# Org Mode Manual

Release 4.54

**by Carsten Dominik**

This manual is for Org-mode (version 4.54).

# Table of Contents

# 1 Introduction

## 1.1 Summary

Org-mode is a mode for keeping notes, maintaining ToDo lists, and doing project planning with a fast and effective plain-text system.

Org-mode develops organizational tasks around NOTES files that contain information about projects as plain text. Org-mode is implemented on top of outline-mode, which makes it possible to keep the content of large files well structured. Visibility cycling and structure editing help to work with the tree. Tables are easily created with a built-in table editor. Org-mode supports ToDo items, deadlines, time stamps, and scheduling. It dynamically compiles entries into an agenda that utilizes and smoothly integrates much of the Emacs calendar and diary. Plain text URL-like links connect to websites, emails, Usenet messages, BBDB entries, and any files related to the projects. For printing and sharing of notes, an Org-mode file can be exported as a structured ASCII file, as HTML, or (todo and agenda items only) as an iCalendar file. It can also serve as a publishing tool for a set of linked webpages.

An important design aspect that distinguishes Org-mode from other packages like Planner/Muse is that it encougages to store every piece of information only once. In Planner, you have project pages, day pages and possibly other files, duplicating some information such as tasks. In Org-mode, you only have notes files. In your notes you mark entries as tasks, label them with tags and timestamps. All necessary lists like a schedule for the day, the agenda for a meeting, tasks lists selected by tags etc are created dynamically when you need them.

Org-mode keeps simple things simple. When first fired up, it should feel like a straightforward, easy to use outliner. Complexity is not imposed, but a large amount of functionality is available when you need it. Org-mode can be used on different levels and in different ways, for example:

- as an outline extension with visibility cycling and structure editing
- as an ASCII system and table editor for taking structured notes
- as an ASCII table editor with spreadsheet-like capabilities
- as a TODO list editor
- as a full agenda and planner with deadlines and work scheduling
- as an environment to implement David Allen's GTD system
- as a simple hypertext system, with HTML export
- as a publishing tool to create a set of interlinked webpages

Org-mode's automatic, context sensitive table editor can be integrated into any major mode by activating the minor Orgtbl-mode.

There is a website for Org-mode which provides links to the newest version of Org-mode, as well as additional information, frequently asked questions (FAQ), links to tutorials etc. This page is located at http://www.astro.uva.nl/~dominik/Tools/org/.

## 1.2 Installation

**Important:** If Org-mode is part of the Emacs distribution or an XEmacs package, please skip this section and go directly to Section 1.3 [Activation], page 2.

If you have downloaded Org-mode from the Web, you must take the following steps to install it: Go into the Org-mode distribution directory and edit the top section of the file '`Makefile`'. You must set the name of the Emacs binary (likely either '`emacs`' or '`xemacs`'), and the paths to the directories where local Lisp and Info files are kept. If you don't have access to the system-wide directories, create your own two directories for these files, enter them into the Makefile, and make sure Emacs finds the Lisp files by adding the following line to '`.emacs`':

```
(setq load-path (cons "~/path/to/lispdir" load-path))
```

**XEmacs users now need to install the file '`noutline.el`' from the '`xemacs`' subdirectory of the Org-mode distribution. Use the command:**

**make install-noutline**

Now byte-compile and install the Lisp files with the shell commands:

```
make
make install
```

If you want to install the info documentation, use this command:

```
make install-info
```

Then add to '`.emacs`':

```
;; This line only if org-mode is not part of the X/Emacs distribution.
(require 'org-install)
```

## 1.3 Activation

Add the following lines to your '`.emacs`' file. The last two lines define *global* keys for the commands `org-store-link` and `org-agenda` - please choose suitable keys yourself.

```
;; The following lines are always needed.  Choose your own keys.
(add-to-list 'auto-mode-alist '("\\.org$" . org-mode))
(define-key global-map "\C-cl" 'org-store-link)
(define-key global-map "\C-ca" 'org-agenda)
```

Furthermore, you must activate `font-lock-mode` in org-mode buffers, because significant functionality depends on font-locking being active. You can do this with either one of the following two lines (XEmacs user must use the second option):

```
(global-font-lock-mode 1)                     ; for all buffers
(add-hook 'org-mode-hook 'turn-on-font-lock)  ; org-mode buffers only
```

With this setup, all files with extension '`.org`' will be put into Org-mode. As an alternative, make the first line of a file look like this:

```
MY PROJECTS    -*- mode: org; -*-
```

which will select Org-mode for this buffer no matter what the file's name is. See also the variable `org-insert-mode-line-in-empty-file`.

## 1.4 Feedback

If you find problems with Org-mode, or if you have questions, remarks, or ideas about it, please contact the maintainer Carsten Dominik at dominik at science dot uva dot nl.

For bug reports, please provide as much information as possible, including the version information of Emacs (*C-h v emacs-version* ⟨RET⟩) and Org-mode (*C-h v org-version* ⟨RET⟩), as well as the Org-mode related setup in '`.emacs`'. If an error occurs, a backtrace can be very useful (see below on how to create one). Often a small example file helps, along with clear information about:

1. What exactly did you do?
2. What did you expect to happen?
3. What happened instead?

Thank you for helping to improve this mode.

### How to create a useful backtrace

If working with Org-mode produces an error with a message you don't understand, you may have hit a bug. The best way to report this is by providing, in addition to what was mentioned above, a *Backtrace*. This is information from the built-in debugger about where and how the error occurred. Here is how to produce a useful backtrace:

1. Start a fresh Emacs or XEmacs, and make sure that it will load the original Lisp code in '`org.el`' instead of the compiled version in '`org.elc`'. The backtrace contains much more information if it is produced with uncompiled code. To do this, either rename '`org.elc`' to something else before starting Emacs, or ask Emacs explicitly to load '`org.el`' by using the command line

   ```
   emacs -l /path/to/org.el
   ```

2. Go to the `Options` menu and select `Enter Debugger on Error` (XEmacs has this option in the `Troubleshooting` sub-menu).
3. Do whatever you have to do to hit the error. Don't forget to to document the steps you take.
4. When you hit the error, a '`*Backtrace*`' buffer will appear on the screen. Safe this buffer to a file (for example using *C-x C-w* and attach) it to your bug report.

# 2 Document Structure

Org-mode is based on outline mode and provides flexible commands to edit the structure of the document.

## 2.1 Outlines

Org-mode is implemented on top of outline-mode. Outlines allow to organize a document in a hierarchical structure, which (at least for me) is the best representation of notes and thoughts. Overview over this structure is achieved by folding (hiding) large parts of the document to show only the general document structure and the parts currently being worked on. Org-mode greatly simplifies the use of outlines by compressing the entire show/hide functionality into a single command `org-cycle`, which is bound to the ⟨TAB⟩ key.

## 2.2 Headlines

Headlines define the structure of an outline tree. The headlines in Org-mode start with one or more stars, on the left margin. For example:

```
* Top level headline
** Second level
*** 3rd level
    some text
*** 3rd level
    more text
* Another top level headline
```

Some people find the many stars too noisy and would prefer an outline that has whitespace followed by a single star as headline starters. Section 12.5 [Clean view], page 65 describes a setup to realize this.

## 2.3 Visibility cycling

Outlines make it possible to hide parts of the text in the buffer. Org-mode uses just two commands, bound to ⟨TAB⟩ and *S-*⟨TAB⟩ to change the visibility in the buffer.

⟨TAB⟩ *Subtree cycling*: Rotate current subtree between the states

```
,-> FOLDED -> CHILDREN -> SUBTREE --.
'-----------------------------------'
```

The cursor must be on a headline for this to work[1]. When the cursor is at the beginning of the buffer and the first line is not a headline, then ⟨TAB⟩ actually runs global cycling (see below)[2]. Also when called with a prefix argument (*C-u* ⟨TAB⟩), global cycling is invoked.

[1] see, however, the option `org-cycle-emulate-tab`.

[2] see the option `org-cycle-global-at-bob`.

`S-`⟨TAB⟩

`C-u` ⟨TAB⟩ *Global cycling*: Rotate the entire buffer between the states

```
,-> OVERVIEW -> CONTENTS -> SHOW ALL --.
'--------------------------------------'
```

Note that inside tables, `S-`⟨TAB⟩ jumps to the previous field.

`C-c C-a` Show all.

`C-c C-r` Reveal context around point, showing the current entry, the following heading and the hierarchy above. Useful for working near a location exposed by a sparse tree command (see Section 2.7 [Sparse trees], page 7) or an agenda command (see Section 8.8 [Agenda commands], page 44).

When Emacs first visits an Org-mode file, the global state is set to OVERVIEW, i.e. only the top level headlines are visible. This can be configured through the variable `org-startup-folded`, or on a per-file basis by adding one of the following lines anywhere in the buffer:

```
#+STARTUP: overview
#+STARTUP: content
#+STARTUP: showall
```

## 2.4 Motion

The following commands jump to other headlines in the buffer.

`C-c C-n` Next heading.

`C-c C-p` Previous heading.

`C-c C-f` Next heading same level.

`C-c C-b` Previous heading same level.

`C-c C-u` Backward to higher level heading.

`C-c C-j` Jump to a different place without changing the current outline visibility. Shows the document structure in a temporary buffer, where you can use visibility cycling (⟨TAB⟩) to find your destination. After pressing ⟨RET⟩, the cursor moves to the selected location in the original buffer, and the headings hierarchy above it is made visible.

## 2.5 Structure editing

`M-`⟨RET⟩ Insert new heading with same level as current. If the cursor is in a plain list item, a new item is created (see Section 2.8 [Plain lists], page 8). To force creation of a new headline, use a prefix arg, or first press ⟨RET⟩ to get to the beginning of the next line. When this command is used in the middle of a line, the line is split and the rest of the line becomes the new headline. If the command is used at the beginning of a headline, the new headline is created before the current line. If at the beginning of any other line, the content of that

line is made the new heading. If the command is used at the end of a folded subtree (i.e. behind the ellipses at the end of a headline), then a headline like the current one will be inserted after the end of the subtree.

`M-S-⟨RET⟩` Insert new TODO entry with same level as current heading.

`M-⟨left⟩` Promote current heading by one level.

`M-⟨right⟩` Demote current heading by one level.

`M-S-⟨left⟩` Promote the current subtree by one level.

`M-S-⟨right⟩` Demote the current subtree by one level.

`M-S-⟨up⟩` Move subtree up (swap with previous subtree of same level).

`M-S-⟨down⟩` Move subtree down (swap with next subtree of same level).

`C-c C-x C-w`
`C-c C-x C-k`
Kill subtree, i.e. remove it from buffer but save in kill ring.

`C-c C-x M-w`
Copy subtree to kill ring.

`C-c C-x C-y`
Yank subtree from kill ring. This does modify the level of the subtree to make sure the tree fits in nicely at the yank position. The yank level can also be specified with a prefix arg, or by yanking after a headline marker like '`****`'.

When there is an active region (transient-mark-mode), promotion and demotion work on all headlines in the region. To select a region of headlines, it is best to place both point and mark at the beginning of a line, mark at the beginning of the first headline, and point at the line just after the last headline to change. Note that when the cursor is inside a table (see Chapter 3 [Tables], page 10), the Meta-Cursor keys have different functionality.

## 2.6 Archiving

When a project represented by a (sub)tree is finished, you may want to move the tree out of the way and to stop it from contributing to the agenda. Org-mode knows two ways of archiving. You can mark a tree with the ARCHIVE tag, or you can move an entire (sub)tree to a different location.

### 2.6.1 The ARCHIVE tag

A headline that is marked with the ARCHIVE tag (see Chapter 7 [Tags], page 36) stays at its location in the outline tree, but behaves in the following way:

– It does not open when you attempt to do so with a visibility cycling command (see Section 2.3 [Visibility cycling], page 4). You can force cycling archived subtrees with `C-⟨TAB⟩`, or by setting the option `org-cycle-open-archived-trees`. Also normal outline commands like `show-all` will open archived subtrees.

- During sparse tree construction (see Section 2.7 [Sparse trees], page 7), matches in archived subtrees are not exposed, unless you configure the option `org-sparse-tree-open-archived-trees`.
- During agenda view construction (see Chapter 8 [Agenda views], page 39), the content of archived trees is ignored unless you configure the option `org-agenda-skip-archived-trees`.
- Archived trees are not exported (see Chapter 10 [Exporting], page 53), only the headline is. Configure the details using the variable `org-export-with-archived-trees`.

The following commands help managing the ARCHIVE tag:

*C-c C-x C-a*
: Toggle the ARCHIVE tag for the current headline. When the tag is set, the headline changes to a shadowish face, and the subtree below it is hidden.

*C-u C-c C-x C-a*
: Check if any direct children of the current headline should be archived. To do this, each subtree is checked for open TODO entries. If none are found, the command offers to set the ARCHIVE tag for the child. If the cursor is *not* on a headline when this command is invoked, the level 1 trees will be checked.

*C-TAB*
: Cycle a tree even if it is tagged with ARCHIVE.

### 2.6.2 Moving subtrees

Once an entire project is finished, you may want to move it to a different location, either in the current file, or even in a different file, the archive file.

*C-c $*
: Archive the subtree starting at the cursor position to the location given by `org-archive-location`.

*C-u C-c $*
: Check if any direct children of the current headline could be moved to the archive. To do this, each subtree is checked for open TODO entries. If none are found, the command offers to move it to the archive location. If the cursor is *not* on a headline when this command is invoked, the level 1 trees will be checked.

The default archive location is a file in the same directory as the current file, with the name derived by appending '`_archive`' to the current file name. For information and examples on how to change this, see the documentation string of the variable `org-archive-location`.

## 2.7 Sparse trees

An important feature of Org-mode is the ability to construct *sparse trees* for selected information in an outline tree. A sparse tree means that the entire document is folded as much as possible, but the selected information is made visible along with the headline structure above it[3]. Just try it out and you will see immediately how it works.

[3] See also the variables `org-show-hierarchy-above` and `org-show-following-heading`.

Org-mode contains several commands creating such trees. The most basic one is `org-occur`:

`C-c /` Occur. Prompts for a regexp and shows a sparse tree with all matches. If the match is in a headline, the headline is made visible. If the match is in the body of an entry, headline and body are made visible. In order to provide minimal context, also the full hierarchy of headlines above the match is shown, as well as the headline following the match. Each match is also highlighted; the highlights disappear when the bufer is changes an editing command, or by pressing `C-c C-c`. When called with a `C-u` prefix argument, previous highlights are kept, so several calls to this command can be stacked.

For frequently used sparse trees of specific search strings, you can use the variable `org-agenda-custom-commands` to define fast keyboard access to specific sparse trees. These commands will then be accessible through the agenda dispatcher (see Section 8.2 [Agenda dispatcher], page 40). For example:

```
(setq org-agenda-custom-commands
      '(("f" occur-tree "FIXME")))
```

will define the key `C-c a f` as a shortcut for creating a sparse tree matching the string '`FIXME`'.

Other commands use sparse trees as well. For example `C-c C-v` creates a sparse TODO tree (see Section 5.1 [TODO basics], page 26).

To print a sparse tree, you can use the Emacs command `ps-print-buffer-with-faces` which does not print invisible parts of the document[4]. Or you can use the command `C-c C-e v` to export only the visible part of the document and print the resulting file.

## 2.8 Plain lists

Within an entry of the outline tree, hand-formatted lists can provide additional structure. They also provide a way to create lists of checkboxes (see Section 5.5 [Checkboxes], page 29). Org-mode supports editing such lists, and the HTML exporter (see Chapter 10 [Exporting], page 53) does parse and format them.

Org-mode knows ordered and unordered lists. Unordered list items start with '`-`', '`+`', or '`*`'[5] as bullets. Ordered list items start with '`1.`' or '`1)`'. Items belonging to the same list must have the same indentation on the first line. In particular, if an ordered list reaches number '`10.`', then the 2–digit numbers must be written left-aligned with the other numbers in the list. Indentation also determines the end of a list item. It ends before the next line that is indented like the bullet/number, or less. For example:

[4] This does not work under XEmacs, because XEmacs uses selective display for outlining, not text properties.

[5] When using '`*`' as a bullet, lines must be indented or they will be seen as top-level headlines. Also, when you are hiding leading stars to get a clean outline view, plain list items starting with a star are visually indistinguishable from true headlines. In short: even though '`*`' is supported, it may be better not to use it for plain list items

```
** Lord of the Rings
   My favorite scenes are (in this order)
   1. The attack of the Rohirrim
   2. Eowyns fight with the witch king
      + this was already my favorite scene in the book
      + I really like Miranda Otto.
   3. Peter Jackson being shot by Legolas
       - on DVD only
      He makes a really funny face when it happens.
   But in the end, not individual scenes matter but the film as a whole.
```

Org-mode supports these lists by tuning filling and wrapping commands to deal with them correctly[6].

The following commands act on items when the cursor is in the first line of an item (the line with the bullet or number).

⟨TAB⟩ Items can be folded just like headline levels if you set the variable `org-cycle-include-plain-lists`. The level of an item is then given by the indentation of the bullet/number. Items are always subordinate to real headlines, however; the hierarchies remain completely separated.

*M*-⟨RET⟩ Insert new item at current level. With prefix arg, force a new heading (see Section 2.5 [Structure editing], page 5). If this command is used in the middle of a line, the line is *split* and the rest of the line becomes the new item. If this command is executed in the *whitespace before a bullet or number*, the new item is created *before* the current item. If the command is executed in the white space before the text that is part of an item but does not contain the bullet, a bullet is added to the current line.

*M-S*-⟨RET⟩ Insert a new item with a checkbox (see Section 5.5 [Checkboxes], page 29).

*S*-⟨up⟩
*S*-⟨down⟩ Jump to the previous/next item in the current list.

*M-S*-⟨up⟩
*M-S*-⟨down⟩ Move the item including subitems up/down (swap with previous/next item of same indentation). If the list is ordered, renumbering is automatic.

*M-S*-⟨left⟩
*M-S*-⟨right⟩ Decrease/increase the indentation of the item, including subitems. Initially, the item tree is selected based on current indentation. When these commands are executed several times in direct succession, the initially selected region is used, even if the new indentation would imply a different hierarchy. To use the new hierarchy, break the command chain with a cursor motion or so.

*C-c C-c* If there is a checkbox (see Section 5.5 [Checkboxes], page 29) in the item line, toggle the state of the checkbox. Otherwise, if this is an ordered list, renumber the ordered list at the cursor.

