The Org Manual

by Carsten Dominik

This manual is for Org (version 6.05b).

Table of Contents

1 Introduction

1.1 Summary

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

Org develops organizational tasks around NOTES files that contain lists or information about projects as plain text. Org 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 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 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 web pages.

An important design aspect that distinguishes Org from for example Planner/Muse is that it encourages 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, 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 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 is a toolbox and can be used in different ways, for example as:

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

Org's automatic, context sensitive table editor with spreadsheet capabilities can be integrated into any major mode by activating the minor Orgtbl mode. Using a translation step, it can be used to maintain tables in arbitrary file types, for example in LaTeX. The structure editing and list creation capabilities can be used outside Org with the minor Orgstruct mode.

There is a website for Org which provides links to the newest version of Org, as well as additional information, frequently asked questions (FAQ), links to tutorials etc. This page is located at http://orgmode.org.

1.2 Installation

Important: *If Org 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 from the Web, either as a distribution '`.zip`' or '`.tar`' file, or as a GIT archive, you must take the following steps to install it: Go into the unpacked Org 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, you can simply run Org directly from the distribution directory by adding the '`lisp`' subdirectory to the Emacs load path. To do this, add the following line to '`.emacs`':

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

If you plan to use code from the '`contrib`' subdirectory, do a similar step for this directory:

```
(setq load-path (cons "~/path/to/orgdir/contrib/lisp" load-path))
```

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

make install-noutline

Now byte-compile the Lisp files with the shell command:

```
make
```

If you are running Org from the distribution directory, this is all. If you want to install into the system directories, use

```
make install
make install-info
```

Then add to '`.emacs`':

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

1.3 Activation

Important: *If you use copy-and-paste to copy lisp code from the PDF documentation as viewed by some PDF viewers to your .emacs file, the single quote character comes out incorrectly and the code will not work. You need to fix the single quotes by hand, or copy from Info documentation.*

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

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

Furthermore, you must activate `font-lock-mode` in Org 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 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, or if you have questions, remarks, or ideas about it, please mail to the Org mailing list `emacs-orgmode@gnu.org`. If you are not a member of the mailing list, your mail will be reviewed by a moderator and then passed through to the list.

For bug reports, please provide as much information as possible, including the version information of Emacs (`C-h v emacs-version` $\langle\overline{\text{RET}}\rangle$) and Org (`C-h v org-version` $\langle\overline{\text{RET}}\rangle$), as well as the Org 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 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 document the steps you take.

4. When you hit the error, a '`*Backtrace*`' buffer will appear on the screen. Save this
   buffer to a file (for example using `C-x C-w`) and attach it to your bug report.

1.5 Typesetting conventions used in this manual

Org uses three types of keywords: TODO keywords, tags, and property names. In this
manual we use the following conventions:

`TODO`
`WAITING`   TODO keywords are written with all capitals, even if they are user-defined.

`boss`
`ARCHIVE`   User-defined tags are written in lowercase; built-in tags with special meaning
             are written with all capitals.

`Release`
`PRIORITY`   User-defined properties are capitalized; built-in properties with special meaning
             are written with all capitals.

2 Document Structure

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

2.1 Outlines

Org is implemented on top of Outline mode. Outlines allow a document to be organized in a hierarchical structure, which (at least for me) is the best representation of notes and thoughts. An overview of 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 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 start with one or more stars, on the left margin[1]. 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 14.5 [Clean view], page 109 describes a setup to realize this.

An empty line after the end of a subtree is considered part of it and will be hidden when the subtree is folded. However, if you leave at least two empty lines, one empty line will remain visible after folding the subtree, in order to structure the collapsed view. See the variable `org-cycle-separator-lines` to modify this behavior.

2.3 Visibility cycling

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

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

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

---

[1] See the variable `org-special-ctrl-a/e` to configure special behavior of `C-a` and `C-e` in headlines.

The cursor must be on a headline for this to work[2]. 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)[3]. Also when called with a prefix argument (`C-u` ⟨TAB⟩), global cycling is invoked.

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

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

When `S-`⟨TAB⟩ is called with a numeric prefix argument N, the CONTENTS view up to headlines of level N will be shown. 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 that has been exposed by a sparse tree command (see Section 2.7 [Sparse trees], page 10) or an agenda command (see Section 10.5 [Agenda commands], page 73). With a prefix argument show, on each level, all sibling headings.

`C-c C-x b` Show the current subtree in an indirect buffer[4]. With a numeric prefix argument N, go up to level N and then take that tree. If N is negative then go up that many levels. With a `C-u` prefix, do not remove the previously used indirect buffer.

When Emacs first visits an Org 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
```

Forthermore, any entries with a 'VISIBILITY' property (see Chapter 7 [Properties and Columns], page 45) will get their visibility adapted accordingly. Allowed values for this property are `folded`, `children`, `content`, and `all`.

`C-u C-u` ⟨TAB⟩
            Switch back to the startup visibility of the buffer, i.e. whatever is requested by startup options and 'VISIBILITY' properties in individual entries.

2.4 Motion

The following commands jump to other headlines in the buffer.

---

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

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

[4] The indirect buffer (see the Emacs manual for more information about indirect buffers) will contain the entire buffer, but will be narrowed to the current tree. Editing the indirect buffer will also change the original buffer, but without affecting visibility in that 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 the following keys to find your destination:

⟨TAB⟩	Cycle visibility.
⟨down⟩ / ⟨up⟩	Next/previous visible headline.
n / p	Next/previous visible headline.
f / b	Next/previous headline same level.
u	One level up.
0-9	Digit argument.
⟨RET⟩	Select this location.

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 11). To force creation of a new headline, use a prefix argument, 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[5]. 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.

`C-⟨RET⟩`   Insert a new heading after the current subtree, same level as the current headline. This command works from anywhere in the entry.

`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).

---

[5] If you do not want the line to be split, customize the variable `org-M-RET-may-split-line`.

`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. With a numeric prefix argument N, kill N sequential subtrees.

`C-c C-x M-w`

> Copy subtree to kill ring. With a numeric prefix argument N, copy the N sequential subtrees.

`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 numeric prefix argument, or by yanking after a headline marker like '****'.

`C-c C-w`   Refile entry to a different location. See Section 9.4 [Refiling notes], page 65.

`C-c ^`   Sort same-level entries. When there is an active region, all entries in the region will be sorted. Otherwise the children of the current headline are sorted. The command prompts for the sorting method, which can be alphabetically, numerically, by time (using the first time stamp in each entry), by priority, or by TODO keyword (in the sequence the keywords have been defined in the setup). Reverse sorting is possible as well. You can also supply your own function to extract the sorting key. With a `C-u` prefix, sorting will be case-sensitive. With two `C-u C-u` prefixes, duplicate entries will also be removed.

`C-x n s`   Narrow buffer to current subtree.

`C-x n w`   Widen buffer to remove a narrowing.

`C-c *`   Turn a normal line or plain list item into a headline (so that it becomes a subheading at its location). Also turn a headline into a normal line by removing the stars. If there is an active region, turn all lines in the region into headlines. Or, if the first line is a headline, remove the stars from all headlines in the region.

   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 14), 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 6 [Tags], page 41) 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 5). 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 10), matches in archived subtrees are not exposed, unless you configure the option `org-sparse-tree-open-archived-trees`.

− During agenda view construction (see Chapter 10 [Agenda Views], page 66), the content of archived trees is ignored unless you configure the option `org-agenda-skip-archived-trees`, in which case these trees will always be included. In the agenda you can press the `v` key to get archives temporarily included.

− Archived trees are not exported (see Chapter 12 [Exporting], page 88), 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 a`   Toggle the ARCHIVE tag for the current headline. When the tag is set, the headline changes to a shadowed face, and the subtree below it is hidden.

`C-u C-c C-x 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. Org can move it to an *Archive Sibling* in the same tree, to a different tree in the current file, or to a different file, the archive file.

`C-c C-x A`   Move the current entry to the *Archive Sibling*. This is a sibling of the entry with the heading '`Archive`' and the tag '`ARCHIVE`' (see Section 2.6.1 [ARCHIVE tag], page 9). The entry becomes a child of that sibling and in this way retains a lot of its original context, including inherited tags and approximate position in the outline.

`C-c C-x C-s`

Archive the subtree starting at the cursor position to the location given by `org-archive-location`. Context information that could be lost like the file name, the category, inherited tags, and the TODO state will be store as properties in the entry.

`C-u C-c C-x C-s`
> 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`. There is also an in-buffer option for setting this variable, for example[6]:

    #+ARCHIVE: %s_done::

If you would like to have a special ARCHIVE location for a single entry or a (sub)tree, give the entry an `:ARCHIVE:` property with the location as the value (see Chapter 7 [Properties and Columns], page 45).

When a subtree is moved, it receives a number of special properties that record context information like the file from where the entry came, it's outline path the archiving time etc. Configure the variable `org-archive-save-context-info` to adjust the amount of information added.

2.7 Sparse trees

An important feature of Org mode is the ability to construct *sparse trees* for selected information in an outline tree, so that the entire document is folded as much as possible, but the selected information is made visible along with the headline structure above it[7]. Just try it out and you will see immediately how it works.

Org mode contains several commands creating such trees, all these commands can be accessed through a dispatcher:

`C-c /`   This prompts for an extra key to select a sparse-tree creating command.

`C-c / r`   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 buffer is changed by an editing command[8], 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.

---

[6] For backward compatibility, the following also works: If there are several such lines in a file, each specifies the archive location for the text below it. The first such line also applies to any text before its definition. However, using this method is *strongly* deprecated as it is incompatible with the outline structure of the document. The correct method for setting multiple archive locations in a buffer is using a property.

[7] See also the variables `org-show-hierarchy-above`, `org-show-following-heading`, `org-show-siblings`, and `org-show-entry-below` for detailed control on how much context is shown around each match.

[8] depending on the option `org-remove-highlights-with-change`

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 10.2 [Agenda dispatcher], page 67). 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'.

The other sparse tree commands select headings based on TODO keywords, tags, or properties and will be discussed later in this manual.

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[9]. 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 5.6 [Checkboxes], page 39). Org supports editing such lists, and the HTML exporter (see Chapter 12 [Exporting], page 88) parses and formats them.

Org knows ordered lists, unordered lists, and description lists.

- *Unordered* list items start with '`-`', '`+`', or '`*`'[10] as bullets.

- *Ordered* list items start with a numeral followed by either a period or a right parenthesis, such as '`1.`' or '`1)`'.

- *Description* list items are like unordered list items, but contain the separator ' `::` ' to separate the description *term* from the desciption.

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. Empty lines are part of the previous item, so you can have several paragraphs in one item. If you would like an empty line to terminate all currently open plain lists, configure the variable `org-empty-line-terminates-plain-lists`. Here is an example:

---

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

[10] 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 to not 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.
   Important actors in this film are:
   - Elijah Wood :: He plays the Frodo
   - Sean Austin :: He plays the Sam, Frodos friend.  I still remember
     him very well from his role as Mikey Walsh a in the Goonies.
```

Org supports these lists by tuning filling and wrapping commands to deal with them correctly[11], and by exporting them properly (see Chapter 12 [Exporting], page 88).

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.

　　　　　　If `org-cycle-include-plain-lists` has not been set, ⟨TAB⟩ fixes the indentation of the current line in a heuristic way.

`M-`⟨RET⟩　　Insert new item at current level. With a prefix argument, force a new heading (see Section 2.5 [Structure editing], page 7). 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[12]. 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 5.6 [Checkboxes], page 39).

`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

---

[11] Org only changes the filling settings for Emacs. For XEmacs, you should use Kyle E. Jones' 'filladapt.el'. To turn this on, put into '.emacs': `(require 'filladapt)`

[12] If you do not want the line to be split, customize the variable `org-M-RET-may-split-line`.

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 5.6 [Checkboxes], page 39) in the item line, toggle the state of the checkbox. If not, this command makes sure that all the items on this list level use the same bullet. Furthermore, if this is an ordered list, make sure the numbering is OK.

`C-c -`   Cycle the entire list level through the different itemize/enumerate bullets ('-', '+', '*', '1.', '1)'). With a numeric prefix argument N, select the Nth bullet from this list. If there is an active region when calling this, all lines will be converted to list items. If the first line already was a list item, any item markers will be removed from the list. Finally, even without an active region, a normal line will be converted into a list item.

2.9 Drawers

Sometimes you want to keep information associated with an entry, but you normally don't want to see it. For this, Org mode has *drawers*. Drawers need to be configured with the variable `org-drawers`[13]. Drawers look like this:

```
** This is a headline
   Still outside the drawer
   :DRAWERNAME:
       This is inside the drawer.
   :END:
   After the drawer.
```

Visibility cycling (see Section 2.3 [Visibility cycling], page 5) on the headline will hide and show the entry, but keep the drawer collapsed to a single line. In order to look inside the drawer, you need to move the cursor to the drawer line and press ⟨TAB⟩ there. Org mode uses a drawer for storing properties (see Chapter 7 [Properties and Columns], page 45), and another one for storing clock times (see Section 8.4 [Clocking work time], page 58).

2.10 The Orgstruct minor mode

If you like the intuitive way the Org mode structure editing and list formatting works, you might want to use these commands in other modes like Text mode or Mail mode as well. The minor mode Orgstruct mode makes this possible. You can always toggle the mode with `M-x orgstruct-mode`. To turn it on by default, for example in Mail mode, use

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

When this mode is active and the cursor is on a line that looks to Org like a headline of the first line of a list item, most structure editing commands will work, even if the same keys normally have different functionality in the major mode you are using. If the cursor is not in one of those special lines, Orgstruct mode lurks silently in the shadow.

---

[13] You can define drawers on a per-file basis with a line like `#+DRAWERS: HIDDEN PROPERTIES STATE`

3 Tables

Org comes with a fast and intuitive table editor. Spreadsheet-like calculations are supported in connection with the Emacs 'calc' package (see the Emacs Calculator manual for more information about the Emacs calculator).

3.1 The built-in table editor

Org makes it easy to format tables in plain ASCII. Any line with '|' as the first non-whitespace 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 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 every line contains a comma, comma-separated values (CSV) are assumed. If not, lines are split at whitespace into fields. You can use a prefix argument to force a specific separator: `C-u` forces CSV, `C-u C-u` forces TAB, and a numeric argument N indicates that at least N consecutive spaces, or alternatively a TAB will be the separator.

If there is no active region, this command creates an empty Org 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 the current row. With a prefix argument, the line is created below the current one.

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

C-c ^   Sort the table lines in the region. The position of point indicates the column to be used for sorting, and the range of lines is the range between the nearest horizontal separator lines, or the entire table. If point is before the first column, you will be prompted for the sorting column. If there is an active region, the mark specifies the first line and the sorting column, while point should be in the last line to be included into the sorting. The command prompts for the sorting type (alphabetically, numerically, or by time). When called with a prefix argument, alphabetic sorting will be case-sensitive.

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
M-RET   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 numeric prefix argument 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 argument, the current field is made blank, and the content is appended to the field above.

Calculations

`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 14.7.1 [Cooperation], page 111).

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 17). When called with a `C-u` prefix, just make the full field visible, so that it can be edited in place.

`M-x org-table-import`

Import a file as a table. The table should be TAB- or whitespace separated. Useful, for example, to import a spreadsheet 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.

`C-c |`   Tables can also be imported by pasting tabular text into the Org buffer, selecting the pasted text with `C-x C-x` and then using the `C-c |` command (see above under *Creation and conversion*.

`M-x org-table-export`

Export the table, by default as a TAB-separated file. Useful for data exchange with, for example, spreadsheet or database programs. The format used to export the file can be configured in the variable `org-table-export-default-format`. You may also use properties `TABLE_EXPORT_FILE` and `TABLE_EXPORT_FORMAT` to specify the file name and the format for table export in a subtree. Org supports quite general formats for exported tables. The exporter format is the same as the format used by Orgtbl radio tables, see Section B.2.3 [Translator functions], page 121 for a detailed description.

    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 tool-tip 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 Column groups

When Org exports tables, it does so by default without vertical lines because that is visually more satisfying in general. Occasionally however, vertical lines can be useful to structure a table into groups of columns, much like horizontal lines can do for groups of rows. In order to specify column groups, you can use a special row where the first field contains only '`/`'. The further fields can either contain '`<`' to indicate that this column should start a group, '`>`' to indicate the end of a column, or '`<>`' to make a column a group of its own. Boundaries between column groups will upon export be marked with vertical lines. Here is an example:

```
|   |   N | N^2 | N^3 | N^4 | sqrt(n) | sqrt[4](N) |
|---+-----+-----+-----+-----+---------+------------|
| / | <>  |  <  |     |  >  |    <    |     >      |
| # |  1  |  1  |  1  |  1  |     1   |       1    |
| # |  2  |  4  |  8  | 16  | 1.4142  |    1.1892  |
| # |  3  |  9  | 27  | 81  | 1.7321  |    1.3161  |
|---+-----+-----+-----+-----+---------+------------|
```

---

[1] This feature does not work on XEmacs.

```
#+TBLFM: $3=$2^2::$4=$2^3::$5=$2^4::$6=sqrt($2)::$7=sqrt(sqrt(($2))
```

It is also sufficient to just insert the column group starters after every vertical line you'd like to have:

```
|  N | N^2 | N^3 | N^4 | sqrt(n) | sqrt[4](N) |
|----+-----+-----+-----+---------+------------|
| /  | <   |     |     | <       |            |
```

3.4 The Orgtbl minor mode

If you like the intuitive way the Org 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)
```

Furthermore, with some special setup, it is possible to maintain tables in arbitrary syntax with Orgtbl mode. For example, it is possible to construct LaTeX tables with the underlying ease and power of Orgtbl mode, including spreadsheet capabilities. For details, see Section B.2 [Tables in arbitrary syntax], page 118.

3.5 The spreadsheet

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. While fully featured, Org's implementation is not identical to other spreadsheets. For example, Org knows the concept of a *column formula* that will be applied to all non-header fields in a column without having to copy the formula to each relevant field.

3.5.1 References

To compute fields in the table from other fields, formulas must reference other fields or ranges. In Org, fields can be referenced by name, by absolute coordinates, and by relative coordinates. To find out what the coordinates of a field are, press `C-c ?` in that field, or press `C-c }` to toggle the display of a grid.

Field references

Formulas can reference the value of another field in two ways. Like in any other spreadsheet, you may reference fields with a letter/number combination like `B3`, meaning the 2nd field in the 3rd row.

Org also uses another, more general operator that looks like this:

```
@row$column
```

Column references can be absolute like '1', '2',...'N', or relative to the current column like '+1' or '-2'.

The row specification only counts data lines and ignores horizontal separator lines (hlines). You can use absolute row numbers '1'...'N', and row numbers relative to the current row like '+3' or '-1'. Or specify the row relative to one of the hlines: 'I' refers to the first hline[2], 'II' to the second etc. '-I' refers to the first such line above the current line, '+I' to the first such line below the current line. You can also write 'III+2' which is the second data line after the third hline in the table. Relative row numbers like '-3' will not cross hlines if the current line is too close to the hline. Instead, the value directly at the hline is used.

'0' refers to the current row and column. Also, if you omit either the column or the row part of the reference, the current row/column is implied.

Org's references with *unsigned* numbers are fixed references in the sense that if you use the same reference in the formula for two different fields, the same field will be referenced each time. Org's references with *signed* numbers are floating references because the same reference operator can reference different fields depending on the field being calculated by the formula.

Here are a few examples:

