A LaTeX Package for Typesetting Crossword Puzzles and More*

Gerd Neugebauer
Brechklinge 10
69256 Mauer (Germany)
Net: gene@gerd-neugebauer.de

Documentation date: 2025/01/23

Abstract

cwpuzzle.dtx provides a package to typeset crossword puzzles. The leading philosophy is that the puzzle and the solution are typeset from the same source.

The package can be used to produce several types of puzzles like the classical crossword puzzle, a number puzzle, and fill-in puzzles. In addition to the block separated puzzles the thick line delimited puzzles are supported as well.

Contents

*This file documents cwpuzzle.dtx version 1.13 as of 2025/01/23.

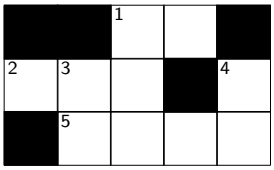
1 About Crossword Puzzles

Crossword puzzles are can be a an amusing but also a challenging hobby. Unfortunately at the time of this writing I am not aware of any good package to typeset crossword puzzles with LaTeX. Thus I decided to make one which at least fits my needs.

There are several types of crossword puzzles among. This package can only be used to typeset several of them. The basic assumption in this package is that puzzles are rectangular arrangements of boxes. Some of these boxes are black and others are prepared to take single letters. Each word in the grid is enclosed in black boxes or the outside.

Optionally there may be rectangular regions left blank inside the puzzle. They can be used to place ads or other informative texts inside the puzzle.

1.1 Classical Crossword Puzzles

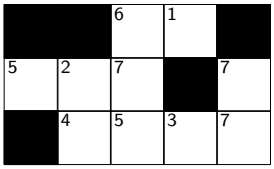


Across 1 unit of measure 2 ∗ 5 sectioning unit

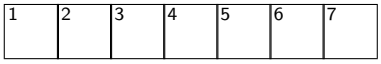
Down 1 η 3 unit of measure 4 non-proportional font

The "classical" type of a crossword puzzle words are marked with numbers and each word is accompanied with a clue which should help (or confuse) the reader. Those clues are listed after the frame of the puzzle.

1.2 Number Crossword Puzzles

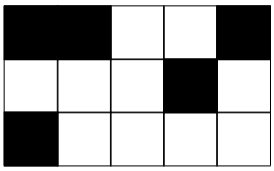


The following letters are used: AEPRSTX



The "number puzzle" variant contains only numbers instead of letters. Different numbers denote different letters. There are no clues. The reader is assumed to find a complete list of letters by filling appropriate words into the grid. Sometimes a word is already entered into the grid to ease the start.

1.3 Fill-In Crossword Puzzles

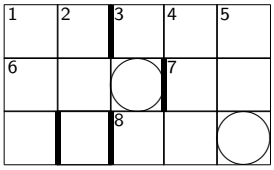


Words of length 2: EX SP TT
Words of length 3: AST ETA
Words of length 4: PART

The "fill-in puzzle" variant consists of a frame containing only black and white boxes. Additionally a list of words is given which have to be put into the frame until none is left and the frame is completed.

1.4 Line delimited Crossword Puzzles

The crossword puzzles we have seen before had the property that words are either delimited by the outer border or by a solid block. In addition line delimited puzzles are common. In this case a thicker line is drawn to indicate the end of a line.

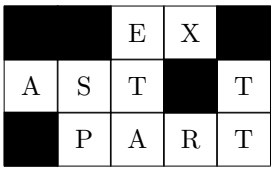


Across 1 unit of measure 3 math function 6 list of tables 7 μ 8 key

Down 1 class option 2 math function 3 η 4 all in angle brackets 5 LaTeX place picture element

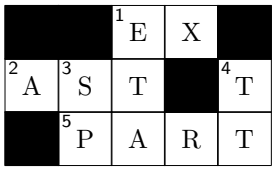
In the example above we can see another feature. This feature is that two letters are circled. This can be used to indicate letters for a solution word of the crossword puzzle.

1.5 Solutions

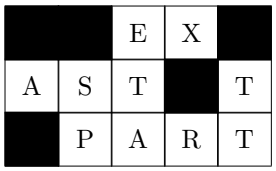


Often it is not only desirable to typeset the unsolved crossword puzzle but also the solution. This means that all the letters have to be filled in. This should be possible with the same source as the questions to avoid typos or redundancies leading to additional work.

Several variants of solutions come to mind. Primarily the solution should show the letters and suppress any clues. One major distinction is also whether or not the numbers of the words should be shown in the solution as well.



Finally there are the lists of letters in numbered puzzles. In the solution they will show the letters in them as well.



The following letters are used: AEPRSTX

X	S	R	P	A	E	T

2 Input of Crossword Puzzles

The basic idea behind this package is that a crossword puzzle is specified in a separate file. The actual appearance of the puzzle is controlled by several options. Thus it should be possible to produce the unsolved and the solved puzzle from the same source. Before we describe the various options we will have a look at the basic environments and macros used to specified a crossword puzzle.

`Puzzle` This package provides the environment `Puzzle` which typesets the frame of a crossword puzzle. This environment takes two arguments. These arguments are the number of columns and the number of the rows of the puzzle. This means that essentially only rectangular puzzles can be typeset.

The example from section 1.1 has been entered as follows:

```
\begin{Puzzle}{5}{3}%
  |*   |*   |[1]E|X  |*   |.
  |[2]A|[3]S|T   |*  |[4]T|.
  |*   |[5]P|A   |R  |T   |.
\end{Puzzle}
```

In this example we can see that inside the `Puzzle` environment there is one special character. This is the bar `|`. This bar is an active character in TEX. Thus you can think of it like a macro.

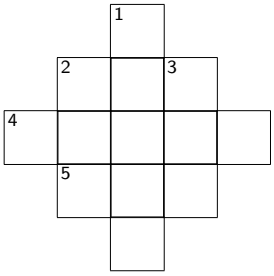
The `|` macro takes three arguments. The first two arguments are optional, i.e. enclosed in brackets if present. The first optional argument denotes the number for numbered boxes. The second optional argument specifies the formatting of the cell.

The third argument is either empty `{}` or it consists of a single character. This argument describes the action to be performed.

- If this argument is a letter then it is simply shown in the solution and suppressed in the unsolved crossword puzzle.

- If this argument is an asterisk `*` then a black box is produced.
- If this argument is a dot `.` then this marks the end of the current row. The next box is typeset at the beginning of the following row.
- If this argument is empty `{}` then a white box is typeset. This box does not contain a letter, nor does it have a frame. This macro can be used to leave room for larger boxed with ads. Alternatively this can be used to disable certain boxes to make a non-rectangular crossword puzzle.

```
\begin{Puzzle}{5}{5}
  |{}  |{}  |[1]S|.
  |{}  |[2]M|I   |[3]D|.
  |[4]T|I   |M   |E   |S   |.
  |{}  |[5]N|E   |G   |.
  |{}  |{}  |Q   |.
\end{Puzzle}
```



Across: 2 | 4 × 5 ¬ **Down:** 1 ≃ 2 log-like function 3 log-like function

Note that white-space is ignored after the arguments but not between the bar and the arguments.

