maframed[framecolor=red]{z}\suber{z}}\
\stopformula
```

$$x \times y^{\boxed{z}}$$

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 TEXT

2 TEXT

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 ConT_EXt 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`.

—— math characters – default —————

α	1 alpha	N	0 Nu	\bigcup	3 bigcup
β	1 beta	Ξ	0 Xi	\int	3 intop
γ	1 gamma	O	0 Omicron	\prod	3 prod
δ	1 delta	Π	0 Pi	Σ	3 sum
ϵ	1 epsilon	R	0 Rho	\otimes	3 bigotimes
ζ	1 zeta	Σ	0 Sigma	\oplus	3 bigoplus
η	1 eta	T	0 Tau	\odot	3 bigodot
θ	1 theta	Υ	0 Upsilon	\oint	3 ointop
ι	1 iota	Φ	0 Phi	\sqcup	3 bigsqcup
κ	1 kappa	X	0 Chi	\int	2 smallint
λ	1 lambda	Ψ	0 Psi	\triangleleft	1 triangleleft
μ	1 mu	Ω	0 Omega	\triangleright	1 triangleright
ν	1 nu	\aleph	2 aleph	\triangle	2 bigtriangleup
ξ	1 xi	\imath	1 imath	∇	2 bigtriangledown
\omicron	1 omicron	j	1 jmath	\wedge	2 wedge
π	1 pi	ℓ	1 ell	\vee	2 vee
ρ	1 rho	\wp	1 wp	\cap	2 cap
σ	1 sigma	\Re	2 Re	\cup	2 cup
τ	1 tau	\Im	2 Im	\ddagger	2 ddagger
υ	1 upsilon	∂	1 partial	\dagger	2 dagger
ϕ	1 phi	∞	2 infity	\sqcap	2 sqcap
χ	1 chi	\prime	2 prime	\sqcup	2 sqcup
ψ	1 psi	\emptyset	2 emptyset	\uplus	2 uplus
ω	1 omega	∇	2 nabla	\amalg	2 amalg
ε	1 varepsilon	\top	2 top	\diamond	2 diamond
ϑ	1 vartheta	\perp	2 bot	\bullet	2 bullet
ϖ	1 varpi	\triangle	2 triangle	\wr	2 wr
ϱ	1 varrho	\forall	2 forall	\div	2 div
ς	1 varsigma	\exists	2 exists	\odot	2 odot
φ	1 varphi	\neg	2 neg	\oslash	2 oslash
A	0 Alpha	\flat	1 flat	\otimes	2 otimes
B	0 Beta	\natural	1 natural	\ominus	2 ominus
Γ	0 Gamma	\sharp	1 sharp	\oplus	2 oplus
Δ	0 Delta	\clubsuit	2 clubsuit	\mp	2 mp
E	0 Epsilon	\diamond	2 diamondsuit	\pm	2 pm
Z	0 Zeta	\heartsuit	2 heartsuit	\circ	2 circ
H	0 Eta	\spadesuit	2 spadesuit	\bigcirc	2 bigcirc
Θ	0 Theta	\amalg	3 coprod	\setminus	2 setminus
I	0 Iota	\bigvee	3 bigvee	\cdot	2 cdot
K	0 Kappa	\bigwedge	3 bigwedge	$*$	2 ast
Λ	0 Lambda	\oplus	3 biguplus	\times	2 times
M	0 Mu	\bigcap	3 bigcap	\star	1 star

\propto	2	propto	\leftharpoonup	1	leftharpoonupdown	\lceil	2	rceil
\sqsubseteq	2	sqssubseteq	\rightarrow	1	rightharpoonup	\lfloor	2	lfloor
\sqsupseteq	2	sqsupseteq	\rightharpoonup	1	rightharpoonupdown	\rfloor	2	rfloor
\parallel	2	parallel	\hookleftarrow	1	lhook	$\sqrt{}$	2	sqrt
\mid	2	mid	\hookrightarrow	1	rhook	\lvert	2	lvert
\dashv	2	dashv	\cdot	1	ldotp	\rvert	2	rvert
\vdash	2	vdash	\cdot	2	cdotp	\parallel	2	lVert
\nearrow	2	nearrow	$:$	0	colon	\parallel	2	rVert
\searrow	2	searrow	$\acute{}$	0	acute	\dagger	2	dag
\nwarrow	2	nwarrow	$\grave{}$	0	grave	\ddagger	2	ddag
\swarrow	2	swarrow	$\ddot{}$	0	ddot	\S	2	S
\Leftrightarrow	2	Leftrightarrow	\sim	0	tilde	\P	2	P
\Leftarrow	2	Leftarrow	\circ	0	mathring	\bigcirc	2	Orb
\Rightarrow	2	Rightarrow	$\bar{}$	0	bar	\cdot	1	mathperiod
\leq	2	leq	$\breve{}$	0	breve	\cdot	1	textperiod
\geq	2	geq	$\check{}$	0	check	$,$	1	mathcomma
$>$	2	succ	$\hat{}$	0	hat	$,$	1	textcomma
$<$	2	prec	$\vec{}$	1	vec	Γ	0	varGamma
\approx	2	approx	\cdot	0	dot	Δ	0	varDelta
\succeq	2	succeq	\sim	3	widetilde	Θ	0	varTheta
\preceq	2	preceq	$\widehat{}$	3	widehat	Λ	0	varLambda
\supset	2	supset	\smile	3	lmoustache	Ξ	0	varXi
\subset	2	subset	\frown	3	rmoustache	Π	0	varPi
\supseteq	2	supseteq	$($	0	lgroup	Σ	0	varSigma
\subseteq	2	subseqeq	$)$	0	rgroup	Υ	0	varUpsilon
\in	2	in	$ $	2	arrowvert	Φ	0	varPhi
\ni	2	ni	\parallel	2	Arrowvert	Ψ	0	varPsi
\gg	2	gg	\lrcorner	3	bracevert	Ω	0	varOmega
\ll	2	ll	\parallel	2	Vert	\searrow	2	internalAnd
$/$	2	not	$ $	2	vert	\boxdot	C	boxdot
\leftrightarrow	2	leftrightharrow	\uparrow	2	uparrow	\boxplus	C	boxplus
\leftarrow	2	leftarrow	\downarrow	2	downarrow	\boxtimes	C	boxtimes
\rightarrow	2	rightarrow	\updownarrow	2	updownarrow	\square	C	square
\mapsto	2	mapstochar	\Uparrow	2	Uparrow	\square	C	Box
\sim	2	sim	\Downarrow	2	Downarrow	\blacksquare	C	blacksquare
\simeq	2	simeq	\Updownarrow	2	Updownarrow	\cdot	C	centerdot
\perp	2	perp	\backslash	2	backslash	\diamond	C	Diamond
\equiv	2	equiv	\langle	2	langle	\diamond	C	lozenge
\asymp	2	asymp	\rangle	2	rangle	\blacklozenge	C	blacklozenge
\smile	1	smile	$\{$	2	lbrace	\circlearrowright	C	circlearrowright
\frown	1	frown	$\}$	2	rbrace	\circlearrowleft	C	circlearrowleft
\leftharpoonup	1	leftharpoonup	\lceil	2	lceil	\rightleftharpoons	C	rightleftharpoons

\leftrightharpoons	C leftrightarpoons	\curlyeqprec	C curlyeqprec	$\overline{\wedge}$	C doublebarwedge
\boxminus	C boxminus	\curlyeqsucc	C curlyeqsucc	\angle	C angle
\Vdash	C Vdash	\preccurlyeq	C preccurlyeq	\sphericalangle	C measuredangle
\Vvdash	C Vvdash	\leqq	C leqq	\sphericalangle	C sphericalangle
\vDash	C vDash	\leqslant	C leqslant	\varpropto	C varpropto
\twoheadrightarrow	C twoheadrightarrow	\lessgtr	C lessgtr	\smile	C smallsmile
\twoheadleftarrow	C twoheadleftarrow	\backprime	C backprime	\frown	C smallfrown
\leftleftarrows	C leftleftarrows	\cdot	C dabar@	\subseteq	C Subset
\rightrightarrows	C rightrightarrows	$\dot{=}$	C risingdotseq	\supset	C Supset
\Uparrow	C upuparrows	$\dot{=}$	C fallingdotseq	\cup	C Cup
\Downarrow	C downdownarrows	\succcurlyeq	C succcurlyeq	\cup	C doublecup
\upharpoonright	C upharpoonright	\geqq	C geqq	\cap	C Cap
\restriction	C restriction	\geqslant	C geqslant	\cap	C doublecap
\downharpoonright	C downharpoonright	\gtrless	C gtrless	\wedge	C curlywedge
\upharpoonleft	C upharpoonleft	\sqsubset	C sqsubset	\vee	C curlyvee
\downharpoonleft	C downharpoonleft	\sqsupset	C sqsupset	\times	C leftthreetimes
\rightarrowtail	C rightharpoonright	\triangleright	C vartriangleright	\times	C rightthreetimes