```
@2$3        2nd row, 3rd column
C2          same as previous
$5          column 5 in the current row
E&          same as previous
@2          current column, row 2
@-1$-3      the field one row up, three columns to the left
@-I$2       field just under hline above current row, column 2
```

Range references

You may reference a rectangular range of fields by specifying two field references connected by two dots '..'. If both fields are in the current row, you may simply use '$2..$7', but if at least one field is in a different row, you need to use the general @row$column format at least for the first field (i.e the reference must start with '@' in order to be interpreted correctly). Examples:

```
$1..$3         First three fields in the current row.
$P..$Q         Range, using column names (see under Advanced)
@2$1..@4$3     6 fields between these two fields.
A2..C4         Same as above.
@-1$-2..@-1    3 numbers from the column to the left, 2 up to current row
```

Range references return a vector of values that can be fed into Calc vector functions. Empty fields in ranges are normally suppressed, so that the vector contains only the non-empty fields (but see the 'E' mode switch below). If there are no non-empty fields, '[0]' is returned to avoid syntax errors in formulas.

---

[2] Note that only hlines are counted that *separate* table lines. If the table starts with a hline above the header, it does not count.

Named references

'`$name`' is interpreted as the name of a column, parameter or constant. Constants are defined globally through the variable `org-table-formula-constants`, and locally (for the file) through a line like

```
#+CONSTANTS: c=299792458. pi=3.14 eps=2.4e-6
```

Also properties (see Chapter 7 [Properties and Columns], page 45) can be used as constants in table formulas: For a property '`:Xyz:`' use the name '`$PROP_Xyz`', and the property will be searched in the current outline entry and in the hierarchy above it. If you have the '`constants.el`' package, it will also be used to resolve constants, including natural constants like '`$h`' for Planck's constant, and units like '`$km`' for kilometers[3]. Column names and parameters can be specified in special table lines. These are described below, see Section 3.5.8 [Advanced features], page 24. All names must start with a letter, and further consist of letters and numbers.

3.5.2 Formula syntax for Calc

A formula can be any algebraic expression understood by the Emacs '`Calc`' package. **Note that '`calc`' has the 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 according to the rules described above. The range vectors can be directly fed into the Calc vector functions like '`vmean`' and '`vsum`'.

A formula can contain an optional mode string after a semicolon. This string consists of flags to influence Calc and other modes during execution. By default, Org 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`.

`p20`	switch the internal precision to 20 digits
`n3 s3 e2 f4`	normal, scientific, engineering, or fixed display format
`D R`	angle modes: degrees, radians
`F S`	fraction and symbolic modes
`N`	interpret all fields as numbers, use 0 for non-numbers
`T`	force text interpretation
`E`	keep empty fields in ranges

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
`$0;%.1f`	Reformat current cell to 1 decimal
`($3-32)*5/9`	Degrees F -> C conversion

---

[3] '`Constant.el`' can supply the values of constants in two different unit systems, SI and cgs. Which one is used depends on the value of the variable `constants-unit-system`. You can use the `#+STARTUP` options `constSI` and `constcgs` to set this value for the current buffer.

`$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
`vmean($2..$7);EN`	Same, but treat empty fields as 0
`taylor($3,x=7,2)`	taylor series of $3, at x=7, second degree

Calc also contains a complete set of logical operations. For example

`if($1<20,teen,string(""))`	"teen" if age $1 less than 20, else empty

3.5.3 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 the Calc's functionality is not enough. 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 specify modes and a printf format after a semicolon. With Emacs Lisp forms, you need to be conscious about the way field references are interpolated into the form. By default, a reference will be interpolated as a Lisp string (in double quotes) containing the field. If you provide the 'N' mode switch, all referenced elements will be numbers (non-number fields will be zero) and interpolated as Lisp numbers, without quotes. If you provide the 'L' flag, all fields will be interpolated literally, without quotes. I.e., if you want a reference to be interpreted as a string by the Lisp form, enclose the reference operator itself in double quotes, like `"$3"`. Ranges are inserted as space-separated fields, so you can embed them in list or vector syntax. A few examples, note how the 'N' mode is used when we do computations in lisp.

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);N`
Compute the sum of columns 1-4, like Calc's `vsum($1..$4)`
   `'(apply '+ '($1..$4));N`

3.5.4 Field formulas

To assign a formula to a particular field, type it directly into the field, preceded by ':=', for example ':=$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 this field, evaluated, and the current field replaced with the result.

Formulas are stored in a special line starting with '`#+TBLFM:`' directly below the table. If you typed the equation in the 4th field of the 3rd data line in the table, the formula will look like '`@3$4=$1+$2`'. When inserting/deleting/swapping column and rows with the appropriate commands, *absolute references* (but not relative ones) in stored formulas are modified in order to still reference the same field. Of cause this is not true if you edit the table structure with normal editing commands - then you must fix the equations yourself.

Instead of typing an equation into the field, you may also use the following command

`C-u C-c =`   Install a new formula for the current field. The command prompts for a formula, with default taken from the '`#+TBLFM:`' line, applies it to the current field and stores it.

3.5.5 Column formulas

Often in a table, the same formula should be used for all fields in a particular column. Instead of having to copy the formula to all fields in that column, Org allows to assign a single formula to an entire column. If the table contains horizontal separator hlines, everything before the first such line is considered part of the table *header* and will not be modified by column formulas.

To assign a formula to a column, type it directly into any field in the column, 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 will only remember the most recently used formula. In the '`TBLFM:`' line, column formulas will look like '`$4=$1+$2`'.

Instead of typing an equation into the field, you may also use the following command:

`C-c =`   Install a new formula for the current column and replace current field with the result of the formula. The command prompts for a formula, with default taken from the '`#+TBLFM`' line, applies it to the current field and stores it. With a numeric prefix argument(e.g. `C-5 C-c =`) the command will apply it to that many consecutive fields in the current column.

3.5.6 Editing and debugging formulas

You can edit individual formulas in the minibuffer or directly in the field. Org can also prepare a special buffer with all active formulas of a table. When offering a formula for editing, Org converts references to the standard format (like `B3` or `D&`) if possible. If you prefer to only work with the internal format (like `@3$2` or `$4`), configure the variable `org-table-use-standard-references`.

`C-c =`
`C-u C-c =`   Edit the formula associated with the current column/field in the minibuffer. See Section 3.5.5 [Column formulas], page 22 and Section 3.5.4 [Field formulas], page 21.

`C-u C-u C-c =`
             Re-insert the active formula (either a field formula, or a column formula) into the current field, so that you can edit it directly in the field. The advantage over editing in the minibuffer is that you can use the command `C-c ?`.

`C-c ?`   While editing a formula in a table field, highlight the field(s) referenced by the reference at the cursor position in the formula.

`C-c }`   Toggle the display of row and column numbers for a table, using overlays. These are updated each time the table is aligned, you can force it with `C-c C-c`.

`C-c {`   Toggle the formula debugger on and off. See below.

`C-c '`   Edit all formulas for the current table in a special buffer, where the formulas
              will be displayed one per line. If the current field has an active formula, the
              cursor in the formula editor will mark it. While inside the special buffer, Org
              will automatically highlight any field or range reference at the cursor position.
              You may edit, remove and add formulas, and use the following commands:

> `C-c C-c`
> `C-x C-s`   Exit the formula editor and store the modified formulas. With `C-u`
>   prefix, also apply the new formulas to the entire table.
>
> `C-c C-q`   Exit the formula editor without installing changes.
>
> `C-c C-r`   Toggle all references in the formula editor between standard (like
>   `B3`) and internal (like `@3$2`).
>
> ⟨TAB⟩   Pretty-print or indent lisp formula at point. When in a line con-
>   taining a lisp formula, format the formula according to Emacs Lisp
>   rules. Another ⟨TAB⟩ collapses the formula back again. In the open
>   formula, ⟨TAB⟩ re-indents just like in Emacs lisp mode.
>
> `M-`⟨TAB⟩   Complete Lisp symbols, just like in Emacs lisp mode.
>
> `S-`⟨up⟩`/`⟨down⟩`/`⟨left⟩`/`⟨right⟩
>   Shift the reference at point. For example, if the reference is `B3` and
>   you press `S-`⟨right⟩, it will become `C3`. This also works for relative
>   references, and for hline references.
>
> `M-S-`⟨up⟩`/`⟨down⟩
>   Move the test line for column formulas in the Org buffer up and
>   down.
>
> `M-`⟨up⟩`/`⟨down⟩
>   Scroll the window displaying the table.
>
> `C-c }`   Turn the coordinate grid in the table on and off.

Making a table field blank does not remove the formula associated with the field, because
that is stored in a different line (the 'TBLFM' line) - 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.

Debugging formulas

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 `Tbl` menu and repeat the calcula-
tion, for example by pressing `C-u C-u C-c =` ⟨RET⟩ in a field. Detailed information will be
displayed.

3.5.7 Updating the table

Recalculation of a table is normally not automatic, but needs to be triggered by a command. See for a way to make recalculation at least semi-automatically.

In order to recalculate a line of a table or the entire table, use the following commands:

`C-c *`   Recalculate the current row by first applying the stored column formulas from left to right, and all field formulas in the current row.

`C-u C-c *`
`C-u C-c C-c`
            Recompute the entire table, line by line. Any lines before the first hline are left alone, assuming that these are part of the table header.

`C-u C-u C-c *`
`C-u C-u C-c C-c`
            Iterate the table by recomputing it until no further changes occur. This may be necessary if some computed fields use the value of other fields that are computed *later* in the calculation sequence.

3.5.8 Advanced features

If you want the recalculation of fields to happen automatically, or if you want to be able to assign *names* to fields and columns, you need to reserve the first column of the table for special marking characters.

`C-#`   Rotate the calculation mark in first column through the states '', '#', '*', '!', '$'. The meaning of these characters is discussed below. When there is an active region, change all marks in the region.

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..@-I);%.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 fields that have a formula assigned to the field itself. 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 '$m1' to refer to the value '10'. Also, if you assign a formula to a names field, it will be stored as '$name=...'.

'_'   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.

'#'   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 '*'.

'/'   Do not export this line. Useful for lines that contain the narrowing '<N>' markers.

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 of degree n at location x for a couple of functions.

```
|---+------------+---+-----+----------------------------------|
|   | 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
```

4 Hyperlinks

Like HTML, Org provides links inside a file, external links to other files, Usenet articles, emails, and much more.

4.1 Link format

Org 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 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 29). The preferred match for such a link is a dedicated target: the same string in double angular brackets. Targets may be located anywhere; sometimes it is convenient to put them into a comment line. For example

>   `# <<My Target>>`

In HTML export (see Section 12.5 [HTML export], page 94), such targets will become named anchors for direct access through '`http`' links[1].

If no dedicated target exists, Org 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 is usually not exported, so the first such target should 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 29, for more commands creating links.

Following a link pushes a mark onto Org'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

Org can automatically turn any occurrences of certain target names in normal text into a link. 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 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.

4.3 External links

Org supports links to files, websites, Usenet and email messages, BBDB database entries and links to both IRC conversations and their logs. 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
irc:/irc.com/#emacs/bob                         IRC 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 27), for example:

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

If the description is a file name or URL that points to an image, HTML export (see Section 12.5 [HTML export], page 94) will inline the image as a clickable button. If there is no description at all and the link points to an image, that image will be inlined into the exported HTML file.

Org 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 if you need to remove ambiguities about the end of the link, enclose them in angular brackets.

4.4 Handling links

Org provides methods to create a link in the correct syntax, to insert it into an Org 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 buffer (see below). For Org 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 IRC links, if you set the variable org-irc-link-to-logs to non-nil then *C-c l* will store a 'file:/' style link to the relevant point in the logs for the current conversation. Otherwise an 'irc:/' style link to the user/channel/server under the point will be stored. For any other files, the link will point to the file, with a search string (see Section 4.7 [Search options], page 31) 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 4.8 [Custom searches], page 32. 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. All links stored during the current session are part of the history for this prompt, so you can access them with ⟨up⟩ and ⟨down⟩ (or *M-p/n*). Completion, on the other hand, will help you to insert valid link prefixes like 'http:' or 'ftp:', including the prefixes defined through link abbreviations (see Section 4.6 [Link abbreviations], page 31). The link will be inserted into the buffer[2], along with a descriptive text. If some text was selected when this command is called, the selected text becomes the default description. Note that you don't have to use this command to insert a link. Links in Org are plain text, and you can type or paste them straight into the buffer. By

---

[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 argument to *C-c C-l*, or configure the option org-keep-stored-link-after-insertion.

using this command, the links are automatically enclosed in double brackets, and you will be asked for the optional descriptive text.

`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 sub-directory 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.

`C-c C-x C-n`
`C-c C-x C-p`

Move forward/backward to the next link in the buffer. At the limit of the buffer, the search fails once, and then wraps around. The key bindings for this are really too long, you might want to bind this also to `C-n` and `C-p`

```
(add-hook 'org-load-hook
  (lambda ()
    (define-key 'org-mode-map "\C-n" 'org-next-link)
    (define-key 'org-mode-map "\C-p" 'org-previous-link)))
```

---

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

4.5 Using links outside Org

You can insert and follow links that have Org syntax not only in Org, but in any Emacs buffer. For this, you should create two global commands, like this (please select suitable global keys yourself):

```
(global-set-key "\C-c L" 'org-insert-link-global)
(global-set-key "\C-c o" 'org-open-at-point-global)
```

4.6 Link abbreviations

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:

```
(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 author is doing besides Emacs hacking with [[ads:Dominik,C]].

If you need special abbreviations just for a single Org 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 14.1 [Completion], page 106 can be used after '[' to complete link abbreviations.

4.7 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 29) 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*.

---

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

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 27. In HTML export (see Section 12.5 [HTML export], page 94), such a file link will become an HTML reference to the corresponding named anchor in the linked file.

*My Target
            In an Org 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.

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.

4.8 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 actually uses this mechanism for BibTeX database files, and you can use the corresponding code as an implementation example. See the file '`org-bibtex.el`'.

5 TODO Items

Org mode does not maintain TODO lists as separate documents[1]. Instead, TODO items are an integral part of the notes file, because TODO items usually come up while taking notes! With Org mode, simply mark any entry in a tree as being a TODO item. In this way, information is not duplicated, and the entire context from which the TODO item emerged is always present.

Of course, this technique for managing TODO items scatters them throughout your notes file. Org mode compensates for this by providing methods to give you an overview of all the things that you have to do.

5.1 Basic TODO functionality

Any headline becomes a TODO item when it starts 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 among

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

The same rotation can also be done "remotely" from the timeline and agenda buffers with the *t* command key (see Section 10.5 [Agenda commands], page 73).

C-u C-c C-t

Select a specific keyword using completion or (if it has been set up) the fast selection interface. For the latter, you need to assign keys to TODO states, see Section 5.2.5 [Per-file keywords], page 36 and Section 6.2 [Setting tags], page 41 for more information.

S-⟨right⟩
S-⟨left⟩   Select the following/preceding TODO state, similar to cycling. Useful mostly if more than two TODO states are possible (see Section 5.2 [TODO extensions], page 34).

C-c C-v
C-c / t   View TODO items in a *sparse tree* (see Section 2.7 [Sparse trees], page 10). Folds the entire buffer, but shows all TODO items and the headings hierarchy above them. With a prefix argument, search for a specific TODO. You will be prompted for the keyword, and you can also give a list of keywords like KWD1|KWD2|.... With numeric prefix argument N, show the tree for the Nth keyword in the variable `org-todo-keywords`. With two prefix arguments, find all TODO and DONE entries.

C-c a t   Show the global TODO list. Collects the TODO items from all agenda files (see Chapter 10 [Agenda Views], page 66) into a single buffer. The new buffer will be in `agenda-mode`, which provides commands to examine and manipulate

---

[1] Of course, you can make a document that contains inly long lists of TODO items, but this is not required.

the TODO entries from the new buffer (see Section 10.5 [Agenda commands], page 73). See Section 10.3.2 [Global TODO list], page 69, for more information.

`S-M-`⟨RET⟩   Insert a new TODO entry below the current one.

5.2 Extended use of TODO keywords

By default, marked TODO entries have one of only two states: TODO and DONE. Org mode allows you to classify TODO items in more complex ways with *TODO keywords* (stored in `org-todo-keywords`). With special setup, the TODO keyword system can work differently in different files.

Note that *tags* are another way to classify headlines in general and TODO items in particular (see Chapter 6 [Tags], page 41).

5.2.1 TODO keywords as workflow states

You can use TODO keywords to indicate different *sequential* states in the process of working on an item, for example[2]:

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

The vertical bar separates the TODO keywords (states that *need action*) from the DONE states (which need *no further action*. If you don't provide the separator bar, the last state is used as the DONE state. With this setup, the command `C-c C-t` will cycle an entry from TODO to FEEDBACK, then to VERIFY, and finally to DONE and DELEGATED. You may also use a numeric prefix argument to quickly select a specific state. For example `C-3 C-c C-t` will change the state immediately to VERIFY. Or you can use `S-left` to go backward through the sequence. If you define many keywords, you can use in-buffer completion (see Section 14.1 [Completion], page 106) or even a special one-key selection scheme (see Section 5.2.4 [Fast access to TODO states], page 35) to insert these words into the buffer. Changing a TODO state can be logged with a timestamp, see Section 5.3.2 [Tracking TODO state changes], page 37 for more information.

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". 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 '((type "Fred" "Sara" "Lucy" "|" "DONE")))
```

In this case, different keywords do not indicate a sequence, but rather different types. So the normal work flow would be to assign a task to a person, and later to mark it DONE.

---

[2] Changing this variable only becomes effective after restarting Org mode in a buffer.

Org mode supports this style by adapting the workings of the command `C-c C-t`[3]. When used several times in succession, it will still cycle through all names, in order to first select the right type for a task. But when you return to the item after some time and execute `C-c C-t` again, it will switch from any 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 numeric prefix argument as well when creating the global TODO list: `C-3 C-c t`.

5.2.3 Multiple keyword sets in one file

Sometimes you may want to use different sets of TODO keywords in parallel. For example, you may want to have the basic TODO/DONE, but also a workflow for bug fixing, and a separate state indicating that an item has been canceled (so it is not DONE, but also does not require action). Your setup would then look like this:

```
(setq org-todo-keywords
      '((sequence "TODO" "|" "DONE")
        (sequence "REPORT" "BUG" "KNOWNCAUSE" "|" "FIXED")
        (sequence "|" "CANCELED")))
```

The keywords should all be different, this helps Org mode to keep track of which subsequence should be used for a given entry. In this setup, `C-c C-t` only operates within a subsequence, so it switches from DONE to (nothing) to TODO, and from FIXED to (nothing) to REPORT. Therefore you need a mechanism to initially select the correct sequence. Besides the obvious ways like typing a keyword or using completion, you may also apply the following commands:

`C-S-⟨right⟩`
`C-S-⟨left⟩`   These keys jump from one TODO subset to the next. In the above example, `C-S-⟨right⟩` would jump from TODO or DONE to REPORT, and any of the words in the second row to CANCELED.

`S-⟨right⟩`
`S-⟨left⟩`   `S-⟨left⟩` and `S-⟨right⟩` and walk through *all* keywords from all sets, so for example `S-⟨right⟩` would switch from DONE to REPORT in the example above.

5.2.4 Fast access to TODO states

If you would like to quickly change an entry to an arbitrary TODO state instead of cycling through the states, you can set up keys for single-letter access to the states. This is done by adding the section key after each keyword, in parenthesis. For example:

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "|" "DONE(d)")
        (sequence "REPORT(r)" "BUG(b)" "KNOWNCAUSE(k)" "|" "FIXED(f)")
        (sequence "|" "CANCELED(c)")))
```

---

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

If you then press `C-u C-c C-t` followed by the selection key, the entry will be switched
to this state. (SPC) can be used to remove any TODO keyword from an entry. Should you
like this way of selecting TODO states a lot, you might want to set the variable `org-use-fast-todo-selection` to `t` and make this behavior the default. Check also the variable
`org-fast-tag-selection-include-todo`, it allows to change the TODO state through the
tags interface (see Section 6.2 [Setting tags], page 41), in case you like to mingle the two
concepts.