[6] Org-mode only changes the filling settings for Emacs. For XEmacs, you should use Kyle E. Jones' '`filladapt.el`'. To turn is on, put into '`.emacs`':

```
(require 'filladapt)
```

# 3 Tables

Org-mode has a very fast and intuitive table editor built-in. Spreadsheet-like calculations are supported in connection with the Emacs 'calc' package.

## 3.1 The built-in table editor

Org-mode makes it easy to format tables in plain ASCII. Any line with '|' as the first non-white character is considered part of a table. '|' is also the column separator. A table might look like this:

```
| Name  | Phone | Age |
|-------+-------+-----|
| Peter |  1234 |  17 |
| Anna  |  4321 |  25 |
```

A table is re-aligned automatically each time you press ⟨TAB⟩ or ⟨RET⟩ or *C-c C-c* inside the table. ⟨TAB⟩ also moves to the next field (⟨RET⟩ to the next row) and creates new table rows at the end of the table or before horizontal lines. The indentation of the table is set by the first line. Any line starting with '|-' is considered as a horizontal separator line and will be expanded on the next re-align to span the whole table width. So, to create the above table, you would only type

```
|Name|Phone|Age|
|-
```

and then press ⟨TAB⟩ to align the table and start filling in fields.

When typing text into a field, Org-mode treats ⟨DEL⟩, ⟨Backspace⟩, and all character keys in a special way, so that inserting and deleting avoids shifting other fields. Also, when typing *immediately after the cursor was moved into a new field with* ⟨TAB⟩*,* *S-*⟨TAB⟩ *or* ⟨RET⟩, the field is automatically made blank. If this behavior is too unpredictable for you, configure the variables `org-enable-table-editor` and `org-table-auto-blank-field`.

**Creation and conversion**

*C-c |* Convert the active region to table. If every line contains at least one TAB character, the function assumes that the material is tab separated. If not, lines are split at whitespace into fields. You can use a prefix argument to indicate the minimum number of consecutive spaces required to identify a field separator (default: just one).
If there is no active region, this command creates an empty Org-mode table. But it's easier just to start typing, like *|Name|Phone|Age* ⟨RET⟩ *|-* ⟨TAB⟩.

**Re-aligning and field motion**

*C-c C-c* Re-align the table without moving the cursor.

⟨TAB⟩ Re-align the table, move to the next field. Creates a new row if necessary.

*S-*⟨TAB⟩ Re-align, move to previous field.

⟨RET⟩ Re-align the table and move down to next row. Creates a new row if necessary. At the beginning or end of a line, ⟨RET⟩ still does NEWLINE, so it can be used to split a table.

**Column and row editing**

`M-⟨left⟩`
`M-⟨right⟩` Move the current column left/right.

`M-S-⟨left⟩` Kill the current column.

`M-S-⟨right⟩` Insert a new column to the left of the cursor position.

`M-⟨up⟩`
`M-⟨down⟩` Move the current row up/down.

`M-S-⟨up⟩` Kill the current row or horizontal line.

`M-S-⟨down⟩` Insert a new row above (with arg: below) the current row.

`C-c -` Insert a horizontal line below current row. With prefix arg, the line is created above the current line.

`C-c ^` Sort the table lines in the region. Point and mark must be in the first and last line to be included, and must be in the column that should be used for sorting. The command prompts for numerical versus alphanumerical sorting.

**Regions**

`C-c C-x M-w`
Copy a rectangular region from a table to a special clipboard. Point and mark determine edge fields of the rectangle. The process ignores horizontal separator lines.

`C-c C-x C-w`
Copy a rectangular region from a table to a special clipboard, and blank all fields in the rectangle. So this is the "cut" operation.

`C-c C-x C-y`
Paste a rectangular region into a table. The upper right corner ends up in the current field. All involved fields will be overwritten. If the rectangle does not fit into the present table, the table is enlarged as needed. The process ignores horizontal separator lines.

`C-c C-q` Wrap several fields in a column like a paragraph. If there is an active region, and both point and mark are in the same column, the text in the column is wrapped to minimum width for the given number of lines. A prefix ARG may be used to change the number of desired lines. If there is no region, the current field is split at the cursor position and the text fragment to the right of the cursor is prepended to the field one line down. If there is no region, but you specify a prefix ARG, the current field is made blank, and the content is appended to the field above.

**Calculations**

`C-c =` Install a new formula for the current column and replace current field with the result of the formula.

`C-u C-c =` Install a new formula for the current field, which must be a named field. Evaluate the formula and replace the field content with the result.

`C-c '` Edit all formulas associated with the current table in a separate buffer.

`C-c *` Recalculate the current row by applying the stored formulas from left to right. When called with a `C-u` prefix, recalculate the entire table, starting with the first non-header line (i.e. below the first horizontal separator line). For details, see Section 3.3 [Table calculations], page 13.

`C-#` Rotate the calculation mark in first column through the states ‘’, ‘`#`’, ‘`*`’, ‘`!`’, ‘`$`’. For the meaning of these marks see Section 3.3.4 [Advanced features], page 15. When there is an active region, change all marks in the region.

`C-c ?` Which table column is the cursor in? Displays number `>0` in echo area.

`C-c +` Sum the numbers in the current column, or in the rectangle defined by the active region. The result is shown in the echo area and can be inserted with `C-y`.

`S-`⟨RET⟩ When current field is empty, copy from first non-empty field above. When not empty, copy current field down to next row and move cursor along with it. Depending on the variable `org-table-copy-increment`, integer field values will be incremented during copy. This key is also used by CUA-mode (see Section 12.7.1 [Cooperation], page 67).

**Miscellaneous**

`C-c ‘` Edit the current field in a separate window. This is useful for fields that are not fully visible (see Section 3.2 [Narrow columns], page 12). When called with a `C-u` prefix, just make the full field visible, so that it can be edited in place.

`C-c` ⟨TAB⟩ This is an alias for `C-u C-c ‘` to make the current field fully visible.

`M-x org-table-import`
Import a file as a table. The table should be TAB- or whitespace separated. Useful, for example, to import an Excel table or data from a database, because these programs generally can write TAB-separated text files. This command works by inserting the file into the buffer and then converting the region to a table. Any prefix argument is passed on to the converter, which uses it to determine the separator.

`M-x org-table-export`
Export the table as a TAB-separated file. Useful for data exchange with, for example, Excel or database programs.

If you don’t like the automatic table editor because it gets in your way on lines which you would like to start with ‘`|`’, you can turn it off with

```
(setq org-enable-table-editor nil)
```

Then the only table command that still works is `C-c C-c` to do a manual re-align.

## 3.2 Narrow columns

The width of columns is automatically determined by the table editor. Sometimes a single field or a few fields need to carry more text, leading to inconveniently wide columns.

To limit[1] the width of a column, one field anywhere in the column may contain just the string '<N>' where 'N' is an integer specifying the width of the column in characters. The next re-align will then set the width of this column to no more than this value.

```
|---+------------------------------|               |---+--------|
|   |                              |               |   | <6>    |
| 1 | one                          |               | 1 | one    |
| 2 | two                          |     ----\     | 2 | two    |
| 3 | This is a long chunk of text |     ----/     | 3 | This=> |
| 4 | four                         |               | 4 | four   |
|---+------------------------------|               |---+--------|
```

Fields that are wider become clipped and end in the string '=>'. Note that the full text is still in the buffer, it is only invisible. To see the full text, hold the mouse over the field - a tooltip window will show the full content. To edit such a field, use the command *C-c '* (that is *C-c* followed by the backquote). This will open a new window with the full field. Edit it and finish with *C-c C-c*.

When visiting a file containing a table with narrowed columns, the necessary character hiding has not yet happened, and the table needs to be aligned before it looks nice. Setting the option `org-startup-align-all-tables` will realign all tables in a file upon visiting, but also slow down startup. You can also set this option on a per-file basis with:

```
#+STARTUP: align
#+STARTUP: noalign
```

## 3.3 Calculations in tables

The table editor makes use of the Emacs 'calc' package to implement spreadsheet-like capabilities. It can also evaluate Emacs Lisp forms to derive fields from other fields. Org-mode has two levels of complexity for table calculations. On the basic level, tables do only horizontal computations, so a field can be computed from other fields *in the same row*, and Org-mode assumes that there is only one formula for each column. This is very efficient to work with and enough for many tasks. On the complex level, columns and individual fields can be named for easier referencing in formulas, individual named fields can have their own formula associated with them, and recalculation can be automated.

### 3.3.1 Formula syntax

A formula can be any algebraic expression understood by the Emacs 'calc' package. Note that 'calc' has the slightly non-standard convention that '/' has lower precedence than '*', so that 'a/b*c' is interpreted as 'a/(b*c)'. Before evaluation by `calc-eval` (see section "Calling calc from Your Lisp Programs" in *GNU Emacs Calc Manual*), variable substitution takes place:

| | |
|---|---|
| `$` | refers to the current field |
| `$3` | refers to the field in column 3 of the current row |
| `$3..$7` | a vector of the fields in columns 3-7 of current row |

[1] This feature does not work on XEmacs.

| | |
|---|---|
| `$P1..$P3` | vector of column range, using column names |
| `&2` | second data field above the current, in same column |
| `&5-2` | vector from fifth to second field above current |
| `&III-II` | vector of fields between 2nd and 3rd hline above |
| `&III` | vector of fields between third hline above and current field |
| `$name` | a named field, parameter or constant |

The range vectors can be directly fed into the calc vector functions like '`vmean`' and '`vsum`'.

'`$name`' is interpreted as the name of a column, parameter or constant. Constants are defined globally through the variable `org-table-formula-constants`. If you have the '`constants.el`' package, it will also be used to resolve constants, including natural constants like '`$h`' for Planck's constant, and units like '`$km`' for kilometers. Column names and parameters can be specified in special table lines. These are described below, see Section 3.3.4 [Advanced features], page 15.

A formula can contain an optional mode string after a semicolon. This string consists of flags to influence calc's modes[2] during execution, e.g. '`p20`' to switch the internal precision to 20 digits, '`n3`', '`s3`', '`e2`' or '`f4`' to switch to normal, scientific, engineering, or fixed display format, respectively, and '`D`', '`R`', '`F`', and '`S`' to turn on degrees, radians, fraction and symbolic modes, respectively. In addition, you may provide a `printf` format specifier to reformat the final result. A few examples:

| | |
|---|---|
| `$1+$2` | Sum of first and second field |
| `$1+$2;%.2f` | Same, format result to two decimals |
| `exp($2)+exp($1)` | Math functions can be used |
| `$;%.1f` | Reformat current cell to 1 decimal |
| `($3-32)*5/9` | Degrees F -> C conversion |
| `$c/$1/$cm` | Hz -> cm conversion, using '`constants.el`' |
| `tan($1);Dp3s1` | Compute in degrees, precision 3, display SCI 1 |
| `sin($1);Dp3%.1e` | Same, but use printf specifier for display |
| `vmean($2..$7)` | Compute column range mean, using vector function |
| `vsum(&III)` | Sum numbers from 3rd hline above, up to here |
| `taylor($3,x=7,2)` | taylor series of $3, at x=7, second degree |

### 3.3.2 Emacs Lisp forms as formulas

It is also possible to write a formula in Emacs lisp; this can be useful for string manipulation and control structures. If a formula starts with a single quote followed by an opening parenthesis, then it is evaluated as a lisp form. The evaluation should return either a string or a number. Just as with '`calc`' formulas, you can provide a format specifier after a semicolon. A few examples:

swap the first two characters of the content of column 1

```
'(concat (substring "$1" 1 2) (substring "$1" 0 1) (substring "$1" 2))
```

Add columns 1 and 2, equivalent to the calc's `$1+$2`

```
'(+ $1 $2)
```

[2] By default, Org-mode uses the standard calc modes (precision 12, angular units degrees, fraction and symbolic modes off). The display format, however, has been changed to `(float 5)` to keep tables compact. The default settings can be configured using the variable `org-calc-default-modes`.

### 3.3.3 Column formulas

To apply a formula to a field, type it directly into the field, preceded by an equal sign, like '`=$1+$2`'. When you press ⟨TAB⟩ or ⟨RET⟩ or `C-c C-c` with the cursor still in the field, the formula will be stored as the formula for the current column, evaluated and the current field replaced with the result. If the field contains only '`=`', the previously stored formula for this column is used.

For each column, Org-mode will remember the most recently used formula. The information is stored in a special line starting with '`#+TBLFM`' directly below the table. When adding/deleting/moving columns with the appropriate commands, the stored equations will be modified accordingly. When a column used in a calculation is removed, references to this column become invalid and will cause an error upon applying the equation.

Instead of typing an equation into the field, you may also use the command `C-c =`. It prompts for a formula (with default taken from the '`#+TBLFM:`' line) and applies it to the current field. A numerical prefix (e.g. `C-5 C-c =`) will apply it to that many consecutive fields in the current column.

To recompute all the fields in a line, use the command `C-c *`. It re-applies all stored equations to the current row, from left to right. With a `C-u` prefix, this will be done to every line in the table, so use this command it you want to make sure the entire table is up-to-date. `C-u C-c C-c` is another way to update the entire table. Global updating does not touch the line(s) above the first horizontal separator line, assuming that this is the table header.

### 3.3.4 Advanced features

If you want the recalculation of fields to happen automatically, or if you want to be able to assign a formula to an individual field (instead of an entire column) you need to reserve the first column of the table for special marking characters. Here is an example of a table that collects exam results of students and makes use of these features:

```
|---+---------+--------+--------+--------+-------+------|
|   | Student | Prob 1 | Prob 2 | Prob 3 | Total | Note |
|---+---------+--------+--------+--------+-------+------|
| ! |         |     P1 |     P2 |     P3 |   Tot |      |
| # | Maximum |     10 |     15 |     25 |    50 | 10.0 |
| ^ |         |     m1 |     m2 |     m3 |    mt |      |
|---+---------+--------+--------+--------+-------+------|
| # | Peter   |     10 |      8 |     23 |    41 |  8.2 |
| # | Sara    |      6 |     14 |     19 |    39 |  7.8 |
| # | Sam     |      2 |      4 |      3 |     9 |  1.8 |
|---+---------+--------+--------+--------+-------+------|
|   | Average |        |        |        |  29.7 |      |
| ^ |         |        |        |        |    at |      |
| $ | max=50  |        |        |        |       |      |
|---+---------+--------+--------+--------+-------+------|
#+TBLFM: $6=vsum($P1..$P3)::$7=10*$Tot/$max;%.1f::$at=vmean(&II);%.1f
```

**Important**: Please note that for these special tables, recalculating the table with `C-u C-c *` will only affect rows that are marked '`#`' or '`*`', and named fields. The column formulas are not applied in rows with empty first field.

The marking characters have the following meaning:

'`!`' The fields in this line define names for the columns, so that you may refer to a column as '`$Tot`' instead of '`$6`'.

'`^`' This row defines names for the fields *above* the row. With such a definition, any formula in the table may use '`$m1`' to refer to the value '`10`'. Also, named fields can have their own formula associated with them.

'`_`' Similar to '`^`', but defines names for the fields in the row *below*.

'`$`' Fields in this row can define *parameters* for formulas. For example, if a field in a '`$`' row contains '`max=50`', then formulas in this table can refer to the value 50 using '`$max`'. Parameters work exactly like constants, only that they can be defined on a per-table basis. Changing a parameter and then recalculating the table can be useful.

'`#`' Fields in this row are automatically recalculated when pressing ⟨TAB⟩ or ⟨RET⟩ or `S-`⟨TAB⟩ in this row. Also, this row is selected for a global recalculation with `C-u C-c *`. Unmarked lines will be left alone by this command.

'`*`' Selects this line for global recalculation with `C-u C-c *`, but not for automatic recalculation. Use this when automatic recalculation slows down editing too much.

'' Unmarked lines are exempt from recalculation with `C-u C-c *`. All lines that should be recalculated should be marked with '`#`' or '`*`'.

### 3.3.5 Named-field formulas

A named field can have its own formula associated with it. In the example above, this is used for the '`at`' field that contains the average result of the students. To enter a formula for a named field, just type it into the buffer, preceded by '`:=`'. Or use `C-u C-c =`. This equation will be stored below the table like '`$name=...`'. Any recalculation in the table (even if only requested for the current line) will also update all named field formulas.

### 3.3.6 Editing and debugging formulas

To edit a column or field formula, use the commands `C-c =` and `C-u C-c =`, respectively. The currently active expression is then presented as default in the minibuffer, where it may be edited.

Note that making a table field blank does not remove the formula associated with the field - during the next recalculation the field will be filled again. To remove a formula from a field, you have to give an empty reply when prompted for the formula, or to edit the '`#+TBLFM`' line.

You may edit the '`#+TBLFM`' directly and re-apply the changed equations with `C-c C-c` in that line, or with the normal recalculation commands in the table.

In particular for large tables with many formulas, it is convenient to use the command *C-c '* to edit the formulas of the current table in a separate buffer. That buffer will show the formulas one per line, and you are free to edit, add and remove formulas. Press *C-c ?* on a '`$...`' expression to get information about its interpretation. Exiting the buffer with *C-c C-c* only stores the modified formulas below the table. Exiting with *C-u C-c C-c* also applies them to the entire table. *C-c C-q* exits without installing the changes.

When the evaluation of a formula leads to an error, the field content becomes the string '`#ERROR`'. If you would like see what is going on during variable substitution and calculation in order to find a bug, turn on formula debugging in the menu and repeat the calculation, for example by pressing *C-c =* ⟨RET⟩ in a field. Detailed information will be displayed.

### 3.3.7 Appetizer

Finally, just to whet your appetite on what can be done with the fantastic '`calc`' package, here is a table that computes the Taylor series for a couple of functions (homework: try that with Excel :-)

```
|---+-------------+---+-----+--------------------------------------|
|   | Func        | n | x   | Result                               |
|---+-------------+---+-----+--------------------------------------|
| # | exp(x)      | 1 | x   | 1 + x                                |
| # | exp(x)      | 2 | x   | 1 + x + x^2 / 2                      |
| # | exp(x)      | 3 | x   | 1 + x + x^2 / 2 + x^3 / 6            |
| # | x^2+sqrt(x) | 2 | x=0 | x*(0.5 / 0) + x^2 (2 - 0.25 / 0) / 2 |
| # | x^2+sqrt(x) | 2 | x=1 | 2 + 2.5 x - 2.5 + 0.875 (x - 1)^2    |
| * | tan(x)      | 3 | x   | 0.0175 x + 1.77e-6 x^3               |
|---+-------------+---+-----+--------------------------------------|
#+TBLFM: $5=taylor($2,$4,$3);n3
```

## 3.4 The Orgtbl minor mode

If you like the intuitive way the Org-mode table editor works, you might also want to use it in other modes like text-mode or mail-mode. The minor mode Orgtbl-mode makes this possible. You can always toggle the mode with *M-x orgtbl-mode*. To turn it on by default, for example in mail mode, use

```
(add-hook 'mail-mode-hook 'turn-on-orgtbl)
```

## 3.5 The '`table.el`' package

Complex ASCII tables with automatic line wrapping, column- and row-spanning, and alignment can be created using the Emacs table package by Takaaki Ota (http://sourceforge.net/projects/table, and also part of Emacs 22). When ⟨TAB⟩ or *C-c C-c* is pressed in such a table, Org-mode will call `table-recognize-table` and move the cursor into the table. Inside a table, the keymap of Org-mode is inactive. In order to execute Org-mode-related commands, leave the table.