The formatting of the cell is controlled by the second optional argument of the `|` macro. This optional argument may contain a list of several characters. Each of these characters is interpreted from left to right. The following list describes the meaning of the built-in characters.

`|[8][f]X` The letter f produces a simple frame around the cell. This is the default if nothing is specified.

8

`|[8][.]X` The character . produces no additional rendering it can be used to overwrite the default rendering which is to place a frame around the cell.

`|[8][*]X` The character * produces a black box. This is the same effect which can be achieved by providing the character * to be filled into the cell for the solution.

`|[8][O]X` The letter O produces an oval as drawn with the LaTeX macro `\oval`.

`|[8][o]X` The letter o produces an oval inside a frame. T is is an abbreviation for the two letters fO.

`|[8][t]X`

The letter t produces a frame with a thicker line at the top. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][T]X`

The letter T produces a thicker line at the top of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][b]X`

The letter b produces a frame with a thicker line at the bottom. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][B]X`

The letter B produces a thicker line at the bottom of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][l]X`

The letter l produces a frame with a thicker line at the left side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][L]X`

The letter L produces a thicker line at the left side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][r]X`

The letter r produces a frame with a thicker line at the right side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][R]X`

The letter R produces a thicker line at the right side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

`|[8][/]X`

The character / produces a line crossing the cell from lower left to upper right.

`|[8][,]X`

The character , produces a line crossing the cell from upper left to lower right.

`|[8][S]X`

The character S produces the solution, i.e. the content of the cell is typeset. No decorations are placed around. For this purpose it should be combined with some other formatting characters.

Whenever you try to use an undefined specification for the cell frame a warning is printed and the letter is ignored.

`\PuzzleDefineCell` You can define additional cell renderings of your own. For this purpose the macro `\PuzzleDefineCell` is provided. It takes two arguments. The first argument contains the key under which the rendering should be addressed in the optional second argument of the macro `|`. The second argument contains the replacement text like in `\newcommand`. This replacement text can make use of two arguments. They are addressed with `#1` and `#2`. The first one contains the x coordinate of the cell to be rendered. The second one its y coordinate.

The following example shows for instance the definition of a new cell type addressed by the key + which draws a thick frame around the cell.

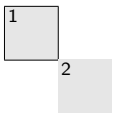
```
\PuzzleDefineCell{+}{
  \PuzzleThicklines
```

```
  \put(#1,#2){\framebox(1,1){}}
}
```

Finally we show how to define a cell type consisting of a colored box. The box itself is drawn with the help of the macro `\colorbox`. Thus it is necessary to load the package `color` which defines this macro.

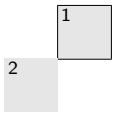
The two invocations show the combination with the f specifier. The specifier f is the default and used only if the user does not provide the optional argument. Thus we need to provide the f we we want to have it additionally.

```
\definecolor{gray}{gray}{.9}
\PuzzleDefineCell{c}{{%
  \put(#1,#2){\makebox(1,1){%
    \fboxsep=0pt
    \colorbox{gray}{\makebox(1,1){}}}}
}}
\begin{Puzzle}{2}{2}
  |[1][cf]X | {}     |.
  |{}       |[2][c]X |.
\end{Puzzle}
```



`\DefineColorCell` The macro `\DefineColorCell` simplifies this task by encapsulating the definition above. It has two arguments. The first one is the key character and the second one is th name of the color to use for the background.

```
\definecolor{gray}{gray}{.9}
\PuzzleDefineColorCell{c}{gray}
\begin{Puzzle}{2}{2}
  |{}      |[1][cf]X  |.
  |[2][c]X |{}        |.
\end{Puzzle}
```

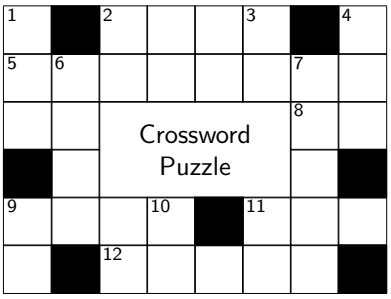


`\Frame` The macro `\Frame` can be used to typeset ads or other text into larger boxes inside the frame of the crossword puzzle. For this purpose five arguments are required. The first two arguments are used to specify the lower left corner of the frame. The lower left corner has the coordinates 0,0 and the numbers increase upwards and to the right.

The third argument is the width of the frame and the fourth argument is the height of the frame measured in number of boxes. Finally, the fifth argument contains the text to be typeset. Per default it is typeset in a mini-page of the appropriate width centered horizontally and vertically.

```
\begin{Puzzle}{8}{6}
  \Frame{2}{2}{4}{2}{\sffamily Crossword\\Puzzle}
  |[1]E|*   |[2]N |U    |L   |[3]L |*   |[4]V|.
  |[5]T|[6]R|I    |A    |N   |G    |[7]L|E   |.
  |A   |U   |{}   |{}   |{}  |{}   |[8]C|C   |.
  |*   |L   |{}   |{}   |{}  |{}   |E   |*   |.
  |[9]B|E   |T    |[10]A|*   |[11]L|I   |M   |.
  |F   |*   |[12]L|A    |B   |E    |L   |*   |.
```

```
\end{Puzzle}
```



Across: 2 empty 5 △ 8 carbon copy (letter.sty) 9 β 11 limes 12 mark it

Down: 1 η 2 $\ni$ 3 logarithm 4 $\vec{}$ 6 black rectangle 7 $\lceil$ 9 bold face 10 Å 11 $\leq$

`PuzzleClues` The clues in the classical crossword puzzle are typeset with the use of the environment `PuzzleClues`. This environment takes one argument which is typeset before the clues. The environment takes roughly the half of the text width and make a mini-page with this width. Thus two invocations of this environment are typeset side by side.

Alternatively if the solution is typeset then the environment `PuzzleClues` has no effect.

```
\begin{PuzzleClues}{\textbf{Across}}%
  \Clue{1}{EX}{unit of measure}%
  \Clue{2}{AST}{\(\ast\)}%
  \Clue{5}{PART}{sectioning unit}%
\end{PuzzleClues}%
\begin{PuzzleClues}{\textbf{Down}}%
  \Clue{1}{ETA}{\(\eta\)}%
  \Clue{3}{SP}{unit of measure}%
  \Clue{4}{TT}{nonproportional font}%
\end{PuzzleClues}%
```

`Clue` The environment `PuzzleClues` defines one local macro. This macro is named `\Clue` and takes three arguments. The first argument is the number of the word. This should correspond to the number in the puzzle frame. The second argument is the word itself. Currently this not used at all. Finally the third argument is the clue for the word.