5.2.5 Setting up keywords for individual files

It can be very useful to use different aspects of the TODO mechanism in different files.
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 CANCELED
```

or

```
#+TYP_TODO: Fred Sara Lucy Mike | DONE
```

A setup for using several sets in parallel would be:

```
#+SEQ_TODO: TODO | DONE
#+SEQ_TODO: REPORT BUG KNOWNCAUSE | FIXED
#+SEQ_TODO: | CANCELED
```

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

Remember that the keywords after the vertical bar (or the last keyword if no bar is
there) must always mean that the item is DONE (although you may use a different word).
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[4].

5.2.6 Faces for TODO keywords

Org mode highlights TODO keywords with special faces: `org-todo` for keywords indi-
cating that an item still has to be acted upon, and `org-done` for keywords indicating that an
item is finished. If you are using more than 2 different states, you might want to use special
faces for some of them. This can be done using the variable `org-todo-keyword-faces`. For
example:

```
(setq org-todo-keyword-faces
      '(("TODO"      . org-warning)
        ("DEFERRED"  . shadow)
        ("CANCELED"  . (:foreground "blue" :weight bold))))
```

While using a list with face properties as shown for CANCELED *should* work, this does
not aways seem to be the case. If necessary, define a special face and use that.

---

[4] 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.

5.3 Progress logging

Org mode can automatically record a time stamp and possibly a note when you mark a TODO item as DONE, or even each time you change the state of a TODO item. This system is highly configurable, settings can be on a per-keyword basis and can be localized to a file or even a subtree. For information on how to clock working time for a task, see Section 8.4 [Clocking work time], page 58.

5.3.1 Closing items

The most basic logging is to keep track of *when* a certain TODO item was finished. This is achieved with[1].

```
(setq org-log-done 'time)
```

Then each time you turn an entry from a TODO (not-done) state into any of the DONE states, a line 'CLOSED: [timestamp]' will be inserted just after the headline. If you turn the entry back into a TODO item through further state cycling, that line will be removed again. If you want to record a note along with the timestamp, use[2]

```
(setq org-log-done 'note)
```

You will then be prompted for a note, and that note will be stored below the entry with a 'Closing Note' heading.

In the timeline (see Section 10.3.4 [Timeline], page 70) and in the agenda (see Section 10.3.1 [Weekly/daily agenda], page 68), you can then use the l key to display the TODO items with a 'CLOSED' timestamp on each day, giving you an overview of what has been done.

5.3.2 Tracking TODO state changes

When TODO keywords are used as workflow states (see Section 5.2.1 [Workflow states], page 34), you might want to keep track of when a state change occurred and maybe take a note about this change. Since it is normally too much to record a note for every state, Org mode expects configuration on a per-keyword basis for this. This is achieved by adding special markers '!' (for a time stamp) and '@' (for a note) in parenthesis after each keyword. For example, with the setting

```
(setq org-todo-keywords
  '((sequence "TODO(t)" "WAIT(w@/!)" "|" "DONE(d!)" "CANCELED(c@)")))
```

you not only define global TODO keywords and fast access keys, but also request that a time is recorded when the entry is turned into DONE[3], and that a note is recorded when switching to WAIT or CANCELED. The setting for WAIT is even more special: The '!' after the slash means that in addition to the note taken when entering the state, a time

---

[1] The corresponding in-buffer setting is: #+STARTUP: logdone

[2] The corresponding in-buffer setting is: #+STARTUP: lognotedone

[3] It is possible that Org mode will record two time stamps when you are using both org-log-done and state change logging. However, it will never prompt for two notes - if you have configured both, the state change recording note will take precedence and cancel the 'Closing Note'.

stamp should be recorded when *leaving* the WAIT state, if and only if the *target* state does not configure logging for entering it. So it has no effect when switching from WAIT to DONE, because DONE is configured to record a timestamp only. But when switching from WAIT back to TODO, the '/!' in the WAIT setting now triggers a timestamp even though TODO has no logging configured.

You can use the exact same syntax for setting logging preferences local to a buffer:

```
#+SEQ_TODO: TODO(t) WAIT(w@/!) | DONE(d!) CANCELED(c@)
```

In order to define logging settings that are local to a subtree or a single item, define a LOGGING property in this entry. Any non-empty LOGGING property resets all logging settings to nil. You may then turn on logging for this specific tree using STARTUP keywords like `lognotedone` or `logrepeat`, as well as adding state specific settings like `TODO(!)`. For example

```
* TODO Log each state with only a time
  :PROPERTIES:
  :LOGGING: TODO(!) WAIT(!) DONE(!) CANCELED(!)
  :END:
* TODO Only log when switching to WAIT, and when repeating
  :PROPERTIES:
  :LOGGING: WAIT(@) logrepeat
  :END:
* TODO No logging at all
  :PROPERTIES:
  :LOGGING: nil
  :END:
```

5.4 Priorities

If you use Org mode extensively, you may end up enough TODO items that it starts to make sense to prioritize them. Prioritizing can be done by placing a *priority cookie* into the headline of a TODO item, like this

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

By default, Org mode supports three 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 10.3.1 [Weekly/daily agenda], page 68); outside the agenda, they have no inherent meaning to Org mode.

Priorities can be attached to any outline tree entries; they do not need to be TODO items.

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 10.5 [Agenda commands], page 73).

```
S-⟨up⟩
S-⟨down⟩
```
Increase/decrease priority of current headline[4]. Note that these keys are also used to modify time stamps (see Section 8.2 [Creating timestamps], page 53). Furthermore, these keys are also used by CUA mode (see Section 14.7.2 [Conflicts], page 113).

You can change the range of allowed priorities by setting the variables `org-highest-priority`, `org-lowest-priority`, and `org-default-priority`. For an individual buffer, you may set these values (highest, lowest, default) like this (please make sure that the highest priority is earlier in the alphabet than the lowest priority):

```
#+PRIORITIES: A C B
```

5.5 Breaking tasks down into subtasks

It is often advisable to break down large tasks into smaller, manageable subtasks. You can do this by creating an outline tree below a TODO item, with detailed subtasks on the tree[5]. To keep the overview over the fraction of subtasks that are already completed, insert either '[/]' or '[%]' anywhere in the headline. These cookies will be updates each time the todo status of a child changes. For example:

```
* Organize Party [33%]
** TODO Call people [1/2]
*** TODO Peter
*** DONE Sarah
** TODO Buy food
** DONE Talk to neighbor
```

If you would like a TODO entry to automatically change to DONE when all chilrden are done, you can use the following setup:

```
(defun org-summary-todo (n-done n-not-done)
  "Switch entry to DONE when all subentries are done, to TODO otherwise."
  (let (org-log-done org-log-states)   ; turn off logging
    (org-todo (if (= n-not-done 0) "DONE" "TODO"))))

(add-hook 'org-after-todo-statistics-hook 'org-summary-todo)
```

Another possibility is the use of checkboxes to identify (a hierarchy of) a large number of subtasks (see Section 5.6 [Checkboxes], page 39).

5.6 Checkboxes

Every item in a plain list (see Section 2.8 [Plain lists], page 11) can be made into a checkbox by starting it with the string '[ ]'. This feature is similar to TODO items (see Chapter 5 [TODO Items], page 33), but is 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

---

[4] See also the option `org-priority-start-cycle-with-default'`.

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

steps. Or you can use them in a shopping list. To toggle a checkbox, use `C-c C-c`, or use the mouse (thanks to Piotr Zielinski's '`org-mouse.el`').

Here is an example of a checkbox list.

```
* TODO Organize party [2/4]
  - [-] call people [1/3]
    - [ ] Peter
    - [X] Sarah
    - [ ] Sam
  - [X] order food
  - [ ] think about what music to play
  - [X] talk to the neighbors
```

Checkboxes work hierarchically, so if a checkbox item has children that are checkboxes, toggling one of the children checkboxes will make the parent checkbox reflect if none, some, or all of the children are checked.

The '`[2/4]`' and '`[1/3]`' in the first and second line are cookies indicating how many checkboxes present in this entry have been checked off, and the total number of checkboxes are present. 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 the headline/item on which the cookie appear. You have to insert the cookie yourself by typing either '`[/]`' or '`[%]`'. With '`[/]`' you get an '`n out of m`' result, as in the examples above. With '`[%]`' 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. With a prefix argument, set it to '`[-]`', which is considered to be an intermediate state.

`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 there is 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 11).

`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 Tags

An excellent way to implement labels and contexts for cross-correlating information is to assign *tags* to headlines. Org mode has extensive support for tags.

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

6.1 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:`' even though the final heading is not explicitly marked with those tags. You can also set tags that all entries in a file should inherit as if these tags would be defined in a hypothetical level zero that surounds the entire file.

```
#+FILETAGS: :Peter:Boss:Secret:
```

To limit tag inheritance to specific tags, or to turn it off entirely, use the variable `org-use-tag-inheritance`.

When a headline matches during a tags search while tag inheritance is turned on, all the sublevels in the same tree will match as well[1]. The list of matches may then become very long. If you only want to see the first tags match in a subtree, configure the variable `org-tags-match-list-sublevels`.

6.2 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 33).

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

---

[1] This is only true if the the search does not involve more complex tests including properties (see Section 7.3 [Property searches], page 47).

```
#+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, add an empty TAGS option line to that file:

```
#+TAGS:
```

By default Org mode uses the standard minibuffer completion facilities for entering tags. However, it also implements another, quicker, tag selection method called *fast tag selection*. This allows you to select and deselect tags with just a single key press. For this to work well you should assign unique letters to most of your commonly used tags. You can do this globally by configuring the variable `org-tag-alist` in your '.emacs' file. For example, you may find the need to tag many items in different files with ':@home:'. In this case you can set something like:

```
(setq org-tag-alist '(("@work" . ?w) ("@home" . ?h) ("laptop" . ?l)))
```

If the tag is only relevant to the file you are working on then you can, instead, set the TAGS option line as:

```
#+TAGS: @work(w)  @home(h)  @tennisclub(t)  laptop(l)  pc(p)
```

You can also group together tags that are mutually exclusive. By using braces, as in:

```
#+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. Multiple such groups are allowed.

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

To set these mutually exclusive groups in the variable `org-mode-alist` you must use the dummy tags `:startgroup` and `:endgroup` instead of the braces. The previous example would be set globally by the following configuration:

```
(setq org-tag-alist '((:startgroup . nil)
                       ("@work" . ?w) ("@home" . ?h)
                       ("@tennisclub" . ?t)
                       (:endgroup . nil)
                       ("laptop" . ?l) ("pc" . ?p)))
```

If at least one tag has a selection key then 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 valid 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.

---

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

`C-g`   Abort without installing changes.

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

`!`   Turn off 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). If you are using expert
             mode, the first `C-c` will display the selection window.

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 key press 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 (in effect: start selection with `C-c C-c C-c` instead of `C-c C-c`). If you set
the variable to the value `expert`, the special window is not even shown for single-key tag
selection, it comes up only when you press an extra `C-c`.

6.3 Tag searches

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

`C-c \`
`C-c / T`   Create a sparse tree with all headlines matching a tags search. With a `C-u`
             prefix argument, ignore headlines that are not a TODO line.

`C-c a m`   Create a global list of tag matches from all agenda files. See Section 10.3.3
             [Matching tags and properties], page 70.

`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 headlines tagged '`:work:`', but discard those also tagged '`:boss:`'.

'`work|laptop`'
             Selects lines tagged '`:work:`' or '`:laptop:`'.

'`work|laptop&night`'
             Like before, 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 34), 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. To make sure that only lines are checked that actually have any TODO keyword, use `C-c a M`, or equivalently start the TODO part after the slash with '`!`'. 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`'.

Any element of the tag/todo match can be a regular expression - in this case it must be enclosed in curly braces. For example, '`work+{^boss.*}`' matches headlines that contain the tag '`:work:`' and any tag *starting* with '`boss`'.

You can also require a headline to be of a certain level or category, by writing instead of any TAG an expression like '`LEVEL=3`' or '`CATEGORY="work"`', respectively. For example, a search '`+LEVEL=3+boss/-DONE`' lists all level three headlines that have the tag '`boss`' and are *not* marked with the TODO keyword DONE.

7 Properties and Columns

Properties are a set of key-value pairs associated with an entry. There are two main applications for properties in Org mode. First, properties are like tags, but with a value. Second, you can use properties to implement (very basic) database capabilities in an Org buffer. For an example of the first application, imagine maintaining a file where you document bugs and plan releases of a piece of software. Instead of using tags like `:release_1:`, `:release_2:`, one can use a property, say `:Release:`, that in different subtrees has different values, such as `1.0` or `2.0`. For an example of the second application of properties, imagine keeping track of your music CD's, where properties could be things such as the album artist, date of release, number of tracks, and so on.

Properties can be conveniently edited and viewed in column view (see Section 7.5 [Column view], page 48).

Properties are like tags, but with a value. For example, in a file where you document bugs and plan releases of a piece of software, instead of using tags like `:release_1:`, `:release_2:`, it can be more efficient to use a property `:Release:` with a value `1.0` or `2.0`. Second, you can use properties to implement (very basic) database capabilities in an Org buffer, for example to create a list of Music CD's you own. You can edit and view properties conveniently in column view (see Section 7.5 [Column view], page 48).

7.1 Property syntax

Properties are key-value pairs. They need to be inserted into a special drawer (see Section 2.9 [Drawers], page 13) with the name `PROPERTIES`. Each property is specified on a single line, with the key (surrounded by colons) first, and the value after it. Here is an example:

```
* CD collection
** Classic
*** Goldberg Variations
    :PROPERTIES:
    :Title:     Goldberg Variations
    :Composer:  J.S. Bach
    :Artist:    Glen Gould
    :Publisher: Deutsche Grammphon
    :NDisks:    1
    :END:
```

You may define the allowed values for a particular property ':`Xyz`:' by setting a property ':`Xyz_ALL`:'. This special property is *inherited*, so if you set it in a level 1 entry, it will apply to the entire tree. When allowed values are defined, setting the corresponding property becomes easier and is less prone to typing errors. For the example with the CD collection, we can predefine publishers and the number of disks in a box like this:

```
* CD collection
  :PROPERTIES:
  :NDisks_ALL:  1 2 3 4
  :Publisher_ALL: "Deutsche Grammophon" Phillips EMI
```

```
      :END:
```

If you want to set properties that can be inherited by any entry in a file, use a line like

```
      #+PROPERTY: NDisks_ALL 1 2 3 4
```

Property values set with the global variable `org-global-properties` can be inherited by all entries in all Org files.

The following commands help to work with properties:

`M-`⟨TAB⟩   After an initial colon in a line, complete property keys. All keys used in the current file will be offered as possible completions.

`C-c C-x p`   Set a property. This prompts for a property name and a value. If necessary, the property drawer is created as well.

`M-x org-insert-property-drawer`
         Insert a property drawer into the current entry. The drawer will be inserted early in the entry, but after the lines with planning information like deadlines.

`C-c C-c`   With the cursor in a property drawer, this executes property commands.

`C-c C-c s`   Set a property in the current entry. Both the property and the value can be inserted using completion.

`S-`⟨left⟩`/`⟨right⟩
         Switch property at point to the next/previous allowed value.

`C-c C-c d`   Remove a property from the current entry.

`C-c C-c D`   Globally remove a property, from all entries in the current file.

`C-c C-c c`   Compute the property at point, using the operator and scope from the nearest column format definition.

7.2 Special properties

Special properties provide alternative access method to Org mode features discussed in the previous chapters, like the TODO state or the priority of an entry. This interface exists so that you can include these states into columns view (see Section 7.5 [Column view], page 48), or to use them in queries. The following property names are special and should not be used as keys in the properties drawer:

`TODO`	The TODO keyword of the entry.
`TAGS`	The tags defined directly in the headline.
`ALLTAGS`	All tags, including inherited ones.
`PRIORITY`	The priority of the entry, a string with a single letter.
`DEADLINE`	The deadline time string, without the angular brackets.
`SCHEDULED`	The scheduling time stamp, without the angular brackets.
`TIMESTAMP`	The first keyword-less time stamp in the entry.
`TIMESTAMP_IA`	The first inactive time stamp in the entry.
`CLOCKSUM`	The sum of CLOCK intervals in the subtree. `org-clock-sum` must be run first to compute the values.

7.3 Property searches

To create sparse trees and special lists with selection based on properties, the same commands are used as for tag searches (see Section 6.3 [Tag searches], page 43), and the same logic applies. For example, here is a search string:

```
+work-boss+PRIORITY="A"+Coffee="unlimited"+Effort<2+With={Sarah\|Denny}
```

If the comparison value is a plain number, a numerical comparison is done, and the allowed operators are '<', '=', '>', '<=', '>=', and '<>'. If the comparison value is enclosed in double quotes, a string comparison is done, and the same operators are allowed. If the comparison value is enclosed in curly braces, a regexp match is performed, with '=' meaning that the regexp matches the property value, and '<>' meaning that it does not match. So the search string in the example finds entries tagged ':work:' but not ':boss:', which also have a priority value 'A', a ':Coffee:' property with the value 'unlimited', an 'Effort' property that is numerically smaller than 2, and a ':With:' property that is matched by the regular expression 'Sarah\|Denny'.

You can configure Org mode to use property inheritance during a search, but beware that this can slow down searches considerably. See Section 7.4 [Property inheritance], page 47 for details.

There is also a special command for creating sparse trees based on a single property:

C-c / p   Create a sparse tree based on the value of a property. This first prompts for the name of a property, and then for a value. A sparse tree is created with all entries that define this property with the given value. If you enclose the value into curly braces, it is interpreted as a regular expression and matched against the property values.

7.4 Property Inheritance

The outline structure of Org mode documents lends itself for an inheritance model of properties: If the parent in a tree has a certain property, the children can inherit this property. Org mode does not turn this on by default, because it can slow down property searches significantly and is often not needed. However, if you find inheritance useful, you can turn it on by setting the variable `org-use-property-inheritance`. It may be set to `t`, to make all properties inherited from the parent, to a list of properties that should be inherited, or to a regular expression that matches inherited properties.

Org mode has a few properties for which inheritance is hard-coded, at least for the special applications for which they are used:

COLUMNS   The `:COLUMNS:` property defines the format of column view (see Section 7.5 [Column view], page 48). It is inherited in the sense that the level where a `:COLUMNS:` property is defined is used as the starting point for a column view table, independently of the location in the subtree from where columns view is turned on.

CATEGORY   For agenda view, a category set through a `:CATEGORY:` property applies to the entire subtree.

ARCHIVE   For archiving, the :ARCHIVE: property may define the archive location for the
            entire subtree (see Section 2.6.2 [Moving subtrees], page 9).

LOGGING   The LOGGING property may define logging settings for an entry or a subtree
            (see Section 5.3.2 [Tracking TODO state changes], page 37).

7.5 Column view

A great way to view and edit properties in an outline tree is *column view*. In column
view, each outline item is turned into a table row. Columns in this table provide access to
properties of the entries. Org mode implements columns by overlaying a tabular structure
over the headline of each item. While the headlines have been turned into a table row, you
can still change the visibility of the outline tree. For example, you get a compact table by
switching to CONTENTS view (*S-*⟨TAB⟩ *S-*⟨TAB⟩, or simply *c* while column view is active),
but you can still open, read, and edit the entry below each headline. Or, you can switch to
column view after executing a sparse tree command and in this way get a table only for the
selected items. Column view also works in agenda buffers (see Chapter 10 [Agenda Views],
page 66) where queries have collected selected items, possibly from a number of files.

7.5.1 Defining columns

Setting up a column view first requires defining the columns. This is done by defining a
column format line.

7.5.1.1 Scope of column definitions

To define a column format for an entire file, use a line like

```
#+COLUMNS: %25ITEM %TAGS %PRIORITY %TODO
```

To specify a format that only applies to a specific tree, add a :COLUMNS: property to the
top node of that tree, for example:

```
** Top node for columns view
   :PROPERTIES:
   :COLUMNS: %25ITEM %TAGS %PRIORITY %TODO
   :END:
```

If a :COLUMNS: property is present in an entry, it defines columns for the entry itself,
and for the entire subtree below it. Since the column definition is part of the hierarchical
structure of the document, you can define columns on level 1 that are general enough for
all sublevels, and more specific columns further down, when you edit a deeper part of the
tree.

7.5.1.2 Column attributes

A column definition sets the attributes of a column. The general definition looks like
this:

```
      %[width]property[(title)][{summary-type}]
```

Except for the percent sign and the property name, all items are optional. The individual parts have the following meaning:

`width`	An integer specifying the width of the column in characters. If omitted, the width will be determined automatically.
`property`	The property that should be edited in this column.
`(title)`	The header text for the column. If omitted, the property name is used.
`{summary-type}`	The summary type. If specified, the column values for parent nodes are computed from the children. Supported summary types are:

`{+}`	Sum numbers in this column.
`{+;%.1f}`	Like '+', but format result with '`%.1f`'.
`{$}`	Currency, short for '`+;%.2f`'.
`{:}`	Sum times, HH:MM:SS, plain numbers are hours.
`{X}`	Checkbox status, [X] if all children are [X].
`{X/}`	Checkbox status, [n/m].
`{X%}`	Checkbox status, [n%].

Here is an example for a complete columns definition, along with allowed values.

```
:COLUMNS:  %20ITEM %9Approved(Approved?){X} %Owner %11Status \¹
                %10Time_Estimate{:} %CLOCKSUM
:Owner_ALL:    Tammy Mark Karl Lisa Don
:Status_ALL:   "In progress" "Not started yet" "Finished" ""
:Approved_ALL: "[ ]" "[X]"
```

The first column, '`%25ITEM`', means the first 25 characters of the item itself, i.e. of the headline. You probably always should start the column definition with the '`ITEM`' specifier. The other specifiers create columns '`Owner`' with a list of names as allowed values, for '`Status`' with four different possible values, and for a checkbox field '`Approved`'. When no width is given after the '`%`' character, the column will be exactly as wide as it needs to be in order to fully display all values. The '`Approved`' column does have a modified title ('`Approved?`', with a question mark). Summaries will be created for the '`Time_Estimate`' column by adding time duration expressions like HH:MM, and for the '`Approved`' column, by providing an '`[X]`' status if all children have been checked. The '`CLOCKSUM`' column is special, it lists the sum of CLOCK intervals in the subtree.

7.5.2 Using column view

Turning column view on and off

C-c C-x C-c

> Create the column view for the local environment. This command searches the hierarchy, up from point, for a `:COLUMNS:` property that defines a format. When one is found, the column view table is established for the entire tree, starting from the entry that contains the `:COLUMNS:` property. If none is found,

---

[1] Please note that the COLUMNS definition must be on a single line - it is wrapped here only because of formatting constraints.

the format is taken from the `#+COLUMNS` line or from the variable `org-columns-default-format`, and column view is established for the current entry and its subtree.

`r`   Recreate the column view, to include recent changes made in the buffer.

`g`   Same as `r`.

`q`   Exit column view.

Editing values

⟨left⟩ ⟨right⟩ ⟨up⟩ ⟨down⟩
             Move through the column view from field to field.

`S-`⟨left⟩`/`⟨right⟩
             Switch to the next/previous allowed value of the field. For this, you have to have specified allowed values for a property.

`1..9,0`   Directly select the nth allowed value, `0` selects the 10th value.
`n / p`   Same as `S-`⟨left⟩`/`⟨right⟩

`e`   Edit the property at point. For the special properties, this will invoke the same interface that you normally use to change that property. For example, when editing a TAGS property, the tag completion or fast selection interface will pop up.

`C-c C-c`   When there is a checkbox at point, toggle it.

`v`   View the full value of this property. This is useful if the width of the column is smaller than that of the value.

`a`   Edit the list of allowed values for this property. If the list is found in the hierarchy, the modified values is stored there. If no list is found, the new value is stored in the first entry that is part of the current column view.

Modifying the table structure

`< / >`   Make the column narrower/wider by one character.

`S-M-`⟨right⟩   Insert a new column, to the right of the current column.

`S-M-`⟨left⟩   Delete the current column.

7.5.3 Capturing column view

Since column view is just an overlay over a buffer, it cannot be exported or printed directly. If you want to capture a column view, use this `columnview` dynamic block (see Section B.3 [Dynamic blocks], page 123). The frame of this block looks like this:

```
* The column view
#+BEGIN: columnview :hlines 1 :id "label"

#+END:
```

This dynamic block has the following parameters:

`:id`   This is most important parameter. Column view is a feature that is often localized to a certain (sub)tree, and the capture block might be in a different location in the file. To identify the tree whose view to capture, you can use 3 values:

`local`	use the tree in which the capture block is located
`global`	make a global view, including all headings in the file
`"label"`	call column view in the tree that has an `:ID:` property with the value *label*. You can use `M-x org-id-copy` to create a globally unique ID for the current entry and copy it to the kill-ring.

`:hlines`   When `t`, insert a hline after every line. When a number N, insert a hline before each headline with level `<= N`.

`:vlines`   When set to `t`, enforce column groups to get vertical lines.

`:maxlevel`
               When set to a number, don't capture entries below this level.

`:skip-empty-rows`
               When set to `t`, skip row where the only non-empty specifier of the column view is `ITEM`.

The following commands insert or update the dynamic block:

`C-c C-x r`   Insert a dynamic block capturing a column view. You will be prompted for the scope or id of the view.

`C-c C-c`
`C-c C-x C-u`
               Update dynamical block at point. The cursor needs to be in the `#+BEGIN` line of the dynamic block.

`C-u C-c C-x C-u`
               Update all dynamic blocks (see Section B.3 [Dynamic blocks], page 123). This is useful if you have several clock table blocks in a buffer.

7.6 The Property API

There is a full API for accessing and changing properties. This API can be used by Emacs Lisp programs to work with properties and to implement features based on them. For more information see Section B.5 [Using the property API], page 125.

8 Dates and Times

To assist project planning, TODO items can be labeled with a date and/or a time. The specially formatted string carrying the date and time information is called a *timestamp* in Org mode. This may be a little confusing because timestamp is often used as indicating when something was created or last changed. However, in Org mode this term is used in a much wider sense.

8.1 Timestamps, deadlines and scheduling

A time stamp is a specification of a date (possibly with time or a range of times) in a special format, either '`<2003-09-16 Tue>`' or '`<2003-09-16 Tue 09:39>`' or '`<2003-09-16 Tue 12:00-12:30>`'[1]. A time stamp can appear anywhere in the headline or body of an Org tree entry. Its presence causes entries to be shown on specific dates in the agenda (see). We distinguish:

Plain time stamp; Event; Appointment
> A simple time stamp just assigns a date/time to an item. This is just like writing down an appointment or event in a paper agenda. 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>
> * Discussion on climate change <2006-11-02 Thu 20:00-22:00>
> ```

Time stamp with repeater interval
> A time stamp may contain a *repeater interval*, indicating that it applies not only on the given date, but again and again after a certain interval of N days (d), weeks (w), months(m), or years(y). The following will show up in the agenda every Wednesday:
>
> ```
> * Pick up Sam at school <2007-05-16 Wed 12:30 +1w>
> ```

Diary-style sexp entries
> For more complex date specifications, Org mode supports using the special sexp diary entries implemented in the Emacs calendar/diary package. For example
>
> ```
> * The nerd meeting on every 2nd Thursday of the month
>   <%%(diary-float t 4 2)>
> ```

Time/Date range
> Two time stamps connected by '`--`' denote a 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>
> ```

---

[1] This is the standard ISO date/time format. To use an alternative format, see Section 8.2.2 [Custom time format], page 55.

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 they do *not* trigger an entry to show up in the agenda.

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

8.2 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 that will not cause an agenda entry.

`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, go to 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 10.3.1 [Weekly/daily agenda], page 68).

`S-`⟨left⟩

`S-`⟨right⟩   Change date at cursor by one day. These key bindings conflict with CUA mode (see Section 14.7.2 [Conflicts], page 113).

`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 5.4 [Priorities], page 38). The key bindings also conflict with CUA mode (see Section 14.7.2 [Conflicts], page 113).

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

8.2.1 The date/time prompt

When Org mode prompts for a date/time, the default is shown as an ISO date, and the prompt therefore seems to ask for an ISO date. But it will in fact accept any string containing some date and/or time information, and it is really smart about interpreting your input. 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 derive anything you have not specified from the *default date and time*. The default is usually the current date and time, but when modifying an existing time stamp, or when entering the second stamp of a range, it is taken from the stamp in the buffer. When filling in information, Org mode assumes that most of the time you will want to enter a date in the future: If you omit the month/year and the given day/month is *before* today, it will assume that you mean a future date[2].

For example, lets assume that today is **June 13, 2006**. Here is how various inputs will be interpreted, the items filled in by Org mode are in **bold**.

```
3-2-5          --> 2003-02-05
14             --> 2006-06-14
12             --> 2006-07-12
Fri            --> nearest Friday (defaultdate or later)
sep 15         --> 2006-11-15
feb 15         --> 2007-02-15
sep 12 9       --> 2009-09-12
12:45          --> 2006-06-13 12:45
22 sept 0:34   --> 2006-09-22 0:34
w4             --> ISO week for of the current year 2006
2012 w4 fri    --> Friday of ISO week 4 in 2012
2012-w04-5     --> Same as above
```

Furthermore you can specify a relative date by giving, as the *first* thing in the input: a plus/minus sign, a number and a letter [dwmy] to indicate change in days weeks, months, years. With a single plus or minus, the date is always relative to today. With a double plus or minus, it is relative to the default date. If instead of a single letter, you use the abbreviation of day name, the date will be the nth such day. E.g.

```
+0             --> today
.              --> today
+4d            --> four days from today
+4             --> same as above
+2w            --> two weeks from today
++5            --> five days from default date
+2tue          --> second tuesday from now.
```

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[3]. When you exit the date prompt, either by clicking on a date in the calendar, or by pressing ⟨RET⟩, the date selected in the calendar will be combined with the information entered at the prompt. You can control the calendar fully from the minibuffer:

`> / <`	Scroll calendar forward/backward by one month.
`mouse-1`	Select date by clicking on it.
`S-`⟨right⟩`/`⟨left⟩	One day forward/backward.

---

[2] See the variable `org-read-date-prefer-future`.

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

S-⟨down⟩/⟨up⟩	One week forward/backward.
M-S-⟨right⟩/⟨left⟩	One month forward/backward.
⟨RET⟩	Choose date in calendar.

The actions of the date/time prompt may seem complex, but I assure you they will grow on you, and you will start getting annoyed by pretty much any other way of entering a date/time out there. To help you understand what is going on, the current interpretation of your input will be displayed live in the minibuffer[4].

8.2.2 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.
- If the time stamp contains a range of clock times or a repeater, these will not be overlayed, but remain in the buffer as they were.
- 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.

8.3 Deadlines and scheduling

A time stamp may be preceded by special keywords to facilitate planning:

DEADLINE
> Meaning: the task (most likely a TODO item, though not necessarily) is supposed to be finished on that date.
>
> On the deadline date, the task will be listed in the agenda. In addition, the agenda for *today* will carry a warning about the approaching or missed deadline,

---

[4] If you find this distracting, turn the display of with `org-read-date-display-live`.

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>
```

You can specify a different lead time for warnings for a specific deadlines using the following syntax. Here is an example with a warning period of 5 days `DEADLINE: <2004-02-29 Sun -5d>`.

SCHEDULED

Meaning: you are planning to start working on that task on the given date.

The headline will be listed under the given date[5]. 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>
```

Important: Scheduling an item in Org mode should *not* be understood in the same way that we understand *scheduling a meeting*. Setting a date for a meeting is just a simple appointment, you should mark this entry with a simple plain time stamp, to get this item shown on the date where it applies. This is a frequent mis-understanding from Org-users. In Org mode, *scheduling* means setting a date when you want to start working on an action item.

You may use time stamps with repeaters in scheduling and deadline entries. Org mode will issue early and late warnings based on the assumption that the time stamp represents the *nearest instance* of the repeater. However, the use of diary sexp entries like `<%%(diary-float t 42)>` in scheduling and deadline timestamps is limited. Org mode does not know enough about the internals of each sexp function to issue early and late warnings. However, it will show the item on each day where the sexp entry matches.

8.3.1 Inserting deadlines or schedules

The following commands allow to quickly insert a deadline or to schedule an item:

`C-c C-d`   Insert 'DEADLINE' keyword along with a stamp. The insertion will happen in the line directly following the headline. When called with a prefix arg, an existing deadline will be removed from the entry.

`C-c / d`   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 / d` 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

---

[5] It will still be listed on that date after it has been marked DONE. If you don't like this, set the variable `org-agenda-skip-scheduled-if-done`.

       removed. When called with a prefix argument, remove the scheduling date
       from the entry.

`C-c C-x C-k`

       Mark the current entry for agenda action. After you have marked the entry
       like this, you can open the agenda or the calendar to find an appropriate date.
       With the cursor on the selected date, press `k s` or `k d` to schedule the marked
       item.

8.3.2 Repeated tasks

Some tasks need to be repeated again and again. Org mode helps to organize such tasks
using a so-called repeater in a DEADLINE, SCHEDULED, or plain time stamp. In the
following example

```
** TODO Pay the rent
   DEADLINE: <2005-10-01 Sat +1m>
```

the `+1m` is a repeater; the intended interpretation is that the task has a deadline on
<2005-10-01> and repeats itself every (one) month starting from that time. If you need
both a repeater and a special warning period in a deadline entry, the repeater comes first
and the warning period last: `DEADLINE: <2005-10-01 Sat +1m -3d>`.

Deadlines and scheduled items produce entries in the agenda when they are over-due,
so it is important to be able to mark such an entry as completed once you have done so.
When you mark a DEADLINE or a SCHEDULE with the TODO keyword DONE, it will
no longer produce entries in the agenda. The problem with this is, however, that then also
the *next* instance of the repeated entry will not be active. Org mode deals with this in the
following way: When you try to mark such an entry DONE (using `C-c C-t`), it will shift
the base date of the repeating time stamp by the repeater interval, and immediately set
the entry state back to TODO. In the example above, setting the state to DONE would
actually switch the date like this:

```
** TODO Pay the rent
   DEADLINE: <2005-11-01 Tue +1m>
```

A timestamp[6] will be added under the deadline, to keep a record that you actually acted
on the previous instance of this deadline.

As a consequence of shifting the base date, this entry will no longer be visible in the
agenda when checking past dates, but all future instances will be visible.

With the '`+1m`' cookie, the date shift will always be exactly one month. So if you have not
payed the rent for three months, marking this entry DONE will still keep it as an overdue
deadline. Depending on the task, this may not be the best way to handle it. For example,
if you forgot to call you father for 3 weeks, it does not make sense to call him 3 times in a
single day to make up for it. Finally, there are tasks like changing batteries which should
always repeat a certain time *after* the last time you did it. For these tasks, Org mode has
special repeaters markers with '`++`' and '`.+`'. For example:

---

[6] You can change this using the option `org-log-repeat`, or the `#+STARTUP` options `logrepeat`,
`lognoterepeat`, and `nologrepeat`. With `lognoterepeat`, you will aslo be prompted for a note.