`C-c C-c` Recognize '`table.el`' table. Works when the cursor is in a table.el table.

`C-c ~` Insert a table.el table. If there is already a table at point, this command converts it between the table.el format and the Org-mode format. See the documentation string of the command `org-convert-table` for the restrictions under which this is possible.

# 4 Hyperlinks

Just like HTML, Org-mode provides links inside a file, and external links to other files, Usenet articles, emails, and much more.

## 4.1 Link format

Org-mode will recognize plain URL-like links and activate them as clickable links. The general link format, however, looks like this:

`[[link][description]]` or alternatively `[[link]]`

Once a link in the buffer is complete (all brackets present), Org-mode will change the display so that '`description`' is displayed instead of '`[[link][description]]`' and '`link`' is displayed instead of '`[[link]]`'. Links will be highlighted in the face `org-link`, which by default is an underlined face. You can directly edit the visible part of a link. Note that this can be either the '`link`' part (if there is no description) or the '`description`' part. To edit also the invisible '`link`' part, use *C-c C-l* with the cursor on the link.

If you place the cursor at the beginning or just behind the end of the displayed text and press ⟨BACKSPACE⟩, you will remove the (invisible) bracket at that location. This makes the link incomplete and the internals are again displayed as plain text. Inserting the missing bracket hides the link internals again. To show the internal structure of all links, use the menu entry `Org->Hyperlinks->Literal links`.

## 4.2 Internal links

If the link does not look like a URL, it is considered to be internal in the current file. Links such as '`[[My Target]]`' or '`[[My Target][Find my target]]`' lead to a text search in the current file. The link can be followed with *C-c C-o* when the cursor is on the link, or with a mouse click (see Section 4.4 [Handling links], page 21). The preferred match for such a link is a dedicated target: the same string in double angular brackets. Targets may be located anywhere; often it is convenient to put them into a comment line. For example

```
# <<My Target>>
```

In HTML export (see Section 10.2 [HTML export], page 53), such targets will become named anchors for direct access through '`http`' links[1].

If no dedicated target exists, Org-mode will search for the words in the link. In the above example the search would be for '`my target`'. Links starting with a star like '`*My Target`' restrict the search to headlines. When searching, Org-mode will first try an exact match, but then move on to more and more lenient searches. For example, the link '`[[*My Targets]]`' will find any of the following:

```
** My targets
** TODO my targets are bright
** my 20 targets are
```

[1] Note that text before the first headline will never be exported, so the first such target must be after the first headline.

To insert a link targeting a headline, in-buffer completion can be used. Just type a star followed by a few optional letters into the buffer and press `M-⟨TAB⟩`. All headlines in the current buffer will be offered as completions. See Section 4.4 [Handling links], page 21, for more commands creating links.

Following a link pushes a mark onto Org-mode's own mark ring. You can return to the previous position with `C-c &`. Using this command several times in direct succession goes back to positions recorded earlier.

### 4.2.1 Radio targets

You can configure Org-mode to link any occurrences of certain target names in normal text. So without explicitly creating a link, the text connects to the target radioing its position. Radio targets are enclosed by triple angular brackets. For example, a target '`<<<My Target>>>`' causes each occurrence of '`my target`' in normal text to become activated as a link. The Org-mode file is scanned automatically for radio targets only when the file is first loaded into Emacs. To update the target list during editing, press `C-c C-c` with the cursor on or at a target.

### 4.2.2 CamelCase words as links

Org-mode also supports CamelCase words as links. This feature is not turned on by default because of the inconsistencies this system suffers from. It is also possible that this feature will disappear entirely in a future version of Org-mode. To activate CamelCase words as links, you need to customize the option `org-activate-links`. A CamelCase word then leads to a text search such that '`CamelCaseLink`' is equivalent to '`[[camel case link]]`'.

## 4.3 External links

Org-mode supports links to files, websites, Usenet and email messages, and BBDB database entries. External links are URL-like locators. They start with a short identifying string followed by a colon. There can be no space after the colon. The following list shows examples for each link type.

| | |
|---|---|
| `http://www.astro.uva.nl/~dominik` | on the web |
| `file:/home/dominik/images/jupiter.jpg` | file, absolute path |
| `file:papers/last.pdf` | file, relative path |
| `news:comp.emacs` | Usenet link |
| `mailto:adent@galaxy.net` | Mail link |
| `vm:folder` | VM folder link |
| `vm:folder#id` | VM message link |
| `vm://myself@some.where.org/folder#id` | VM on remote machine |
| `wl:folder` | WANDERLUST folder link |
| `wl:folder#id` | WANDERLUST message link |
| `mhe:folder` | MH-E folder link |
| `mhe:folder#id` | MH-E message link |

| | |
|---|---|
| `rmail:folder` | RMAIL folder link |
| `rmail:folder#id` | RMAIL message link |
| `gnus:group` | GNUS group link |
| `gnus:group#id` | GNUS article link |
| `bbdb:Richard Stallman` | BBDB link |
| `shell:ls *.org` | A shell command |
| `elisp:(find-file-other-frame "Elisp.org")` | An elisp form to evaluate |

A link should be enclosed in double brackets and may contain a descriptive text to be displayed instead of the url (see Section 4.1 [Link format], page 19), for example:

```
[[http://www.gnu.org/software/emacs/][GNU Emacs]]
```

Org-mode also finds external links in the normal text and activates them as links. If spaces must be part of the link (for example in '`bbdb:Richard Stallman`'), or you need to remove ambiguities about the end of the link, enclose them in angular brackets.

## 4.4 Handling links

Org-mode provides methods to create a link in the correct syntax, to insert it into an org-mode file, and to follow the link.

*C-c l* Store a link to the current location. This is a *global* command which can be used in any buffer to create a link. The link will be stored for later insertion into an Org-mode buffer (see below). For Org-mode files, if there is a '`<<target>>`' at the cursor, the link points to the target. Otherwise it points to the current headline. For VM, RMAIL, WANDERLUST, MH-E, GNUS and BBDB buffers, the link will indicate the current article/entry. For W3 and W3M buffers, the link goes to the current URL. For any other files, the link will point to the file, with a search string (see Section 4.6 [Search options], page 23) pointing to the contents of the current line. If there is an active region, the selected words will form the basis of the search string. If the automatically created link is not working correctly or accurately enough, you can write custom functions to select the search string and to do the search for particular file types - see Section 4.7 [Custom searches], page 24. The key binding *C-c l* is only a suggestion - see Section 1.2 [Installation], page 2.

*C-c C-l* Insert a link. This prompts for a link to be inserted into the buffer. You can just type a link, using text for an internal link, or one of the link type prefixes mentioned in the examples above. Through completion, all links stored during the current session can be accessed[2]. The link will be inserted into the buffer, along with a descriptive text. Note that you don't have to use this command to insert a link. Links in Org-mode are plain text, and you can type or paste them straight into the buffer. By using this command, the links are automatically enclosed in double brackets, and you will be asked for the optional descriptive text. If the link is a '`file:`' link and the linked file is located in the same

[2] After insertion of a stored link, the link will be removed from the list of stored links. To keep it in the list later use, use a triple *C-u* prefix to *C-c C-l*, or configure the option `org-keep-stored-link-after-insertion`.

directory as the current file or a subdirectory of it, the path of the file will be inserted relative to the current directory.

*C-u C-c C-l*
: When *C-c C-l* is called with a *C-u* prefix argument, a link to a file will be inserted and you may use file name completion to select the name of the file. The path to the file is inserted relative to the directory of the current org file, if the linked file is in the current directory or in a subdirectory of it, or if the path is written relative to the current directory using '`../`'. Otherwise an absolute path is used, if possible with '`~/`' for your home directory. You can force an absolute path with two *C-u* prefixes.

*C-c C-l* with cursor on existing link
: When the cursor is on an existing link, *C-c C-l* allows you to edit the link and description parts of the link.

*C-c C-o*
: Open link at point. This will launch a web browser for URLs (using `browse-url-at-point`), run vm/mh-e/wanderlust/rmail/gnus/bbdb for the corresponding links, and execute the command in a shell link. When the cursor is on an internal link, this commands runs the corresponding search. When the cursor is on a TAG list in a headline, it creates the corresponding TAGS view. If the cursor is on a time stamp, it compiles the agenda for that date. Furthermore, it will visit text and remote files in '`file:`' links with Emacs and select a suitable application for local non-text files. Classification of files is based on file extension only. See option `org-file-apps`. If you want to override the default application and visit the file with Emacs, use a *C-u* prefix.

*mouse-2*
*mouse-1*
: On links, *mouse-2* will open the link just as *C-c C-o* would. Under Emacs 22, also *mouse-1* will follow a link.

*mouse-3*
: Like *mouse-2*, but force file links to be opened with Emacs, and internal links to be displayed in another window[3].

*C-c %*
: Push the current position onto the mark ring, to be able to return easily. Commands following an internal link do this automatically.

*C-c &*
: Jump back to a recorded position. A position is recorded by the commands following internal links, and by *C-c %*. Using this command several times in direct succession moves through a ring of previously recorded positions.

## 4.5 Link abbreviatons

Long URLs can be cumbersome to type, and often many similar links are needed in a document. For this you can use link abbreviations. An abbreviated link looks like this

```
[[linkword::tag][description]]
```

where the tag is optional. Such abbreviations are resolved according to the information in the variable `org-link-abbrev-alist` that relates the linkwords to replacement text. Here is an example:

[3] See the variable `org-display-internal-link-with-indirect-buffer`

```
(setq org-link-abbrev-alist
  '(("bugzilla" . "http://10.1.2.9/bugzilla/show_bug.cgi?id=")
    ("google"   . "http://www.google.com/search?q=")
    ("ads"      . "http://adsabs.harvard.edu/cgi-bin/
                   nph-abs_connect?author=%s&db_key=AST")))
```

If the replacement text contains the string '`%s`', it will be replaced with the tag. Otherwise the tag will be appended to the string in order to create the link. You may also specify a function that will be called with the tag as the only argument to create the link.

With the above setting, you could link to a specific bug with `[[bugzilla::129]]`, search the web for OrgMode with `[[google::OrgMode]]` and find out what the Org-mode author is doing besides Emacs hacking with `[[ads::Dominik,C]]`.

If you need special abbreviations just for a single Org-mode buffer, you can define them in the file with

```
#+LINK: bugzilla  http://10.1.2.9/bugzilla/show_bug.cgi?id=
#+LINK: google    http://www.google.com/search?q=%s
```

In-buffer completion see Section 12.1 [Completion], page 63 can be used after '`[`' to complete link abbreviations.

## 4.6 Search options in file links

File links can contain additional information to make Emacs jump to a particular location in the file when following a link. This can be a line number or a search option after a double[4] colon. For example, when the command `C-c l` creates a link (see Section 4.4 [Handling links], page 21) to a file, it encodes the words in the current line as a search string that can be used to find this line back later when following the link with `C-c C-o`.

Here is the syntax of the different ways to attach a search to a file link, together with an explanation:

```
[[file:~/code/main.c::255]]
[[file:~/xx.org::My Target]]
[[file:~/xx.org::*My Target]]
[[file:~/xx.org::/regexp/]]
```

`255` Jump to line 255.

`My Target` Search for a link target '`<<My Target>>`', or do a text search for '`my target`', similar to the search in internal links, see Section 4.2 [Internal links], page 19. In HTML export (see Section 10.2 [HTML export], page 53), such a file link will become an HTML reference to the corresponding named anchor in the linked file.

`*My Target`
In an Org-mode file, restrict search to headlines.

`/regexp/` Do a regular expression search for `regexp`. This uses the Emacs command `occur` to list all matches in a separate window. If the target file is in Org-mode, `org-occur` is used to create a sparse tree with the matches.

[4] For backward compatibility, line numbers can also follow a single colon.

As a degenerate case, a file link with an empty file name can be used to search the current file. For example, `<file:::find me>` does a search for '`find me`' in the current file, just as '`[[find me]]`' would.

## 4.7 Custom Searches

The default mechanism for creating search strings and for doing the actual search related to a file link may not work correctly in all cases. For example, BibTeX database files have many entries like '`year="1993"`' which would not result in good search strings, because the only unique identification for a BibTeX entry is the citation key.

If you come across such a problem, you can write custom functions to set the right search string for a particular file type, and to do the search for the string in the file. Using `add-hook`, these functions need to be added to the hook variables `org-create-file-search-functions` and `org-execute-file-search-functions`. See the docstring for these variables for more information. Org-mode actually uses this mechanism for BibTeX database files, and you can use the corresponding code as an implementation example. Search for '`BibTeX links`' in the source file.

## 4.8 Remember

Another way to create org entries with links to other files is through the *Remember* package by John Wiegley. *Remember* lets you store quick notes with little interruption of your work flow. See http://www.emacswiki.org/cgi-bin/wiki/RememberMode for more information. The notes produced by *Remember* can be stored in different ways, and Org-mode files are a good target. Org-mode allows you to file away notes either to a default file, or directly to the correct location in your Org-mode outline tree. The following customization will tell *Remember* to use org files as target, and to create annotations compatible with Org-mode links.

```
(setq org-directory "~/path/to/my/orgfiles/")
(setq org-default-notes-file "~/.notes")
(setq remember-annotation-functions '(org-remember-annotation))
(setq remember-handler-functions '(org-remember-handler))
(add-hook 'remember-mode-hook 'org-remember-apply-template)
```

In combination with Org-mode, you can use templates to generate different types of remember notes. For example, if you would like to use one template to create general TODO entries, and another one for journal entries, you could use:

```
(setq org-remember-templates
      '((?t "* TODO %?\n  %i\n  %a" "~/org/TODO.org")
        (?j "* %U %?\n\n  %i\n  %a" "~/org/JOURNAL.org")))
```

In these entries, the character specifies how to select the template, the first string specifies the template, and the (optional) second string specifies a default file (overruling `org-default-notes-file`) as a target for this note.

When you call *`M-x remember`* to remember something, org will prompt for a key to select the template and then prepare the buffer like

```
* TODO
  <file:link to where you called remember>
```

or

```
* [2006-03-21 Tue 15:37]

  <file:link to where you called remember>
```

See the variable `org-remember-templates` for more details.

When you are finished composing a note with remember, you have to press `C-c C-c` to file the note away. The handler first prompts for a target file - if you press ⟨RET⟩, the value of `org-default-notes-file` is used. Then the command offers the headings tree of the selected file. You can either immediately press ⟨RET⟩ to get the note appended to the file. Or you can use vertical cursor motion (⟨up⟩ and ⟨down⟩) and visibility cycling (⟨TAB⟩) to find a better place. Pressing ⟨RET⟩ or ⟨left⟩ or ⟨right⟩ leads to the following result.

| **Cursor position** | **Key** | **Note gets inserted** |
|---|---|---|
| buffer-start | ⟨RET⟩ | as level 2 heading at end of file |
| on headline | ⟨RET⟩ | as sublevel of the heading at cursor |
| | ⟨left⟩ | as same level, before current heading |
| | ⟨right⟩ | as same level, after current heading |
| not on headline | ⟨RET⟩ | at cursor position, level taken from context. Or use prefix arg to specify level manually. |

So a fast way to store the note is to press `C-c C-c` ⟨RET⟩ ⟨RET⟩ to append it to the default file. Even shorter would be `C-u C-c C-c`, which does the same without even showing the tree. But with little extra effort, you can push it directly to the correct location.

Before inserting the text into a tree, the function ensures that the text has a headline, i.e. a first line that starts with a '`*`'. If not, a headline is constructed from the current date and some additional data. If the variable `org-adapt-indentation` is non-nil, the entire text is also indented so that it starts in the same column as the headline (after the asterisks).

# 5 TODO items

Org-mode does not maintain TODO lists as a separate document. TODO items are an integral part of the notes file, because TODO items usually come up while taking notes! With Org-mode, you simply mark any entry in a tree as being a TODO item. In this way, the information is not duplicated, and the entire context from which the item emerged is always present when you check.

Of course, this technique causes TODO items to be scattered throughout your file. Org-mode provides methods to give you an overview over all things you have to do.

## 5.1 Basic TODO functionality

Any headline can become a TODO item by starting it with the word TODO, for example:

```
*** TODO Write letter to Sam Fortune
```

The most important commands to work with TODO entries are:

`C-c C-t` Rotate the TODO state of the current item between

```
,-> (unmarked) -> TODO -> DONE --.
'--------------------------------'
```

The same rotation can also be done "remotely" from the timeline and agenda buffers with the `t` command key (see Section 8.8 [Agenda commands], page 44).

`S-`⟨right⟩
`S-`⟨left⟩ Select the following/preceding TODO state, similar to cycling. Mostly useful if more than two TODO states are possible (see Section 5.2 [TODO extensions], page 26).

`C-c C-v` View TODO items in a *sparse tree* (see Section 2.7 [Sparse trees], page 7). Folds the entire buffer, but shows all TODO items and the headings hierarchy above them. With prefix arg, show also the DONE entries. With numerical prefix N, show the tree for the Nth keyword in the variable `org-todo-keywords`.

`C-c a t` Show the global TODO list. This collects the TODO items from all agenda files (see Chapter 8 [Agenda views], page 39) into a single buffer. The buffer is in `agenda-mode`, so there are commands to examine and manipulate the TODO entries directly from that buffer (see Section 8.8 [Agenda commands], page 44). See Section 8.4 [Global TODO list], page 41, for more information.

## 5.2 Extended use of TODO keywords

The default implementation of TODO entries is just two states: TODO and DONE. You can, however, use the TODO feature for more complicated things by configuring the variables `org-todo-keywords` and `org-todo-interpretation`. Using special setup, you can even use TODO keywords in different ways in different org files.

Note that *tags* are another way to classify headlines in general and TODO items in particular (see Chapter 7 [Tags], page 36).