If the unsolved puzzle is typeset then the first and the third argument are used. Otherwise all arguments are silently absorbed.

`\PuzzleLetters` The macro `\PuzzleLetters` can be used to typeset the list of used letters in numbered crossword puzzles. It has one argument which are the used letters (preferably in alphabetical order.

`\PuzzleNumbers` The macro `\PuzzleNumbers` can be used to generate a numbered list of boxes for the numbered crossword puzzles. The user is supposed to collect the found letters here.

`PuzzleWords` The environment `PuzzleWords` can be sued to typeset the list of words for a fill-in puzzle. It takes one argument. This is the length of the words listed. For

each length there should be an invocation of this environment. The words in this environment are supposed to be ordered alphabetically.

`\Word` The macro `\Word` is defined inside the environment `PuzzleWords`. It takes one argument which is the word itself.

```
\begin{PuzzleWords}{2}
  \Word{EX}%
  \Word{SP}%
  \Word{TT}%
\end{PuzzleWords}%
\begin{PuzzleWords}{3}
  \Word{AST}%
  \Word{ETA}%
\end{PuzzleWords}%
\begin{PuzzleWords}{4}
  \Word{PART}%
\end{PuzzleWords}%
```

3 Other Grid-based Puzzles

In addition to the crossword puzzles other puzzles based on a grid can also be typeset with this package. The basic principle is the same. Just some minor simplifications have been provided.

3.1 Sudoku

A Sudoku is a puzzle on a 9×9 grid. It is filled with nine numbers. Each number occurs only once in each row, each column and each of the nine 3×3 boxes. Initially some of the numbers are shown. The goal is to fill in all missing digits.

	2						9	
3		1	9		6	5		2
			8		4			
	9						5	
5			2		3			6
	7						2	
			4		7			
8		2	5		1	7		3
	5						8	

`Sudoku` The input for a Sudoku is given in a specialized environment. Since the size is fixed there is no need to specify a size. We separate the cells with a pipe symbol and mark the end of a line with a dot. To mark those cells contianing the hints we preceed the number with an asterisk.

The Sudoku shown above is typeset from the following source:

```
\begin{Sudoku}
  | 7|*2| 4| 1| 3| 5| 6|*9| 8|.
  |*3| 8|*1|*9| 7|*6|*5| 4|*2|.
  | 9| 6| 5|*8| 2|*4| 1| 3| 7|.
  | 2|*9| 6| 7| 1| 8| 3|*5| 4|.
  |*5| 1| 8|*2| 4|*3| 9| 7|*6|.
  | 4|*7| 3| 6| 5| 9| 8|*2| 1|.
  | 6| 3| 9|*4| 8|*7| 2| 1| 5|.
  |*8| 4|*2|*5| 9|*1|*7| 6|*3|.
  | 1|*5| 7| 3| 6| 2| 4|*8| 9|.
\end{Sudoku}
```

As for crossword puzzles the macro `\PuzzleSolution` can be used to switch to solution mode. Then all numbers are shown. The parameters to modify the appearance of a puzzle work here as well.

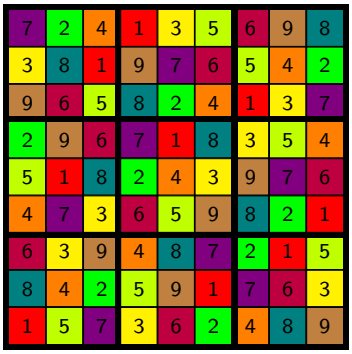
7	2	4	1	3	5	6	9	8
3	8	1	9	7	6	5	4	2
9	6	5	8	2	4	1	3	7
2	9	6	7	1	8	3	5	4
5	1	8	2	4	3	9	7	6
4	7	3	6	5	9	8	2	1
6	3	9	4	8	7	2	1	5
8	4	2	5	9	1	7	6	3
1	5	7	3	6	2	4	8	9

Since the characters in the solution are numbers only we can use the following definition to colorize the cells in the solution.

```
\def\PuzzleSolutionContent#1{\fboxsep=0pt
\def\myColor{}%
\ifcase#1
\def\myColor{white}%0
\or\def\myColor{red}%1
\or\def\myColor{green}%2
\or\def\myColor{yellow}%3
\or\def\myColor{orange}%4
\or\def\myColor{lime}%5
\or\def\myColor{purple}%6
\or\def\myColor{violet}%7
```

```
\or\def\myColor{teal}%8
\or\def\myColor{brown}%9
\else\def\myColor{white}%*
\fi
\colorbox{\myColor}{\makebox(1,1){#1}}}
```

With the help of the package xcolor for the predefined colors we get the following output:

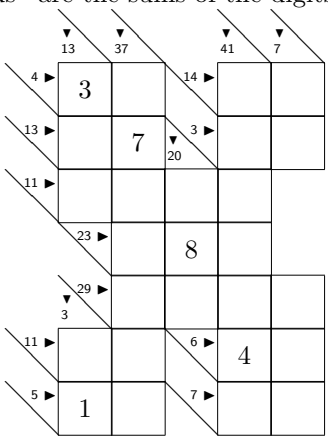


Note that this definition has to be restricted to a local block if you have solutions for other puzzle types in the same document.

3.2 Kakuro

In a Kakuro the cells are filled with digits. Each "word" consists of different digits; i.e. a number can not be repeated in a consecutive horizontal or vertical sequence of numbers.

The hints for the "words" are the sums of the digits.



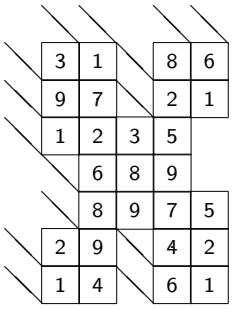
`Kakuro` The input for a Kakuro is given in a specialized environment. It takes the width and the height as arguments. We separate the cells with a pipe symbol and mark the end of a line with a dot. To mark those cells contianing the initial digits we preceed the digit with an asterisk.

The hints are enteres in angle brackets. They contain the horizontal and vertical sums separated by a colon. The sums can be empty if none should be typeset.