\leftarrowtail	C leftarrowtail	\triangleright	C rhd	\subseteq	C subseteqq
\leftrightarrows	C leftrightarrows	\triangleleft	C lhd	\supseteq	C supseteqq
\rightleftarrows	C rightleftarrows	\triangleleft	C vartriangleleft	\bumpeq	C bumpeq
\Lsh	C Lsh	\triangleright	C trianglerighteq	\bumpeq	C Bumpeq
\Rsh	C Rsh	\triangleright	C unrhd	\lll	C lless
\rightsquigarrow	C rightsquigarrow	\triangleleft	C trianglelefteq	\lll	C lll
\leadsto	C leadsto	\triangleleft	C unlhd	\ggg	C gggtr
\leftrightsquigarrow	C leftrightsquigarrow	\star	C bigstar	\ggg	C ggg
\looparrowleft	C looparrowleft	\emptyset	C between	\ulcorner	C ulcorner
\looparrowright	C looparrowright	\blacktriangledown	C blacktriangledown	\urcorner	C urcorner
\circ	C circeq	\blacktriangleright	C blacktriangleright	\textcircled{S}	C circledS
\succsim	C succsim	\blacktriangleleft	C blacktriangleleft	\pitchfork	C pitchfork
\gtrsim	C gtrsim	\triangle	C vartriangle	$\dot{+}$	C dotplus
\gtrapprox	C gtrapprox	\triangle	C triangleup	\sim	C backsim
\multimap	C multimap	\blacktriangle	C blacktriangle	\backsimeq	C backsimeq
\therefore	C therefore	∇	C triangledown	\llcorner	C llcorner
\because	C because	\equiv	C eqcirc	\lrcorner	C lrcorner
\doteqdot	C doteqdot	\lesseqgtr	C lesseqgtr	\complement	C complement
\doteq	C Doteq	\gtreqless	C gtreqless	\intercal	C intercal
\triangleq	C triangleq	\lesseqgtr	C lesseqgtr	\textcircled{c}	C circledcirc
\precsim	C precsim	\gtreqless	C gtreqless	$\textcircled{*}$	C circledast
\lesssim	C lesssim	\Rightarrow	C Rrightarrow	$\textcircled{-}$	C circleddash
\lessapprox	C lessapprox	\Leftarrow	C Lleftarrow	\Dlvertneqq	D lvertneqq
\eqslantless	C eqslantless	\veebar	C veebar	\Dgvertneqq	D gvertneqq
\eqslantgtr	C eqslantgtr	$\bar{\wedge}$	C barwedge	\nleq	D nleq

\nless	D ngeq	\supsetneq	D varsupsetneq	\emptyset	D varnothing
\nless	D nless	\nsubseteq	D nsubseteqq	\nexists	D nexists
\ngtr	D ngtr	\nsupseteq	D nsupseteqq	\dashv	D Finv
\nprec	D nprec	\subsetneq	D subsetneqq	\oslash	D Game
\nsucc	D nsucc	\supseteq	D supsetneqq	\mathcal{O}	D mho
\lneqq	D lneqq	\varsubsetneq	D varsubsetneqq	\eth	D eth
\gneqq	D gneqq	\supsetneq	D varsupsetneqq	\approx	D eqsim
\nleqslant	D nleqslant	\subsetneq	D subsetneq	\beth	D beth
\ngeqslant	D ngeqslant	\supsetneq	D supsetneq	\beth	D gimel
\lneq	D lneq	\nsubseteq	D nsubseteq	\daleth	D daleth
\gneq	D gneq	\nsupseteq	D nsupseteq	\lessdot	D lessdot
\npreceq	D npreceq	\nparallel	D nparallel	\gtrdot	D gtrdot
\nsucceq	D nsucceq	\nmid	D nmid	\ltimes	D ltimes
\precnsim	D precnsim	\nshortmid	D nshortmid	\rtimes	D rtimes
\succnsim	D succnsim	\nshortparallel	D nshortparallel	\mid	D shortmid
\lnsim	D lnsim	\nvDash	D nvDash	\parallel	D shortparallel
\gnsim	D gnsim	\nVDash	D nVDash	\setminus	D smallsetminus
\nleqq	D nleqq	\nDash	D nDash	\sim	D thicksim
\ngeqq	D ngeqq	\nVDash	D nVDash	\approx	D thickapprox
\precneqq	D precneqq	\ntrianglerighteq	D ntrianglerighteq	\approx	D approxeq
\succneqq	D succneqq	\ntrianglelefteq	D ntrianglelefteq	\approx	D succapprox
\precnapprox	D precnapprox	\ntriangleleft	D ntriangleleft	\approx	D precapprox
\succnapprox	D succnapprox	\ntriangleright	D ntriangleright	\curvearrowleft	D curvearrowleft
\lnapprox	D lnapprox	\nleftarrow	D nleftarrow	\curvearrowright	D curvearrowright
\gnapprox	D gnapprox	\rightarrow	D nrightarrow	F	D digamma
\nsim	D nsim	\Leftarrow	D nLeftarrow	\varkappa	D varkappa
\ncong	D ncong	\Rightarrow	D nRightarrow	\Bbbk	D Bbbk
\diagup	D diagup	\Leftrightarrow	D nLeftrightarrow	\hslash	D hslash
\diagdown	D diagdown	\leftrightharpoonup	D nleftrightharpoonup	\hbar	D hbar
\varsubsetneq	D varsubsetneq	\divideontimes	D divideontimes	\eth	D backepsilon

19 MetaPost

In a ConT_EXt document we can use MetaPost code directly. For example:

```
\startMPgraphic
  fill unitsquare scaled 100 withcolor (.2,.3,.4) ;
\stopMPgraphic
```

A direct relation with the ConT_EXt 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 referred 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 ConT_EXt command line interface T_EXexec, 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

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 ConT_EXt files

D.1 Files in tex/context/base

filename(s)	title	subtitle
attr-ini.tex (lua)	Attribute Macros	Initialization
char-act.lua	Character Macros	Active characters via lua
char-cmp.lua	Character Macros	Lua character composition
char-def.tex (lua)	Character Macros	Unicode Support
char-ini.tex (lua)	Character Macros	Character Support (Initialization)
char-map.lua	Character Macros	Case mapping
char-mth.lua	Character Macros	Math named character table
char-syn.lua	Character Macros	Named character synonyms
char-tok.lua	Character Macros	Lua token handlers
char-utf.tex (lua)	Character Macros	Unicode Support (UTF)
colo-ema.tex	Color Macros	Emacs Colors
colo-ext.tex	Color Macros	Extras
colo-hex.tex	Color Macros	Hex Colors
colo-ini.tex	Color Macros	Initialization
colo-new.tex (mkii,mkiv,lua)	Color Macros	Initialization
colo-rgb.tex	Color Macros	RGB
colo-run.tex	Color Macros	Runtime loaded commands
colo-xwi.tex	Color Macros	X Windows
cont-cs.tex	Context	Czech Format Generation
cont-cz.tex	Context	Czech Format Generation
cont-de.tex	Context	German Format Generation
cont-en.tex (lua)	Context	English Format Generation
cont-err.tex	System Files	Just A warning
cont-fil.tex	Miscellaneous Macros	File Synonyms
cont-fr.tex	Context	German Format Generation
cont-gb.tex	Context	English Format Generation
cont-it.tex	Context	Italian Format Generation