### 5.2.1 TODO keywords as workflow states

You can use TODO keywords to indicate different states in the process of working on an item, for example:

```
(setq org-todo-keywords '("TODO" "FEEDBACK" "VERIFY" "DONE")
      org-todo-interpretation 'sequence)
```

Changing these variables only becomes effective in a new Emacs session. With this setup, the command `C-c C-t` will cycle an entry from TODO to FEEDBACK, then to VERIFY, and finally to DONE. You may also use a prefix argument to quickly select a specific state. For example `C-3 C-c C-t` will change the state immediately to VERIFY. If you define many keywords, you can use in-buffer completion (see Section 12.1 [Completion], page 63) to insert these words into the buffer.

### 5.2.2 TODO keywords as types

The second possibility is to use TODO keywords to indicate different types of action items. For example, you might want to indicate that items are for "work" or "home". If you are into David Allen's *Getting Things DONE*, you might want to use todo types 'NEXTACTION', 'WAITING', 'MAYBE'. Or, when you work with several people on a single project, you might want to assign action items directly to persons, by using their names as TODO keywords. This would be set up like this:

```
(setq org-todo-keywords '("Fred" "Sara" "Lucy" "Mike" "DONE")
      org-todo-interpretation 'type)
```

In this case, different keywords do not indicate a sequence, but rather different types. So it is normally not useful to change from one type to another. Therefore, in this case the behavior of the command `C-c C-t` is changed slightly[1]. When used several times in succession, it will still cycle through all names. But when you return to the item after some time and execute `C-c C-t` again, it will switch from each name directly to DONE. Use prefix arguments or completion to quickly select a specific name. You can also review the items of a specific TODO type in a sparse tree by using a numeric prefix to `C-c C-v`. For example, to see all things Lucy has to do, you would use `C-3 C-c C-v`. To collect Lucy's items from all agenda files into a single buffer, you would use the prefix arg as well when creating the global todo list: `C-3 C-c t`.

### 5.2.3 Setting up TODO keywords for individual files

It can be very useful to use different aspects of the TODO mechanism in different files, which is not possible with the global settings described above. For file-local settings, you need to add special lines to the file which set the keywords and interpretation for that file only. For example, to set one of the two examples discussed above, you need one of the following lines, starting in column zero anywhere in the file:

```
#+SEQ_TODO: TODO FEEDBACK VERIFY DONE
#+TYP_TODO: Fred Sara Lucy Mike DONE
```

[1] This is also true for the `t` command in the timeline and agenda buffers.

To make sure you are using the correct keyword, type '`#+`' into the buffer and then use *M-*⟨TAB⟩ completion.

Remember that the last keyword must always mean that the item is DONE (although you may use a different word). Also note that in each file, only one of the two aspects of TODO keywords can be used. After changing one of these lines, use *C-c C-c* with the cursor still in the line to make the changes known to Org-mode[2].

If you want to use very many keywords, for example when working with a large group of people, you may split the names over several lines:

```
#+TYP_TODO: Fred Sara Lucy Mike
#+TYP_TODO: Luis George Jules Jessica
#+TYP_TODO: Kim Arnold Peter
#+TYP_TODO: DONE
```

## 5.3 Priorities

If you use Org-mode extensively to organize your work, you may end up with a number of TODO entries so large that you'd like to prioritize them. This can be done by placing a *priority cookie* into the headline, like this

```
*** TODO [#A] Write letter to Sam Fortune
```

With its standard setup, Org-mode supports priorities '`A`', '`B`', and '`C`'. '`A`' is the highest priority. An entry without a cookie is treated as priority '`B`'. Priorities make a difference only in the agenda (see Section 8.3 [Weekly/Daily agenda], page 40).

*C-c ,* Set the priority of the current headline. The command prompts for a priority character '`A`', '`B`' or '`C`'. When you press ⟨SPC⟩ instead, the priority cookie is removed from the headline. The priorities can also be changed "remotely" from the timeline and agenda buffer with the `,` command (see Section 8.8 [Agenda commands], page 44).

*S-*⟨up⟩
*S-*⟨down⟩ Increase/decrease priority of current headline. Note that these keys are also used to modify time stamps (see Section 6.2 [Creating timestamps], page 31). Furthermore, these keys are also used by CUA-mode (see Section 12.7.2 [Conflicts], page 68).

## 5.4 Breaking tasks down into subtasks

It is often advisable to break down large tasks into smaller, managable subtasks. You can do this by creating an outline tree below a TODO item, with detailed subtasks on the tree[3]. Another possibility is the use of checkboxes to ideantify (a hierarchy of) a large number of subtasks (see Section 5.5 [Checkboxes], page 29).

[2] Org-mode parses these lines only when Org-mode is activated after visiting a file. *C-c C-c* with the cursor in a line starting with '`#+`' is simply restarting Org-mode for the current buffer.

[3] To keep subtasks out of the global TODO list, see the `org-agenda-todo-list-sublevels`.

## 5.5 Checkboxes

Every item in a plain list (see Section 2.8 [Plain lists], page 8) can be made a checkbox by starting it with the string '`[ ]`'. This feature is similar to TODO items (see Chapter 5 [TODO items], page 26), but more lightweight. Checkboxes are not included into the global TODO list, so they are often great to split a task into a number of simple steps. Or you can use them in a shopping list. To toggle a checkbox, use *C-c C-c*, or try Piotr Zielinski's '`org-mouse.el`'. Here is an example of a checkbox list.

```
* TODO Organize party [3/6]
  - call people [1/3]
    - [ ] Peter
    - [X] Sarah
    - [ ] Sam
  - [X] order food
  - [ ] think about what music to play
  - [X] talk to the neighbors
```

The '`[3/6]`' and '`[1/3]`' in the first and second line are cookies indicating how many checkboxes are present in this entry, and how many of them have been checked off. This can give you an idea on how many checkboxes remain, even without opening a folded entry. The cookies can be placed into a headline or into (the first line of) a plain list item. Each cookie covers all checkboxes structurally below that headline/item. You have to insert the cookie yourself by typing either '`[/]`' or '`[%]`'. In the first case you get an '`n out of m`' result, in the second case you get information about the percentage of checkboxes checked (in the above example, this would be '`[50%]`' and '`[33%], respectively`').

The following commands work with checkboxes:

*C-c C-c* Toggle checkbox at point.

*C-c C-x C-b*
Toggle checkbox at point.

- If there is an active region, toggle the first checkbox in the region and set all remaining boxes to the same status as the first. If you want to toggle all boxes in the region independently, use a prefix argument.
- If the cursor is in a headline, toggle checkboxes in the region between this headline and the next (so *not* the entire subtree).
- If no active region, just toggle the checkbox at point.

*M-S-*⟨RET⟩ Insert a new item with a checkbox. This works only if the cursor is already in a plain list item (see Section 2.8 [Plain lists], page 8).

*C-c #* Update the checkbox statistics in the current outline entry. When called with a *C-u* prefix, update the entire file. Checkbox statistic cookies are updated automatically if you toggle checkboxes with *C-c C-c* and make new ones with *M-S-*⟨RET⟩. If you delete boxes or add/change them by hand, use this command to get things back into synch. Or simply toggle any checkbox twice with *C-c C-c*.

# 6 Timestamps

Items can be labeled with timestamps to make them useful for project planning.

## 6.1 Time stamps, deadlines and scheduling

A time stamp is a specification of a date (possibly with time) in a special format, either '`<2003-09-16 Tue>`' or '`<2003-09-16 Tue 09:39>`'[1]. A time stamp can appear anywhere in the headline or body of an org-tree entry. Its presence allows entries to be shown on specific dates in the agenda (see Section 8.3 [Weekly/Daily agenda], page 40). We distinguish:

*Plain time stamp*
: A simple time stamp just assigns a date/time to an item. This is just like writing down an appointment in a paper agenda, or like writing down an event in a diary, when you want to take note of when something happened. In the timeline and agenda displays, the headline of an entry associated with a plain time stamp will be shown exactly on that date.

```
* Meet Peter at the movies <2006-11-01 Wed 19:15>
```

*Inactive time stamp*
: Just like a plain time stamp, but with square brackets instead of angular ones. These time stamps are inactive in the sense that thay do *not* trigger an entry to show up in the agenda.

```
* Gillian comes late for the fifth time [2006-11-01 Wed]
```

*Time stamp range*
: Two time stamps connected by '`--`' denote a time range. The headline will be shown on the first and last day of the range, and on any dates that are displayed and fall in the range. Here is an example:

```
** Meeting in Amsterdam
   <2004-08-23 Mon>--<2004-08-26 Thu>
```

*Time stamp with SCHEDULED keyword*
: If a time stamp is preceded by the word '`SCHEDULED:`', it means you are planning to start working on that task on the given date. So this is not about recording an event, but about planning your work. The headline will be listed under the given date. In addition, a reminder that the scheduled date has passed will be present in the compilation for *today*, until the entry is marked DONE. I.e., the task will automatically be forwarded until completed.

```
*** TODO Call Trillian for a date on New Years Eve.
    SCHEDULED: <2004-12-25 Sat>
```

*Time stamp with DEADLINE keyword*
: If a time stamp is preceded by the word '`DEADLINE:`', the task (most likely a TODO item) is supposed to be finished on that date, and it will be listed

[1] This is the standard ISO date/time format. If you cannot get used to these, see Section 6.3 [Custom time format], page 33

then. In addition, the compilation for *today* will carry a warning about the approaching or missed deadline, starting `org-deadline-warning-days` before the due date, and continuing until the entry is marked DONE. An example:

```
*** TODO write article about the Earth for the Guide
    The editor in charge is <bbdb:Ford Prefect>
    DEADLINE: <2004-02-29 Sun>
```

*Time stamp with CLOSED keyword*
: When `org-log-done` is non-nil, Org-mode will automatically insert a special time stamp each time a TODO entry is marked done (see Section 6.4 [Progress logging], page 33). This time stamp is enclosed in square brackets instead of angular brackets.

*Time range with CLOCK keyword*
: When using the clock to time the work that is being done on specific items, time ranges preceded by the CLOCK keyword are inserted automatically into the file. The time stamps are enclosed in square brackets instead of angular brackets. See Section 6.4.2 [Clocking work time], page 34.

## 6.2 Creating timestamps

For Org-mode to recognize time stamps, they need to be in the specific format. All commands listed below produce time stamps in the correct format.

`C-c .`
: Prompt for a date and insert a corresponding time stamp. When the cursor is at a previously used time stamp, it is updated to NOW. When this command is used twice in succession, a time range is inserted.

`C-u C-c .`
: Like `C-c .`, but use the alternative format which contains date and time. The default time can be rounded to multiples of 5 minutes, see the option `org-time-stamp-rounding-minutes`.

`C-c !`
: Like `C-c .`, but insert an inactive time stamp not triggering the agenda.

`C-c <`
: Insert a time stamp corresponding to the cursor date in the Calendar.

`C-c >`
: Access the Emacs calendar for the current date. If there is a timestamp in the current line, goto the corresponding date instead.

`C-c C-o`
: Access the agenda for the date given by the time stamp or -range at point (see Section 8.3 [Weekly/Daily agenda], page 40).

`C-c C-d`
: Insert '`DEADLINE`' keyword along with a stamp. The insertion will happen in the line directly following the headline.

`C-c C-w`
: Create a sparse tree with all deadlines that are either past-due, or which will become due within `org-deadline-warning-days`. With `C-u` prefix, show all deadlines in the file. With a numeric prefix, check that many days. For example, `C-1 C-c C-w` shows all deadlines due tomorrow.

`C-c C-s`
: Insert '`SCHEDULED`' keyword along with a stamp. The insertion will happen in the line directly following the headline. Any CLOSED timestamp will be removed.

*S-*⟨left⟩
*S-*⟨right⟩ Change date at cursor by one day. These key bindings conflict with CUA-mode (see Section 12.7.2 [Conflicts], page 68).

*S-*⟨up⟩
*S-*⟨down⟩ Change the item under the cursor in a timestamp. The cursor can be on a year, month, day, hour or minute. Note that if the cursor is in a headline and not at a time stamp, these same keys modify the priority of an item. (see Section 5.3 [Priorities], page 28). The key bindings also conflict with CUA-mode (see Section 12.7.2 [Conflicts], page 68).

*C-c C-y* Evaluate a time range by computing the difference between start and end. With prefix arg, insert result after the time range (in a table: into the following column).

### 6.2.1 The date/time prompt

When Org-mode prompts for a date/time, the prompt suggests to enter an ISO date. But it will in fact accept any string containing some date and/or time information. You can, for example, use *C-y* to paste a (possibly multi-line) string copied from an email message. Org-mode will find whatever information is in there and will replace anything not specified with the current date and time. For example:

```
3-2-5         --> 2003-02-05
feb 15        --> currentyear-02-15
sep 12 9      --> 2009-09-12
12:45         --> today 12:45
22 sept 0:34  --> currentyear-09-22 0:34
12            --> currentyear-currentmonth-12
Fri           --> nearest Friday (today or later)
```

The function understands English month and weekday abbreviations. If you want to use unabbreviated names and/or other languages, configure the variables `parse-time-months` and `parse-time-weekdays`.

Parallel to the minibuffer prompt, a calendar is popped up[2]. You can control the calendar fully from the minibuffer:

`<` Scroll calendar backwards by one month.

`>` Scroll calendar forwards by one month.

*mouse-1* Select date by clicking on it.

*S-*⟨right⟩ One day forward.

*S-*⟨left⟩ One day back.

*S-*⟨down⟩ One week forward.

*S-*⟨up⟩ One week back.

*M-S-*⟨right⟩ One month forward.

[2] If you don't need/want the calendar, configure the variable `org-popup-calendar-for-date-prompt`.

`M-S-⟨left⟩` One month back.

`⟨RET⟩` Choose date in calendar (only if nothing was typed into minibuffer).

## 6.3 Custom time format

Org-mode uses the standard ISO notation for dates and times as it is defined in ISO 8601. If you cannot get used to this and require another representation of date and time to keep you happy, you can get it by customizing the variables `org-display-custom-times` and `org-time-stamp-custom-formats`.

`C-c C-x C-t`
: Toggle the display of custom formats for dates and times.

Org-mode needs the default format for scanning, so the custom date/time format does not *replace* the default format - instead it is put *over* the default format using text properties. This has the following consequences:

- You cannot place the cursor onto a time stamp anymore, only before or after.
- The `S-⟨up⟩/⟨down⟩` keys can no longer be used to adjust each component of a time stamp. If the cursor is at the beginning of the stamp, `S-⟨up⟩/⟨down⟩` will change the stamp by one day, just like `S-⟨left⟩/⟨right⟩`. At the end of the stamp, the time will be changed by one minute.
- When you delete a time stamp character-by-character, it will only disappear from the buffer after *all* (invisible) characters belonging to the ISO timestamp have been removed.
- If the custom time stamp format is longer than the default and you are using dates in tables, table alignment will be messed up. If the custom format is shorter, things do work as expected.

## 6.4 Progress Logging

Org-mode can automatically record a time stamp when you mark a TODO item as DONE. You can also measure precisely the time you spent on specific items in a project by starting and stopping a clock when you start and stop working on an aspect of a project.

### 6.4.1 Closing items

If you want to keep track of *when* a certain TODO item was finished, turn on logging with

```
(setq org-log-done t)
```

Then each time you turn a TODO entry into DONE using either `C-c C-t` in the Org-mode buffer or `t` in the agenda buffer, a line '`CLOSED: [timestamp]`' will be inserted just after the headline. If you turn the entry back into a TODO item again through further state cycling, that line will be removed again. In the timeline (see Section 8.6 [Timeline], page 42) and in the agenda (see Section 8.3 [Weekly/Daily agenda], page 40), you can then use the `l` key to display the TODO items closed on each day, giving you an overview of what has been

done on a day. See the variable `org-log-done` for the possibility to record an additional note together with a timestamp.

### 6.4.2 Clocking work time

Org-mode allows you to clock the time you spent on specific tasks in a project. When you start working on an item, you can start the clock. When you stop working on that task, or when you mark the task done, the clock is stopped and the corresponding time interval is recorded. It also computes the total time spent on each subtree of a project.

*C-c C-x C-i*
: Start the clock on the current item (clock-in). This inserts the CLOCK keyword together with a timestamp.

*C-c C-x C-o*
: Stop the clock (clock-out). The inserts another timestamp at the same location where the clock was last started. It also directly computes the resulting time in inserts it after the time range as '`=> HH:MM`'. See the variable `org-log-done` for the possibility to record an additional note together with a the clock-out time stamp.

*C-c C-y*
: Recompute the time interval after changing one of the time stamps. This is only necessary if you edit the time stamps directly. If you change them with *S-*⟨cursor⟩ keys, the update is automatic.

*C-c C-t*
: Changing the TODO state of an item to DONE automatically stops the clock if it is running in this same item.

*C-c C-x C-x*
: Cancel the current clock. This is useful if a clock was started by mistake, or if you ended up working on something else.

*C-c C-x C-d*
: Display time summaries for each subtree in the current buffer. This puts overlays at the end of each headline, showing the total time recorded under that heading, including the time of any subheadings. You can use visibility cycling to study the tree, but the overlays disappear when you change the buffer (see variable `org-remove-highlights-with-change`) or press *C-c C-c*.

*C-c C-x C-r*
: Insert a dynamic block (see Section A.2 [Dynamic blocks], page 70) containing a clock report as an org-mode table into the current file.

```
#+BEGIN: clocktable :maxlevel 2 :emphasize nil

#+END: clocktable
```

If such a block already exists, its content is replaced by the new table. The '`BEGIN`' line can specify options:

| | |
|---|---|
| `:maxlevels` | Maximum level depth to which times are listed in the table. |
| `:emphasize` | When `t`, emphasize level one and level two items |
| `:block` | The time block to consider. This block is specified relative |

to the current time and may be any of these keywords: `today`, `yesterday`, `thisweek`, `lastweek`, `thismonth`, `lastmonth`, `thisyear`, or `lastyear`.

`:tstart` A time string specifying when to start considering times

`:tend` A time string specifying when to stop considering times

So to get a clock summary for the current day, you could write

```
#+BEGIN: clocktable :maxlevel 2 :block today

#+END: clocktable
```

and to use a specific time range you could write[3]

```
#+BEGIN: clocktable :tstart "<2006-08-10 Thu 10:00>"
                    :tend "<2006-08-10 Thu 12:00>"

#+END: clocktable
```

*C-u C-c C-x C-u*

Update all dynamic blocks (see Section A.2 [Dynamic blocks], page 70). This is useful if you have several clocktable blocks in a buffer.

The *l* key may be used in the timeline (see Section 8.6 [Timeline], page 42) and in the agenda (see Section 8.3 [Weekly/Daily agenda], page 40) to show which tasks have been worked on or closed during a day.

[3] Note that all parameters must be specified in a single line - the line is broken here only to fit it onto the manual.