The Kakuro shown above is typeset from the following source:

```
\begin{Kakuro}{6}{9}
  |  -   |<:13> |<:37> |  -   |<:41> |<:7>  |  -   |.
  |<4:>  |* 3   |  1   |<14:> |  8   |  6   |  -   |.
  |<13:> |  9   |* 7   |<3:20>|  2   |  1   |  -   |.
  |<11:> |  1   |  2   |  3   |  5   |  -   |  -   |.
  |  -   |<23:> |  6   |* 8   |  9   |  -   |  -   |.
  |  -   |<29:3>|  8   |  9   |  7   |  5   |  -   |.
  |<11:> |  2   |  9   |<6:>  |* 4   |  2   |  -   |.
  |<5:>  |  1   |  4   |<7:>  |  6   |  1   |  -   |.
  |  -   |  -   |  -   |  -   |  -   |  -   |  -   |.
\end{Kakuro}
```

As for crossword puzzles the macro `\PuzzleSolution` can be used to switch to solution mode. Then all numbers are shown.




```
\PuzzleSolution
\PuzzleUnitlength=14pt
\sffamily\footnotesize
\begin{Kakuro}{6}{9}
  |  -  |<:13> |<:37>|  -   |<:41>|<:7> | - |.
  |<4:> |* 3   |  1  |<14:> |  8  |  6  | - |.
  |<13:>|  9   |* 7  |<3:20>|  2  |  1  | - |.
  |<11:>|  1   |  2  |  3   |  5  |  -  | - |.
  |  -  |<23:> |  6  |* 8   |  9  |  -  | - |.
  |  -  |<29:3>|  8  |  9   |  7  |  5  | - |.
  |<11:>|  2   |  9  |<6:>  |* 4  |  2  | - |.
  |<5:> |  1   |  4  |<7:>  |  6  |  1  | - |.
  |  -  |  -   |  -  |  -   |  -  |  -  | - |.
\end{Kakuro}
```

4 Parameters and Options

The package cwpuzzle can be controlled by a rich set of macros. In addition some settings can be performed with style options. The following style options are recognized:

numbered The solution numbering is turned on.

nocenter The puzzle is not typeset in a centered paragraph of its own.

unboxed The clues are not enclosed in mini-pages and centered on the page.

normalsize The puzzle is set in normalsize. This is the default.

small The puzzle is set in small. The size of the cell and the font size of the solution are adjusted accordingly.

large The puzzle is set in large. The size of the cell and the font size of the solution are adjusted accordingly.

german

ngerman The build in texts are switched to german variants. The defaults are English. This options is also in effect when given to the document class. The style inherits it from there.

The style options can be passed to the style in the usual way:

```
\usepackage[nocenter,unboxed,small]{cwpuzzle}
```

The fine tuning can be achieved with the help of several macros. Those macros are described below.

`\PuzzleUnitlength` The length `\PuzzleUnitlength` determines the width and height of each single box in the frame of a crossword puzzle. The default value is `20pt`.

`\PuzzleBlackBox` The macro `\PuzzleBlackBox` contains the commands to produce the black boxes. It has to produce at most of width and height of `\PuzzleUnitlength`. Per default it just produces a black rectangle of this size.

The following list shows some variants which can be achieved by redefining the macro `\PuzzleBlackBox`.

```
\renewcommand{\PuzzleBlackBox}{\rule{.75\PuzzleUnitlength}%
                                    {.75\PuzzleUnitlength}}
```

```
\renewcommand{\PuzzleBlackBox}{\framebox(.75,.75){%
                                  \framebox(.5,.5){}}}
```

Additional effects can be achieved by using shades of gray (with the `graphics` package).

`\PuzzleFont` The macro `\PuzzleFont` contains the font changing command issued before the frame of the crossword puzzle.

`\PuzzleNumberFont` The macro `\PuzzleNumberFont` contains the font changing command issued before a number in the frame of the crossword puzzle is typeset.

`\PuzzleClueFont` The macro `\PuzzleClueFont` contains the font changing command issued before the clues are typeset.

`\PuzzleWordsText` The macro `\PuzzleWordsText` contains the text which is typeset at the beginning of the environment `PuzzleWords`. It has one argument which contains the length of the words listed.

`\PuzzleLettersText` The macro `\PuzzleLettersText` contains the text which is typeset at the beginning of the macro `\PuzzleLetters`.

`\PuzzleSolution` The macro `\PuzzleSolution` arranges everything that the following puzzles

are typeset in the “solution” mode, i.e. the letters are shown and the clues are suppressed.

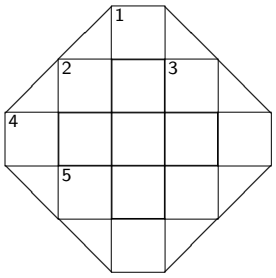
This macros has one optional argument which has to be `true` or `false`. This argument determines whether or not the numbers should also be shown in the solution. The default is `false` which means that the numbers are suppressed in the solution.

`\PuzzleUnsolved` The macro `\PuzzleUnsolved` arranges everything that the following puzzles are typeset in the “unsolved” mode, i.e. the letters are suppressed and the clues are shown.

`\PuzzlePutNumber` The macro `\PuzzlePutNumber` is a configuration macro which typesets the number in a cell. The first argument is the x coordinate. The second argument is the y coordinate. The third argument is the number to be typeset. The coordinates are integer numbers. The coordinate (0,0) is the lower left corner.

`\PuzzleHook` The macro `\PuzzleHook` is called at the end of the `Puzzle` environment. It can be used to place additional graphical elements in the puzzle frame.

The following example shows a crossword puzzle which we have seen before and the definition for the `\PuzzleHook`.




```
\newcommand\PuzzleHook{
  \put(0,2){\line(1,-1){2}}
  \put(0,3){\line(1,1){2}}
  \put(5,2){\line(-1,-1){2}}
  \put(5,3){\line(-1,1){2}}
}
```

`\PuzzleLineThickness` The macro `\PuzzleLineThickness` contains the width of the line used to frame the cells.

`\PuzzlePre` This macro contains the code to be inserted before a puzzle is typeset. It is initialized to begin a new paragraph and center the puzzle.

`\PuzzlePost` This macro contains the code to be inserted after a puzzle is typeset. It is initialized to end the paragraph and center the puzzle.

`\PuzzleCluePre` This macro contains the code to be inserted before the clues are typeset in normal mode. It is initialized to end the paragraph and center the puzzle.

`\PuzzleCluePost` This macro contains the code to be inserted after the clues are typeset in normal mode. It is initialized to end the paragraph and center the puzzle.

`\PuzzleContent` This macro contains the content of a cell during formatting this cell. This enables the cell formating macro to access it.

`\PuzzleSolutionContent` This macro processes the letter in solution mode. It takes one argument, the letter. This macro can be redefinied to achieve special effects for the solution.

`\SudokuLinethickness` This macro contains the thickness of the thick lines in a sudoku.

`\KakuroNumberFont` This macro contains the definition of the font switching macros used when typesetting a Kakuro hint.

`\KakuroHintType` This macro contains the cell type used when typesetting a Kakuro hint. It can be used to redefine the appearance.

5 Further Plans

5.1 General

Maybe I will add a mode for further variants of crossword puzzles sometimes.

Maybe I can add support for further languages if someone provides the appropriate texts. Contributions are welcome.

5.2 The Related Program

There is a related program written in Perl/Tk. This program can be used to manually construct crossword puzzles and save them in a format suitable for this package. Other features include the creation of a proper frame and filling with words.