```
** TODO Call Father
   DEADLINE: <2008-02-10 Sun ++1w>
   Marking this DONE will shift the date by at least one week,
   but also by as many weeks as it takes to get this date into
   the future.  However, it stays on a Sunday, even if you called
   and marked it done on Saturday.
** TODO Check the batteries in the smoke detectors
   DEADLINE: <2005-11-01 Tue .+1m>
   Marking this DONE will shift the date to one month after
   today.
```

You may have both scheduling and deadline information for a specific task - just make sure that the repeater intervals on both are the same.

8.4 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. If this is not the first clocking of this item, the multiple CLOCK lines will be wrapped into a `:CLOCK:` drawer (see also the variable `org-clock-into-drawer`). When called with a `C-u` prefix argument, select the task from a list of recently clocked tasks. With two `C-u C-u` prefixes, clock into the task at point and mark it as the default task. The default task will always be available when selecting a clocking task, with letter `d`.

`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-note-clock-out` for the possibility to record an additional note together with the clock-out time stamp[7].

`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.

---

[7] The corresponding in-buffer setting is: `#+STARTUP: lognoteclock-out`

`C-c C-x C-j`

> Jump to the entry that contains the currently running clock. With a `C-u` prefix arg, select the target task from a list of recently clocked tasks.

`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 B.3 [Dynamic blocks], page 123) containing a clock report as an Org mode table into the current file. When the cursor is at an existing clock table, just update it. When called with a prefix argument, jump to the first clock report in the current document and update it.
>
>   #+BEGIN: clocktable :maxlevel 2 :emphasize nil :scope file
>   #+END: clocktable
>
> If such a block already exists at point, its content is replaced by the new table. The 'BEGIN' line can specify options:

`:maxlevel`	Maximum level depth to which times are listed in the table.
`:emphasize`	When `t`, emphasize level one and level two items
`:scope`	The scope to consider. This can be any of the following:

`nil`	the current buffer or narrowed region
`file`	the full current buffer
`subtree`	the subtree where the clocktable is located
`treeN`	the surrounding level N tree, for example `tree3`
`tree`	the surrounding level 1 tree
`agenda`	all agenda files
`("file"..)`	scan these files
`file-with-archives`	current file and its archives
`agenda-with-archives`	all agenda files, including archives

`:block`	The time block to consider. This block is specified either absolute, or relative to the current time and may be any of these formats:

`2007-12-31`	New year eve 2007
`2007-12`	December 2007
`2007-W50`	ISO-week 50 in 2007
`2007`	the year 2007
`today, yesterday, today-N`	a relative day
`thisweek, lastweek, thisweek-N`	a relative week
`thismonth, lastmonth, thismonth-N`	a relative month
`thisyear, lastyear, thisyear-N`	a relative year

> Use `S-`⟨left⟩/⟨right⟩ keys to shift the time interval.

`:tstart`	A time string specifying when to start considering times
`:tend`	A time string specifying when to stop considering times
`:step`	`week` or `day`, to split the table into chunks.

To use this, `:block` or `:tstart`, `:tend` are needed.

`:link`   Link the item headlines in the table to their origins

So to get a clock summary of the current level 1 tree, for the current day, you could write

```
#+BEGIN: clocktable :maxlevel 2 :block today :scope tree1 :link t
#+END: clocktable
```

and to use a specific time range you could write[8]

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

`C-c C-c`
`C-c C-x C-u`
            Update dynamical block at point. The cursor needs to be in the `#+BEGIN` line of the dynamic block.

`C-u C-c C-x C-u`
            Update all dynamic blocks (see Section B.3 [Dynamic blocks], page 123). This is useful if you have several clock table blocks in a buffer.

`S-⟨left⟩`
`S-⟨right⟩`   Shift the current `:block` interval and update the table. The cursor needs to be in the `#+BEGIN: clocktable` line for this command. If `:block` is `today`, it will be shifted to `today-1` etc.

The `l` key may be used in the timeline (see Section 10.3.4 [Timeline], page 70) and in the agenda (see Section 10.3.1 [Weekly/daily agenda], page 68) to show which tasks have been worked on or closed during a day.

8.5 Effort estimates

If you want to plan your work in a very detailed way, or if you need to produce offers with quotations of the estimated work effort, you may want to assign effort estimates to entries. If you are also clocking your work, you may later want to compare the planned effort with the actual working time, a great way to improve planning estimates. Effort estimates are stored in a special property 'Effort'[9]. Clearly the best way to work with effort estimates is through column view (see Section 7.5 [Column view], page 48). You should start by setting up discrete values for effort estimates, and a `COLUMNS` format that displays these values together with clock sums (if you want to clock your time). For a specific buffer you can use

```
#+PROPERTY: Effort_ALL 0 0:10 0:30 1:00 2:00 3:00 4:00 5:00 6:00 7:00 8:00
#+COLUMNS: %40ITEM(Task) %17Effort(Estimated Effort){:} %CLOCKSUM
```

or you can set up these values globally by customizing the variables `org-global-properties` and `org-columns-default-format`. In particular if you want to use this setup also in the agenda, a global setup may be advised.

---

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

[9] You may change the property being used with the variable `org-effort-property`.

The way to assign estimates to individual items is then to switch to column mode, and to use S-right and S-left to change the value. The values you enter will immediately be summed up in the hierarchy. In the column next to it, any clocked time will be displayed.

If you switch to column view in the daily/weekly agenda, the effort column will summarize the estimated work effort for each day[10], and you can use this to find space in your schedule. To get an overview of the entire part of the day that is committed, you can set the option `org-agenda-columns-add-appointments-to-effort-sum`. The appointments on a day that take place over a specified time interval will then also be added to the load estimate of the day.

---

[10] Please note the pitfalls of summing hierarchical data in a flat list (see Section 10.7 [Agenda column view], page 83).

9 Remember

The *Remember* package by John Wiegley lets you store quick notes with little interruption of your work flow. See <http://www.emacswiki.org/cgi-bin/wiki/RememberMode> for more information. It is an excellent way to add new notes and tasks to Org files. Org significantly expands the possibilities of *remember*: You may define templates for different note types, and associate target files and headlines with specific templates. It also allows you to select the location where a note should be stored interactively, on the fly.

9.1 Setting up Remember

The following customization will tell *remember* to use org files as target, and to create annotations compatible with Org links.

```
(org-remember-insinuate)
(setq org-directory "~/path/to/my/orgfiles/")
(setq org-default-notes-file (concat org-directory "/notes.org"))
(define-key global-map "\C-cr" 'org-remember)
```

The last line binds the command `org-remember` to a global key[1]. `org-remember` basically just calls `remember`, but it makes a few things easier: If there is an active region, it will automatically copy the region into the remember buffer. It also allows to jump to the buffer and location where remember notes are being stored: Just call `org-remember` with a prefix argument. If you use two prefix arguments, Org jumps to the location where the last remember note was stored.

You can also call `org-remember` in a special way from the agenda, using the `k r` key combination. With this access, any time stamps inserted by the selected remember template (see below) will default to the cursor date in the agenda, rather than to the current date.

9.2 Remember templates

In combination with Org, 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, another one for journal entries, and a third one for collecting random ideas, you could use:

```
(setq org-remember-templates
 '(("Todo" ?t "* TODO %?\n  %i\n  %a" "~/org/TODO.org" "Tasks")
   ("Journal" ?j "* %U %?\n\n  %i\n  %a" "~/org/JOURNAL.org")
   ("Idea" ?i "* %^{Title}\n  %i\n  %a" "~/org/JOURNAL.org" "New Ideas")))
```

In these entries, the first string is just a name, and the character specifies how to select the template. It is useful if the character is also the first letter of the name. The next string specifies the template. Two more (optional) strings give the file in which, and the headline under which the new note should be stored. The file (if not present or `nil`) defaults to `org-default-notes-file`, the heading to `org-remember-default-headline`. If the file name is not an absolute path, it will be interpreted relative to `org-directory`. The heading can

---

[1] Please select your own key, `C-c r` is only a suggestion.

also be the symbols `top` or `bottom` to send note as level 1 entries to the beginning or end of the file, respectively.

An optional sixth element specifies the contexts in which the user can select the template. This element can be a list of major modes or a function. `org-remember` will first check whether the function returns `t` or if we are in any of the listed major mode, and exclude templates fo which this condition is not fulfilled. Templates that do not specify this element at all, or that use `nil` or `t` as a value will always be selectable.

So for example:

```
(setq org-remember-templates
 '(("Bug" ?b "* BUG %?\n  %i\n  %a" "~/org/BUGS.org" "Bugs" (emacs-lisp-mode))
   ("Journal" ?j "* %U %?\n\n  %i\n  %a" "~/org/JOURNAL.org" "X" my-check)
   ("Idea" ?i "* %^{Title}\n  %i\n  %a" "~/org/JOURNAL.org" "New Ideas")))
```

The first template will only be available when invoking `org-remember` from an buffer in `emacs-lisp-mode`. The second template will only be available when the function `my-check` returns `t`. The third template will be proposed in any context.

When you call `M-x org-remember` (or `M-x remember`) to remember something, Org will prompt for a key to select the template (if you have more than one template) and then prepare the buffer like

```
* TODO
  [[file:link to where you called remember]]
```

During expansion of the template, special %-escapes allow dynamic insertion of content:

`%^{prompt}`	prompt the user for a string and replace this sequence with it. You may specify a default value and a completion table with %^{prompt|default|completion2|completion3...} The arrow keys access a prompt-specific history.
`%a`	annotation, normally the link created with `org-store-link`
`%A`	like `%a`, but prompt for the description part
`%i`	initial content, the region when remember is called with C-u. The entire text will be indented like `%i` itself.
`%t`	time stamp, date only
`%T`	time stamp with date and time
`%u, %U`	like the above, but inactive time stamps
`%^t`	like `%t`, but prompt for date. Similarly `%^T`, `%^u`, `%^U` You may define a prompt like `%^{Birthday}t`
`%n`	user name (taken from `user-full-name`)
`%c`	Current kill ring head.
`%x`	Content of the X clipboard.
`%^C`	Interactive selection of which kill or clip to use.
`%^L`	Like `%^C`, but insert as link.
`%^g`	prompt for tags, with completion on tags in target file.
`%^G`	prompt for tags, with completion all tags in all agenda files.
`%:keyword`	specific information for certain link types, see below
`%[pathname]`	insert the contents of the file given by `pathname`
`%(sexp)`	evaluate elisp `(sexp)` and replace with the result
`%!`	immediately store note after completing the template

> (skipping the `C-c C-c` that normally triggers storing)
> %&   jump to target location immediately after storing note

For specific link types, the following keywords will be defined[2]:

```
Link type            |  Available keywords
---------------------+------------------------------------------------
bbdb                 |  %:name %:company
bbdb                 |  %::server %:port %:nick
vm, wl, mh, rmail    |  %:type %:subject %:message-id
                     |  %:from %:fromname %:fromaddress
                     |  %:to   %:toname   %:toaddress
                     |  %:fromto (either "to NAME" or "from NAME")³
gnus                 |  %:group, for messages also all email fields
w3, w3m              |  %:url
info                 |  %:file %:node
calendar             |  %:date"
```

To place the cursor after template expansion use:

> %?   After completing the template, position cursor here.

If you change your mind about which template to use, call `org-remember` in the remember buffer. You may then select a new template that will be filled with the previous context information.

9.3 Storing notes

When you are finished preparing a note with *remember*, you have to press `C-c C-c` to file the note away. If you have started the clock in the remember buffer, you will first be asked if you want to clock out now[4]. If you answer `n`, the clock will continue to run after the note was filed away.

The handler will then store the note in the file and under the headline specified in the template, or it will use the default file and headlines. The window configuration will be restored, sending you back to the working context before the call to `remember`. To re-use the location found during the last call to `remember`, exit the remember buffer with `C-u C-u C-c C-c`, i.e. specify a double prefix argument to `C-c C-c`.

If you want to store the note directly to a different place, use `C-u C-c C-c` instead to exit remember[5]. The handler will then first prompt for a target file - if you press ⌊RET⌋, the value specified for the template is used. Then the command offers the headings tree of the selected file, with the cursor position at the default headline (if you had specified one in the template). You can either immediately press ⌊RET⌋ to get the note placed there. Or you can use the following keys to find a different location:

---

[2] If you define your own link types (see Section B.1 [Adding hyperlink types], page 117), any property you store with `org-store-link-props` can be accessed in remember templates in a similar way.

[3] This will always be the other, not the user. See the variable `org-from-is-user-regexp`.

[4] To avoid this query, configure the variable `org-remember-clock-out-on-exit`.

[5] Configure the variable `org-remember-store-without-prompt` to make this behavior the default.

⟨TAB⟩	Cycle visibility.
⟨down⟩ / ⟨up⟩	Next/previous visible headline.
n / p	Next/previous visible headline.
f / b	Next/previous headline same level.
u	One level up.

Pressing ⟨RET⟩ or ⟨left⟩ or ⟨right⟩ then leads to the following result.

Cursor position	Key	Note gets inserted
on headline	⟨RET⟩	as sublevel of the heading at cursor, first or last depending on `org-reverse-note-order`.
	⟨left⟩/⟨right⟩	as same level, before/after current heading
buffer-start	⟨RET⟩	as level 2 heading at end of file or level 1 at beginning depending on `org-reverse-note-order`.
not on headline	⟨RET⟩	at cursor position, level taken from context.

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 you have indented the text of the note below the headline, the indentation will be adapted if inserting the note into the tree requires demotion from level 1.

9.4 Refiling notes

Remember is usually used to quickly capture notes and tasks into one or a few capture lists. When reviewing the captured data, you may want to refile some of the entries into a different list, for example into a project. Cutting, finding the right location and then pasting the note is cumbersome. To simplify this process, you can use the following special command:

`C-c C-w` Refile the entry at point. This command offers possible locations for refiling the entry and lets you select one with completion. The item is filed below the target heading as a subitem. Depending on `org-reverse-note-order`, it will be either the first of last subitem.
By default, all level 1 headlines in the current buffer are considered to be targets, but you can have more complex definitions across a number of files. See the variable `org-refile-targets` for details. If you would like to select a location via a file-pathlike completion along the outline path, see the variable `org-refile-use-outline-path`.

`C-u C-c C-w`
Use the refile interface to jump to a heading.

`C-u C-u C-c C-w`
Jump to the location where `org-refile` last moved a tree to.

10 Agenda Views

Due to the way Org 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 of open action items, or of events that are important for a particular date, this information must be collected, sorted and displayed in an organized way.

Org can select items based on various criteria, and display them in a separate buffer. Seven 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*, showings headlines based on the tags associated with them,
- a *timeline view* that shows all events in a single Org file, in time-sorted view,
- a *keyword search view* that shows all entries from multiple files that contain specified keywords.
- a *stuck projects view* showing projects that currently don't move along, and
- *custom views* that are special tag/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 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`.

10.1 Agenda files

The information to be shown is normally collected from all *agenda files*, the files listed in the variable `org-agenda-files`[1]. If a directory is part of this list, all files with the extension '`.org`' in this directory will be part of the list.

Thus even if you only work with a single Org 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 a prefix argument, file is added/moved to the end.

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

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

---

[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 `<` before selecting a command will actually limit the command to the current file, and ignore `org-agenda-files` until the next dispatcher command.

`M-x org-iswitchb`
> Command to use an `iswitchb`-like interface to switch to and between Org buffers.

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

If you would like to focus the agenda temporarily onto a file not in this list, or onto just one file in the list or even only a subtree in a file, this can be done in different ways. For a single agenda command, you may press < once or several times in the dispatcher (see Section 10.2 [Agenda dispatcher], page 67). To restrict the agenda scope for an extended period, use the following commands:

`C-c C-x <` Permanently restrict the agenda to the current subtree. When with a prefix argument, or with the cursor before the first headline in a file, the agenda scope is set to the entire file. This restriction remains in effect until removed with `C-c C-x >`, or by typing either < or > in the agenda dispatcher. If there is a window displaying an agenda view, the new restriction takes effect immediately.

`C-c C-x <` Remove the permanent restriction created by `C-c C-x <`.

When working with 'Speedbar', you can use the following commands in the Speedbar frame:

< in the speedbar frame
> Permanently restrict the agenda to the item at the cursor in the Speedbar frame, either an Org file or a subtree in such a file. If there is a window displaying an agenda view, the new restriction takes effect immediately.

> in the speedbar frame
> Lift the restriction again.

10.2 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 10.3.1 [Weekly/daily agenda], page 68).

`t / T`   Create a list of all TODO items (see Section 10.3.2 [Global TODO list], page 69).

`m / M`   Create a list of headlines matching a TAGS expression (see Section 10.3.3 [Matching tags and properties], page 70).

`L`   Create the timeline view for the current buffer (see Section 10.3.4 [Timeline], page 70).

`s`   Create a list of entries selected by a boolean expression of keywords and/or regular expressions that must or must not occur in the entry.

/   Search for a regular expression in all agenda files and additionally in the files
               listed in `org-agenda-multi-occur-extra-files`. This uses the Emacs com-
               mand `multi-occur`. A prefix argument can be used to specify the number of
               context lines for each match, default is 1.

`# / !`   Create a list of stuck projects (see Section 10.3.6 [Stuck projects], page 71).

`<`   Restrict an agenda command to the current buffer[3]. After pressing `<`, you still
               need to press the character selecting the command.

`< <`   If there is an active region, restrict the following agenda command to the region.
               Otherwise, restrict it to the current subtree[4]. After pressing `< <`, 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 10.6 [Custom agenda views], page 77.

10.3 The built-in agenda views

In this section we describe the built-in views.

10.3.1 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 numeric prefix[5] (like `C-u 2 1 C-c a a`)
               you may set the number of days to be displayed (see also the variable `org-`
               `agenda-ndays`)

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 10.5 [Agenda commands], page 73.

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. It can be very
useful to combine output from Org with the diary.

---

[3] For backward compatibility, you can also press `1` to restrict to the current buffer.

[4] For backward compatibility, you can also press `0` to restrict to the current buffer.

[5] For backward compatibility, the universal prefix `C-u` causes all TODO entries to be listed before the
     agenda. This feature is deprecated, use the dedicated TODO list, or a block agenda instead.

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.

If you are using the diary only for sexp entries and holidays, it is faster to not use the above setting, but instead to copy or even move the entries into an Org file. Org mode evaluates diary-style sexp entries, and does it faster because there is no overhead for first creating the diary display. Note that the sexp entries must start at the left margin, no white space is allowed before them. For example, the following segment of an Org file will be processed and entries will be made in the agenda:

```
* Birthdays and similar stuff
#+CATEGORY: Holiday
%%(org-calendar-holiday)   ; special function for holiday names
#+CATEGORY: Ann
%%(diary-anniversary 14  5 1956) Arthur Dent is %d years old
%%(diary-anniversary  2 10 1869) Mahatma Gandhi would be %d years old
```

Appointment reminders

Org can interact with Emacs appointments notification facility.

To add all the appointments of your agenda files, use the command `org-agenda-to-appt`. This commands also lets you filter through the list of your appointments and add only those belonging to a specific category or matching a regular expression. See the docstring for details.

10.3.2 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 10 [Agenda Views], page 66) 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 10.5 [Agenda commands], page 73).

`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, and you may also specify several keywords by separating them with '|' as boolean OR operator. 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 10.2 [Agenda dispatcher], page 67).

Matching specific TODO keywords can also be done as part of a tags search (see Section 6.3 [Tag searches], page 43).

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 10.5 [Agenda commands], page 73.

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 8.1 [Timestamps], page 52) 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.

10.3.3 Matching tags and properties

If headlines in the agenda files are marked with *tags* (see Chapter 6 [Tags], page 41), 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 6 [Tags], page 41). If you often need a specific search, define a custom command for it (see Section 10.2 [Agenda dispatcher], page 67).

`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 6.3 [Tag searches], page 43.

The commands available in the tags list are described in Section 10.5 [Agenda commands], page 73.

10.3.4 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 10.5 [Agenda commands],
page 73.

10.3.5 Keyword search

This agenda view is a general text search facility for Org mode entries. It is particularly
useful to find notes.

`C-c a s`   This is a special search that lets you select entries by keywords or regular
             expression, using a boolean logic. For example, the search string

                   +computer +wifi -ethernet -{8\.11[bg]}

             will search for note entries that contain the keywords `computer` and `wifi`, but
             not the keyword `ethernet`, and which are also not matched by the regular
             expression `8\.11[bg]`, meaning to exclude both 8.11b and 8.11g.

             Note that in addition to the agenda files, this command will also search the files
             listed in `org-agenda-text-search-extra-files`.

10.3.6 Stuck projects

If you are following a system like David Allen's GTD to organize your work, one of the
"duties" you have is a regular review to make sure that all projects move along. A *stuck*
project is a project that has no defined next actions, so it will never show up in the TODO
lists Org mode produces. During the review, you need to identify such projects and define
next actions for them.

`C-c a #`   List projects that are stuck.

`C-c a !`   Customize the variable `org-stuck-projects` to define what a stuck project is
             and how to find it.

You almost certainly will have to configure this view before it will work for you. The
built-in default assumes that all your projects are level-2 headlines, and that a project is
not stuck if it has at least one entry marked with a TODO keyword TODO or NEXT or
NEXTACTION.

Lets assume that you, in your own way of using Org mode, identify projects with a tag
PROJECT, and that you use a TODO keyword MAYBE to indicate a project that should
not be considered yet. Lets further assume that the TODO keyword DONE marks finished
projects, and that NEXT and TODO indicate next actions. The tag @SHOP indicates
shopping and is a next action even without the NEXT tag. Finally, if the project contains
the special word IGNORE anywhere, it should not be listed either. In this case you would
start by identifying eligible projects with a tags/todo match '+PROJECT/-MAYBE-DONE', and
then check for TODO, NEXT, @SHOP, and IGNORE in the subtree to identify projects
that are not stuck. The correct customization for this is

```
(setq org-stuck-projects
      '("+PROJECT/-MAYBE-DONE" ("NEXT" "TODO") ("@SHOP")
                               "\\<IGNORE\\>"))
```

10.4 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 10.4.1 [Categories], page 72) 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.

10.4.1 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[6]:

    #+CATEGORY: Thesis

If you would like to have a special CATEGORY for a single entry or a (sub)tree, give the entry a `:CATEGORY:` property with the location as the value (see Chapter 7 [Properties and Columns], page 45).

The display in the agenda buffer looks best if the category is not longer than 10 characters.

10.4.2 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 10.3.1 [Weekly/daily agenda], page 68), 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
```

---

[6] For backward compatibility, the following also works: If there are several such lines in a file, each specifies the category for the text below it. The first category also applies to any text before the first CATEGORY line. However, using this method is *strongly* deprecated as it is incompatible with the outline structure of the document. The correct method for setting multiple categories in a buffer is using a property.