# 7 Tags

If you wish to implement a system of labels and contexts for cross-correlating information, an excellent way is to assign *tags* to headlines. Org-mode has extensive support for using tags.

Every headline can contain a list of tags, at the end of the headline. Tags are normal words containing letters, numbers, '_', and '@'. Tags must be preceded and followed by a single colon; like ':WORK:'. Several tags can be specified like ':WORK:URGENT:'.

## 7.1 Tag inheritance

*Tags* make use of the hierarchical structure of outline trees. If a heading has a certain tag, all subheadings will inherit the tag as well. For example, in the list

```
* Meeting with the French group      :WORK:
** Summary by Frank                  :BOSS:NOTES:
*** TODO Prepare slides for him      :ACTION:
```

the final heading will have the tags ':WORK:', ':BOSS:', ':NOTES:', and ':ACTION:'. When executing tag searches and Org-mode finds that a certain headline matches the search criterion, it will not check any sublevel headline, assuming that these likely also match, and that the list of matches can become very long. This may not be what you want, however, and you can influence inheritance and searching using the variables `org-use-tag-inheritance` and `org-tags-match-list-sublevels`.

## 7.2 Setting tags

Tags can simply be typed into the buffer at the end of a headline. After a colon, *M-*⟨TAB⟩ offers completion on tags. There is also a special command for inserting tags:

*C-c C-c* Enter new tags for the current headline. Org-mode will either offer completion or a special single-key interface for setting tags, see below. After pressing ⟨RET⟩, the tags will be inserted and aligned to `org-tags-column`. When called with a *C-u* prefix, all tags in the current buffer will be aligned to that column, just to make things look nice. TAGS are automatically realigned after promotion, demotion, and TODO state changes (see Section 5.1 [TODO basics], page 26).

Org will support tag insertion based on a *list of tags*. By default this list is constructed dynamically, containing all tags currently used in the buffer. You may also globally specify a hard list of tags with the variable `org-tag-alist`. Finally you can set the default tags for a given file with lines like

```
#+TAGS: @WORK @HOME @TENNISCLUB
#+TAGS: Laptop Car PC Sailboat
```

If you have globally defined your preferred set of tags using the variable `org-tag-alist`, but would like to use a dynamic tag list in a specific file: Just add an empty TAGS option line to that file:

```
#+TAGS:
```

The default support method for entering tags is minibuffer completion. However, Org-mode also implements a much better method: *fast tag selection*. This method allows to select and deselect tags with a single key per tag. To function efficiently, you should assign unique keys to most tags. This can be done globally with

```
(setq org-tag-alist '(("@WORK" . ?w) ("@HOME" . ?h) ("Laptop" . ?l)))
```

or on a per-file basis with

```
#+TAGS: @WORK(w)  @HOME(h)  @TENNISCLUB(t)  Laptop(l)  PC(p)
```

You can also group together tags that are mutually exclusive. With curly braces[1]

```
#+TAGS: { @WORK(w)  @HOME(h)  @TENNISCLUB(t) }  Laptop(l)  PC(p)
```

you indicate that at most one of '`@WORK`', '`@HOME`', and '`@TENNISCLUB`' should be selected.

Don't forget to press *C-c C-c* with the cursor in one of these lines to activate any changes.

If at least one tag has a selection key, pressing *C-c C-c* will automatically present you with a special interface, listing inherited tags, the tags of the current headline, and a list of all legal tags with corresponding keys[2]. In this interface, you can use the following keys:

*a-z...* Pressing keys assigned to tags will add or remove them from the list of tags in the current line. Selecting a tag in a group of mutually exclusive tags will turn off any other tags from that group.

⟨TAB⟩ Enter a tag in the minibuffer, even if the tag is not in the predefined list. You will be able to complete on all tags present in the buffer.

⟨SPC⟩ Clear all tags for this line.

⟨RET⟩ Accept the modified set.

*C-g* Abort without installing changes.

*q* If *q* is not assigned to a tag, it aborts like *C-g*.

*!* Turn of groups of mutually exclusive tags. Use this to (as an exception) assign several tags from such a group.

*C-c* Toggle auto-exit after the next change (see below).

This method lets you assign tags to a headline with very few keys. With the above setup, you could clear the current tags and set '`@HOME`', '`Laptop`' and '`PC`' tags with just the following keys: *C-c C-c* ⟨SPC⟩ *h l p* ⟨RET⟩. Switching from '`@HOME`' to '`@WORK`' would be done with *C-c C-c w* ⟨RET⟩ or alternatively with *C-c C-c C-c w*. Adding the non-predefined tag '`Sarah`' could be done with *C-c C-c* ⟨TAB⟩ *S a r a h* ⟨RET⟩ ⟨RET⟩.

If you find that most of the time, you need only a single keypress to modify your list of tags, set the variable `org-fast-tag-selection-single-key`. Then you no longer have to press ⟨RET⟩ to exit fast tag selection - it will immediately exit after the first change. If you then occasionally need more keys, press *C-c* to turn off auto-exit for the current tag selection process.

---

[1] In `org-mode-alist` use `'(:startgroup)` and `'(:endgroup)`, respectively. Several groups are allowed.

[2] Keys will automatically be assigned to tags which have no configured keys.

## 7.3 Tag searches

Once a tags system has been set up, it can be used to collect related information into special lists.

`C-c \` Create a sparse tree with all headlines matching a tags search.

`C-c a m` Create a global list of tag matches from all agenda files. See Section 8.5 [Matching headline tags], page 42.

`C-c a M` Create a global list of tag matches from all agenda files, but check only TODO items and force checking subitems (see variable `org-tags-match-list-sublevels`).

A *tags* search string can use Boolean operators '`&`' for AND and '`|`' for OR. '`&`' binds more strongly than '`|`'. Parenthesis are currently not implemented. A tag may also be preceded by '`-`', to select against it, and '`+`' is syntactic sugar for positive selection. The AND operator '`&`' is optional when '`+`' or '`-`' is present. Examples:

'`+WORK-BOSS`'
: Select all headlines that are tagged '`:WORK:`', but discard those also tagged '`:BOSS:`'.

'`WORK|LAPTOP`'
: Selects lines tagged '`:WORK:`' or '`:LAPTOP:`'.

'`WORK|LAPTOP&NIGHT`'
: Like the previous example, but require the '`:LAPTOP:`' lines to be tagged also '`NIGHT`'.

If you are using multi-state TODO keywords (see Section 5.2 [TODO extensions], page 26), it can be useful to also match on the TODO keyword. This can be done by adding a condition after a slash to a tags match. The syntax is similar to the tag matches, but should be applied with consideration: For example, a positive selection on several TODO keywords can not meaningfully be combined with boolean AND. However, *negative selection* combined with AND can be meaningful. Examples:

'`WORK/WAITING`'
: Select '`:WORK:`'-tagged TODO lines with the specific TODO keyword '`WAITING`'.

'`WORK/-WAITING-NEXT`'
: Select '`:WORK:`'-tagged TODO lines that are neither '`WAITING`' nor '`NEXT`'

'`WORK/+WAITING|+NEXT`'
: Select '`:WORK:`'-tagged TODO lines that are either '`WAITING`' or '`NEXT`'.

# 8 Agenda Views

Due to the way Org-mode works, TODO items, time-stamped items, and tagged headlines can be scattered throughout a file or even a number of files. To get an overview over open action items, or over events that are important for a particular date, this information must be collected, sorted and displayed in an organized way.

Org-mode can select items based on various criteria, and display them in a separate buffer. Five different view types are provided:

- an *agenda* that is like a calendar and shows information for specific dates
- a *TODO list* that covers all unfinished action items,
- a *tags view* that shows information based on the tags associated with headlines in the outline tree,
- a *timeline view* that shows all events in a single Org-mode file, in time-sorted view
- *custom views* that are special tag and keyword searches and combinations of different views.

The extracted information is displayed in a special *agenda buffer*. This buffer is read-only, but provides commands to visit the corresponding locations in the original Org-mode files, and even to edit these files remotely.

Two variables control how the agenda buffer is displayed and whether the window configuration is restored when the agenda exits: `org-agenda-window-setup` and `org-agenda-restore-windows-after-quit`.

## 8.1 Agenda files

The information to be shown is collected from all *agenda files*, the files listed in the variable `org-agenda-files`[1]. Thus even if you only work with a single Org-mode file, this file should be put into that list[2]. You can customize `org-agenda-files`, but the easiest way to maintain it is through the following commands

`C-c [` Add current file to the list of agenda files. The file is added to the front of the list. If it was already in the list, it is moved to the front. With prefix arg, file is added/moved to the end.

`C-c ]` Remove current file from the list of agenda files.

`C-,` Cycle through agenda file list, visiting one file after the other.

The Org menu contains the current list of files and can be used to visit any of them.

[1] If the value of that variable is not a list, but a single file name, then the list of agenda files will be maintained in that external file.

[2] When using the dispatcher, pressing *1* before selecting a command will actually limit the command to the current file, and ignore `org-agenda-files` until the next dispatcher command.

## 8.2 The agenda dispatcher

The views are created through a dispatcher that should be bound to a global key, for example `C-c a` (see Section 1.2 [Installation], page 2). In the following we will assume that `C-c a` is indeed how the dispatcher is accessed and list keyboard access to commands accordingly. After pressing `C-c a`, an additional letter is required to execute a command. The dispatcher offers the following default commands:

`a` Create the calendar-like agenda (see Section 8.3 [Weekly/Daily agenda], page 40).

`t / T` Create a list of all TODO items (see Section 8.4 [Global TODO list], page 41).

`m / M` Create a list of headlines matching a TAGS expression (see Section 8.5 [Matching headline tags], page 42).

`L` Create the timeline view for the current buffer (see Section 8.6 [Timeline], page 42).

`1` Restrict an agenda command to the current buffer. After pressing `1`, you still need to press the character selecting the command.

`0` If there is an active region, restrict the following agenda command to the region. Otherwise, restrict it to the current subtree. After pressing `0`, you still need to press the character selecting the command.

You can also define custom commands that will be accessible through the dispatcher, just like the default commands. This includes the possibility to create extended agenda buffers that contain several blocks together, for example the weekly agenda, the global TODO list and a number of special tags matches. See Section 8.9 [Custom agenda views], page 46.

## 8.3 The weekly/daily agenda

The purpose of the weekly/daily *agenda* is to act like a page of a paper agenda, showing all the tasks for the current week or day.

`C-c a a` Compile an agenda for the current week from a list of org files. The agenda shows the entries for each day. With a `C-u` prefix (or when the variable `org-agenda-include-all-todo` is `t`), all unfinished TODO items (including those without a date) are also listed at the beginning of the buffer, before the first date.

Remote editing from the agenda buffer means, for example, that you can change the dates of deadlines and appointments from the agenda buffer. The commands available in the Agenda buffer are listed in Section 8.8 [Agenda commands], page 44.

### 8.3.1 Calendar/Diary integration

Emacs contains the calendar and diary by Edward M. Reingold. The calendar displays a three-month calendar with holidays from different countries and cultures. The diary

allows you to keep track of anniversaries, lunar phases, sunrise/set, recurrent appointments (weekly, monthly) and more. In this way, it is quite complementary to Org-mode. It can be very useful to combine output from Org-mode with the diary.

In order to include entries from the Emacs diary into Org-mode's agenda, you only need to customize the variable

```
(setq org-agenda-include-diary t)
```

After that, everything will happen automatically. All diary entries including holidays, anniversaries etc will be included in the agenda buffer created by Org-mode. ⟨SPC⟩, ⟨TAB⟩, and ⟨RET⟩ can be used from the agenda buffer to jump to the diary file in order to edit existing diary entries. The `i` command to insert new entries for the current date works in the agenda buffer, as well as the commands `S`, `M`, and `C` to display Sunrise/Sunset times, show lunar phases and to convert to other calendars, respectively. `c` can be used to switch back and forth between calendar and agenda.

## 8.4 The global TODO list

The global TODO list contains all unfinished TODO items, formatted and collected into a single place.

`C-c a t` Show the global TODO list. This collects the TODO items from all agenda files (see Chapter 8 [Agenda views], page 39) into a single buffer. The buffer is in `agenda-mode`, so there are commands to examine and manipulate the TODO entries directly from that buffer (see Section 8.8 [Agenda commands], page 44).

`C-c a T` Like the above, but allows selection of a specific TODO keyword. You can also do this by specifying a prefix argument to `C-c a t`. With a `C-u` prefix you are prompted for a keyword. With a numeric prefix, the Nth keyword in `org-todo-keywords` is selected. The `r` key in the agenda buffer regenerates it, and you can give a prefix argument to this command to change the selected TODO keyword, for example `3 r`. If you often need a search for a specific keyword, define a custom command for it (see Section 8.2 [Agenda dispatcher], page 40). Matching specific TODO keywords can also be done as part of a tags search (see Section 7.3 [Tag searches], page 38).

Remote editing of TODO items means that you can change the state of a TODO entry with a single key press. The commands available in the TODO list are described in Section 8.8 [Agenda commands], page 44.

Normally the global todo list simply shows all headlines with TODO keywords. This list can become very long. There are two ways to keep it more compact:

- Some people view a TODO item that has been *scheduled* for execution (see Section 6.1 [Time stamps], page 30) as no longer *open*. Configure the variable `org-agenda-todo-ignore-scheduled` to exclude scheduled items from the global TODO list.
- TODO items may have sublevels to break up the task into subtasks. In such cases it may be enough to list only the highest level TODO headline and omit the sublevels from the global list. Configure the variable `org-agenda-todo-list-sublevels` to get this behavior.

## 8.5 Matching headline tags

If headlines in the agenda files are marked with *tags* (see Chapter 7 [Tags], page 36), you can select headlines based on the tags that apply to them and collect them into an agenda buffer.

`C-c a m` Produce a list of all headlines that match a given set of tags. The command prompts for a selection criterion, which is a boolean logic expression with tags, like '`+WORK+URGENT-WITHBOSS`' or '`WORK|HOME`' (see Chapter 7 [Tags], page 36). If you often need a specific search, define a custom command for it (see Section 8.2 [Agenda dispatcher], page 40).

`C-c a M` Like `C-c a m`, but only select headlines that are also TODO items and force checking subitems (see variable `org-tags-match-list-sublevels`). Matching specific todo keywords together with a tags match is also possible, see Section 7.3 [Tag searches], page 38.

The commands available in the tags list are described in Section 8.8 [Agenda commands], page 44.

## 8.6 Timeline for a single file

The timeline summarizes all time-stamped items from a single Org-mode file in a *time-sorted view*. The main purpose of this command is to give an overview over events in a project.

`C-c a L` Show a time-sorted view of the org file, with all time-stamped items. When called with a `C-u` prefix, all unfinished TODO entries (scheduled or not) are also listed under the current date.

The commands available in the timeline buffer are listed in Section 8.8 [Agenda commands], page 44.

## 8.7 Presentation and sorting

Before displaying items in an agenda view, Org-mode visually prepares the items and sorts them. Each item occupies a single line. The line starts with a *prefix* that contains the *category* (see Section 8.7.1 [Categories], page 42) of the item and other important information. You can customize the prefix using the option `org-agenda-prefix-format`. The prefix is followed by a cleaned-up version of the outline headline associated with the item.

### 8.7.1 Categories

The category is a broad label assigned to each agenda item. By default, the category is simply derived from the file name, but you can also specify it with a special line in the buffer, like this:

```
#+CATEGORY: Thesis
```

If there are several such lines in a file, each specifies the category for the text below it (but the first category also applies to any text before the first CATEGORY line). The display in the agenda buffer looks best if the category is not longer than 10 characters.

### 8.7.2 Time-of-Day Specifications

Org-mode checks each agenda item for a time-of-day specification. The time can be part of the time stamp that triggered inclusion into the agenda, for example as in '`<2005-05-10 Tue 19:00>`'. Time ranges can be specified with two time stamps, like '`<2005-05-10 Tue 20:30>--<2005-05-10 Tue 22:15>`'.

In the headline of the entry itself, a time(range) may also appear as plain text (like '`12:45`' or a '`8:30-1pm`'. If the agenda integrates the Emacs diary (see Section 8.3.1 [Calendar/Diary integration], page 40), time specifications in diary entries are recognized as well.

For agenda display, Org-mode extracts the time and displays it in a standard 24 hour format as part of the prefix. The example times in the previous paragraphs would end up in the agenda like this:

```
 8:30-13:00 Arthur Dent lies in front of the bulldozer
12:45...... Ford Prefect arrives and takes Arthur to the pub
19:00...... The Vogon reads his poem
20:30-22:15 Marwin escorts the Hitchhikers to the bridge
```

If the agenda is in single-day mode, or for the display of today, the timed entries are embedded in a time grid, like

```
 8:00...... ------------------
 8:30-13:00 Arthur Dent lies in front of the bulldozer
10:00...... ------------------
12:00...... ------------------
12:45...... Ford Prefect arrives and takes Arthur to the pub
14:00...... ------------------
16:00...... ------------------
18:00...... ------------------
19:00...... The Vogon reads his poem
20:00...... ------------------
20:30-22:15 Marwin escorts the Hitchhikers to the bridge
```

The time grid can be turned on and off with the variable `org-agenda-use-time-grid`, and can be configured with `org-agenda-time-grid`.

### 8.7.3 Sorting of agenda items

Before being inserted into a view, the items are sorted. How this is done depends on the type of view.

- For the daily/weekly agenda, the items for each day are sorted. The default order is to first collect all items containing an explicit time-of-day specification. These entries will be shown at the beginning of the list, as a *schedule* for the day. After that, items

remain grouped in categories, in the sequence given by `org-agenda-files`. Within each category, items are sorted by priority (see Section 5.3 [Priorities], page 28), which is composed of the base priority (2000 for priority '`A`', 1000 for '`B`', and 0 for '`C`'), plus additional increments for overdue scheduled or deadline items.

- For the TODO list, items remain in the order of categories, but within each category, sorting takes place according to priority (see Section 5.3 [Priorities], page 28).
- For tags matches, items are not sorted at all, but just appear in the sequence in which they are found in the agenda files.

Sorting can be customized using the variable `org-agenda-sorting-strategy`.

## 8.8 Commands in the agenda buffer

Entries in the agenda buffer are linked back to the org file or diary file where they originate. You are not allowed to edit the agenda buffer itself, but commands are provided to show and jump to the original entry location, and to edit the org-files "remotely" from the agenda buffer. In this way, all information is stored only once, removing the risk that your agenda and note files may diverge.

Some commands can be executed with mouse clicks on agenda lines. For the other commands, the cursor needs to be in the desired line.