cont-log.tex	Miscellaneous Macros	T _E X Logos
cont-mtx.tex	Miscellaneous Macros	Experimental MetaT _E X Macros
cont-new.tex (mkii,mkiv)	Miscellaneous Macros	New Macros
cont-nl.tex	Context	Dutch Format Generation
cont-old.tex	Miscellaneous Macros	Old Macros
cont-ro.tex	Context	Romanian Format Generation
cont-uk.tex	Context	English Format Generation
context.tex	Context	Format Generation
cont-sys.ori	Miscellaneous Macros	System Specific Setups
cont-usr.ori	User Format Specifications	System Specific Setups
context-characters.lmx	Lua-enabled html pages	companion to comm-xml.tex
context-debug.lmx	Lua-enabled html pages	companion to comm-xml.tex
context-error.lmx	Lua-enabled html pages	companion to comm-xml.tex
context.css	Lua-enabled html pages	CSS setups
core-bar.tex	Core Macros	Margin Bars and alike
core-blk.tex	Core Macros	Blockmoves

core-box.tex	Core Macros	Boxes
core-buf.tex (mkii,mkiv,lua)	Core Macros	Buffers
core-con.tex (mkii,mkiv,lua)	Core Macros	Conversion Macros
core-ctx.tex (mkii,mkiv,lua)	Core Macros	Job Control
core-dat.tex	Core Macros	Database Support
core-def.tex	Core Macros	Defaults
core-des.tex	Core Macros	Descriptions
core-fig.tex	Core Macros	Figure Inclusion
core-fil.tex	Core Macros	File Support
core-fld.tex	Core Macros	Fill in fields
core-fnt.tex	Core Macros	Font Support
core-gen.tex	Core Macros	General
core-grd.tex	Core Macros	Grid Snapping (Experimental)
core-hlp.tex	Core Macros	Help (Experimental)
core-inc.tex (mkii,mkiv,lua)	Core Macros	Figure Inclusion
core-ini.tex	Core Macros	Additional Initialization
core-ins.tex	Insertion Macros	Insertions
core-int.tex	Core Macros	Interaction
core-itm.tex	Core Macros	itemgroups
core-job.tex (mkii,mkiv,lua)	Core Macros	Job Handling
core-lme.tex	Core Macros	LAst Minute Extensions
core-lnt.tex	Core Macros	Line Notes
core-lst.tex	Core Macros	Lists
core-ltb.tex	Core Macros	Line Tables
core-mak.tex	Core Macros	General Makeup Commands
core-mar.tex	Core Macros	Markings
core-mat.tex	Core Macros	Math Fundamentals
core-mis.tex	Core Macros	Miscellaneous
core-nav.tex	Core Macros	Navigation
core-new.tex	Core Macros	New ones
core-not.tex	Core Macros	Note Handling
core-ntb.tex	Core Macros	Natural Tables
core-num.tex	Core Macros	Numbering
core-obj.tex (mkii,mkiv,lua)	Core Macros	Object Handling
core-par.tex	Core Macros	Paragraph Tricks
core-pgr.tex	Core Macros	Positioning Support
core-pos.tex (mkii,mkiv,lua)	Core Macros	Positioning Support
core-ref.tex	Core Macros	Cross Referencing
core-reg.tex (mkii,mkiv,lua)	Core Macros	Register Management
core-rul.tex (mkii,mkiv,lua)	Core Macros	Ruled Stuff Handling
core-sec.tex	Core Macros	Sectioning
core-snc.tex	Core Macros	Synchronization Support
core-spa.tex (mkii,mkiv,lua)	Core Macros	Spacing
core-stg.tex	Core Macros	Strategies
core-swd.tex	Core Macros	Section Worlds
core-syn.tex (mkii,mkiv,lua)	Core Macros	Synonyms and Sorts
core-sys.tex (mkii,mkiv)	Core Macros	System
core-tab.tex	Core Macros	T <small>A</small> B <small>L</small> E Embedding
core-tbl.tex	Core Macros	Text Flow Tabulation
core-trf.tex	Core Macros	Transformations
core-tsp.tex	Core Macros	Splitting Tables

core-two.tex	(mkii,mkiv,lua)	Core Macros	Two Pass Data
core-uti.tex	(mkii,mkiv,lua)	Core Macros	Utility File Handling
core-var.tex		Core Macros	Variables
core-ver.tex	(mkii,mkiv)	Core Macros	Verbatim
core-vis.tex		Core Macros	Visualization
enco-032.tex		Encoding Macros	Unicode Goodies
enco-037.tex		Unicode Macros	Encoding for vector 37
enco-acc.tex		Encoding Macros	Composed Characters Commands
enco-agr.tex		Unicode Macros	Ancient Greek
enco-ans.tex		Encoding Macros	YandY texnansi Encoding
enco-cas.tex		Encoding Macros	Named Glyph Case Mapping
enco-chi.tex		Encoding Macros	Traditional and Simplified Chinese
enco-com.tex		Encoding Macros	Composed Characters Commands
enco-cyr.tex		Encoding Macros	Cyrillic
enco-def.tex		Encoding Macros	Default Character Definitions
enco-ec.tex		Encoding Macros	LaTeX EC Encoding
enco-ecm.tex		Encoding Macros	Glyphs that may not be present in EC
enco-el.tex		Encoding Macros	EuroLetter
enco-fde.tex		Encoding Macros	German Input Filter
enco-ffr.tex		Encoding Macros	French Input Filter
enco-fpl.tex		Encoding Macros	Polish Input Filter
enco-fro.tex		Encoding Macros	Romanian Input Filter
enco-fsl.tex		Encoding Macros	Slovenian Specialities
enco-grk.tex		Encoding Macros	Greek
enco-heb.tex		Encoding Macros	Hebrew
enco-ibm.tex		Encoding Macros	Hebrew
enco-ibm.tex		Backward Compatibility	IBM (DOS) Regime
enco-il2.tex		Encoding Macros	Czech and Slovak ISO Latin 2 Encoding
enco-ini.tex	(mkii,mkiv)	Encoding Macros	Initialization
enco-lat.tex		Backward Compatibility	Latin2 Regime
enco-mis.tex		Encoding Macros	Missing Glyphs
enco-pdf.tex		Encoding Macros	YandY texnansi Encoding
enco-pfr.tex	(mkii,mkiv)	Encoding Macros	PDF Font Resource Inclusion
enco-pol.tex		Encoding Macros	Polish Mixed Encoding
enco-qx.tex		Encoding Macros	Polish QX Encoding
enco-raw.tex		Encoding Macros	plain ASCII Encoding
enco-run.tex		Encoding Macros	Runtime Macros
enco-t5.tex		Encoding Macros	New Vietnamese Encoding
enco-tbo.tex		Encoding Macros	TeXBaseOne Encoding
enco-uc.tex		Encoding Macros	Unicode (backwards mapping)
enco-utf.tex		Encoding Macros	UTF-* Encoding
enco-vis.tex		Backward Compatibility	Vietnamese Regime
enco-vna.tex		Encoding Macros	Vietnamese Accents
enco-win.tex		Backward Compatibility	Windows Regime
enco-x5.tex		Encoding Macros	Vietnamese Encoding
filt-bas.tex		Filter Macros	A Base Collection