Right now I have not prepared a distribution of this program yet since this program requires dictionaries which I can not distribute legally.

The crossword examples in this documentation have been computed with the help of the `cwp` program.

6 The Implementation

The crossword puzzle is basically implemented with the LaTeX picture environment. This gives us enough flexibility and provides an high enough abstraction such that we do not have to fiddle around with to many low level details.

The natural unit in a crossword puzzle is a box which is empty or black. Thus the `unitlength` is set to the width (and height) of such a box.

6.1 Basic Definitions and Parameters

First we identify this package.

```
\ProvidesPackage{cwpuzzle}[\filedate gene]
```

Next we load the package amssymb beended for the triangles used in Kakuros

```
\RequirePackage{amssymb}
```

The dimen register `\PuzzleUnitlength` stores the height and width of a box of the puzzle. The default is `20pt` which is also shown in this documentation.

```
\newdimen\PuzzleUnitlength
\PuzzleUnitlength=20pt
```

`\PuzzleClueFont` The macro `\PuzzleClueFont` contains font changing commands issued before the clues are typeset.

```
\newcommand\PuzzleClueFont{\footnotesize}
```

`\PuzzleFont` The macro `\PuzzleFont` contains font changing commands issued before the puzzle is typeset.

```
\newcommand\PuzzleFont{\rmfamily\normalsize}
```

`\PuzzleNumberFont` The macro `\PuzzleNumberFont` contains font changing commands issued before the numbers in a puzzle are typeset.

```
\newcommand\PuzzleNumberFont{\sffamily\scriptsize}
```

`\PuzzleHook` Puzzles are typeset with the LaTeX picture environment. At the end of this environment the macro `\PuzzleHook` is called. The package produces an empty default. Users may want to use this place to typeset additional elements on top of the puzzle.

The puzzle uses a `\unitlength` of `\PuzzleUnitlength`. Thus it is rather easy to address the boxes in the puzzle.

```
\let\PuzzleHook=\relax
```

6.2 The Frame of the Crossword Puzzle

To describe the coordinates where the next box should be typeset we need two counters for the coordinates. These counters are now allocated (even though we could use temporary counters from LaTeX).

```
\newcount\Puzzle@X
\newcount\Puzzle@Y
```

```
\begingroup
\catcode`\|=13
\gdef\Puzzle@@solution{
  \let|=\Puzzle@Box@@solution
  \let\Frame=\Puzzle@Frame@@solution
}
\gdef\Puzzle@@normal{
  \let|=\Puzzle@Box@@normal
  \let\Frame=\Puzzle@Frame@@normal
}
\endgroup
```

`Puzzle` The environment `Puzzle` typesets the frame of a crossword puzzle. It is implemented utilizing a `picture` environment. The unitlength is set to the `\PuzzleUnitlength`. Thus the navigation is fairly easy. The basic unit is width and height of a single box.

The macros which are local to the environment are activated. Thus we avoid collisions with other packages where the same macro names might be used.

Finally the counter which contain the x and the y coordinate have to be initialized.

The last action in the `picture` environment is the expansion of the macro `\PuzzleHook`. This can be used to include additional material in the `picture` environment. Primarily I have use this to include the ads. But now there is the macro `\Frame` for this purpose.

```
\newenvironment{Puzzle}[2]{\PuzzlePre
  \catcode`\|=13
  \@nameuse{Puzzle@@\Puzzle@TYPE}%
  \unitlength=\PuzzleUnitlength
  \linethickness{\PuzzleLineThickness}%
  \Puzzle@Y=#2
  \begin{picture}(#1,#2)
    \Puzzle@Box@@normal.
}{%
    \PuzzleHook
  \end{picture}\PuzzlePost
}
```

`\PuzzleLineThickness` The macro `\PuzzleLineThickness` contains the width of the line used to frame the cells.

```
\newcommand\PuzzleLineThickness{.25pt}
```

`\PuzzlePre` This macro contains the code to be inserted before a puzzle is typeset. It is initialized to begin a new paragraph and center the puzzle.

```
\newcommand\PuzzlePre{%
    \par\noindent\mbox{}\hfill
}
```

\PuzzlePost This macro contains the code to be inserted after a puzzle is typeset. It is initialized to end the paragraph and center the puzzle.

```
\newcommand\PuzzlePost{%
    \hfill\null\par\noindent
}
```

\Puzzle@Frame@@normal The macro Puzzle@Frame is used to place additional rectangular regions into the puzzle frame. This frame can contain arbitrary text which is typeset in a centered environment.

This macro takes five arguments. The first two arguments are the coordinates of the upper left corner of the frame. The coordinates are logical coordinates starting from the lower left corner of the puzzle. The next two arguments are the width and the height of the frame given in the number of boxes covered. Finally the fifth argument contains the text which should appear in this frame.

```
\newcommand\Puzzle@Frame@@normal[5]{\put(#1,#2){\framebox(#3,#4){%
      \begin{minipage}{#3\unitlength}\begin{center} #5
      \end{center}\end{minipage}}}}
```

\Puzzle@Frame@@solution For the solution the framed ads are simply ignored.

```
\newcommand\Puzzle@Frame@@solution[5]{}
```

\PuzzleBlackBox The macro \PuzzleBlackBox is called to typeset the black boxes. It should produce a box of at most width and height of \PuzzleUnitlength.

```
\newcommand\PuzzleBlackBox{\rule{\PuzzleUnitlength}{\PuzzleUnitlength}}
```

\Puzzle@Box@@normal The macro \Puzzle@Box@@normal performs all tasks when a box should be typeset in "normal" mode. The arguments are evaluated and the appropriate type of box typeset or other actions performed.

```
\newcommand\Puzzle@Box@@normal[1][]{%
  \def\Puzzle@tmp@{#1}%
  \Puzzle@Box@@normal@
}
```

\Puzzle@Box@@normal@ The macro \Puzzle@Box@@normal@ performs all tasks when a box should be typeset in "normal" mode. The arguments are evaluated and the appropriate type of box typeset or other actions performed.

```
\newcommand\Puzzle@Box@@normal@[2][f]{%
  \def\PuzzleContent{#2}%
  \def\Puzzle@tmp{#2}%
  \if\Puzzle@tmp.
    \Puzzle@X=0
    \advance\Puzzle@Y-1
  \else
    \ifx\Puzzle@tmp\@empty
    \else
      \if\Puzzle@tmp*
        \Puzzle@Cell@Loop *#1{}%
      \else
```

```
        \Puzzle@Cell@Loop #1{}%
      \fi
    \fi
    \ifx\@empty\Puzzle@tmp@\else
      \PuzzlePutNumber{\Puzzle@X}{\Puzzle@Y}{\Puzzle@tmp@}%
    \fi
    \advance\Puzzle@X 1
  \fi
}
```