**Motion**

*n* Next line (same as ⟨up⟩).

*p* Previous line (same as ⟨down⟩).

**View/GoTo org file**

*mouse-3*

⟨SPC⟩ Display the original location of the item in another window.

*L* Display original location and recenter that window.

*mouse-2*

*mouse-1*

⟨TAB⟩ Go to the original location of the item in another window. Under Emacs 22, *mouse-1* will also works for this.

⟨RET⟩ Go to the original location of the item and delete other windows.

*f* Toggle Follow mode. In Follow mode, as you move the cursor through the agenda buffer, the other window always shows the corresponding location in the org file. The initial setting for this mode in new agenda buffers can be set with the variable `org-agenda-start-with-follow-mode`.

*l* Toggle Logbook mode. In Logbook mode, entries that where marked DONE while logging was on (variable `org-log-done`) are shown in the agenda, as are entries that have been clocked on that day.

**Change display**

*o* Delete other windows.

*w* Switch to weekly view (7 days displayed together).

`d` Switch to daily view (just one day displayed).

`D` Toggle the inclusion of diary entries. See Section 8.3.1 [Calendar/Diary integration], page 40.

`g` Toggle the time grid on and off. See also the variables `org-agenda-use-time-grid` and `org-agenda-time-grid`.

`r` Recreate the agenda buffer, for example to reflect the changes after modification of the time stamps of items with S-⟨left⟩ and S-⟨right⟩. When the buffer is the global todo list, a prefix argument is interpreted to create a selective list for a specific TODO keyword.

`s` Save all Org-mode buffers in the current Emacs session.

⟨right⟩ Display the following `org-agenda-ndays` days. For example, if the display covers a week, switch to the following week. With prefix arg, go forward that many times `org-agenda-ndays` days.

⟨left⟩ Display the previous dates.

`.` Goto today.

**Remote editing**

`0-9` Digit argument.

`t` Change the TODO state of the item, both in the agenda and in the original org file.

`T` Show all tags associated with the current item. Because of inheritance, this may be more than the tags listed in the line itself.

`:` Set tags for the current headline.

`a` Toggle the ARCHIVE tag for the current headline.

`,` Set the priority for the current item. Org-mode prompts for the priority character. If you reply with ⟨SPC⟩, the priority cookie is removed from the entry.

`p` Display weighted priority of current item.

`+`
`S-`⟨up⟩ Increase the priority of the current item. The priority is changed in the original buffer, but the agenda is not resorted. Use the `r` key for this.

`-`
`S-`⟨down⟩ Decrease the priority of the current item.

`C-c C-s` Schedule this item

`C-c C-d` Set a deadline for this item.

`S-`⟨right⟩ Change the time stamp associated with the current line by one day into the future. With prefix argument, change it by that many days. For example, `3 6 5 S-`⟨right⟩ will change it by a year. The stamp is changed in the original org file, but the change is not directly reflected in the agenda buffer. Use the `r` key to update the buffer.

`S-⟨left⟩` Change the time stamp associated with the current line by one day into the past.

`>` Change the time stamp associated with the current line to today. The key `>` has been chosen, because it is the same as `S-.` on my keyboard.

`I` Start the clock on the current item. If a clock is running already, it is stopped first.

`O` Stop the previously started clock.

`X` Cancel the currently running clock.

**Calendar commands**

`c` Open the Emacs calendar and move to the date at the agenda cursor.

`c` When in the calendar, compute and show the Org-mode agenda for the date at the cursor.

`i` Insert a new entry into the diary. Prompts for the type of entry (day, weekly, monthly, yearly, anniversary, cyclic) and creates a new entry in the diary, just as `i d` etc. would do in the calendar. The date is taken from the cursor position.

`M` Show the phases of the moon for the three months around current date.

`S` Show sunrise and sunset times. The geographical location must be set with calendar variables, see documentation of the Emacs calendar.

`C` Convert the date at cursor into many other cultural and historic calendars.

`H` Show holidays for three month around the cursor date.

`C-c C-x C-c`
Export a single iCalendar file containing entries from all agenda files.

**Quit and Exit**

`q` Quit agenda, remove the agenda buffer.

`x` Exit agenda, remove the agenda buffer and all buffers loaded by Emacs for the compilation of the agenda. Buffers created by the user to visit org files will not be removed.

## 8.9 Custom agenda views

Custom agenda commands serve two purposes: to store and quickly access frequently used TODO and tags searches, and to create special composite agenda buffers. Custom agenda commands will be accessible through the dispatcher (see Section 8.2 [Agenda dispatcher], page 40), just like the default commands.

### 8.9.1 Storing searches

The first application of custom searches is the definition of keyboard shortcuts for frequently used searches, either creating an agenda buffer, or a sparse tree (the latter covering

of course only the current buffer). Custom commands are configured in the variable `org-agenda-custom-commands`. You can customize this variable, for example by pressing `C-c a C`. You can also directly set it with Emacs Lisp in '`.emacs`'. The following example contains all valid search types:

```
(setq org-agenda-custom-commands
      '(("w" todo "WAITING")
        ("W" todo-tree "WAITING")
        ("u" tags "+BOSS-URGENT")
        ("v" tags-todo "+BOSS-URGENT")
        ("U" tags-tree "+BOSS-URGENT")
        ("f" occur-tree "\\<FIXME\\>")))
```

The initial single-character string in each entry defines the character you have to press after the dispatcher command `C-c a` in order to access the command. The second parameter is the search type, followed by the string or regular expression to be used for the matching. The example above will therefore define:

`C-c a w` as a global search for TODO entries with '`WAITING`' as the TODO keyword

`C-c a W` as the same search, but only in the current buffer and displaying the results as a sparse tree

`C-c a u` as a global tags search for headlines marked '`:BOSS:`' but not '`:URGENT:`'

`C-c a v` as the same search as `C-c a u`, but limiting the search to headlines that are also TODO items

`C-c a U` as the same search as `C-c a u`, but only in the current buffer and displaying the result as a sparse tree

`C-c a f` to create a sparse tree (again: current buffer only) with all entries containing the word '`FIXME`'.

### 8.9.2 Block agenda

Another possibility is the construction of agenda views that comprise the results of *several* commands, each of which creates a block in the agenda buffer. The available commands include `agenda` for the daily or weekly agenda (as created with `C-c a a`), `alltodo` for the global todo list (as constructed with `C-c a t`), and the matching commands discussed above: `todo`, `tags`, and `tags-todo`. Here are two examples:

```
(setq org-agenda-custom-commands
      '(("h" "Agenda and Home-related tasks"
         ((agenda)
          (tags-todo "HOME")
          (tags "GARDEN")))
        ("o" "Agenda and Office-related tasks"
         ((agenda)
          (tags-todo "WORK")
          (tags "OFFICE")))))
```

This will define `C-c a h` to create a multi-block view for stuff you need to attend to at home. The resulting agenda buffer will contain your agenda for the current week, all TODO items

that carry the tag 'HOME', and also all lines tagged with 'GARDEN'. Finally the command *C-c a o* provides a similar view for office tasks.

### 8.9.3 Setting Options for custom commands

Org-mode contains a number of variables regulating agenda construction and display. The global variables define the behavior for all agenda commands, including the custom commands. However, if you want to change some settings just for a single custom view, you can do so. Setting options requires inserting a list of variable names and values at the right spot in `org-agenda-custom-commands`. For example:

```
(setq org-agenda-custom-commands
      '(("w" todo "WAITING"
         ((org-agenda-sorting-strategy '(priority-down))
          (org-agenda-prefix-format "  Mixed: ")))
        ("U" tags-tree "+BOSS-URGENT"
         ((org-show-following-heading nil)
          (org-show-hierarchy-above nil)))))
```

Now the *C-c a w* command will sort the collected entries only by priority, and the prefix format is modified to just say ' `Mixed:`' instead of giving the category of the entry. The sparse tags tree of *C-c a U* will now turn out ultra-compact, because neither the headline hierarchy above the match, nor the headline following the match will be shown.

For command sets creating a block agenda, `org-agenda-custom-commands` has two separate spots for setting options. You can add options that should be valid for just a single command in the set, and options that should be valid for all commands in the set. The former are just added to the command entry, the latter must come after the list of command entries. Going back to the block agenda example (see Section 8.9.2 [Block agenda], page 47), let's change the sorting strategy for the *C-c a h* commands to `priority-down`, but let's sort the results for GARDEN tags query in the opposite order, `priority-up`. This would look like this:

```
(setq org-agenda-custom-commands
      '(("h" "Agenda and Home-related tasks"
         ((agenda)
          (tags-todo "HOME")
          (tags "GARDEN" ((org-agenda-sorting-strategy '(priority-up)))))
         ((org-agenda-sorting-strategy '(priority-down))))
        ("o" "Agenda and Office-related tasks"
         ((agenda)
          (tags-todo "WORK")
          (tags "OFFICE")))))
```

As you see, the values and parenthesis setting is a little complex. When in doubt, use the customize interface to set this variable - it fully supports its structure. Just one caveat: When setting options in this interface, the *values* are just lisp expressions. So if the value is a string, you need to add the double quotes around the value yourself.

### 8.9.4 Creating agenda views in batch processing

If you want to print or otherwise reprocess agenda views, it can be useful to create an agenda from the command line. This is the purpose of the function `org-batch-agenda`. It takes as a parameter one of the strings that are the keys in `org-agenda-custom-commands`. For example, to directly print the current TODO list, you could use

```
emacs -batch -l ~/.emacs -eval '(org-batch-agenda "t")' | lpr
```

You may also modify parameters on the fly like this:

```
emacs -batch -l ~/.emacs                                      \
   -eval '(org-batch-agenda "a"                               \
            org-agenda-ndays 300                              \
            org-agenda-include-diary nil                      \
            org-agenda-files (quote ("~/org/project.org")))'  \
   | lpr
```

which will produce a 300 day agenda, fully restricted to the Org file '`~/org/projects.org`', not even including the diary.

# 9 Embedded LaTeX

Plain ASCII is normally sufficient for almost all note taking. One exception, however, are scientific notes which need to be able to contain mathematical symbols and the occasional formula. LaTeX[1] is widely used to typeset scientific documents. Org-mode supports embedding LaTeX code into its files, because many academics are used to read LaTeX source code, and because it can be readily processed into images for HTML production.

It is not necessary to mark LaTeX macros and code in any special way. If you observe a few conventions, Org-mode knows how to find it and what to do with it.

## 9.1 Math symbols

You can use LaTeX macros to insert special symbols like '`\alpha`' to indicate the Greek letter, or '`\to`' to indicate an arrow. Completion for these macros is available, just type '\' and maybe a few letters, and press *M-*⟨TAB⟩ to see possible completions. Unlike LaTeX code, Org-mode allows these macros to be present without surrounding math delimiters, for example:

```
Angles are written as Greek letters \alpha, \beta and \gamma.
```

During HTML export (see Section 10.2 [HTML export], page 53), these symbols are translated into the proper syntax for HTML, for the above examples this is '`&alpha;`' and '`&rarr;`', respectively.

## 9.2 Subscripts and Superscripts

Just like in LaTeX, '`^`' and '`_`' are used to indicate super- and subscripts. Again, these can be used without embedding them in math-mode delimiters. To increase the readability of ASCII text, it is not necessary (but OK) to surround multi-character sub- and superscripts with curly braces. For example

```
The mass if the sun is M_sun = 1.989 x 10^30 kg.  The radius of
the sun is R_{sun} = 6.96 x 10^8 m.
```

To avoid interpretation as raised or lowered text, you can quote '`^`' and '`_`' with a backslash: '`\_`' and '`\^`'.

During HTML export (see Section 10.2 [HTML export], page 53), subscript and superscripts are surrounded with `<sub>` and `<sup>` tags, respectively.

## 9.3 LaTeX fragments

With symbols, sub- and superscripts, HTML is pretty much at its end when it comes to representing mathematical formulas[2]. More complex expressions need a dedicated formula

[1] LaTeX is a macro system based on Donald E. Knuth's TeX system. Many of the features described here as "LaTeX" are really from TeX, but for simplicity I am blurring this distinction.

[2] Yes, there is MathML, but that is not yet fully supported by many browsers, and there is no decent converter for turning LaTeX of ASCII representations of formulas into MathML. So for the time being, converting formulas into images seems the way to go.

processor. To this end, Org-mode can contain arbitrary LaTeX fragments. It provides commands to preview the typeset result of these fragments, and upon export to HTML, all fragments will be converted to images and inlined into the HTML document. For this to work you need to be on a system with a working LaTeX installation. You also need the '`dvipng`' program, available at `http://sourceforge.net/projects/dvipng/`.

LaTeX fragments don't need any special marking at all. The following snippets will be identified as LaTeX source code:

- Environments of any kind. The only requirement is that the `\begin` statement appears on a new line, preceded by only whitespace.
- Text within the usual LaTeX math delimiters. To avoid conflicts with currency specifications, single '`$`' characters are only recognized as math delimiters if the enclosed text contains at most two line breaks, is directly attached to the '`$`' characters with no whitespace in between, and if the closing '`$`' is followed by whitespace or punctuation. For the other delimiters, there is no such restriction, so when in doubt, use '`\(...\)`' as inline math delimiters.

For example:

```
\begin{equation}                          % arbitrary environments,
x=\sqrt{b}                                % even tables, figures
\end{equation}                            % etc

If $a^2=b$ and \( b=2 \), then the solution must be
either $$ a=+\sqrt{2} $$ or \[ a=-\sqrt{2} \].
```

If you need any of the delimiter ASCII sequences for other purposes, you can configure the option `org-format-latex-options` to deselect the ones you do not wish to have interpreted by the LaTeX converter.

## 9.4 Processing LaTeX fragments

LaTeX fragments can be processed to produce a preview images of the typeset expressions:

*C-c C-x C-l*
: Produce a preview image of the LaTeX fragment at point and overlay it over the source code. If there is no fragment at point, process all fragments in the current entry (between two headlines). When called with a prefix argument, process the entire subtree. When called with two prefix arguments, or when the cursor is before the first headline, process the entire buffer.

*C-c C-c*
: Remove the overlay preview images.

During HTML export (see Section 10.2 [HTML export], page 53), all LaTeX fragments are converted into images and inlined into the document if the following setting is active:

```
(setq org-export-with-LaTeX-fragments t)
```

## 9.5 Using CDLaTeX to enter math

CDLaTeX-mode is a minor mode that is normally used in combination with a major LaTeX mode like AUCTeX in order to speed-up insertion of environments and math templates. Inside Org-mode, you can make use of some of the features of cdlatex-mode. You need to install '`cdlatex.el`' and '`texmathp.el`' (the latter comes also with AUCTeX) from `http://www.astro.uva.nl/~dominik/Tools/cdlatex`. Don't turn cdlatex-mode itself under Org-mode, but use the light version `org-cdlatex-mode` that comes as part of Org-mode. Turn it on for the current buffer with `M-x org-cdlatex-mode`, or for all Org-mode files with

```
(add-hook 'org-mode-hook 'turn-on-org-cdlatex)
```

When this mode is enabled, the following features are present (for more details see the documentation of cdlatex-mode):

- Environment templates can be inserted with *`C-c {`*.
- The ⟨TAB⟩ key will do template expansion if the cursor is inside a LaTeX fragment[3]. For example, ⟨TAB⟩ will expand `fr` to `\frac{}{}` and position the cursor correctly inside the first brace. Another ⟨TAB⟩ will get you into the second brace. Even outside fragments, ⟨TAB⟩ will expand environment abbreviations at the beginning of a line. For example, if you write '`equ`' at the beginning of a line and press ⟨TAB⟩, this abbreviation will be expanded to an `equation` environment. To get a list of all abbreviations, type *`M-x cdlatex-command-help`*.
- Pressing `_` and `^` inside a LaTeX fragment will insert these characters together with a pair of braces. If you use ⟨TAB⟩ to move out of the braces, and if the braces surround only a single character or macro, they are removed again (depending on the variable `cdlatex-simplify-sub-super-scripts`).
- Pressing the backquote `‘` followed by a character inserts math macros, also outside LaTeX fragments. If you wait more than 1.5 seconds after the backquote, a help window will pop up.
- Pressing the normal quote `’` followed by another character modifies the symbol before point with an accent or a font. If you wait more than 1.5 seconds after the backquote, a help window will pop up. Character modification will work only inside LaTeX fragments, outside the quote is normal.

[3] Org-mode has a method to test if the cursor is inside such a fragment, see the documentation of the function `org-inside-LaTeX-fragment-p`.

# 10 Exporting

Org-mode documents can be exported into a variety of other formats. For printing and sharing of notes, ASCII export produces a readable and simple version of an Org-mode file. HTML export allows you to publish a notes file on the web, while the XOXO format provides a solid base for exchange with a broad range of other applications. To incorporate entries with associated times like deadlines or appointments into a desktop calendar program like iCal, Org-mode can also produce extracts in the iCalendar format. Currently Org-mode only supports export, not import of these different formats.

When exporting, Org-mode uses special conventions to enrich the output produced. See Section 10.5 [Text interpretation], page 55, for more details.

`C-c C-e` Dispatcher for export and publishing commands. Displays a help-window listing the additional key(s) needed to launch an export or publishing command.

## 10.1 ASCII export

ASCII export produces a simple and very readable version of an Org-mode file.

`C-c C-e a` Export as ASCII file. If there is an active region, only the region will be exported. For an org file '`myfile.org`', the ASCII file will be '`myfile.txt`'. The file will be overwritten without warning.

`C-c C-e v a`
Export only the visible part of the document.

In the exported version, the first 3 outline levels will become headlines, defining a general document structure. Additional levels will be exported as itemized lists. If you want that transition to occur at a different level, specify it with a prefix argument. For example,

```
C-1 C-c C-e a
```

creates only top level headlines and does the rest as items. When headlines are converted to items, the indentation of the text following the headline is changed to fit nicely under the item. This is done with the assumption that the first bodyline indicates the base indentation of the body text. Any indentation larger than this is adjusted to preserve the layout relative to the first line. Should there be lines with less indentation than the first, these are left alone.

## 10.2 HTML export

Org-mode contains an HTML (XHTML 1.0 strict) exporter with extensive HTML formatting, in ways similar to John Grubers *markdown* language, but with additional support for tables.

`C-c C-e h` Export as HTML file '`myfile.html`'.

`C-c C-e b` Export as HTML file and open it with a browser.

*C-c C-e v h*
*C-c C-e v b*
: Export only the visible part of the document.

In the exported version, the first 3 outline levels will become headlines, defining a general document structure. Additional levels will be exported as itemized lists. If you want that transition to occur at a different level, specify it with a prefix argument. For example,

```
C-2 C-c C-e b
```

creates two levels of headings and does the rest as items.