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`.

10.4.3 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 5.4 [Priorities], page 38), 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 5.4 [Priorities], page 38).

- 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`, and may also include criteria based on the estimated effort of an entry.

10.5 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⟩ and `C-p`).

`p`   Previous line (same as ⟨down⟩ and `C-n`).

View/Go to 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`.

`b`   Display the entire subtree of the current item in an indirect buffer. With a
             numeric prefix argument N, go up to level N and then take that tree. If N
             is negative, go up that many levels. With a `C-u` prefix, do not remove the
             previously used indirect buffer.

`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.

`v`   Toggle Archives mode. In archives mode, trees that are marked are also scanned
             when producing the agenda. When you call this command with a `C-u` prefix
             argument, even all archive files are included. To exit archives mode, press `v`
             again.

`R`   Toggle Clockreport mode. In clockreport mode, the daily/weekly agenda will al-
             ways show a table with the clocked times for the timespan and file scope covered
             by the current agenda view. The initial setting for this mode in new agenda
             buffers can be set with the variable `org-agenda-start-with-clockreport-`
             `mode`.

Change display

`o`   Delete other windows.

`d w m y`   Switch to day/week/month/year view. When switching to day or week view,
             this setting becomes the default for subsequent agenda commands. Since month
             and year views are slow to create, they do not become the default. A numeric
             prefix argument may be used to jump directly to a specific day of the year, ISO
             week, month, or year, respectively. For example, `32 d` jumps to February 1st, `9`
             `w` to ISO week number 9. When setting day, week, or month view, a year may
             be encoded in the prefix argument as well. For example, `200712 w` will jump to
             week 12 in 2007. If such a year specification has only one or two digits, it will
             be mapped to the interval 1938-2037.

`D`	Toggle the inclusion of diary entries. See Section 10.3.1 [Weekly/daily agenda], page 68.
`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.
`g`	Same as `r`.
`s` `C-x C-s`	Save all Org 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.
`.`	Go to today.
`C-c C-x C-c`	Invoke column view (see Section 7.5 [Column view], page 48) in the agenda buffer. The column view format is taken from the entry at point, or (if there is no entry at point), from the first entry in the agenda view. So whatever the format for that entry would be in the original buffer (taken from a property, from a `#+COLUMNS` line, or from the default variable `org-columns-default-format`), will be used in the agenda.

Query editing

`[ ] { }`	In the *search view* (see Section 10.3.5 [Keyword search], page 71), these keys add new search words (`[` and `]`) or new regular expressions (`{` and `}`) to the query string. The opening bracket/brace will add a positive search term prefixed by '+', indicating that this search term *must* occur/match in the entry. Closing bracket/brace add a negative search term which *must not* occur/match in the entry for it to be selected.

Remote editing

`0-9`	Digit argument.
`C-_`	Undo a change due to a remote editing command. The change is undone both in the agenda buffer and in the remote buffer.
`t`	Change the TODO state of the item, both in the agenda and in the original org file.
`C-k`	Delete the current agenda item along with the entire subtree belonging to it in the original Org file. If the text to be deleted remotely is longer than one line, the kill needs to be confirmed by the user. See variable `org-agenda-confirm-kill`.
`a`	Toggle the ARCHIVE tag for the current headline.

`A`	Move the subtree corresponding to the current entry to its *Archive Sibling*.
`$`	Archive the subtree corresponding to the current headline. This means the entry will be moved to the configured archive location, most likely a different 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. If there is an active region in the agenda, change a tag for all headings in the region.
`,`	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.
`k`	Agenda actions, to set dates for selected items to the cursor date. This command also works in the calendar! The command prompts for an additonal key:

> | | |
> |---|---|
> | m | Mark the entry at point for action. You can also make entries in Org files with `C-c C-x C-k`. |
> | d | Set the deadline of the marked entry to the date at point. |
> | s | Schedule the marked entry at the date at point. |
> | r | Call `org-remember` with the cursor date as default date. |

> Press `r` afterwards to refresh the agenda and see the effect of the command.

`S-`⟨right⟩	Change the time stamp associated with the current line by one day into the future. With a numeric 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.
`J`	Jump to the running clock in another window.

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.
`M-x org-export-icalendar-combine-agenda-files`	Export a single iCalendar file containing entries from all agenda files. This is a globally available command, and also available in the agenda menu.

Exporting to a file

`C-x C-w`	Write the agenda view to a file. Depending on the extension of the selected file name, the view will be exported as HTML (extension '`.html`' or '`.htm`'), Postscript (extension '`.ps`'), or plain text (any other extension). Use the variable `org-agenda-exporter-settings` to set options for '`ps-print`' and for '`htmlize`' to be used during export.

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.

10.6 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 10.2 [Agenda dispatcher], page 67), just like the default commands.

10.6.1 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\\>")
         ("h" . "HOME+Name tags searches") ; description for "h" prefix
         ("hl" tags "+home+Lisa")
         ("hp" tags "+home+Peter")
         ("hk" tags "+home+Kim")))
```

The initial string in each entry defines the keys you have to press after the dispatcher command `C-c a` in order to access the command. Usually this will be just a single character, but if you have many similar commands, you can also define two-letter combinations where the first character is the same in several combinations and serves as a prefix key[7]. 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`'

`C-c a h`   as a prefix command for a HOME tags search where you have to press an additional key (`l`, `p` or `k`) to select a name (Lisa, Peter, or Kim) as additional tag to match.

10.6.2 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:

---

[7] You can provide a description for a prefix key by inserting a cons cell with the prefix and the description.

```
(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.

10.6.3 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)))
        ("N" search ""
         ((org-agenda-files '("~org/notes.org"))
          (org-agenda-text-search-extra-files nil)))))
```

Now the `C-c a w` command will sort the collected entries only by priority, and the prefix for-
mat 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. The command `C-c
a N` will do a text search limited to only a single file.

For command sets creating a block agenda, `org-agenda-custom-commands` has two sep-
arate 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 com-
mand entries. Going back to the block agenda example (see Section 10.6.2 [Block agenda],
page 78), 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.

10.6.4 Exporting Agenda Views

If you are away from your computer, it can be very useful to have a printed version of some agenda views to carry around. Org mode can export custom agenda views as plain text, HTML[8] postscript, and iCalendar files. If you want to do this only occasionally, use the command

`C-x C-w`   Write the agenda view to a file. Depending on the extension of the selected file name, the view will be exported as HTML (extension '.html' or '.htm'), Postscript (extension '.ps'), iCalendar (extension '.ics'), or plain text (any other extension). Use the variable **org-agenda-exporter-settings** to set options for 'ps-print' and for 'htmlize' to be used during export, for example

```
(setq org-agenda-exporter-settings
      '((ps-number-of-columns 2)
        (ps-landscape-mode t)
        (htmlize-output-type 'css)))
```

If you need to export certain agenda views frequently, you can associate any custom agenda command with a list of output file names[9]. Here is an example that first does define custom commands for the agenda and the global todo list, together with a number of files to which to export them. Then we define two block agenda commands and specify file names for them as well. File names can be relative to the current working directory, or absolute.

---

[8] You need to install Hrvoje Niksic' 'htmlize.el'.

[9] If you want to store standard views like the weekly agenda or the global TODO list as well, you need to define custom commands for them in order to be able to specify file names.

```
(setq org-agenda-custom-commands
      '(("X" agenda "" nil ("agenda.html" "agenda.ps"))
        ("Y" alltodo "" nil ("todo.html" "todo.txt" "todo.ps"))
        ("h" "Agenda and Home-related tasks"
         ((agenda "")
          (tags-todo "home")
          (tags "garden"))
         nil
         ("~/views/home.html"))
        ("o" "Agenda and Office-related tasks"
         ((agenda)
          (tags-todo "work")
          (tags "office"))
         nil
         ("~/views/office.ps" "~/calendars/office.ics")))))
```

The extension of the file name determines the type of export. If it is '`.html`', Org mode will use the '`htmlize.el`' package to convert the buffer to HTML and save it to this file name. If the extension is '`.ps`', `ps-print-buffer-with-faces` is used to produce postscript output. If the extension is '`.ics`', iCalendar export is run export over all files that were used to construct the agenda, and limit the export to entries listed in the agenda now. Any other extension produces a plain ASCII file.

The export files are *not* created when you use one of those commands interactively because this might use too much overhead. Instead, there is a special command to produce *all* specified files in one step:

`C-c a e`   Export all agenda views that have export file names associated with them.

You can use the options section of the custom agenda commands to also set options for the export commands. For example:

```
(setq org-agenda-custom-commands
      '(("X" agenda ""
         ((ps-number-of-columns 2)
          (ps-landscape-mode t)
          (org-agenda-prefix-format " [ ] ")
          (org-agenda-with-colors nil)
          (org-agenda-remove-tags t))
         ("theagenda.ps"))))
```

This command sets two options for the postscript exporter, to make it print in two columns in landscape format - the resulting page can be cut in two and then used in a paper agenda. The remaining settings modify the agenda prefix to omit category and scheduling information, and instead include a checkbox to check off items. We also remove the tags to make the lines compact, and we don't want to use colors for the black-and-white printer. Settings specified in `org-agenda-exporter-settings` will also apply, but the settings in `org-agenda-custom-commands` take precedence.

From the command line you may also use

```
emacs -f org-batch-store-agenda-views -kill
```

or, if you need to modify some parameters

```
emacs -eval '(org-batch-store-agenda-views                    \
              org-agenda-ndays 30                             \
              org-agenda-start-day "2007-11-01"               \
              org-agenda-include-diary nil                    \
              org-agenda-files (quote ("~/org/project.org")))' \
       -kill
```

which will create the agenda views restricted to the file '`~/org/project.org`', without diary entries and with 30 days extent.

10.6.5 Using agenda information outside of Org

Org provides commands to access agenda information for the command line in emacs batch mode. This extracted information can be sent directly to a printer, or it can be read by a program that does further processing of the data. The first of these commands is the function `org-batch-agenda`, that produces an agenda view and sends it as ASCII text to STDOUT. The command takes a single string as parameter. If the string has length 1, it is used as a key to one of the commands you have configured in `org-agenda-custom-commands`, basically any key you can use after `C-c a`. For example, to directly print the current TODO list, you could use

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

If the parameter is a string with 2 or more characters, it is used as a tags/todo match string. For example, to print your local shopping list (all items with the tag '`shop`', but excluding the tag '`NewYork`'), you could use

```
emacs -batch -l ~/.emacs                                     \
      -eval '(org-batch-agenda "+shop-NewYork")' | lpr
```

You may also modify parameters on the fly like this:

```
emacs -batch -l ~/.emacs                                     \
   -eval '(org-batch-agenda "a"                              \
            org-agenda-ndays 30                              \
            org-agenda-include-diary nil                     \
            org-agenda-files (quote ("~/org/project.org")))' \
   | lpr
```

which will produce a 30 day agenda, fully restricted to the Org file '`~/org/projects.org`', not even including the diary.

If you want to process the agenda data in more sophisticated ways, you can use the command `org-batch-agenda-csv` to get a comma-separated list of values for each agenda item. Each line in the output will contain a number of fields separated by commas. The fields in a line are:

`category`	The category of the item
`head`	The headline, without TODO kwd, TAGS and PRIORITY
`type`	The type of the agenda entry, can be

`todo`	selected in TODO match
`tagsmatch`	selected in tags match
`diary`	imported from diary
`deadline`	a deadline

	`scheduled`	scheduled
	`timestamp`	appointment, selected by timestamp
	`closed`	entry was closed on date
	`upcoming-deadline`	warning about nearing deadline
	`past-scheduled`	forwarded scheduled item
	`block`	entry has date block including date
`todo`	The TODO keyword, if any	
`tags`	All tags including inherited ones, separated by colons	
`date`	The relevant date, like 2007-2-14	
`time`	The time, like 15:00-16:50	
`extra`	String with extra planning info	
`priority-l`	The priority letter if any was given	
`priority-n`	The computed numerical priority	

Time and date will only be given if a timestamp (or deadline/scheduled) lead to the selection
of the item.

A CSV list like this is very easy to use in a post processing script. For example, here is a
Perl program that gets the TODO list from Emacs/Org and prints all the items, preceded
by a checkbox:

```perl
#!/usr/bin/perl

# define the Emacs command to run
$cmd = "emacs -batch -l ~/.emacs -eval '(org-batch-agenda-csv \"t\")'";

# run it and capture the output
$agenda = qx{$cmd 2>/dev/null};

# loop over all lines
foreach $line (split(/\n/,$agenda)) {

  # get the individual values
  ($category,$head,$type,$todo,$tags,$date,$time,$extra,
   $priority_l,$priority_n) = split(/,/,$line);

  # proccess and print
  print "[ ] $head\n";
}
```

10.7 Using column view in the agenda

Column view (see Section 7.5 [Column view], page 48) is normally used to view and edit
properties embedded in the hierarchical structure of an Org file. It can be quite useful to
use column view also from the agenda, where entries are collected by certain criteria.

`C-c C-x C-c`
>   Turn on column view in the agenda.

To understand how to use this properly, it is important to realize that the entries in the agenda are no longer in their proper outline environment. This causes the following issues:

1. Org needs to make a decision which `COLUMNS` format to use. Since the entries in the agenda are collected from different files, and different files may have different `COLUMNS` formats, this is a non-trivial problem. Org first checks if the variable `org-overriding-columns-format` is currently set, and if yes takes the format from there. Otherwise it takes the format associated with the first item in the agenda, or, if that item does not have a specific format (defined in a property, or in it's file), it uses `org-columns-default-format`.

2. If any of the columns has a summary type defined (see Section 7.5.1.2 [Column attributes], page 48), turning on column view in the agenda will visit all relevant agenda files and make sure that the computations of this property are up to date. This is also true for the special `CLOCKSUM` property. Org will then sum the values displayed in the agenda. In the daily/weekly agenda, the sums will cover a single day, in all other views they cover the entire block. It is vital to realize that the agenda may show the same entry *twice* (for example as scheduled and as a deadline), and it may show two entries from the same hierarchy (for example a *parent* and it's *child*). In these cases, the summation in the agenda will lead to incorrect results because some values will count double.

3. When the column view in the agenda shows the `CLOCKSUM`, that is always the entire clocked time for this item. So even in the daily/weekly agenda, the clocksum listed in column view may originate from times outside the current view. This has the advantage that you can compare these values with a column listing the planned total effort for a task - one of the major applications for column view in the agenda. If you want information about clocked time in the displayed period use clock table mode (press `R` in the agenda).

11 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 reading 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.

11.1 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 $M-\boxed{\text{TAB}}$ 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 12.5 [HTML export], page 94), these symbols are translated into the proper syntax for HTML, for the above examples this is '`α`' and '`→`', respectively.

11.2 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 12.5 [HTML export], page 94), subscript and superscripts are surrounded with `<sub>` and `<sup>` tags, respectively.

11.3 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 or 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[3]. 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/`. The LaTeX header that will be used when processing a fragment can be configured with the variable `org-format-latex-header`.

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.

11.4 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 12.5 [HTML export], page 94), 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)
```

---

[3] The LaTeX export will not use images for displaying LaTeX fragments but include these fragments directly into the LaTeX code.

11.5 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 tem-
plates. 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 use 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 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[4]. 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 back-
  quote, a help window will pop up. Character modification will work only inside LaTeX
  fragments, outside the quote is normal.

---

[4] 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`.

12 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 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. LaTeX export lets you use Org mode and its structured editing functions to easily create LaTeX files. 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.

12.1 Markup rules

When exporting Org mode documents, the exporter tries to reflect the structure of the document as accurately as possible in the back-end. Since export targets like HTML or LaTeX allow much richer formatting, Org mode has rules how to prepare text for rich export. This section summarizes the markup rule used in an Org mode buffer.

Document title

The title of the exported document is taken from the special line

        #+TITLE: This is the title of the document

If this line does not exist, the title is derived from the first non-empty, non-comment line in the buffer. If no such line exists, or if you have turned off exporting of the text before the first headline (see below), the title will be the file name without extension.

If you are exporting only a subtree by marking is as the region, the heading of the subtree will become the title of the document. If the subtree has a property `EXPORT_TITLE`, that will take precedence.

Headings and sections

The outline structure of the document as described in Chapter 2 [Document Structure], page 5 forms the basis for defining sections of the exported document. However, since the outline structure is also used for (for example) lists of tasks, only the first three outline levels will be used as headings. Deeper levels will become itemized lists. You can change the location of this switch, globally by setting the variable `org-headline-levels`, or on a per file basis with a line

        #+OPTIONS: H:4

Table of contents

The table of contents is normally inserted directly before the first headline of the file. If you would like to get it to a different location, insert the string `[TABLE-OF-CONTENTS]` on a line by itself at the desired location. The depth of the table of contents is by default the same as the number of headline levels, but you can choose a smaller number or turn

off the table of contents entirely by configuring the variable `org-export-with-toc`, or on
a per-file basis with a line like

```
#+OPTIONS: toc:2          (only to two levels in TOC)
#+OPTIONS: toc:nil        (no TOC at all)
```

Text before the first headline

Org mode normally exports the text before the first headline, and even uses the first
line as the document title. The text will be fully marked up. If you need to include literal
HTML or LaTeX code, use the special constructs described below in the sections for the
individual exporters.

Some people like to use the space before the first headline for setup and internal links
and therefore would like to control the exported text before the first headline in a different
way. You can do so by setting the variable `org-export-skip-text-before-1st-heading`
to `t`. On a per-file basis, you can get the same effect with '`#+OPTIONS: skip:t`'.