`\PuzzlePutNumber` The macro `\PuzzlePutNumber` is a configuration macro which typesets the number in a cell. The first argument is the x coordinate. The second argument is the y coordinate. The third argument is the number to be typeset.

```
\def\PuzzlePutNumber#1#2#3{%
  \put(#1,#2){\makebox(1,.95)[tl]{\PuzzleNumberFont\,#3}}%
}
```

`\Puzzle@Cell@Loop` The macro `\Puzzle@Cell@Loop` processes its arguments until an empty argument is found. For each argument it is tried to invoke the corresponding cell drawing macro.

```
\def\Puzzle@Cell@Loop#1{%
  \def\Puzzle@tmp{#1}%
  \ifx\Puzzle@tmp\@empty
    \let\Puzzle@tmp\relax
  \else
    \expandafter\ifx\csname Puzzle@Cell@@#1\endcsname\relax
      \typeout{cwpuzzle: Cell type #1 is undefined. I am ignoring it}%
    \else
      \csname Puzzle@Cell@@#1\endcsname{\Puzzle@X}{\Puzzle@Y}%
    \fi
    \let\Puzzle@tmp\Puzzle@Cell@Loop
  \fi
  \Puzzle@tmp
}
```

`\Puzzle@Box@@solution` The macro `\Puzzle@Box@@solution` performs all tasks when a box should be typeset in "solution" mode. The arguments are evaluated and the appropriate type of box typeset or other actions performed.

```
\newcommand\Puzzle@Box@@solution[1][]{%
  \def\Puzzle@tmp@{#1}%
  \Puzzle@Box@@solution@
}
```

`\Puzzle@Box@@solution@` The macro `\Puzzle@Box@@solution@` performs all tasks when a box should be typeset in "solution" mode. The arguments are evaluated and the appropriate type of box typeset or other actions performed.

```
\newcommand\Puzzle@Box@@solution@[2][f]{%
  \def\Puzzle@tmp{#2}%
  \if\Puzzle@tmp.
```

```
    \Puzzle@X=0
    \advance\Puzzle@Y-1
  \else
    \ifx\Puzzle@tmp\@empty
    \else
      \if\Puzzle@tmp*
        \Puzzle@Cell@Loop *#1{}%
      \else
        \Puzzle@Cell@Loop #1{}%
        \put(\Puzzle@X,\Puzzle@Y){\PuzzleSolutionContent{#2}}%
      \fi
    \fi
    \def\Puzzle@tmp{#1}%
    \ifx\Puzzle@tmp\@empty\else
      \ifPuzzle@SolutionNumbered
        \PuzzlePutNumber{\Puzzle@X}{\Puzzle@Y}{\Puzzle@tmp@}%
      \fi
    \fi
    \advance\Puzzle@X 1
  \fi
}
```

`\PuzzleSolutionContent` The macro `\PuzzleSolutionContent` procees the character in solution mode. Per default it converts the argument to upper caseand sets it in a box of size 1×1.

```
\def\PuzzleSolutionContent#1{\makebox(1,1){\uppercase{#1}}}
```

6.3 Predefined Cell Types

In this section a series of frame types are defined.

`\PuzzleDefineCell` The macro `\PuzzleDefineCell` is a user command to define a new cell type. The first argument contains the key under which the cell type should be addressed. This key should be expandable and should result into a single letter. Special effects can be achieved with keys constituted of non letters or several characters.

The second argument contains the code to be stored for the key given.

```
\newcommand\PuzzleDefineCell[2]{
  \global\@namedef{Puzzle@Cell@@#1}##1##2{#2}%
}
```

`\PuzzleDefineColorCell`

```
\newcommand\PuzzleDefineColorCell[2]{
  \global\@namedef{Puzzle@Cell@@#1}##1##2{%
    \fboxsep=0pt
    \put(##1,##2){\makebox(1,1){\colorbox{#2}{\makebox(1,1){}}}}
   }%
}
```

`\PuzzleThickine` The parameter `\PuzzleThickine` contains the expansion text to be inserted. whenever a thick line is required. This means that this macro arranges everything

that a thick line is drawn. Usually it contains an invocation to `\linethickness`. It can be redefined to achieve other effects like even thicker lines or colored lines. Note that the macro is used inside a group in the predefined cell types.

```
\def\PuzzleThickline{\linethickness{2pt}}
```

`\Puzzle@Cell@@T` The letter T produces a thicker line at the top of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{T}{{%
  \advance#2 1
  \PuzzleThickline
  \put(#1,#2){{\line(1,0){1}}}
}}
```

`\Puzzle@Cell@@t` The letter t produces a frame with a thicker line at the top. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{t}{{%
  \put(#1,#2){\framebox(1,1){}}
  \advance#2 1
  \PuzzleThickline
  \put(#1,#2){{\line(1,0){1}}}
}}
```

`\Puzzle@Cell@@B` The letter B produces a thicker line at the bottom of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{B}{{%
  \PuzzleThickline
  \put(#1,#2){{\line(1,0){1}}}
}}
```

`\Puzzle@Cell@@b` The letter b produces a frame with a thicker line at the bottom. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{b}{{%
  \put(#1,#2){\framebox(1,1){}}
  \PuzzleThickline
  \put(#1,#2){{\line(1,0){1}}}
}}
```

`\Puzzle@Cell@@l` The letter l produces a frame with a thicker line at the left side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{l}{{%
  \put(#1,#2){\framebox(1,1){}}
  \PuzzleThickline
  \put(#1,#2){{\line(0,1){1}}}
}}
```

`\Puzzle@Cell@@L` The letter L produces a thicker line at the left side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{L}{{%
```

```
  \PuzzleThickline
  \put(#1,#2){{\line(0,1){1}}}
}}
```

`\Puzzle@Cell@@r` The letter r produces a frame with a thicker line at the right side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{r}{{%
  \put(#1,#2){\framebox(1,1){}}
  \advance #1 1
  \PuzzleThickline
  \put(#1,#2){{\line(0,1){1}}}
}}
```

`\Puzzle@Cell@@R` The letter R produces a thicker line at the right side of the cell. The thickness of this line is controlled by the macro `\PuzzleThickline`.

```
\PuzzleDefineCell{R}{{%
  \advance #1 1
  \PuzzleThickline
  \put(#1,#2){{\line(0,1){1}}}
}}
```

`\Puzzle@Cell@@f` The letter f produces a simple frame around the cell. This is the default if nothing is specified.

```
\PuzzleDefineCell{f}{{%
  \put(#1,#2){\framebox(1,1){}}
}}
```

`\Puzzle@Cell@@S` The letter S produces the solution without any other formatting around it.

```
\PuzzleDefineCell{S}{{%
  \put(#1,#2){\makebox(1,1){\expandafter\uppercase{\PuzzleContent}}}
}}
```

`\Puzzle@Cell@@.` The character . produces no additional rendering it can be used to overwrite the default rendering which is to place a frame around the cell.

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

`\Puzzle@Cell@@O` The letter O produces an oval as drawn with the LaTeX macro `\oval`.