If you want to include HTML tags which should be interpreted as such, mark them with '`@`' as in '`@<b>bold text@</b>`'. Plain '`<`' and '`>`' are always transformed to '`<`' and '`>`' in HTML export.

Internal links (see Section 4.2 [Internal links], page 19) will continue to work in HTML files only if they match a dedicated '`<<target>>`'. Automatic links created by radio targets (see Section 4.2.1 [Radio targets], page 20) will also work in the HTML file. Links to external files will still work if the HTML file is in the same directory as the Org-mode file. Links to other '`.org`' files will be translated into HTML links under the assumption that an HTML version also exists of the linked file. For information related to linking files while publishing them to a publishing directory see Section 11.1.6 [Publishing links], page 60.

You can also give style information for the exported file. The HTML exporter assigns the following CSS classes to appropriate parts of the document - your style specifications may change these:

| | |
|---|---|
| `.todo` | TODO keywords |
| `.done` | the DONE keyword |
| `.timestamp` | time stamp |
| `.timestamp-kwd` | keyword associated with a time stamp, like SCHEDULED |
| `.tag` | tag in a headline |
| `.target` | target for links |

The default style specification can be configured through the option `org-export-html-style`. If you want to use a file-local style, you may use file variables, best wrapped into a COMMENT section at the end of the outline tree. For example:

```
* COMMENT HTML style specifications

# Local Variables:
# org-export-html-style: "   <style type=\"text/css\">
#       p {font-weight: normal; color: gray; }
#       h1 {color: black; }
#   </style>"
# End:
```

Remember to execute *M-x normal-mode* after changing this to make the new style visible to Emacs. This command restarts org-mode for the current buffer and forces Emacs to re-evaluate the local variables section in the buffer.

## 10.3 XOXO export

Org-mode contains an exporter that produces XOXO-style output. Currently, this exporter only handles the general outline structure and does not interpret any additional Org-mode features.

`C-c C-e x` Export as XOXO file '`myfile.html`'.

`C-c C-e v x`
: Export only the visible part of the document.

## 10.4 iCalendar export

Some people like to use Org-mode for keeping track of projects, but still prefer a standard calendar application for anniversaries and appointments. In this case it can be useful to have deadlines and other time-stamped items in Org-mode files show up in the calendar application. Org-mode can export calendar information in the standard iCalendar format.

`C-c C-e i` Create iCalendar entries for the current file and store them in the same directory, using a file extension '`.ics`'.

`C-c C-e I` Like `C-c C-e i`, but do this for all files in `org-agenda-files`. For each of these files, a separate iCalendar file will be written.

`C-c C-e c` Create a single large iCalendar file from all files in `org-agenda-files` and write it to the file given by `org-combined-agenda-icalendar-file`.

How this calendar is best read and updated, depends on the application you are using. For example, when using iCal under Apple MacOS X, you could create a new calendar '`OrgMode`' (the default name for the calendar created by `C-c C-e c`, see the variables `org-icalendar-combined-name` and `org-combined-agenda-icalendar-file`). Then set Org-mode to overwrite the corresponding file '`~/Library/Calendars/OrgMode.ics`'. You may even use AppleScript to make iCal re-read the calendar files each time a new version of '`OrgMode.ics`' is produced. Here is the setup needed for this:

```
(setq org-combined-agenda-icalendar-file
    "~/Library/Calendars/OrgMode.ics")
(add-hook 'org-after-save-iCalendar-file-hook
 (lambda ()
  (shell-command
   "osascript -e 'tell application \"iCal\" to reload calendars'")))
```

## 10.5 Text interpretation by the exporter

The exporter backends interpret additional structure in the Org-mode file in order to produce better output.

### 10.5.1 Comment lines

Lines starting with '`#`' in column zero are treated as comments and will never be exported. Also entire subtrees starting with the word '`COMMENT`' will never be exported. Finally, any text before the first headline will not be exported either.

`C-c ;` Toggle the COMMENT keyword at the beginning of an entry.

### 10.5.2 Enhancing text for export

Some of the export backends of Org-mode allow for sophisticated text formatting, this is true in particular for the HTML backend. Org-mode has a number of typing conventions that allow to produce a richly formatted output.

- Plain lists '`-`', '`*`' or '`+`' as bullet, or with '`1.`' or '`2)`' as enumerator will be recognized and transformed if the backend supports lists. See See Section 2.8 [Plain lists], page 8.
- You can make words ***bold***, */italic/*, _underlined_, `=code=`, and '`+strikethrough+`'.
- Many TEX macros and entire LATEX fragments are converted into HTML entities or images (see Chapter 9 [Embedded LaTeX], page 50).
- Tables are transformed into native tables under the exporter, if the export backend supports this. Data fields before the first horizontal separator line will be formatted as table header fields.
- If a headline starts with the word '`QUOTE`', the text below the headline will be typeset as fixed-width, to allow quoting of computer codes etc. Lines starting with '`:`' are also typeset in fixed-width font.

  `C-c :` Toggle fixed-width for entry (QUOTE) or region, see below.
- A double backslash *at the end of a line* enforces a line break at this position.

If these conversions conflict with your habits of typing ASCII text, they can all be turned off with corresponding variables (see the customization group `org-export-general`, and the following section which explains how to set export options with special lines in a buffer.

### 10.5.3 Export options

The exporter recognizes special lines in the buffer which provide additional information. These lines may be put anywhere in the file. The whole set of lines can be inserted into the buffer with `C-c C-e t`. For individual lines, a good way to make sure the keyword is correct is to type '`#+`' and then use `M-`⟨TAB⟩ completion (see Section 12.1 [Completion], page 63).

`C-c C-e t` Insert template with export options, see example below.

```
#+TITLE:     the title to be shown (default is the buffer name)
#+AUTHOR:    the author (default taken from user-full-name)
#+EMAIL:     his/her email address (default from user-mail-address)
#+LANGUAGE:  language for HTML, e.g. 'en' (org-export-default-language)
#+TEXT:      Some descriptive text to be inserted at the beginning.
#+TEXT:      Several lines may be given.
```

```
#+OPTIONS:   H:2 num:t toc:t \n:nil @:t ::t |:t ^:t *:nil TeX:t LaTeX:t
```

The OPTIONS line is a compact form to specify export settings. Here you can:

| | |
|---|---|
| `H:` | set the number of headline levels for export |
| `num:` | turn on/off section-numbers |
| `toc:` | turn on/off table of contents |
| `\n:` | turn on/off linebreak-preservation |
| `@:` | turn on/off quoted HTML tags |
| `::` | turn on/off fixed-width sections |
| `\|:` | turn on/off tables |
| `^:` | turn on/off TEX-like syntax for sub- and superscripts. |
| `*:` | turn on/off emphasized text (bold, italic, underlined) |
| `TeX:` | turn on/off simple TEX macros in plain text |
| `LaTeX:` | turn on/off LATEX fragments |

# 11 Publishing

Org-mode includes[1] a publishing management system that allows you to configure automatic HTML conversion of *projects* composed of interlinked org files. This system is called *org-publish*. You can also configure org-publish to automatically upload your exported HTML pages and related attachments, such as images and source code files, to a web server. Org-publish turns org-mode into a web-site authoring tool.

Org-publish has been contributed to Org-mode by David O'Toole.

## 11.1 Configuration

Publishing needs significant configuration to specify files, destination and many other properties of a project.

### 11.1.1 The variable `org-publish-project-alist`

Org-publish is configured almost entirely through setting the value of one variable, called `org-publish-project-alist`. Each element of the list configures one project, and may be in one of the two following forms:

```
("project-name"  :property value :property value ...)
```

or

```
("project-name"  :components ("project-name" "project-name" ...))
```

In both cases, projects are configured by specifying property values. A project defines the set of files that will be published, as well as the publishing configuration to use when publishing those files. When a project takes the second form listed above, the individual members of the "components" property are taken to be components of the project, which group together files requiring different publishing options. When you publish such a "meta-project" all the components will also publish.

### 11.1.2 Sources and destinations for files

Most properties are optional, but some should always be set. In particular, org-publish needs to know where to look for source files, and where to put published files.

| | |
|---|---|
| `:base-directory` | Directory containing publishing source files |
| `:publishing-directory` | Directory (possibly remote) where output files will be published. |
| `:preparation-function` | Function called before starting publishing process, for example to run `make` for updating files to be published. |

[1] '`org-publish.el`' is not yet part of Emacs, so if you are using '`org.el`' as it comes with Emacs, you need to download this file separately. Also make sure org.el is at least version 4.27.

### 11.1.3 Selecting files

By default, all files with extension '`.org`' in the base directory are considered part of the project. This can be modified by setting the properties

`:base-extension` Extension (without the dot!) of source files. This actually is a regular expression.

`:exclude` Regular expression to match file names that should not be published, even though they have been selected on the basis of their extension.

`:include` List of files to be included regardless of `:base-extension` and `:exclude`.

### 11.1.4 Publishing Action

Publishing means that a file is copied to the destination directory and possibly transformed in the process. The default transformation is to export Org-mode files as HTML files, and this is done by the function `org-publish-org-to-html` which calls the HTML exporter (see Section 10.2 [HTML export], page 53). Other files like images only need to be copied to the publishing destination. For non-Org-mode files, you need to specify the publishing function.

`:publishing-function` Function executing the publication of a file. This may also be a list of functions, which will all be called in turn.

The function must accept two arguments: a property list containing at least a `:publishing-directory` property, and the name of the file to be published. It should take the specified file, make the necessary transformation (if any) and place the result into the destination folder. You can write your own publishing function, but `org-publish` provides one for attachments (files that only need to be copied): `org-publish-attachment`.

### 11.1.5 Options for the HTML exporter

The property list can be used to set many export options for the HTML exporter. In most cases, these properties correspond to user variables in Org-mode. The table below lists these properties along with the variable they belong to. See the documentation string for the respective variable for details.

| | |
|---|---|
| `:language` | `org-export-default-language` |
| `:headline-levels` | `org-export-headline-levels` |
| `:section-numbers` | `org-export-with-section-numbers` |
| `:table-of-contents` | `org-export-with-toc` |
| `:archived-trees` | `org-export-with-archived-trees` |
| `:emphasize` | `org-export-with-emphasize` |
| `:sub-superscript` | `org-export-with-sub-superscripts` |
| `:TeX-macros` | `org-export-with-TeX-macros` |
| `:LaTeX-fragments` | `org-export-with-LaTeX-fragments` |
| `:fixed-width` | `org-export-with-fixed-width` |
| `:timestamps .` | `org-export-with-timestamps` |

```
:tags .                  org-export-with-tags
:tables                  org-export-with-tables
:table-auto-headline     org-export-highlight-first-table-line
:style                   org-export-html-style
:convert-org-links       org-export-html-link-org-files-as-html
:inline-images           org-export-html-inline-images
:expand-quoted-html      org-export-html-expand
:timestamp               org-export-html-with-timestamp
:publishing-directory    org-export-publishing-directory
:preamble                org-export-html-preamble
:postamble               org-export-html-postamble
:auto-preamble           org-export-html-auto-preamble
:auto-postamble          org-export-html-auto-postamble
:author                  user-full-name
:email                   user-mail-address
```

When a property is given a value in org-publish-project-alist, its setting overrides the value of the corresponding user variable (if any) during publishing. options set within a file (see Section 10.5.3 [Export options], page 56), however, override everything.

### 11.1.6 Links between published files

To create a link from one Org-mode file to another, you would use something like '`[[file:foo.org][The foo]]`' or simply '`file:foo.org.`' (see Chapter 4 [Hyperlinks], page 19). Upon publishing this link becomes a link to '`foo.html`'. In this way, you can interlink the pages of your "org web" project and the links will work as expected when you publish them to HTML.

You may also link to related files, such as images. Provided you are careful with relative pathnames, and provided you have also configured org-publish to upload the related files, these links will work too. Section 11.2.2 [Complex example], page 61 for an example of this usage.

Sometime an Org-mode file to be published may contain links that are only valid in your production environment, but not in the publishing location. In this case, use the property

`:link-validation-function` Function to validate links

to define a function for checking link validity. This function must accept two arguments, the file name and a directory relative to which the file name is interpreted in the production environment. If this function returns `nil`, then the HTML generator will only insert a description into the HTML file, but no link. One option for this function is `org-publish-validate-link` which checks if the given file is part of any project in `org-publish-project-alist`.

### 11.1.7 Project page index

The following properties may be used to control publishing of an index of files or summary page for a given project.

| | |
|---|---|
| `:auto-index` | When non-nil, publish an index during org-publish-current-project or org-publish-all. |
| `:index-filename` | Filename for output of index. Defaults to '`index.org`' (which becomes '`index.html`'). |
| `:index-title` | Title of index page. Defaults to name of file. |
| `:index-function` | Plugin function to use for generation of index. Defaults to `org-publish-org-index`, which generates a plain list of links to all files in the project. |

## 11.2 Sample configuration

Below we provide two example configurations. The first one is a simple project publishing only a set of Org-mode files. The second example is more complex, with a multi-component project.

### 11.2.1 Example: simple publishing configuration

This example publishes a set of Org-mode files to the '`public_html`' directory on the local machine.

```
(setq org-publish-project-alist
      '(("org"
         :base-directory "~/org/"
         :publishing-directory "~/public_html"
         :section-numbers nil
         :table-of-contents nil
         :style "<link rel=stylesheet
                href=\"../other/mystyle.css\"
                type=\"text/css\">")))
```

### 11.2.2 Example: complex publishing configuration

This more complicated example publishes an entire website, including org files converted to HTML, image files, emacs lisp source code, and stylesheets. The publishing-directory is remote and private files are excluded.

To ensure that links are preserved, care should be taken to replicate your directory structure on the web server, and to use relative file paths. For example, if your org files are kept in '`~/org`' and your publishable images in '`~/images`', you'd link to an image with

```
file:../images/myimage.png
```

On the web server, the relative path to the image should be the same. You can accomplish this by setting up an "images" folder in the right place on the webserver, and publishing images to it.

```
(setq org-publish-project-alist
      '(("orgfiles"
```

```
 :base-directory "~/org/"
 :base-extension "org"
 :publishing-directory "/ssh:user@host:~/html/notebook/"
 :publishing-function org-publish-org-to-html
 :exclude "PrivatePage.org"   ;; regexp
 :headline-levels 3
 :section-numbers nil
 :table-of-contents nil
 :style "<link rel=stylesheet
         href=\"../other/mystyle.css\" type=\"text/css\">"
 :auto-preamble t
 :auto-postamble nil)

("images"
 :base-directory "~/images/"
 :base-extension "jpg\\|gif\\|png"
 :publishing-directory "/ssh:user@host:~/html/images/"
 :publishing-function org-publish-attachment)

("other"
 :base-directory "~/other/"
 :base-extension "css\\|el"
 :publishing-directory "/ssh:user@host:~/html/other/"
 :publishing-function org-publish-attachment)
("website" :components ("orgfiles" "images" "other"))))
```

## 11.3 Triggering publication

Once org-publish is properly configured, you can publish with the following functions:

`C-c C-e c` Prompt for a specific project and publish all files that belong to it.

`C-c C-e p` Publish the project containing the current file.

`C-c C-e f` Publish only the current file.

`C-c C-e a` Publish all projects.

Org uses timestamps to track when a file has changed. The above functions normally only publish changed files. You can override this and force publishing of all files by giving a prefix argument.

# 12 Miscellaneous

## 12.1 Completion

Org-mode supports in-buffer completion. This type of completion does not make use of the minibuffer. You simply type a few letters into the buffer and use the key to complete text right there.

*M-*⟨TAB⟩ Complete word at point

- At the beginning of a headline, complete TODO keywords.
- After '\', complete TEX symbols supported by the exporter.
- After '`*`', complete headlines in the current buffer so that they can be used in search links like '`[[*find this headline]]`'.
- After '`:`', complete tags. The list of tags is taken from the variable `org-tag-alist` (possibly set through the '`#+TAGS`' in-buffer option, see Section 7.2 [Setting tags], page 36), or it is created dynamically from all tags used in the current buffer.
- After '`[`', complete link abbreviations (see Section 4.5 [Link abbreviations], page 22).
- After '`#+`', complete the special keywords like '`TYP_TODO`' or '`OPTIONS`' which set file-specific options for Org-mode. When the option keyword is already complete, pressing *M-*⟨TAB⟩ again will insert example settings for this keyword.
- In the line after '`#+STARTUP: `', complete startup keywords, i.e. valid keys for this line.
- Elsewhere, complete dictionary words using ispell.

## 12.2 Customization

There are more than 100 variables that can be used to customize Org-mode. For the sake of compactness of the manual, we are not describing the variables here. A structured overview of customization variables is available with *M-x org-customize*. Or select `Browse Org Group` from the `Org->Customization` menu. Many settings can also be activated on a per-file basis, by putting special lines into the buffer (see Section 12.3 [In-buffer settings], page 63).

## 12.3 Summary of in-buffer settings

Org-mode uses special lines in the buffer to define settings on a per-file basis. These lines start with a '`#+`' followed by a keyword, a colon, and then individual words defining a setting. Several setting words can be in the same line, but you can also have multiple lines for the keyword. While these settings are described throughout the manual, here is a summary. After changing any of those lines in the buffer, press *C-c C-c* with the cursor

still in the line to activate the changes immediately. Otherwise they become effective only when the file is visited again in a new Emacs session.

*#+STARTUP:*

This line sets options to be used at startup of org-mode, when an Org-mode file is being visited. The first set of options deals with the initial visibility of the outline tree. The corresponding variable for global default settings is `org-startup-folded`, with a default value `t`, which means `overview`.

| | |
|---|---|
| `overview` | top-level headlines only |
| `content` | all headlines |
| `showall` | no folding at all, show everything |

Then there are options for aligning tables upon visiting a file. This is useful in files containing narrowed table columns. The corresponding variable is `org-startup-align-all-tables`, with a default value `nil`.

| | |
|---|---|
| `align` | align all tables |
| `noalign` | don't align tables on startup |

Logging when a TODO item is marked DONE (variable `org-log-done`) can be configured using these options.

| | |
|---|---|
| `logging` | record a timestamp when an item is marked DONE |
| `nologging` | don't record when items are marked DONE |

Here are the options for hiding leading stars in outline headings. The corresponding variables are `org-hide-leading-stars` and `org-odd-levels-only`, both with a default setting `nil` (meaning `showstars` and `oddeven`).

| | |
|---|---|
| `hidestars` | make all but one of the stars starting a headline invisible. |
| `showstars` | show all stars starting a headline |
| `odd` | allow only odd outline levels (1,3,...) |
| `oddeven` | allow all outline levels |

To turn on custom format overlayes over time stamps (variables `org-put-time-stamp-overlays` and `org-time-stamp-overlay-formats`), use

| | |
|---|---|
| `customtime` | overlay custom time format |

*#+SEQ_TODO: #+TYP_TODO:*

These lines set the TODO keywords and their interpretation in the current file. The corresponding variables are `org-todo-keywords` and `org-todo-interpretation`.