If you still want to have some text before the first headline, use the `#+TEXT` construct:

```
#+OPTIONS: skip:t
#+TEXT: This text will go before the *first* headline.
#+TEXT: [TABLE-OF-CONTENTS]
#+TEXT: This goes between the table of contents and the first headline
```

Lists

Plain lists as described in Section 2.8 [Plain lists], page 11 are translated to the back-ends
syntax for such lists. Most back-ends support unordered, ordered, and description lists.

Paragraphs, line breaks, and quoting

Paragraphs are separated by at least one empty line. If you need to enforce a line break
within a paragraph, use '`\\`' at the end of a line.

To keep the line breaks in a region, but otherwise use normal formatting, you can use
this construct, which can also be used to format poetry.

```
#+BEGIN_VERSE
Everything should be made as simple as possible,
but not any simpler -- Albert Einstein
#+END_VERSE
```

When quoting a passage from another document, it is customary to format this as
a paragraph that is indented on both the left and the right margin. You can include
quotations in Org mode documents like this:

```
#+BEGIN_QUOTE
Everything should be made as simple as possible,
but not any simpler -- Albert Einstein
#+END_QUOTE
```

Literal examples

You can include literal examples that should not be subjected to markup. Such examples will be typeset in monospace, so this is well suited for source code and similar examples.

```
#+BEGIN_EXAMPLE
Some example from a text file.
#+END_EXAMPLE
```

For simplicity when using small examples, you can also start the example lines with a colon:

```
: Some example from a text file.
```

If the example is source code from a programming language, or any other text that can be marked up by font-lock in Emacs, you can ask for the example to look like the fontified Emacs buffer[1]. This is done with the 'src' block, where you also need to specify the name of the major mode that should be used to fontify the example:

```
#+BEGIN_SRC emacs-lisp
(defun org-xor (a b)
   "Exclusive or."
   (if a (not b) b))
#+END_SRC
```

`C-c '`   Edit the source code example at point in its native mode. This works by switching to an indirect buffer, narrowing the buffer and switching to the other mode. You need to exit by pressing `C-c '` again.

Include files

During export, you can include the content of another file. For example, to include your .emacs file, you could use:

```
#+INCLUDE: "~/.emacs" src emacs-lisp
```

The optional second and third parameter are the markup ('quote', 'example', or 'src'), and, if the markup is 'src', the language for formatting the contents. The markup is optional, if it is not given, the text will be assumed to be in Org mode format and will be processed normally.

`C-c '`   Visit the include file at point.

Tables

Both the native Org mode tables (see Chapter 3 [Tables], page 14) and tables formatted with the 'table.el' package will be exported properly. For Org mode tables, the lines before the first horizontal separator line will become table header lines.

---

[1] Currently this works only for the HTML back-end, and requires the 'htmlize.el' package version 1.34 or later.

Footnotes

Numbers in square brackets are treated as footnote markers, and lines starting with such a marker are interpreted as the footnote itself. You can use the Emacs package 'footnote.el' to create footnotes[2]. For example:

```
The Org homepage[1] now looks a lot better than it used to.

[1] The link is: http://orgmode.org
```

Emphasis and monospace

You can make words ***bold***, */italic/*, _underlined_, =code= and ~verbatim~, and, if you must, '+strike-through+'. Text in the code and verbatim string is not processed for Org mode specific syntax, it is exported verbatim.

TEX macros and LaTEX fragments

A TEX-like syntax is used to specify special characters. Where possible, these will be transformed into the native format of the exporter back-end. Strings like \alpha will be exported as α in the HTML output, and as α in the LaTEX output. Similarly, \nbsp will become   in HTML and ~ in LaTEX. This applies for a large number of entities, with names taken from both HTML and LaTEX, see the variable org-html-entities for the complete list. If you are unsure about a name, use M-⟨TAB⟩ for completion after having types the backslash and maybe a few characters (see Section 14.1 [Completion], page 106).

LaTEX fragments are converted into images for HTML export, and they are written literally into the LaTEX export. See also Chapter 11 [Embedded LaTeX], page 85.

Finally, '\-' is treated as a shy hyphen, and '--', '---', and '...' are all converted into special commands creating hyphens of different lengths or a compact set of dots.

Horizontal rules

A line consisting of only dashes, and at least 5 of them, will be exported as a horizontal line ('<hr/>' in HTML).

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, regions surrounded by '#+BEGIN_COMMENT' ... '#+END_COMMENT' will not be exported.

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

---

[2] The 'footnote' package uses *C-c !* to invoke its commands. This binding conflicts with the Org mode command for inserting inactive time stamps. You could use the variable footnote-prefix to switch footnotes commands to another key. Or, if you are too used to this binding, you could use org-replace-disputed-keys and org-disputed-keys to change the settings in Org.

12.2 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 14.1 [Completion], page 106).

`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)
#+DATE:      A date, fixed, of a format string for format-time-string
#+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 f:t TeX:t ...
#+LINK_UP:   the ''up'' link of an exported page
#+LINK_HOME: the ''home'' link of an exported page
```

The OPTIONS line is a compact[3] 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, or set level limit (integer)
`\n:`	turn on/off line-break-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. If you write "`^:{}`", `a_{b}` will be interpreted, but the simple `a_b` will be left as it is.
`-:`	turn on/off conversion of special strings.
`f:`	turn on/off footnotes like this[1].
`*:`	turn on/off emphasized text (bold, italic, underlined)
`TeX:`	turn on/off simple TeX macros in plain text
`LaTeX:`	turn on/off LaTeX fragments
`skip:`	turn on/off skipping the text before the first heading
`author:`	turn on/off inclusion of author name/email into exported file
`timestamp:`	turn on/off inclusion creation time into exported file
`d:`	turn on/off inclusion of drawers

These options take effect in both the HTML and LaTeX export, except for `TeX` and `LaTeX`, which are respectively `t` and `nil` for the LaTeX export.

When exporting only a single subtree by selecting it with `C-c @` before calling an export command, the subtree can overrule some of the file's export settings with properties `EXPORT_FILE_NAME`, `EXPORT_TITLE`, `EXPORT_TEXT`, and `EXPORT_OPTIONS`.

---

[3] If you want to configure many options this way, you can use several OPTIONS lines.

12.3 The export dispatcher

All export commands can be reached using the export dispatcher, which is a prefix key that prompts for an additional key specifying the command. Normally the entire file is exported, but if there is an active region that contains one outline tree, the first heading is used as document title and the subtrees are exported.

`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. The prefix arg is passed through to the exporter. If the option `org-export-run-in-background` is set, Org will run the command in the background if that seems useful for the specific command (i.e. commands that write to a file).

`C-c C-e v`   Like `C-c C-e`, but only export the text that is currently visible (i.e. not hidden by outline visibility).

`C-u C-u C-c C-e`

Call an the exporter, but reverse the setting of `org-export-run-in-background`, i.e. request background processing if not set, or force processing in the current Emacs process if st.

12.4 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. For an org file '`myfile.org`', the ASCII file will be '`myfile.txt`'. The file will be overwritten without warning. If there is an active region, only the region will be exported. If the selected region is a single tree[4], the tree head will become the document title. If the tree head entry has or inherits an `EXPORT_FILE_NAME` property, that name will be used for the export.

`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 body line 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.

---

[4] To select the current subtree, use `C-c @`.

12.5 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.

12.5.1 HTML export commands

`C-c C-e h` Export as HTML file 'myfile.html'. For an org file 'myfile.org', the ASCII file will be 'myfile.html'. The file will be overwritten without warning. If there is an active region, only the region will be exported. If the selected region is a single tree[5], the tree head will become the document title. If the tree head entry has or inherits an `EXPORT_FILE_NAME` property, that name will be used for the export.

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

`C-c C-e H` Export to a temporary buffer, do not create a file.

`C-c C-e R` Export the active region to a temporary buffer. With a prefix argument, do not produce the file header and footer, but just the plain HTML section for the region. This is good for cut-and-paste operations.

`C-c C-e v h`
`C-c C-e v b`
`C-c C-e v H`
`C-c C-e v R`
Export only the visible part of the document.

`M-x org-export-region-as-html`
Convert the region to HTML under the assumption that it was Org mode syntax before. This is a global command that can be invoked in any buffer.

`M-x org-replace-region-by-HTML`
Replace the active region (assumed to be in Org mode syntax) by HTML code.

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 numeric prefix argument. For example,

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

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

12.5.2 Quoting HTML tags

Plain '<' and '>' are always transformed to '<' and '>' in HTML export. If you want to include simple HTML tags which should be interpreted as such, mark them with '@' as in '@bold text@'. Note that this really works only for simple tags. For more extensive HTML that should be copied verbatim to the exported file use either

---

[5] To select the current subtree, use `C-c @`.

```
    #+HTML: Literal HTML code for export
```

or

```
    #+BEGIN_HTML
    All lines between these markers are exported literally
    #+END_HTML
```

12.5.3 Links

Internal links (see Section 4.2 [Internal links], page 27) 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 28) 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 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 13.1.6 [Publishing links], page 102.

12.5.4 Images

HTML export can inline images given as links in the Org file, and it can make an image the clickable part of a link. By default[6], images are inlined if a link does not have a description. So '[[file:myimg.jpg]]' will be inlined, while '[[file:myimg.jpg][the image]]' will just produce a link 'the image' that points to the image. If the description part itself is a `file:` link or a `http:` URL pointing to an image, this image will be inlined and activated so that clicking on the image will activate the link. For example, to include a thumbnail that will link to a high resolution version of the image, you could use:

```
    [[file:highres.jpg][file:thumb.jpg]]
```

and you could use `http` addresses just as well.

12.5.5 CSS support

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[7]:

---

[6] but see the variable `org-export-html-inline-images`

[7] Under Emacs 21, the continuation lines for a variable value should have no '#' at the start of the line.

```
* 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.

12.5.6 Javascript supported display of web pages

Sebastian Rose has written a JavaScript program especially designed to enhance the web viewing experience of HTML files created with Org. This program allows to view large files in two different ways. The first one is an *Info*-like mode where each section is displayed separately and navigation can be done with the `n` and `p` keys (and some other keys as well, press `?` for an overview of the available keys). The second view type is a *folding* view much like Org provides it inside Emacs. The script is available at `http://orgmode.org/org-info.js` and you can find the documentation for it at `http://orgmode.org/worg/code/org-info-js/org-info.js.html`. We are serving the script from our site, but if you use it a lot, you might not want to be dependent on `orgmode.org` and prefer to install a local copy on your own web server.

To use the script, you need to make sure that the 'org-jsinfo.el' module gets loaded. It should be loaded by default, try `M-x customize-variable` (RET) `org-modules` (RET) to convince yourself that this is indeed the case. All it then takes to make use of the program is adding a single line to the Org file:

```
#+INFOJS_OPT: view:info toc:nil
```

If this line is found, the HTML header will automatically contain the code needed to invoke the script. Using the line above, you can set the following viewing options:

path:	The path to the script. The default is to grab the script from `http://orgmode.org/org-info.js`, but you might want to have a local copy and use a path like '`../scripts/org-info.js`'.
view:	Initial view when website is first shown. Possible values are:

	info	Info-like interface with one section per page.
	overview	Folding interface, initially showing only top-level.
	content	Folding interface, starting with all headlines visible.
	showall	Folding interface, all headlines and text visible.

sdepth:	Maximum headline level that will still become an independent section for info and folding modes. The default is taken from `org-headline-levels` (= the H switch in `#+OPTIONS`). If this is smaller than in `org-headline-levels`, each info/folding section can still contain children headlines.
toc:	Should the table of content *initially* be visible?

Even when `nil`, you can always get to the toc with *i*.

tdepth: The depth of the table of contents. The defaults are taken from
            the variables `org-headline-levels` and `org-export-with-toc`.

ftoc: Does the css of the page specify a fixed position for the toc?
            If yes, the toc will never be displayed as a section.

ltoc: Should there be short contents (children) in each section?

mouse: Headings are highlighted when the mouse is over them. Should be
            '`underline`' (default) or a background color like '`#cccccc`'.

buttons: Should view-toggle buttons be everywhere? When `nil` (the
            default), only one such button will be present.

You can choose default values for these options by customizing the variable `org-infojs-options`. If you always want to apply the script to your pages, configure the variable `org-export-html-use-infojs`.

12.6 LaTeX export

Org mode contains a LaTeX exporter written by Bastien Guerry.

12.6.1 LaTeX export commands

C-c C-e l   Export as LaTeX file '`myfile.tex`'. For an org file '`myfile.org`', the ASCII
               file will be '`myfile.tex`'. The file will be overwritten without warning. If there
               is an active region, only the region will be exported. If the selected region is
               a single tree[8], the tree head will become the document title. If the tree head
               entry has or inherits an `EXPORT_FILE_NAME` property, that name will be used
               for the export.

C-c C-e L   Export to a temporary buffer, do not create a file.

C-c C-e v l
C-c C-e v L
               Export only the visible part of the document.

M-x org-export-region-as-latex
               Convert the region to LaTeX under the assumption that it was Org mode syntax
               before. This is a global command that can be invoked in any buffer.

M-x org-replace-region-by-latex
               Replace the active region (assumed to be in Org mode syntax) by LaTeX code.

In the exported version, the first 3 outline levels will become headlines, defining a general document structure. Additional levels will be exported as description lists. The exporter can ignore them or convert them to a custom string depending on `org-latex-low-levels`.

If you want that transition to occur at a different level, specify it with a numeric prefix argument. For example,

    C-2 C-c C-e l

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

---

[8] To select the current subtree, use *C-c @*.

12.6.2 Quoting LaTeX code

Embedded LaTeX as described in [Chapter 11 [Embedded LaTeX], page 85](#) will be correctly inserted into the LaTeX file. Furthermore, you can add special code that should only be present in LaTeX export with the following constructs:

```
#+LaTeX: Literal LaTeX code for export
```

or

```
#+BEGIN_LaTeX
All lines between these markers are exported literally
#+END_LaTeX
```

12.6.3 Sectioning structure

By default, the LaTeX output uses the class `article`.

You can change this globally by setting a different value for `org-export-latex-default-class` or locally by adding an option like `#+LaTeX_CLASS: myclass` in your file. The class should be listed in `org-export-latex-classes`, where you can also define the sectioning structure for each class.

12.7 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.

12.8 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 files show up in the calendar application. Org mode can export calendar information in the standard iCalendar format. If you also want to have TODO entries included in the export, configure the variable `org-icalendar-include-todo`.

The iCalendar standard requires each entry to have a globally unique identifier (UID). Org creates these identifiers during export. If you set the variable `org-icalendar-store-UID`, the UID will be stored in the `:ID:` property of the entry and re-used next time you report this entry. Since a single entry can give rise to multiple iCalendar entries (as a timestamp, a deadline, a scheduled item, and as a TODO item), Org adds prefixes to the UID, depending on what triggered the inclusion of the entry. In this way the UID remains unique, but a synchronization program can still figure out from which entry all the different instances originate.

`C-c C-e i`   Create iCalendar entries for the current file and store them in the same direc-
            tory, 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`.

The export will honor SUMMARY, DESCRIPTION and LOCATION properties if the
selected entries have them. If not, the summary will be derived from the headline, and the
description from the body (limited to `org-icalendar-include-body` characters).

How this calendar is best read and updated, depends on the application you are using.
The FAQ covers this issue.

13 Publishing

Org 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 into a web-site authoring tool.

You can also use Org-publish to convert files into LaTeX, or even combine HTML and LaTeX conversion so that files are available in both formats on the server[2].

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

13.1 Configuration

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

13.1.1 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.

13.1.2 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.

---

[1] '`org-publish.el`' is not distributed with Emacs 21, if you are still using Emacs 21, you need you need to download this file separately.

[2] Since LaTeX files on a server are not that helpful, you surely want to perform further conversion on them – e.g. convert them to PDF format.

`:preparation-function`	Function called before starting the publishing process, for example to run `make` for updating files to be published.
`:completion-function`	Function called after finishing the publishing process, for example to change permissions of the resulting files.

13.1.3 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`.

13.1.4 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 files as HTML files, and this is done by the function `org-publish-org-to-html` which calls the HTML exporter (see Section 12.5 [HTML export], page 94). But you also can publish your files in LaTeX by using the function `org-publish-org-to-latex` instead. Other files like images only need to be copied to the publishing destination. For non-Org 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`.