filt-ini.tex		Filter Macros	Initialization
font-afm.lua		MKIV font code	Handling AFM files
font-arb.tex		Backward Compatibility	Old arabtex loader
font-bfm.tex		Font Macros	Mixed Normal and Bold Math
font-chi.tex		Font Macros	Chinese

<code>font-col.tex</code>	(lua)	Font Macros	Fallbacks (collections)
<code>font-def.lua</code>		MKIV font code	Parsing font definitions
<code>font-enc.lua</code>		MKIV font code	Encoding remapper
<code>font-ext.lua</code>		MKIV font code	Font expansion and protrusion
<code>font-fbk.lua</code>		MKIV font code	Font fallbacks (experimental)
<code>font-heb.tex</code>		Backward Compatibility	Old arabtex hebrew loader
<code>font-ini.tex</code>	(mkii,mkiv,lua)	Font Macros	Initialization
<code>font-jap.tex</code>		Font Macros	Japanese
<code>font-map.lua</code>		MKIV font code	Mapfile loader
<code>font-old.lua</code>		MKIV font code	old obsolete stuff
<code>font-otf.lua</code>		MKIV font code	Handling OTF and TTF files
<code>font-run.tex</code>		Font Macros	Runtime Macros
<code>font-set.tex</code>	(lua)	Lua Macros	Font Loading Support
<code>font-syn.lua</code>		MKIV font code	Font filename aliases
<code>font-tfm.lua</code>		MKIV font code	Handling TFM files
<code>font-uni.tex</code>		Font Macros	Unicode Initialization
<code>font-unk.tex</code>		Font Macros	Unknown Defaults
<code>font-vf.lua</code>		MKIV font code	Handling VF files
<code>hand-def.tex</code>		Handling Macros	Default Protruding Factors
<code>hand-ini.tex</code>	(mkii,mkiv)	Handling Macros	Initialization
<code>java-ans.tex</code>		JavaScript Macros	Answer Analization
<code>java-exa.tex</code>		JavaScript Macros	Example Support
<code>java-fil.tex</code>		JavaScript Macros	Filing and Printing
<code>java-fld.tex</code>		JavaScript Macros	Field Support
<code>java-ini.tex</code>		JavaScript Macros	Initialization
<code>java-stp.tex</code>		JavaScript Macros	Stepping
<code>l-aux.lua</code>		Lua libraries	Low-level auxiliary routines
<code>l-boolean.lua</code>		Lua libraries	Handling of lua booleans
<code>l-dimen.lua</code>		Lua libraries	Calculations with T _E X dimensions
<code>l-dir.lua</code>		Lua libraries	Operations on directories
<code>l-file.lua</code>		Lua libraries	Operations on files and filenames
<code>l-io.lua</code>		Lua libraries	File IO routines
<code>l-lpeg.lua</code>		Lua libraries	Low-level lpeg helpers
<code>l-math.lua</code>		Lua libraries	A few extra math routines
<code>l-md5.lua</code>		Lua libraries	Using MD5 checksums
<code>l-number.lua</code>		Lua libraries	Number manipulations
<code>l-os.lua</code>		Lua libraries	Operating System interface
<code>l-set.lua</code>		Lua libraries	Set operations
<code>l-string.lua</code>		Lua libraries	String manipulations
<code>l-table.lua</code>		Lua libraries	Table manipulations
<code>l-unicode.lua</code>		Lua libraries	Unicode helper routines
<code>l-url.lua</code>		Lua libraries	URL handling
<code>l-utils.lua</code>		Lua libraries	Utility file handling
<code>l-xml.lua</code>		Lua libraries	XML parser
<code>l-xml-edu.lua</code>		Lua libraries	old XML parser (obsolete)
<code>l-xmlctx.lua</code>		Lua libraries	CTX helper routines
<code>lang-all.xml</code>		XML databases	Language data
<code>lang-alt.tex</code>		Language Macros	Altaic Languages
<code>lang-ana.tex</code>		Language Macros	Anatolian Languages
<code>lang-ara.tex</code>		Language Macros	Arabic Languages
<code>lang-art.tex</code>		Language Macros	Artificial Languages

lang-bal.tex	Language Macros	Baltic Languages
lang-cel.tex	Language Macros	Celtic Languages
lang-chi.tex	Language Macros	Chinese
lang-ctx.tex	Language Macros	Generic Patterns
lang-cyr.tex	Language Macros	Cyrillic Languages
lang-dis.tex	Language Macros	Distribution Patterns
lang-frd.tex	Language Macros	Language Frequency Table Data
lang-frq.tex	Language Macros	Language Frequency Table Support
lang-ger.tex	Language Macros	Germanic Languages
lang-grk.tex	Language Macros	Uralic Languages
lang-ind.tex	Language Macros	Indo Iranian Languages
lang-ini.tex (mkii,mkiv,lua)	Language Macros	Initialization
lang-ita.tex	Language Macros	Italic Languages
lang-jap.tex	Language Macros	Japanese
lang-lab.tex	Language Macros	Language Head and Label Texts
lang-mis.tex	Language Macros	Language Options
lang-run.tex	Language Macros	Runtime Macros
lang-sla.tex (mkii,mkiv)	Language Macros	Slavic Languages
lang-spa.tex	Language Macros	Spacing
lang-spe.tex	Language Macros	Specifics
lang-sla.tex (mkii,mkiv)	Language Macros	Uralic Languages
lang-tst.lua	Language Macros	Experimental lua tests
lang-url.tex (mkii,mkiv,lua)	Language Macros	Language Options
lang-vn.tex	Language Macros	Vietnamese
luat-cbk.lua	Lua MkIV core	Callbacks
luat-crl.lua	Lua MkIV core	Curl interface
luat-deb.tex (lua)	Communication Macros	Initialization
luat-env.tex (lua)	Lua Macros	ConT _E Xt features
luat-exe.lua	Lua MkIV core	Executing external commands
luat-ini.tex (lua)	Lua Macros	Initialization
luat-inp.lua	Lua MkIV core	Finding files
luat-iop.lua	Lua MkIV core	I/O processing
luat-lib.tex (lua)	Lua Macros	Unicode Support
luat-kps.lua	Lua MkIV core	Kpathsea emulation
luat-lmx.tex (lua)	Lua Macros	LMX Support
luat-log.tex (lua)	Lua Macros	LOGO Support
luat-lua.lua	Lua MkIV core	Low-level lua routines
luat-rmp.lua	Lua MkIV core	XML remapping
luat-sta.lua	Lua MkIV core	System states
luat-tex.lua	Lua MkIV core	Process logging and statistics
luat-tmp.lua	Lua MkIV core	Caching information
luat-tra.lua	Lua MkIV core	Tracing routines
luat-tre.lua	Lua MkIV core	XML Tree routines