```
\PuzzleDefineCell{O}{{%
  \put(\the#1.5,\the#2.5){\oval(1,1){}}
}}
```

`\Puzzle@Cell@@o` The letter o produces an oval inside a frame. T is is an abbreviation for the two letters fO.

```
\PuzzleDefineCell{o}{{%
  \put(#1,#2){\framebox(1,1){}}
  \put(\the#1.5,\the#2.5){\oval(1,1){}}
}}
```

`\Puzzle@Cell@@*` The letter * produces a solid black box.

```
\PuzzleDefineCell{*}{%
  \put(#1,#2){\framebox(1,1){\PuzzleBlackBox}}
}
```

`\Puzzle@Cell@@/` The character / produces a line crossing the cell from lower left to upper right.

```
\PuzzleDefineCell{/}{{%
  \put(#1,#2){{\line(1,1){1}}}
}}
```

`\Puzzle@Cell@@,` The character , produces a line crossing the cell from upper left to lower right.

```
\PuzzleDefineCell{,}{{%
  \advance#2 1
  \put(#1,#2){{\line(1,-1){1}}}
}}
```

6.4 Clues

`\Puzzle@Clue@@normal` The first and the third argument are shown as clue. This macro is used for unsolved puzzles.

```
\newcommand\Puzzle@Clue@@normal[3]{\textsf{#1} #3 }
```

`\Puzzle@Clue@@solution` In solutions clues are simply suppressed. Thus all three arguments are discarded.

```
\newcommand\Puzzle@Clue@@solution[3]{}
```

`Puzzle@Clues@@normal` The environment `Puzzle@Clues@@normal` is mapped to `PuzzleClues` in "normal" mode. It typesets its contents in a mini-page of approximately half text width.

```
\newenvironment{Puzzle@Clues@@normal}[1]{%
  \let\Clue\Puzzle@Clue@@normal
  \PuzzleCluePre
  \PuzzleClueFont{#1}%
}{\PuzzleCluePost }
```

`\PuzzleCluePre` The macro `\PuzzleCluePre` contains the code which is inserted before the clues are typeset in normal mode.

```
\newcommand\PuzzleCluePre{%
  \null\hfill
  \begin{minipage}[t]{.45\textwidth}%
}
```

`\PuzzleCluePost` The macro `\PuzzleCluePost` contains the code which is inserted after the clues are typeset in normal mode.

```
\newcommand\PuzzleCluePost{
  \end{minipage}\hfill\null
}
```

Puzzle@Clues@@solution The environment Puzzle@Clues@@solution is mapped to PuzzleClues in "solution" mode. It just suppresses any output.

```
\newenvironment{Puzzle@Clues@@solution}[1]{%
  \let\Clue\Puzzle@Clue@@solution
}{}
```

\PuzzleWordsText The macro \PuzzleWordsText is the text typeset at the beginning of the environment PuzzleWords. It takes one argument which is the length of the words listed.

```
\newcommand\PuzzleWordsText[1]{Words of length #1: }
```

Puzzle@Words@@normal The environment Puzzle@Words@@normal will be mapped to the environment PuzzleWords in "normal" mode. It just arranges that words are typeset after the \PuzzleWordsText has shown the length of the words. Finally a new paragraph is started.

```
\newenvironment{Puzzle@Words@@normal}[1]{%
  \PuzzleWordsText{#1}%
  \let\Word\relax
}{\par}
```

Puzzle@Words@@solution The environment Puzzle@Words@@solution will be mapped to the environment PuzzleWords in "solution" mode. It arranges things that the contents is silently ignored.

```
\newenvironment{Puzzle@Words@@solution}[1]{%
  \newcommand\Word[1]{}%
}{}
```

6.5 Numbers

\PuzzleNumbers The macro \PuzzleNumbers will produce a list of boxes with numbers for letters. It is intended for numbered crossword puzzles.

```
\newcommand\PuzzleNumbers[1]{\begingroup
  \@nameuse{Puzzle@@\Puzzle@TYPE}%
  \Puzzle@Y=0
  \Puzzle@X=1
  \unitlength=\PuzzleUnitlength
  \Puzzle@Numbers#1.\endgroup}
```

\Puzzle@Numbers The macro \Puzzle@Numbers loops through the arguments until it finds a dot. For each argument it produces a box, either with the numbers or with the letters or both, depending on the current settings.

The loop is implemented via recursion. The box is typeset by the | macro which takes care of the current settings. For this purpose this character has to be made active temporarily.

```
\begingroup
\catcode`\|=13
\gdef\Puzzle@Numbers#1{%
```

```
  \if#1.
    \let\next\relax
  \else
    \begin{picture}(1,1)
      \xdef\X{\the\Puzzle@X}%
      \Puzzle@X=0
      |[\X]{#1}%
    \end{picture}%
    \let\next\Puzzle@Numbers
    \advance\Puzzle@X 1
  \fi
  \next
}
\endgroup
```

\PuzzleLettersText The macro \PuzzleLettersText contains the text typeset at the beginning of the \PuzzleLetters environment.

```
\newcommand\PuzzleLettersText{The following letters are used: }
```

\PuzzleLetters The macro \PuzzleLetters is intended to show the letters used in a numbered crossword puzzle. The argument is the (sorted) list of characters used.

```
\newcommand\PuzzleLetters[1]{\PuzzleLettersText #1\par}
```

\Puzzle@TYPE The macro \Puzzle@TYPE contains the type of the puzzle. It is used find the appropriate initialization macro.

```
\newcommand\Puzzle@TYPE{normal}
```

\PuzzleSolution The macro \PuzzleSolution arranges everything that the following puzzles are typeset in the "solution" mode, i.e. the letters are shown and the clues are suppressed.

This macros has one optional argument which has to be `true` or `false`. This argument determines whether or not the numbers should also be shown in the solution. The default is `false` which means that the numbers are suppressed in the solution.

```
\newcommand\PuzzleSolution[1][false]{%
  \@nameuse{Puzzle@SolutionNumbered#1}%
  \let\Kakuro@HINT\Kakuro@nohint
  \let\PuzzleClues\Puzzle@Clues@@solution
  \let\endPuzzleClues\endPuzzle@Clues@@solution
  \let\PuzzleWords\Puzzle@Words@@solution
  \let\endPuzzleWords\endPuzzle@Words@@solution
  \def\Puzzle@TYPE{solution}%
}
```

\PuzzleUnsolved The macro \PuzzleUnsolved arranges everything that the following puzzles are typeset in the "unsolved" mode, i.e. the letters are suppressed and the clues are shown.