*#+TAGS: TAG1(c1) TAG2(c2)*

These lines (several such lines are allowed) specify the legal tags in this file, and (potentially) the corresponding *fast tag selection* keys. The corresponding variable is `org-tag-alist`.

*#+LINK: linkword replace*

These lines (several are allowed) specify link abbreviations. See Section 4.5 [Link abbreviations], page 22. The corresponding variable is `org-link-abbrev-alist`.

*#+CATEGORY:*

This line sets the category for the agenda file. The category applies for all subsequent lines until the next '`#+CATEGORY`' line, or the end of the file.

`#+TBLFM:` This line contains the formulas for the table directly above the line.

`#+TITLE:, #+AUTHOR:, #+EMAIL:, #+LANGUAGE:, #+TEXT:, #+OPTIONS:`
These lines provide settings for exporting files. For more details see Section 10.5.3 [Export options], page 56.

## 12.4 The very busy C-c C-c key

The key `C-c C-c` has many purposes in org-mode, which are all mentioned scattered throughout this manual. One specific function of this key is to add *tags* to a headline (see Chapter 7 [Tags], page 36). In many other circumstances it means something like *Hey Org-mode, look here and update according to what you see here.* Here is a summary of what this means in different contexts.

- If there are highlights in the buffer from the creation of a sparse tree, or from clock display, remove these highlights.
- If the cursor is in one of the special `#+KEYWORD` lines, this triggers scanning the buffer for these lines and updating the information.
- If the cursor is inside a table, realign the table. This command works even if the automatic table editor has been turned off.
- If the cursor is on a `#+TBLFM` line, re-apply the formulas to the entire table.
- If the cursor is inside a table created by the '`table.el`' package, activate that table.
- If the current buffer is a remember buffer, close the note and file it. With a prefix argument, file it, without further interaction, to the default location.
- If the cursor is on a `<<<target>>>`, update radio targets and corresponding links in this buffer.
- If the cursor is in a plain list item with a checkbox, toggle the status of the checkbox.
- If the cursor is on a numbered item in a plain list, renumber the ordered list.

## 12.5 A cleaner outline view

Some people find it noisy and distracting that the Org-mode headlines are starting with a potentially large number of stars. For example the tree from Section 2.2 [Headlines], page 4:

```
* Top level headline
** Second level
*** 3rd level
    some text
*** 3rd level
    more text
* Another top level headline
```

Unfortunately this is deeply ingrained into the code of Org-mode and cannot be easily changed. You can, however, modify the display in such a way that all leading stars become invisible and the outline more easy to read. To do this, customize the variable `org-hide-leading-stars` like this:

```
(setq org-hide-leading-stars t)
```

or change this on a per-file basis with one of the lines (anywhere in the buffer)

```
#+STARTUP: showstars
#+STARTUP: hidestars
```

Press *C-c C-c* with the cursor in a '`STARTUP`' line to activate the modifications.

With stars hidden, the tree becomes:

```
* Top level headline
 * Second level
  * 3rd level
    some text
  * 3rd level
    more text
* Another top level headline
```

Note that the leading stars are not truly replaced by whitespace, they are only fontified with the face `org-hide` that uses the background color as font color. If are are not using either white or black background, you may have to customize this face to get the wanted effect. Another possibility is to set this font such that the extra stars are *almost* invisible, for example using the color `grey90` on a white background.

Things become cleaner still if you skip all the even levels and use only odd levels 1, 3, 5..., effectively adding two stars to go from one outline level to the next:

```
* Top level headline
  * Second level
    * 3rd level
      some text
    * 3rd level
      more text
* Another top level headline
```

In order to make the structure editing and export commands handle this convention correctly, use

```
(setq org-odd-levels-only t)
```

or set this on a per-file basis with one of the following lines (don't forget to press *C-c C-c* with the cursor in the startup line to activate changes immediately).

```
#+STARTUP: odd
#+STARTUP: oddeven
```

You can convert an Org-mode file from single-star-per-level to the double-star-per-level convention with *M-x org-convert-to-odd-levels RET* in that file. The reverse operation is *M-x org-convert-to-oddeven-levels*.

## 12.6 Using org-mode on a tty

Org-mode uses a number of keys that are not accessible on a tty. This applies to most special keys like cursor keys, ⟨TAB⟩ and ⟨RET⟩, when these are combined with modifier keys like ⟨Meta⟩ and/or ⟨Shift⟩. Org-mode uses these bindings because it needs to provide keys

for a large number of commands, and because these keys appeared particularly easy to remember. In order to still be able to access the core functionality of Org-mode on a tty, alternative bindings are provided. Here is a complete list of these bindings, which are obviously more cumbersome to use. Note that sometimes a work-around can be better. For example changing a time stamp is really only fun with `S-⟨cursor⟩` keys. On a tty you would rather use `C-c .` to re-insert the timestamp.

| Default | Alternative 1 | Alternative 2 |
|---|---|---|
| `S-⟨TAB⟩` | `C-u ⟨TAB⟩` | |
| `M-⟨left⟩` | `C-c C-x l` | `⟨Esc⟩ ⟨left⟩` |
| `M-S-⟨left⟩` | `C-c C-x L` | |
| `M-⟨right⟩` | `C-c C-x r` | `⟨Esc⟩ ⟨right⟩` |
| `M-S-⟨right⟩` | `C-c C-x R` | |
| `M-⟨up⟩` | `C-c C-x u` | `⟨Esc⟩ ⟨up⟩` |
| `M-S-⟨up⟩` | `C-c C-x U` | |
| `M-⟨down⟩` | `C-c C-x d` | `⟨Esc⟩ ⟨down⟩` |
| `M-S-⟨down⟩` | `C-c C-x D` | |
| `S-⟨RET⟩` | `C-c C-x c` | |
| `M-⟨RET⟩` | `C-c C-x m` | `⟨Esc⟩ ⟨RET⟩` |
| `M-S-⟨RET⟩` | `C-c C-x M` | |
| `S-⟨left⟩` | `C-c C-x ⟨left⟩` | |
| `S-⟨right⟩` | `C-c C-x ⟨right⟩` | |
| `S-⟨up⟩` | `C-c C-x ⟨up⟩` | |
| `S-⟨down⟩` | `C-c C-x ⟨down⟩` | |

## 12.7 Interaction with other packages

Org-mode lives in the world of GNU Emacs and interacts in various ways with other code out there.

### 12.7.1 Packages that Org-mode cooperates with

‘`calc.el`’ by Dave Gillespie

Org-mode uses the calc package for implementing spreadsheet functionality in its tables (see Section 3.3 [Table calculations], page 13). Org-modes checks for the availability of calc by looking for the function `calc-eval` which should be autoloaded in your setup if calc has been installed properly. As of Emacs 22, calc is part of the Emacs distribution. Another possibility for interaction between the two packages is using calc for embedded calculations. See section “Embedded Mode” in *GNU Emacs Calc Manual*.

‘`constants.el`’ by Carsten Dominik

In a table formula (see Section 3.3 [Table calculations], page 13), it is possible to use names for natural constants or units. Instead of defining your own constants in the variable `org-table-formula-constants`, install the ‘`constants`’ package which defines a large number of constants and units, and lets you use unit prefixes like ‘`M`’ for ‘`Mega`’ etc. You will need version 2.0 of this package,

available at `http://www.astro.uva.nl/~dominik/Tools`. Org-mode checks for the function `constants-get`, which has to be autoloaded in your setup. See the installation instructions in the file '`constants.el`'.

'`cdlatex.el`' by Carsten Dominik
: Org-mode can make use of the cdlatex package to efficiently enter LaTeX fragments into Org-mode files. See Section 9.5 [CDLaTeX mode], page 52.

'`remember.el`' by John Wiegley
: Org mode cooperates with remember, see Section 4.8 [Remember], page 24. '`Remember.el`' is not part of Emacs, find it on the web.

'`table.el`' by Takaaki Ota
: Org mode cooperates with table.el, see Section 3.5 [table.el], page 17. '`table.el`' is part of Emacs 22.

### 12.7.2 Packages that lead to conflicts with Org-mode

'`allout.el`' by Ken Manheimer
: Startup of Org-mode may fail with the error message `(wrong-type-argument keymapp nil)` when there is an outdated version '`allout.el`' on the load path, for example the version distributed with Emacs 21.x. Upgrade to Emacs 22 and this problem will disappear. If for some reason you cannot do this, make sure that org.el is loaded *before* '`allout.el`', for example by putting `(require 'org)` early enough into your '`.emacs`' file.

'`CUA.el`' by Kim. F. Storm
: Keybindings in Org-mode conflict with the *`S-<cursor>`* keys used by CUA-mode (as well as pc-select-mode and s-region-mode) to select and extend the region. If you want to use one of these packages along with Org-mode, configure the variable `org-CUA-compatible`. When set, Org-mode will move the following keybindings in org-mode files, and in the agenda buffer (but not during date selection).

```
S-UP    -> M-p             S-DOWN  -> M-n
S-LEFT  -> M--             S-RIGHT -> M-+
S-RET   -> C-S-RET
```

  Yes, these are unfortunately more difficult to remember. If you want to have other replacement keys, look at the variable `org-disputed-keys`.

'`windmove.el`' by Hovav Shacham
: Also this package uses the *`S-<cursor>`* keys, so everything written in the paragraph above about CUA mode also applies here.

## 12.8 Bugs

Here is a list of things that should work differently, but which I have found too hard to fix.

- If a table field starts with a link, and if the corresponding table column is narrowed (see Section 3.2 [Narrow columns], page 12) to a width too small to display the link, the field would look entirely empty even though it is not. To prevent this, Org-mode throws an error. The work-around is to make the column wide enough to fit the link, or to add some text (at least 2 characters) before the link in the same field.
- Narrowing table columns does not work on XEmacs, because the `format` function does not transport text properties.
- Text in an entry protected with the '`QUOTE`' keyword should not autowrap.
- When the application called by `C-c C-o` to open a file link fails (for example because the application does not exist or refuses to open the file), it does so silently. No error message is displayed.
- The remote-editing commands in the agenda buffer cannot be undone with `undo` called from within the agenda buffer. But you can go to the corresponding buffer (using ⟨TAB⟩ or ⟨RET⟩ and execute `undo` there.
- Recalculating a table line applies the formulas from left to right. If a formula uses *calculated* fields further down the row, multiple recalculation may be needed to get all fields consistent.
- A single letter cannot be made bold, for example '`*a*`'.
- The exporters work well, but could be made more efficient.

# Appendix A Extensions, Hooks and Hacking

This appendix lists extensions for Org-mode written by other authors. It also covers some aspects where users can easily extend the functionality of Org-mode.

## A.1 Third-party extensions for Org-mode

The following extensions for Org-mode have been written by other people:

'`org-publish.el`' by David O'Toole
: This package provides facilities for publishing related sets of Org-mode files together with linked files like images as a webpages. It is highly configurable and can be used for other publishing purposes as well. As of Org-mode version 4.30, '`org-publish.el`' is part of the Org-mode distribution. It is not yet part of Emacs, however, a delay caused by the preparations for the 22.1 release. In the mean time, '`org-publish.el`' can be downloaded from David's site: `http://dto.freeshell.org/e/org-publish.el`.

'`org-mouse.el`' by Piotr Zielinski
: This package implements extended mouse functionality for Org-mode. It allows you to cycle visibility and to edit the document structure with the mouse. Best of all, it provides a context-sensitive menu on ⟨mouse-3⟩ that changes depending on the context of a mouse-click. As of Org-mode version 4.53, '`org-mouse.el`' is part of the Org-mode distribution. It is not yet part of Emacs, however, a delay caused by the preparations for the 22.1 release. In the mean time, '`org-mouse.el`' can be downloaded from Piotr's site: `http://www.cl.cam.ac.uk/~pz215/files/org-mouse.el`.

'`org-blog.el`' by David O'Toole
: A blogging plug-in for '`org-publish.el`'. `http://dto.freeshell.org/notebook/OrgMode.html`.

'`org-blogging.el`' by Bastien Guerry
: Publish Org-mode files as blogs. `http://www.cognition.ens.fr/~guerry/org-blogging.html`.

## A.2 Dynamic blocks

Org-mode documents can contain *dynamic blocks*. These are specially marked regions that are updated by some user-written function. A good example for such a block is the clock table inserted by the command `C-c C-x C-r` (see Section 6.4.2 [Clocking work time], page 34).

Dynamic block are enclosed by a BEGIN-END structure that assigns a name to the block and can also specify parameters for the function producing the content of the block.

```
#+BEGIN: myblock :parameter1 value1 :parameter2 value2 ...

#+END:
```

Dynamic blocks are updated with the following commands

`C-c C-x C-u`
: Update dynamic block at point.

`C-u C-c C-x C-u`
: Update all dynamic blocks in the current file.

Updating a dynamic block means to remove all the text between BEGIN and END, parse the BEGIN line for parameters and then call the specific writer function for this block to insert the new content. For a block with name `myblock`, the writer function is `org-dblock-write:myblock` with as only parameter a property list with the parameters given in the begin line. Here is a trivial example of a block that keeps track of when the block update function was last run:

```
#+BEGIN: block-update-time :format "on %m/%d/%Y at %H:%M"

#+END:
```

The corresponding block writer function could look like this:

```
(defun org-dblock-write:block-update-time (params)
   (let ((fmt (or (plist-get params :format) "%d. %m. %Y")))
     (insert "Last block update at: "
             (format-time-string fmt (current-time)))))
```

If you want to make sure that all dynamic blocks are always up-to-date, you could add the function `org-update-all-dblocks` to a hook, for example `before-save-hook`. `org-update-all-dblocks` is written in a way that is does nothing in buffers that are not in Org-mode.

# Appendix B History and Acknowledgments

Org-mode was borne in 2003, out of frustration over the user interface of the Emacs outline-mode. All I wanted was to make working with an outline tree possible without having to remember more than 10 commands just for hiding and unhiding parts of the outline tree, and to allow to restructure a tree easily. Visibility cycling and structure editing were originally implemented in the package '`outline-magic.el`', but quickly moved to the more general '`org.el`'. TODO entries, basic time stamps, and table support were added next, and highlight the two main goals that Org-mode still has today: To create a new, outline-based, plain text mode with innovative and intuitive editing features, and to incorporate project planning functionality directly into a notes file.

Since the first release, hundreds of emails to me or on `emacs-orgmode@gnu.org` have provided a constant stream of bug reports, feedback, new ideas, and sometimes even patches and add-on code. Many thanks to everyone who has helped to improve this package. I am trying to keep here a list of the people who had significant influence in shaping one or more aspects of Org-mode. The list may not be complete, if I have forgotten someone, please accept my apologies and let me know.

- *Thomas Baumann* contributed the code for links to the MH-E email system.
- *Alex Bochannek* provided a patch for rounding time stamps.
- *Charles Cave*'s suggestion sparked the implementation of templates for Remember.
- *Pavel Chalmoviansky* influenced the agenda treatment of items with specified time.
- *Gregory Chernov* patched support for lisp forms into table calculations and improved XEmacs compatibility, in particular by porting '`nouline.el`' to XEmacs.
- *Sacha Chua* suggested to copy some linking code from Planner.
- *Eddward DeVilla* proposed and tested checkbox statistics.
- *Kees Dullemond* inspired the use of narrowed tabled columns.
- *Christian Egli* converted the documentation into TeXInfo format, patched CSS formatting into the HTML exporter, and inspired the agenda.
- *Nic Ferrier* contributed mailcap and XOXO support.
- *Niels Giessen* had the idea to automatically archive DONE trees.
- *Bastien Guerry* provided extensive feedback.
- *Kai Grossjohann* pointed out key-binding conflicts with other packages.
- *Leon Liu* asked for embedded LaTeX and tested it.
- *Stefan Monnier* provided a patch to keep the Emacs-Lisp compiler happy.
- *Todd Neal* provided patches for links to Info files and elisp forms.
- *Tim O'Callaghan* suggested in-file links, search options for general file links, and TAGS.
- *Oliver Oppitz* suggested multi-state TODO items.
- *Scott Otterson* sparked the introduction of descriptive text for links, among other things.
- *Pete Phillips* helped during the development of the TAGS feature, and provided frequent feedback.
- *T.V. Raman* reported bugs and suggested improvements.

- *Matthias Rempe* (Oelde) provided ideas, Windows support, and quality control.
- *Kevin Rogers* contributed code to access VM files on remote hosts.
- *Frank Ruell* solved the mystery of the `keymapp nil` bug, a conflict with '`allout.el`'.
- *Jason Riedy* sent a patch to fix a bug with export of TODO keywerds.
- *Philip Rooke* created the Org-mode reference card and provided lots of feedback.
- *Christian Schlauer* proposed angular brackets around links, among other things.
- Linking to VM/BBDB/GNUS was inspired by *Tom Shannon*'s '`organizer-mode.el`'.
- *Daniel Sinder* came up with the idea of internal archiving by locking subtrees.
- *Dale Smith* proposed link abbreviations.
- *David O'Toole* wrote '`org-publish.el`' and drafted the manual chapter about publishing.
- *Jürgen Vollmer* contributed code generating the table of contents in HTML output.
- *Chris Wallace* provided a patch implementing the '`QUOTE`' keyword.
- *David Wainberg* suggested archiving, and improvements to the linking system.
- *John Wiegley* wrote '`emacs-wiki.el`' and '`planner.el`'. The development of Org-mode was fully independent, and both systems are really different beasts in their basic ideas and implementation details. I later looked at John's code, however, and learned from his implementation of (i) links where the link itself is hidden and only a description is shown, and (ii) popping up a calendar to select a date.
- *Carsten Wimmer* suggested some changes and helped fix a bug in linking to GNUS.
- *Roland Winkler* requested additional keybindings to make Org-mode work on a tty.
- *Piotr Zielinski* wrote '`org-mouse.el`', proposed angenda blocks and contributed various ideas and code snippets.

# Index

## A



## B



## C



## D



## E

## F



## G



## H



## I



## J



## K



## L



## M

## N



## O



## P



## Q



## R



## S



## T

## U



## V



## W



## X

# Key Index

## D



## F



## G



## H



## I



## L



## M



## N



## O



## P



## Q



## R

## S



## T



## W



## X