luat-uni.tex (lua)	Lua Macros	Unicode Support
luat-zip.lua	Lua MkIV core	Zip file interface
lxml-ini.tex (lua)	Lua based xml Support	Initialization
m-arabtex.tex	Modules	Arabic
m-chart.tex	Modules	Flow Charts
m-chemic.tex	Extra Modules	ppchT _E X (Plain Pictex Context cHemie T _E X)
m-cweb.tex	Extra Modules	cweb Pretty Printing Macros
m-database.tex	Modules	Database Thingies

m-dratex.tex	Extra Modules	DRATEX Loading Macros
m-editsnc.tex	Modules	Editor Synchronization
m-educat.tex	Extra Modules	Educational Extras
m-gamma.tex	Extra Modules	Basic Omega Support
m-gnuplot.tex	Extra Modules	gnuplot Inclusion
m-graph.tex	Extra Modules	MetaPost graph module support
m-layout.tex	Modules	Additional Layouts
m-level.tex	Extra Modules	Catching Nesting Errors
m-narrowtt.tex	Modules	Narrow Verbatim
m-newmat.tex	Math Module	AMS-like math extensions
m-pdfsnct.tex	Modules	Editor Synchronization
m-pictex.tex	Extra Modules	P _T CT _E X Loading Macros
m-plus.tex	Extra Modules	Loading extra features
m-pstricks.tex	Extra Modules	pstricks Connections
m-invull.tex	Extra Modules	Exercise
m-r.tex	Modules	R Support
m-steps.tex	Modules	Step Charts and Tables
m-streams.tex	Modules	Streams
m-subsub.tex	Private Modules	More Section Levels
m-tex4ht.tex	Private Modules	Preliminary tex4ht support
m-timing.tex	Modules	Timing
m-tryout.tex	Extra Modules	Tryout Features
m-units.tex	Extra Modules	Scientific Units
m-visual.tex	Extra Modules	Visualization and Faking
math-ams.tex	Math Macros	AMS Specials
math-cow.tex	Math Macros	Cow Math
math-def.lua	Math Macros	MKIV definitions
math-ent.lua	Math Macros	MKIV Math entities
math-eul.tex	Math Macros	Virtual Euler Specials
math-ext.tex	Math Macros	Extra Macros
math-fou.tex	Math Macros	Fourier Specials
math-ini.tex (mkii,mkiv,lua)	Math Macros	Basic Macros
math-lbr.tex	Math Macros	Lucida Specials
math-mis.tex	Math Macros	Miscellaneous Symbols
math-pln.tex	System Macros	Efficient Plain T _E X loading
math-run.tex	Math Macros	Runtime Macros
math-tex.tex		Plain Specials
math-tim.tex	Math Macros	Mathtime Specials
math-uni.tex	Math Macros	unicode support
meta-clp.tex	MetaPost Graphics	Clipping
meta-dum.tex	MetaPost Graphics	Dummy (External) Graphics
meta-fig.tex (mkii,mkiv)	MetaPost Graphics	Stand Alone Graphics
meta-ini.tex (mkii,mkiv)	MetaPost Graphics	Initialization
meta-mis.tex	MetaPost Graphics	Misc Test Graphics
meta-nav.tex	MetaPost Graphics	Navigational Graphics
meta-pag.tex	MetaPost Graphics	Initialization
meta-pdf.tex (mkii,mkiv,lua)	Support Macros	MetaPost to pdf conversion
meta-pre.tex	MetaPost Graphics	Predefined Goodies
meta-tex.tex (mkii,mkiv)	Support Macros	MetaPost fast text insertion
meta-txt.tex	MetaPost Graphics	Text Tricks
meta-xml.tex	MetaPost Graphics	XML Hacks

mllib-ctx.tex (lua)	MetaPost Integrated Graphics	Basics
mllib-pdf.tex (lua)	MetaPost Integrated Graphics	Conversion to PDF
mllib-pps.tex (lua)	MetaPost Integrated Graphics	Basics
mllib-run.lua	MetaPost Integrated Graphics	job control
mult-com.tex	Multilingual Macros	Commands
mult-con.tex	Multilingual Macros	Constants
mult-fst.tex	Multilingual Macros	Speed Up
mult-ini.tex (mkii,mkiv,lua)	Multilingual Macros	Initialization
mult-sys.tex	Multilingual Macros	System
node-ini.tex (lua)	Character Macros	Node Support (Initialization)
page-app.tex	Core Macros	Independent page building
page-bck.tex	Page Macros	Backgrounds
page-flt.tex	OTR Macros	Floating Bodies
page-flw.tex	OTR Macros	Text Flows
page-imp.tex	Core Macros	Pagebody Building (Imposition)
page-ini.tex	Page Macros	Initializations
page-lay.tex	Page Macros	Layout Specification
page-lin.tex (mkii,mkiv,lua)	Core Macros	Line Numbering
page-log.tex	Page Macros	Logos
page-lyr.tex	Page Macros	Layers
page-mak.tex	Page Macros	Simple MakeUp
page-mar.tex	Core Macros	Marginal Things
page-mul.tex	OTR Macros	Multi Column Output
page-new.tex	Page Macros	Page New
page-nnt.tex	Page Macros	Footnotes
page-num.tex	Core Macros	Numbering
page-one.tex	OTR Macros	Default Routine
page-par.tex	Core Macros	Line Numbering
page-pls.tex	Core Macros	Page Setup
page-run.tex	Page Macros	Runtime Macros
page-set.tex	OTR Macros	Column Sets
page-sid.tex	OTR Macros	Side Floats
page-spr.tex	Page Macros	Spreading
page-str.tex	Core Macros	Page Streams
page-txt.tex	Page Macros	Texts
pdf-def.tex	PDF Font Resources	Character definitions (autogenerated)
pdf-ec.tex	PDF Font Resources	EC encoding
ppchtex (m-chemie).tex	Extra Modules	ppchT _E X (Plain Pictex Context cHemie T _E X)
prop-ini.tex	Property Macros	Initialization
prop-lay.tex	Property Macros	Layers
prop-mis.tex (mkii,mkiv)	Property Macros	Miscellaneous
prop-run.tex	Property Macros	Runtime Macros
regi-8859-1.tex (lua)	Regime Macros	iso-8859-1 (West European)
regi-8859-10.tex (lua)	Regime Macros	iso-8859-10 (Nordic)
regi-8859-13.tex (lua)	Regime Macros	iso-8859-13 (Baltic)
regi-8859-15.tex (lua)	Regime Macros	iso-8859-15 (West European)
regi-8859-16.tex (lua)	Regime Macros	iso-8859-16 (Romanian)
regi-8859-2.tex (lua)	Regime Macros	iso-8859-2 (East European)
regi-8859-3.tex (lua)	Regime Macros	iso-8859-3 (South European)
regi-8859-4.tex (lua)	Regime Macros	iso-8859-4 (North European)
regi-8859-5.tex (lua)	Regime Macros	iso-8859-5 (Cyrillic)

regi-8859-7.tex (lua)	Regime Macros	iso-8859-7 (Greek)
regi-8859-11.lua	Regime Macros	iso-8859-11
regi-8859-14.lua	Regime Macros	iso-8859-14
regi-8859-6.lua	Regime Macros	iso-8859-6
regi-8859-8.lua	Regime Macros	iso-8859-8
regi-cp1255.lua	Regime Macros	code page 1255