```
\newcommand\PuzzleUnsolved{%
```

```
  \let\Kakuro@HINT\Kakuro@hint
  \let\PuzzleClues\Puzzle@Clues@@normal
  \let\endPuzzleClues\endPuzzle@Clues@@normal
  \let\PuzzleWords\Puzzle@Words@@normal
  \let\endPuzzleWords\endPuzzle@Words@@normal
  \xdef\Puzzle@TYPE{normal}}
```

The boolean `Puzzle@SolutionNumbered` determines whether or not the solution should contain numbers. Initially it is set to "false".

```
\newif\ifPuzzle@SolutionNumbered
\Puzzle@SolutionNumberedfalse
```

6.6 Sudoku

The challenge for the sudoku is to implement a convenient input syntax.

`Sudoku` The environemnt `Sudoku` is used to typeset the puzzle. The implementation defines the begin and end macro separately.

```
\begingroup
\catcode`\|=13
\gdef\Sudoku{\begin{Puzzle}{9}{9}%
  \def\PuzzleContent{}
  \let\Puzzle@pipe=|%
  \def\PPa{\Puzzle@pipe[][fS]}%
  \def|##1{\ifx##1*\let\next\PPa\else
    \Puzzle@pipe{##1}\let\next\relax\fi\next}
}
\endgroup
```

The macro `\endSudoku` contains the code to be expanded at the end of the environment. It draws the field with the 3×3 boxes.

```
\gdef\endSudoku{%
  \multiput(0,0)(1,0)9{\framebox(1,1){}}
  \multiput(0,1)(1,0)9{\framebox(1,1){}}
  \multiput(0,2)(1,0)9{\framebox(1,1){}}
  \multiput(0,3)(1,0)9{\framebox(1,1){}}
  \multiput(0,4)(1,0)9{\framebox(1,1){}}
  \multiput(0,5)(1,0)9{\framebox(1,1){}}
  \multiput(0,6)(1,0)9{\framebox(1,1){}}
  \multiput(0,7)(1,0)9{\framebox(1,1){}}
  \multiput(0,8)(1,0)9{\framebox(1,1){}}
  \linethickness{\SudokuLinethickness}%
  \put(0,0){\framebox(9,9){}}
  \put(3,0){\framebox(3,9){}}
  \put(0,3){\framebox(9,3){}}
\end{Puzzle}}
```

`\SudokuLinethickness` The macro `\SudokuLinethickness` contains the thickness od thick lines in a Sudoku.

```
\newcommand\SudokuLinethickness{2pt}
```

6.7 Kakuro

\KakuroNumberFont The macro \KakuroNumberFont is used to typeset the hints, i.e. the horizontal and vertical sums.

```
\newcommand\KakuroNumberFont{\sffamily\tiny}
```

\Kakuro@cell The macro \Kakuro@cell is used to typeset the cells. It analyzes the argument and acts accordingly.

```
\def\Kakuro@cell#1{%
  \ifx#1.       \def\next{\Puzzle@pipe.}%
  \else\ifx#1<  \let\next\Kakuro@HINT
  \else\ifx#1*  \let\next\Kakuro@always
  \else\ifx#1-  \let\next\Kakuro@empty
  \else\Puzzle@pipe#1 \let\next\relax
  \fi\fi\fi\fi
  \next
}%
```

\Kakuro@always The macro \Kakuro@always is used to draw a cell with an inital number.

```
\def\Kakuro@always{\Puzzle@pipe[][fS]}%
```

\Kakuro@empty The macro \Kakuro@empty is used to drae an empty cell.

```
\def\Kakuro@empty{\Puzzle@pipe{}}%
```

\Kakuro@hint The macro \Kakuro@hint is used to draw hints.

```
\def\Kakuro@hint#1:#2>{%
  \def\x{#2}%
  \ifx\x\empty\else
  \put(\Puzzle@X,\Puzzle@Y){%
    \makebox(1,.8)[r]{\parbox{.95\unitlength}{%
      \raggedright\KakuroNumberFont
        $\blacktriangledown$\\#2}}}
  \fi
  \def\x{#1}%
  \ifx\x\empty\else
  \put(\Puzzle@X,\Puzzle@Y){%
    \makebox(1,1.2){\parbox{.95\unitlength}{%
      \raggedleft\KakuroNumberFont
        #1 $\blacktriangleright$\\\mbox{}}}}
  \fi
  \Puzzle@pipe[][\KakuroHintType]{ }}%
```

\Kakuro@nohint The macro \Kakuro@nohint is used to suppress hints in solution mode.

```
\def\Kakuro@nohint#1:#2>{%
  \Puzzle@pipe[][,]{ }}%
```

\Kakuro@HINT The macro \Kakuro@HINT contains the definition to be used to typeset hints. This indirection is needed to suppress hints in solution mode.

```
\let\Kakuro@HINT\Kakuro@hint
```

`Kakuro` The environemnt `Kakuro` is used to typeset the puzzle. The implementation defines the begin and end macro separately to cope with catcode changes.

```
\begingroup
\catcode`\|=13
\gdef\Kakuro#1#2{\begin{Puzzle}{#1}{#2}%
  \catcode`\|=13
  \let\Puzzle@pipe=|
  \let|=\Kakuro@cell
}
\endgroup
```

The macro `\endKakuro` contains the code to be expanded at the end of the environment.

```
\def\endKakuro{\end{Puzzle}}
```

`\KakuroHintType` The macro `\KakuroHintType` contains the cell type for typesetting hint cells. It can be used to achieve a different look and feel.

```
\def\KakuroHintType{,}
```

6.8 Initialization

Finally we arrange that the default behavior is to typeset an unsolved crossword puzzle.

```
\PuzzleUnsolved
```

Now, that everything is in place we can arrange some package options.

```
\DeclareOption{numbered}{\Puzzle@SolutionNumberedtrue}
\DeclareOption{nocenter}{\let\PuzzlePre=\relax
  \let\PuzzlePost=\relax}
\DeclareOption{unboxed}{\let\PuzzleCluePre=\relax
  \let\PuzzleCluePost=\relax}
\DeclareOption{normalsize}{\PuzzleUnitlength=20pt
  \def\PuzzleFont{\rmfamily\normalsize}}
\DeclareOption{small}{\PuzzleUnitlength=16pt
  \def\PuzzleFont{\rmfamily\small}}
\DeclareOption{large}{\PuzzleUnitlength=24pt
  \def\PuzzleFont{\rmfamily\large}}
\DeclareOption{german}{%
  \renewcommand\PuzzleWordsText[1]{Worte der L\"ange #1: }%
  \renewcommand\PuzzleLettersText{Benutzte Buchstaben: }%
}
\DeclareOption{ngerman}{%
  \renewcommand\PuzzleWordsText[1]{Worte der L\"ange #1: }%
  \renewcommand\PuzzleLettersText{Benutzte Buchstaben: }%
}
\ProcessOptions\relax
```

That's all.

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

S

T

W

X