regi-cp1256.lua	Regime Macros	code page 1256
regi-cp1258.lua	Regime Macros	code page 1258
regi-8859-9.tex (lua)	Regime Macros	iso-8859-9 (Turkish)
regi-cp1250.tex (lua)	Regime Macros	cp1250 (East European)
regi-cp1251.tex (lua)	Regime Macros	cp1251 (Cyrillic)
regi-cp1252.tex (lua)	Regime Macros	cp1252 (West European)
regi-cp1253.tex (lua)	Regime Macros	cp1253 (Greek)
regi-cp1254.tex (lua)	Regime Macros	cp1254 (Turkish)
regi-cp1257.tex (lua)	Regime Macros	cp1257 (Windows Baltic)
regi-cyp.tex	Regime Macros	Cyrillic Plus
regi-cyr.tex	Regime Macros	Cyrillic
regi-def.tex	Regime Macros	Default Character Definitions
regi-ibm.tex	Regime Macros	The Good Old MSDOS IBM codepage
regi-ini.tex (mkii,mkiv,lua)	Regime Macros	Initialization
regi-mac.tex	Regime Macros	Mac Encoding
regi-run.mkii	Regime Macros	Runtime Macros
regi-syn.tex	Regime Macros	Synonyms
regi-uni.tex	Regime Macros	Unicode
regi-utf.tex	Regime Macros	UTF-8
regi-vis.tex	Regime Macros	viscii
rlxcache.rlx	RLX files	Cache
rlxtools.rlx	RLX files	Tools
s-abr-01.tex	Style File	General Abbreviations 1
s-abr-02.tex	Style File	General Abbreviations 2
s-abr-03.tex	Style File	General Abbreviations 3
s-cdr-01.tex	Style File	CDROM Cover
s-chi-00.tex	Style File	Basic Chinese Style
s-faq-00.tex	Style File	FAQ Common Macros
s-faq-01.tex	Style File	FAQ Interactive Version
s-faq-02.tex	Style File	FAQ Paper Version
s-faq-03.tex	Style File	FAQ General Framework
s-fnt-01.tex	Style File	Font Environment 1
s-fnt-02.tex	Style File	Font Environment 2
s-fnt-10.tex	Style File	Font Environment definitions
s-grk-00.tex	Style File	CB Greek Support
s-jap-00.tex	Style File	Basic Japanese Style
s-mag-01.tex	Style File	Magazine Base Style
s-map-se.tex	Style File	Maps basis stijl
s-mod-00.tex	Style File	Documentation Base Environment
s-mod-01.tex	Style File	Documentation Paper Environment
s-mod-02.tex	Style File	Documentation Screen Environment
s-pre-00.tex	Style File	Presentation Environment 0
s-pre-01.tex	Style File	Presentation Environment 1
s-pre-02.tex	Style File	Presentation Environment 2
s-pre-03.tex	Style File	Presentation Environment 3

s-pre-04.tex	Style File	Presentation Environment 4
s-pre-05.tex	Style File	Presentation Environment 5
s-pre-06.tex	Style File	Presentation Environment 6
s-pre-07.tex	Style File	Presentation Environment 7
s-pre-08.tex	Style File	Presentation Environment 8
s-pre-09.tex	Style File	Presentation Environment 9
s-pre-10.tex	Style File	Presentation Environment 10
s-pre-13.tex	Style File	Presentation Environment 13
s-pre-14.tex	Style File	Presentation Environment 14
s-pre-15.tex	Style File	Presentation Environment 15
s-pre-16.tex	Style File	Presentation Environment 16
s-pre-19.tex	Style File	Presentation Environment 19
s-pre-22.tex	Style File	Presentation Environment 22
s-pre-20.tex	Style File	Presentation Environment 20
s-pre-30.tex	Style File	Presentation Environment 30
s-pre-50.tex	Style File	Presentation Environment 50
s-pre-60.tex	Style File	Presentation Environment 60
s-pre-61.tex	Style File	Presentation Environment 61
s-pre-62.tex	Style File	Presentation Environment 62
s-pre-63.tex	Style File	Presentation Environment 63
s-pre-64.tex	Style File	Presentation Environment 64
s-ptj-01.tex	Style File	PracTeX Journal Style
s-syn-01.tex	Style File	Preliminary Syntax Stuff
s-sys-01.tex	Style File	Generate List of Math Symbol
sort-def.tex (mkii,mkiv)	Sort Macros	Defaults
sort-ini.tex (mkii,mkiv,lua)	Sort Macros	Initialization
sort-lan.tex (mkii,mkiv,lua)	Sort Macros	Language Definitions
spec-def.tex (mkii,mkiv)	Special Macros	Definitions
spec-dpm.tex	Special Macros	DVIPDFM support
spec-dpx.tex	Special Macros	DVIPDFMx support
spec-dvi.tex	Special Macros	Generic T _E X Solutions
spec-fdf.tex (mkii,mkiv)	pdf Macros	Support Macros
spec-ini.tex	Special Macros	Initialization
spec-mis.tex	Special Macros	Miscellaneous Macros
spec-pdf.tex (lua)	Special Macros	Adobe Acrobat version 2.1
spec-ps.tex	Special Macros	Adobe PostScript
spec-tpd.tex (mkii,mkiv)	Special Macros	pdfT _E X
spec-tr.tex	Special Macros	Thomas Rokicki's DVIPS
spec-tst.tex	PDF Macros	Special Test Macro
spec-var.tex	Special Macros	Variables
spec-win.tex	Special Macros	YandY's DIWINDO
spec-xtx.tex	Special Macros	X _Y T _E X support
spec-xtx.tex	Special Macros	X _Y T _E X support
spec-yy.tex	Special Macros	YandY's DVIPSONE and DVIWINDO
supp-ali.tex	Support Macros	Alignment
supp-box.tex	Support Macros	Boxes
supp-dir.tex	Support Macros	Directional Things
supp-emp.tex	Support Macros	emT _E X specials to pdf conversion
supp-eps.tex	Support Macros	eps tools
supp-fil.tex (mkii,mkiv,lua)	Support Macros	Files
supp-fun.tex	Support Macros	Fun Stuff

supp-ini.tex	Support system Macros	Initializations
supp-lan.tex	Support Macros	Language Options
supp-mat.tex	Support Macros	Math
supp-mis.tex	Support Macros	Missing (For Generic Use)
supp-mpe.tex	Support Macros	METAPOST Special Extensions
supp-mps.tex	Support Macros	MetaPost Inclusion
supp-mrk.tex	Support Macros	Marks
supp-num.tex	Support Macros	Number (Digit) Handling
supp-pat.tex	Language Macros	Loading (Generic) Patterns
supp-pdf.tex	Support Macros	MetaPost to pdf conversion
supp-ran.tex	Support Macros	Random Number Generation
supp-spe.tex	Support Macros	Specials
supp-tpi.tex	Support Macros	tpic Conversion
supp-vis.tex	Support Macros	Visualization
symb-cow.tex	Symbol Libraries	Cow Symbols
symb-eur.tex	Symbol Libraries	Adobe Euro Symbols
symb-glm.tex	Symbol Libraries	Guillemots
symb-ini.tex	Symbol Libraries	Basic Symbols Commands
symb-jmn.tex	Symbol Libraries	Special Navigational Symbols
symb-mis.tex	Symbol Libraries	Miscellaneous
symb-mvs.tex	Symbol Libraries	Martin Vogels Symbole
symb-nav.tex	Symbol Libraries	Navigational Symbols
symb-run.tex	Symbol Libraries	Runtime Macros
symb-uni.tex	Symbol Libraries	Unicode Symbols
symb-was.tex	Symbol Libraries	Roland Waldi's Symbols (wasy-2)
syst-cat.tex (mkii,mkiv)	System Macros	Catcode Handling
syst-chr.tex	System Macros	Character Related Things
syst-con.tex (mkii,mkiv,lua)	System Macros	Conversions
syst-etx.tex	System Macros	Efficient Plain T _E X loading
syst-ext.tex	System Macros	Extras
syst-fnt.tex	System Macros	Font Things
syst-gen.tex	System Macros	General
syst-mtx.tex	System Macros	MetaT _E X specifics
syst-new.tex	Support Macros	New Ones
syst-omg.tex	System Macros	A couple of Omega goodies
syst-pdt.tex	System Macros	pdfT _E X specifics
syst-pln.tex	System Macros	Efficient Plain T _E X loading
syst-prm.tex	System Macros	Primitive Behavior
syst-rtp.tex (mkii,mkiv)	Core Macros	Run Time Processes
syst-str.tex (mkii,mkiv)	System Macros	String Processing
syst-tex.tex	System Macros	Efficient Plain T _E X loading
syst-var.tex	System Macros	Variables
syst-xtx.tex	System Macros	X _Y T _E X specifics
thrd-pic.tex	Third party macros	PicTeX
thrd-ran.tex	Third party macros	Randomizer
thrd-tab.tex	Third party macros	TaBLE
thrd-trg.tex	Third party macros	Trigonometry
todo-mkii.tex	TODO list	MKII
todo-mkiv.tex	TODO list	MKIV
toks-ini.tex (lua)	Character Macros	Token Support (Initialization)
type-akb.tex	Typescript Macros	Adobe's Famous Gang of Fonts

type-buy.tex	Typescript Macros	A Few Commercial Fonts
type-cbg.tex	Typescript Macros	CB Greek
type-cow.tex	Typescript Macros	Cow Fonts
type-def.tex	Typescript Macros	Default Definitions
type-dis.tex	Typescript Macros	Distribution scripts
type-enc.tex	Typescript Macros	Encoding scripts
type-exa.tex	Typescript Macros	Example scripts
type-exp.tex	Typescript Macros	Experimental Definitions
page-new.tex	Page Macros	Fontsite 500
type-ghz.tex	Typescript Macros	Hermann Zapf's Fonts
type-gyr.tex	Typescript Macros	TeXGyre Collection
type-hgz.tex	Typescript Macros	Hermann Zapf's Fonts
type-ini.tex	Typescript Macros	Initialization
type-map.tex	Typescript Macros	Mapfile scripts
type-msw.tex	Typescript Macros	Microsoft Windows Fonts
type-old.tex	Typescript Macros	Ghosts from the Past
type-omg.tex	Typescript Macros	Basic Omega Gamma Definitions
type-one.tex	Typescript Macros	Type One Definitions
type-otf.tex	Typescript Macros	Opentype Definitions
type-pre.tex	Typescript Macros	Compatibility scripts
type-run.tex	Typescript Macros	Runtime Macros
type-siz.tex	Typescript Macros	Sizing scripts
type-spe.tex	Typescript Macros	Special scripts
type-syn.tex	Typescript Macros	Filename scripts
type-tmf.tex	Typescript Macros	Core T _E X Fonts
type-xtx.tex	Typescript Macros	X _Y T _E X's font treasures
typo-ini.tex	Typographic Macros	Initialization
unic-000.tex	Unicode Macros	Vector 0
unic-001.tex	Unicode Macros	Vector 1
unic-002.tex	Unicode Macros	Vector 2
unic-003.tex	Unicode Macros	Vector 3
unic-004.tex	Unicode Macros	Vector 4
unic-005.tex	Unicode Macros	Vector 5
unic-030.tex	Unicode Macros	Vector 30
unic-031.tex	Unicode Macros	Vector 31
unic-032.tex	Unicode Macros	Vector 32
unic-033.tex	Unicode Macros	Vector 33
unic-034.tex	Unicode Macros	Vector 34
unic-037.tex	Unicode Macros	Vector 37
unic-039.tex	Unicode Macros	Vector 39
unic-251.tex	Unicode Macros	Vector 251
unic-cjk.tex	Unicode Macros	CJK Vectors
unic-exp.tex	Unicode Support	Unicode vector expansion
unic-ini.tex (mkii,mkiv)	Unicode Support	Unicode and UTF-8 support
unic-run.tex	Unicode Support	Goodies
verb-c.tex	Verbatim Macros	Pretty C Verbatim
verb-eif.tex	Verbatim Macros	Pretty Eiffel Verbatim
verb-ini.tex	Verbatim Macros	Initialization
verb-js.tex	Verbatim Macros	Pretty JavaScript Verbatim
verb-jv.tex	Verbatim Macros	Pretty Java Verbatim
verb-lua.lua	Verbatim Macros	Pretty Lua verbatim

verb-mp.tex (lua)	Verbatim Macros	Pretty MetaPost Verbatim
verb-pas.tex	Verbatim Macros	Pretty Pascal and Modula Verbatim
verb-pl.tex	Verbatim Macros	Pretty Perl Verbatim
verb-raw.tex	Verbatim Macros	RAW
verb-sql.tex	Verbatim Macros	Pretty sql Verbatim
verb-tex.tex (lua)	Verbatim Macros	Pretty T _E X verbatim
verb-xml.tex	Verbatim Macros	Pretty XML verbatim
x-calcmath.mkiv	Modules	Calculator Math
x-cals.mkiv	XML Modules	Cals table renderer
x-chemml.tex (mkii,mkiv)	XML Modules	Loading CHEMML Filters
x-chemml.xsd	XML Schemas	ChemML (XSD)
x-contml.tex	XML Support	Basic Context commands
x-contml.xsd	XML Schemas	ML (XSD)
x-corres.tex	XML Modules	Handling Correspondence Base
x-corres.rng	XML Schemas	Coresspondence (Relax NG)
x-ct.xml	XML Modules	Structures
x-dir-01.tex	Directory Handling	Overview (1)
x-fdf-00.tex	XML Modules	FDF support
x-fe.tex	foXet	Simple Extensions
x-fig-00.tex	Style File	Figure Base Loading
x-fig-00.dtd	XML Schemas	Figure databases (DTD)
x-fig-00.xsd	XML Schemas	Figure databases (XSD)
x-fig-01.tex	Style File	Figure Base Generation
x-fig-02.tex	Style File	Figure Base Inclusion (I)
x-fig-03.tex	Style File	Figure Base Inclusion (II)
x-fo.tex	foXet	Formatting Objects
x-foxet.tex	foXet	Main foXet loader
x-ldx.tex	XML Modules	LDX support
x-ldx.ctx	CTX process files	Lua Documentation Generator
x-mathml.tex (mkii,mkiv,lua)	XML Modules	Loading MathML Filters
x-mathml.xsd	XML Schemas	MathML (XSD)
x-newmml.tex (mkii,mkiv)	XML Modules	ChemML renderer
x-newmme.tex	XML Modules	MathML Entities
x-newmml.tex (mkii,mkiv)	XML Modules	MathML Renderer
x-newmmo.tex	XML Modules	MathML Renderer/Open Math Extensions
x-newpml.tex	XML Support	Units
x-om2cml.xsl	XSL Transformations	OpenMath to MathML
x-openmath.xsl	XSL Transformations	OpenMath
x-physml.tex	XML Modules	Loading PHYXML Filters
x-physml.xsd	XML Schemas	PhysML (XSD)
x-res-00.tex	Style File	Resource Libraries
x-fig-01.tex	Style File	Figure Base Generation
x-res-02.tex	Style File	Figure Base Inclusion (I)
x-res-03.tex	Style File	Figure Base Inclusion (II)
x-res-04.tex	Style File	Figure Base Loading
x-res-08.tex	Style File	Resource Reporting
x-res-09.tex	Style File	Resource Reporting (2)
x-res-10.tex	Style File	Resource Dummy Generation
x-res-11.tex	Style File	Resource Reporting (3)
x-res-12.tex	Style File	Resource Checking
x-res-20.tex	Style File	Figure Lists

x-res-50.tex	Style File	Multimedia Presentation
x-sch-00.tex	Style File	XML Schema Basics
x-sch-01.tex	Style File	XML Schema Presentation
x-set-01.tex	Setup Mappings	Macro Definitions
x-set-02.tex	Setup Mappings	Macro Definitions
x-set-11.tex (mkii,mkiv)	Setup Definitions	Macro Definitions
x-set-12.tex	Setup Definitions	Macro Definitions
x-sm2om.xsl	XSL Transformations	SimpleMath to OpenMath
x-xml-01.tex	XML Style File	Formatting X?? files
x-xml-02.tex	XML Style File	Pretty Printing
x-xml-11.tex	XML Style File	Formatting X?? files
x-xml-11.tex	XML macros	ChemML
xtag-ent.tex	XML Support	A bunch of Entities
xtag-exp.tex	XML Support	Expansion Related Things
xtag-ext.tex	XML Support	Extra Macros
xtag-hyp.tex	XML Support	hyphenation support
xtag-ini.tex (mkii,mkiv)	XML Support	Initialization
xtag-map.tex	XML Support	Remapping
xtag-mea.tex	XML Support	ISOAMSA entities
xtag-meb.tex	XML Support	ISOAMSB entities
xtag-mec.tex	XML Support	ISOAMSC entities
xtag-meh.tex	XML Support	ISOTECH entities
xtag-men.tex	XML Support	ISOAMSN entities
xtag-meo.tex	XML Support	ISOAMSO entities
xtag-mer.tex	XML Support	ISOAMSR entities
xtag-mmc.tex	XML Support	Math ML
xtag-mml.tex	XML Support	Math ML
xtag-mmp.tex	XML Support	Math ML
xtag-mxa.tex	XML Support	MathML ISOAMSA Entity aliases
xtag-mxb.tex	XML Support	MathML ISOAMSB Entity aliases
xtag-mxc.tex	XML Support	MathML ISOAMSC Entity aliases
xtag-mxh.tex	XML Support	MathML ISOTECH Entity aliases
xtag-mxn.tex	XML Support	MathML ISOAMSN Entity aliases
xtag-mxo.tex	XML Support	MathML ISOAMSO Entity aliases
xtag-mxr.tex	XML Support	MathML ISOAMSR Entity aliases
xtag-pml.tex	XML Support	Physics ML
xtag-pmu.tex	XML Support	Units
xtag-pre.tex	XML Support	Predefined Things
xtag-prs.tex	XML Support	Parsing
xtag-raw.tex	XML Support	reducing specials
xtag-rng.tex	XML Support	Relax NG
xtag-run.tex	XML Support	Visualization
xtag-exp.tex	XML Support	Stacking Data
xtag-utf.tex	XML Support	UTF-8 support
xtag-xsd.tex	XML Support	Schemas
xtag-xsl.tex	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 launching 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 Con \TeX t 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 Con \TeX t and \TeX Live 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 ConT_EXt scripts it's best not to use the suffix since this permits us to change the scripting language. ConT_EXt 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=psstopdf --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=psstopdf --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 ConT_EXt 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 preferred way and most simple way to generate the stubs for the scripts that come with ConT_EXt 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

If 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 <code>kpsewhich</code> 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 its location
--execute	use Ruby's 'exec' instead of 'system'
--batch	<i>not yet implemented</i>

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 <code>texmfstart</code> 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 T_EX 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 T_EX trees. This may take a while, especially when you have a complete tree, like the one on T_EX Live.²⁷

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
```

²⁷ On my computer I use multiple trees parallel to the latest T_EX Live tree. This results in a not that intuitively and predictable search process. The cover of this manual reflects state of those trees.

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.xml somefile.xml
```

or shorter:

```
texmfstart bin:xsltproc kpse:somescript.xml somefile.xml
```

In both cases `somescript.xml` 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
```

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 ConT_EXt 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%
MPPPOOL        = %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 value when 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

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly

within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

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 License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in

quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before re-distributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title,

year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the

combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” 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.