13.1.5 Options for the HTML/LaTeX exporters

The property list can be used to set many export options for the HTML and LaTeX exporters. In most cases, these properties correspond to user variables in Org. 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
:special-strings         org-export-with-special-strings
: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
```

If you use several email addresses, separate them by a semi-column.

Most of the `org-export-with-*` variables have the same effect in both HTML and LaTeX exporters, except for `:TeX-macros` and `:LaTeX-fragments`, respectively `nil` and `t` in the LaTeX export.

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 12.2 [Export options], page 92), however, override everything.

13.1.6 Links between published files

To create a link from one Org file to another, you would use something like '`[[file:foo.org][The foo]]`' or simply '`file:foo.org.`' (see Chapter 4 [Hyperlinks], page 27). 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 13.2.2 [Complex example], page 104 for an example of this usage.

Sometime an Org 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`.

13.1.7 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`   Plug-in 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.

13.2 Sample configuration

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

13.2.1 Example: simple publishing configuration

This example publishes a set of Org 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\">")))
```

13.2.2 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 style sheets. 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 web server, 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"))))
```

13.3 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.

14 Miscellaneous

14.1 Completion

Org 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 ':' in a headline, complete tags. The list of tags is taken from the variable `org-tag-alist` (possibly set through the '#+TAGS' in-buffer option, see Section 6.2 [Setting tags], page 41), or it is created dynamically from all tags used in the current buffer.
- After ':' and not in a headline, complete property keys. The list of keys is constructed dynamically from all keys used in the current buffer.
- After '[', complete link abbreviations (see Section 4.6 [Link abbreviations], page 31).
- 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.

14.2 Customization

There are more than 180 variables that can be used to customize Org. For the sake of compactness of the manual, I am 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 14.3 [In-buffer settings], page 106).

14.3 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.

`#+ARCHIVE: %s_done::`

> This line sets the archive location for the agenda file. It applies for all subsequent lines until the next '`#+ARCHIVE`' line, or the end of the file. The first such line also applies to any entries before it. The corresponding variable is `org-archive-location`.

`#+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. The first such line also applies to any entries before it.

`#+COLUMNS: %25ITEM .....`

> Set the default format for columns view. This format applies when columns view is invoked in location where no `COLUMNS` property applies.

`#+CONSTANTS: name1=value1 ...`

> Set file-local values for constants to be used in table formulas. This line set the local variable `org-table-formula-constants-local`. The global version of this variable is `org-table-formula-constants`.

`#+FILETAGS: :tag1:tag2:tag3:`

> Set tags that can be inherited by any entry in the file, including the top-level entries.

`#+DRAWERS: NAME1 .....`

> Set the file-local set of drawers. The corresponding global variable is `org-drawers`.

`#+LINK: linkword replace`

> These lines (several are allowed) specify link abbreviations. See Section 4.6 [Link abbreviations], page 31. The corresponding variable is `org-link-abbrev-alist`.

`#+PRIORITIES: highest lowest default`

> This line sets the limits and the default for the priorities. All three must be either letters A-Z or numbers 0-9. The highest priority must have a lower ASCII number that the lowest priority.

`#+PROPERTY: Property_Name Value`

> This line sets a default inheritance value for entries in the current buffer, most useful for specifying the allowed values of a property.

`#+SETUPFILE: file`

> This line defines a file that holds more in-buffer setup. Normally this is entirely ignored. Only when the buffer is parsed for option-setting lines (i.e. when starting Org mode for a file, when pressing `C-c C-c` in a settings line, or when exporting), then the contents of this file are parsed as if they had been included in the buffer. In particlar, the file can be any other Org mode file with internal setup. You can visit the file the cursor is in the line with `C-c '`.

`#+STARTUP:`

This line sets options to be used at startup of Org mode, when an Org 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 closing and reinstating TODO items, and clock intervals (variables `org-log-done`, `org-log-note-clock-out`, and `org-log-repeat`) can be configured using these options.

`logdone`	record a timestamp when an item is marked DONE
`lognotedone`	record timestamp and a note when DONE
`nologdone`	don't record when items are marked DONE
`logrepeat`	record a time when reinstating a repeating item
`lognoterepeat`	record a note when reinstating a repeating item
`nologrepeat`	do not record when reinstating repeating item
`lognoteclock-out`	record a note when clocking out
`nolognoteclock-out`	don't record a note when clocking out

Here are the options for hiding leading stars in outline headings, and for indenting outlines. 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
`indent`	virtual indentation according to outline level
`noindent`	no virtual indentation according to outline level
`odd`	allow only odd outline levels (1,3,...)
`oddeven`	allow all outline levels

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

`customtime`	overlay custom time format

The following options influence the table spreadsheet (variable `constants-unit-system`).

`constcgs`	`constants.el` should use the c-g-s unit system
`constSI`	`constants.el` should use the SI unit system

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

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

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

`#+TITLE:, #+AUTHOR:, #+EMAIL:, #+LANGUAGE:, #+TEXT:, #+OPTIONS, #+DATE:`
These lines provide settings for exporting files. For more details see Section 12.2 [Export options], page 92.

`#+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`.

14.4 The very busy C-c C-c key

The key `C-c C-c` has many purposes in Org, which are all mentioned scattered throughout this manual. One specific function of this key is to add *tags* to a headline (see Chapter 6 [Tags], page 41). In many other circumstances it means something like *Hey Org, 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 property line or at the start or end of a property drawer, offer property commands.

− 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.

− If the cursor is on the `#+BEGIN` line of a dynamical block, the block is updated.

14.5 A cleaner outline view

Some people find it noisy and distracting that the Org headlines are starting with a potentially large number of stars, and that text below the headlines is not indented. This is not really a problem when you are writing a book where the outline headings are really section headlines. However, in a more list-oriented outline, it is clear that an indented structure is a lot cleaner, as can be seen by comparing the two columns in the following example:

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

It is non-trivial to make such a look work in Emacs, but Org contains three separate features that, combined, achieve just that.

1. *Indentation of text below headlines*
   You may indent text below each headline to make the left boundary line up with the headline, like

   ```
   *** 3rd level
       more text, now indented
   ```

   A good way to get this indentation is by hand, and Org supports this with paragraph filling, line wrapping, and structure editing[1] preserving or adapting the indentation appropriate. A different approach would be to have a way to automatically indent lines according to outline structure by adding overlays or text properties. But I have not yet found a robust and efficient way to do this in large files.

2. *Hiding leading stars*
   You can modify the display in such a way that all leading stars become invisible. To do this in a global way, configure the variable `org-hide-leading-stars` or change this on a per-file basis with

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

   With hidden stars, the tree becomes:

   ```
   * Top level headline
    * Second level
     * 3rd level
     ...
   ```

   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 you 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.

3. 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. In this way we get the outline view shown at the beginning of this section. In order to make the structure editing and export commands handle this convention correctly, configure the variable `org-odd-levels-only`, or set this on a per-file basis with one of the following lines:

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

---

[1] See also the variable `org-adapt-indentation`.

You can convert an Org 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`.

14.6 Using Org on a tty

Because Org contains a large number of commands, by default much of Org's core commands are bound to keys that are generally not accessible on a tty, such as the cursor keys (⟨left⟩, ⟨right⟩, ⟨up⟩, ⟨down⟩), ⟨TAB⟩ and ⟨RET⟩, in particular when used together with modifiers like ⟨Meta⟩ and/or ⟨Shift⟩. To access these commands on a tty when special keys are unavailable, the following alternative bindings can be used. The tty bindings below will likely be more cumbersome; you may find for some of the bindings below that a customized work-around suits you better. For example, changing a time stamp is really only fun with `S-`⟨cursor⟩ keys, whereas 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` ⟨left⟩	
`S-`⟨right⟩	`C-c` ⟨right⟩	
`S-`⟨up⟩	`C-c` ⟨up⟩	
`S-`⟨down⟩	`C-c` ⟨down⟩	
`C-S-`⟨left⟩	`C-c C-x` ⟨left⟩	
`C-S-`⟨right⟩	`C-c C-x` ⟨right⟩	

14.7 Interaction with other packages

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

14.7.1 Packages that Org cooperates with

'`calc.el`' by Dave Gillespie
> Org uses the Calc package for implementing spreadsheet functionality in its tables (see Section 3.5 [The spreadsheet], page 18). Org 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 3.5 [The spreadsheet], page 18), 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 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 files. See Section 11.5 [CDLaTeX mode], page 87.

'`imenu.el`' by Ake Stenhoff and Lars Lindberg

Imenu allows menu access to an index of items in a file. Org mode supports Imenu - all you need to do to get the index is the following:

```
(add-hook 'org-mode-hook
          (lambda () (imenu-add-to-menubar "Imenu")))
```

By default the index is two levels deep - you can modify the depth using the option `org-imenu-depth`.

'`remember.el`' by John Wiegley

Org cooperates with remember, see Chapter 9 [Remember], page 62. '`Remember.el`' is not part of Emacs, find it on the web.

'`speedbar.el`' by Eric M. Ludlam

Speedbar is a package that creates a special frame displaying files and index items in files. Org mode supports Speedbar and allows you to drill into Org files directly from the Speedbar. It also allows to restrict the scope of agenda commands to a file or a subtree by using the command `<` in the Speedbar frame.

'`table.el`' by Takaaki Ota

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 ⒯ⒶⒷ 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.

'`table.el`' is part of Emacs 22.

'`footnote.el`' by Steven L. Baur

> Org mode recognizes numerical footnotes as provided by this package (see ⟨undefined⟩ [Footnotes], page ⟨undefined⟩).

14.7.2 Packages that lead to conflicts with Org mode

'`allout.el`' by Ken Manheimer

> Startup of Org 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

> Key bindings in Org 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, configure the variable `org-replace-disputed-keys`. When set, Org will move the following key bindings in Org 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-+
> ```

> 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.

'`footnote.el`' by Steven L. Baur

> Org supports the syntax of the footnote package, but only the numerical footnote markers. Also, the default key for footnote commands, `C-c !` is already used by Org. You could use the variable `footnote-prefix` to switch footnotes commands to another key. Or, you could use `org-replace-disputed-keys` and `org-disputed-keys` to change the settings in Org.

14.8 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 17) to a width too small to display the link, the field would look entirely empty even though it is not. To prevent this, Org 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.

- 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. You may use the command `org-table-iterate` (`C-u C-c *`) to recalculate until convergence.

- The exporters work well, but could be made more efficient.

Appendix A Extensions

This appendix lists the extension modules that have been written for Org. Many of these extensions live in the 'contrib' directory of the Org distribution, others are available somewhere on the web.

A.1 Extensions in the 'contrib' directory

'org-annotate-file.el' by *Philip Jackson*

      Annotate a file with org syntax, in a separate file, with links back to the annotated file.

'org-annotation-helper.el' by *Bastien Guerry and Daniel E. German*

      Call *remember* directly from Firefox/Opera, or from Adobe Reader. When activating a special link or bookmark, Emacs receives a trigger to create a note with a link back to the website. Requires some setup, a detailes description is in 'contrib/packages/org-annotation-helper'.

'org-bookmark.el' by *Tokuya Kameshima*

      Support for links to Emacs bookmarks.

'org-depend.el' by *Carsten Dominik*

      TODO dependencies for Org-mode. Make TODO state changes in one entry trigger changes in another, or be blocked by the state of another entry. Also, easily create chains of TODO items with exactly one active item at any time.

'org-elisp-symbol.el' by *Bastien Guerry*

      Org links to emacs-lisp symbols. This can create annotated links that exactly point to the definition location of a variable of function.

'org-eval.el' by *Carsten Dominik*

      The `<lisp>` tag, adapted from Emacs Wiki and Emacs Muse, allows to include text in a document that is the result of evaluating some code. Other scripting languages like `perl` can be supported with this package as well.

'org-expiry.el' by *Bastien Guerry*

      Expiry mechanism for Org entries.

'org-indent.el' by *Carsten Dominik*

      Dynamic indentation of Org outlines. The plan is to indent an outline according to level, but so far this is too hard for a proper and stable implementation. Still, it works somewhat.

'org-interactive-query.el' by *Christopher League*

      Interactive modification of tags queries. After running a general query in Org, this package allows to narrow down the results by adding more tags or keywords.

'org-mairix.el' by *Georg C. F. Greve*

      Hook mairix search into Org for different MUAs.

'org-man.el' by *Carsten Dominik*

      Support for links to manpages in Org-mode.

'`org-mtags.el`' by *Carsten Dominik*

> Support for some Muse-like tags in Org-mode. This package allows you to write `<example>` and `<src>` and other syntax copied from Emacs Muse, right inside an Org file. The goal here is to make it easy to publish the same file using either org-publish or Muse.

'`org-panel.el`' by *Lennard Borgman*

> Simplified and display-aided access to some Org commands.

'`org-registry.el`' by *Bastien Guerry*

> A registry for Org links, to find out from where links point to a given file or location.

'`org2rem.el`' by *Bastien Guerry*

> Convert org appointments into reminders for the '`remind`' program.

'`org-screen.el`' by *Andrew Hyatt*

> Visit screen sessions through Org-mode links.

'`org-toc.el`' by *Bastien Guerry*

> Table of contents in a separate buffer, with fast access to sections and easy visibility cycling.

'`orgtbl-sqlinsert.el`' by *Jason Riedy*

> Convert Org-mode tables to SQL insertions. Documentation for this can be found on the Worg pages.

A.2 Other extensions

> *TO BE DONE*

Appendix B   Hacking

This appendix covers some aspects where users can extend the functionality of Org.

B.1 Adding hyperlink types

Org has a large number of hyperlink types built-in (see Chapter 4 [Hyperlinks], page 27).
If you would like to add new link types, it provides an interface for doing so. Lets look at an
example file 'org-man.el' that will add support for creating links like '[[man:printf][The
printf manpage]]' to show Unix manual pages inside emacs:

```
;;; org-man.el - Support for links to manpages in Org

(require 'org)

(org-add-link-type "man" 'org-man-open)
(add-hook 'org-store-link-functions 'org-man-store-link)

(defcustom org-man-command 'man
  "The Emacs command to be used to display a man page."
  :group 'org-link
  :type '(choice (const man) (const woman)))

(defun org-man-open (path)
  "Visit the manpage on PATH.
PATH should be a topic that can be thrown at the man command."
  (funcall org-man-command path))

(defun org-man-store-link ()
  "Store a link to a manpage."
  (when (memq major-mode '(Man-mode woman-mode))
    ;; This is a man page, we do make this link
    (let* ((page (org-man-get-page-name))
           (link (concat "man:" page))
           (description (format "Manpage for %s" page)))
      (org-store-link-props
       :type "man"
       :link link
       :description description))))

(defun org-man-get-page-name ()
  "Extract the page name from the buffer name."
  ;; This works for both 'Man-mode' and 'woman-mode'.
  (if (string-match " \\(\\S-+\\)\\*" (buffer-name))
      (match-string 1 (buffer-name))
    (error "Cannot create link to this man page")))
```

```
    (provide 'org-man)

    ;;; org-man.el ends here
```
You would activate this new link type in '`.emacs`' with
```
    (require 'org-man)
```
Lets go through the file and see what it does.

1. It does (`require 'org`) to make sure that '`org.el`' has been loaded.
2. The next line calls `org-add-link-type` to define a new link type with prefix '`man`'. The call also contains the name of a function that will be called to follow such a link.
3. The next line adds a function to `org-store-link-functions`, in order to allow the command `C-c l` to record a useful link in a buffer displaying a man page.

The rest of the file defines the necessary variables and functions. First there is a customization variable that determines which emacs command should be used to display man pages. There are two options, `man` and `woman`. Then the function to follow a link is defined. It gets the link path as an argument - in this case the link path is just a topic for the manual command. The function calls the value of `org-man-command` to display the man page.

Finally the function `org-man-store-link` is defined. When you try to store a link with `C-c l`, also this function will be called to try to make a link. The function must first decide if it is supposed to create the link for this buffer type, we do this by checking the value of the variable `major-mode`. If not, the function must exit and return the value `nil`. If yes, the link is created by getting the manual topic from the buffer name and prefixing it with the string '`man:`'. Then it must call the command `org-store-link-props` and set the `:type` and `:link` properties. Optionally you can also set the `:description` property to provide a default for the link description when the link is later inserted into an Org buffer with `C-c C-l`.

B.2 Tables and lists in arbitrary syntax

Since Orgtbl mode can be used as a minor mode in arbitrary buffers, a frequent feature request has been to make it work with native tables in specific languages, for example LaTeX. However, this is extremely hard to do in a general way, would lead to a customization nightmare, and would take away much of the simplicity of the Orgtbl mode table editor.

This appendix describes a different approach. We keep the Orgtbl mode table in its native format (the *source table*), and use a custom function to *translate* the table to the correct syntax, and to *install* it in the right location (the *target table*). This puts the burden of writing conversion functions on the user, but it allows for a very flexible system.

Bastien added the ability to do the same with lists. You can use Org's facilities to edit and structure lists by turning `orgstruct-mode` on, then locally exporting such lists in another format (HTML, LaTeX or Texinfo.)

B.2.1 Radio tables

To define the location of the target table, you first need to create two lines that are comments in the current mode, but contain magic words for Orgtbl mode to find. Orgtbl

mode will insert the translated table between these lines, replacing whatever was there before. For example:

```
/* BEGIN RECEIVE ORGTBL table_name */
/* END RECEIVE ORGTBL table_name */
```

Just above the source table, we put a special line that tells Orgtbl mode how to translate this table and where to install it. For example:

```
#+ORGTBL: SEND table_name translation_function arguments....
```

`table_name` is the reference name for the table that is also used in the receiver lines. `translation_function` is the Lisp function that does the translation. Furthermore, the line can contain a list of arguments (alternating key and value) at the end. The arguments will be passed as a property list to the translation function for interpretation. A few standard parameters are already recognized and acted upon before the translation function is called:

`:skip N`   Skip the first N lines of the table. Hlines do count as separate lines for this parameter!

`:skipcols (n1 n2 ...)`
List of columns that should be skipped. If the table has a column with calculation marks, that column is automatically discarded as well. Please note that the translator function sees the table *after* the removal of these columns, the function never knows that there have been additional columns.

The one problem remaining is how to keep the source table in the buffer without disturbing the normal workings of the file, for example during compilation of a C file or processing of a LaTeX file. There are a number of different solutions:

- The table could be placed in a block comment if that is supported by the language. For example, in C mode you could wrap the table between '`/*`' and '`*/`' lines.
- Sometimes it is possible to put the table after some kind of *END* statement, for example '`\bye`' in TeX and '`\end{document}`' in LaTeX.
- You can just comment the table line by line whenever you want to process the file, and uncomment it whenever you need to edit the table. This only sounds tedious - the command *M-x orgtbl-toggle-comment* does make this comment-toggling very easy, in particular if you bind it to a key.

B.2.2 A LaTeX example of radio tables

The best way to wrap the source table in LaTeX is to use the `comment` environment provided by '`comment.sty`'. It has to be activated by placing `\usepackage{comment}` into the document header. Orgtbl mode can insert a radio table skeleton[1] with the command *M-x orgtbl-insert-radio-table*. You will be prompted for a table name, lets say we use '`salesfigures`'. You will then get the following template:

```
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
```

---

[1] By default this works only for LaTeX, HTML, and Texinfo. Configure the variable `orgtbl-radio-tables` to install templates for other modes.

```
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex
| | |
\end{comment}
```

The `#+ORGTBL: SEND` line tells Orgtbl mode to use the function `orgtbl-to-latex` to convert the table into LaTeX and to put it into the receiver location with name `salesfigures`. You may now fill in the table, feel free to use the spreadsheet features[2]:

```
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex
| Month | Days | Nr sold | per day |
|-------+------+---------+---------|
| Jan   |   23 |      55 |     2.4 |
| Feb   |   21 |      16 |     0.8 |
| March |   22 |     278 |    12.6 |
#+TBLFM: $4=$3/$2;%.1f
% $ (optional extra dollar to keep font-lock happy, see footnote)
\end{comment}
```

When you are done, press `C-c C-c` in the table to get the converted table inserted between the two marker lines.

Now lets assume you want to make the table header by hand, because you want to control how columns are aligned etc. In this case we make sure that the table translator does skip the first 2 lines of the source table, and tell the command to work as a *splice*, i.e. to not produce header and footer commands of the target table:

```
\begin{tabular}{lrrr}
Month & \multicolumn{1}{c}{Days} & Nr.\ sold & per day\\
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
\end{tabular}
%
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex :splice t :skip 2
| Month | Days | Nr sold | per day |
|-------+------+---------+---------|
| Jan   |   23 |      55 |     2.4 |
| Feb   |   21 |      16 |     0.8 |
| March |   22 |     278 |    12.6 |
#+TBLFM: $4=$3/$2;%.1f
\end{comment}
```

---

[2] If the '`#+TBLFM`' line contains an odd number of dollar characters, this may cause problems with font-lock in LaTeX mode. As shown in the example you can fix this by adding an extra line inside the `comment` environment that is used to balance the dollar expressions. If you are using AUCTeX with the font-latex library, a much better solution is to add the `comment` environment to the variable `LaTeX-verbatim-environments`.

The LaTeX translator function `orgtbl-to-latex` is already part of Orgtbl mode. It uses a `tabular` environment to typeset the table and marks horizontal lines with `\hline`. Furthermore, it interprets the following parameters (see also Section B.2.3 [Translator functions], page 121):

`:splice nil/t`

When set to t, return only table body lines, don't wrap them into a tabular environment. Default is nil.

`:fmt fmt`   A format to be used to wrap each field, should contain `%s` for the original field value. For example, to wrap each field value in dollars, you could use `:fmt "$%s$"`. This may also be a property list with column numbers and formats. for example `:fmt (2 "$%s$" 4 "%s\\%%")`. A function of one argument can be used in place of the strings; the function must return a formatted string.

`:efmt efmt`

Use this format to print numbers with exponentials. The format should have `%s` twice for inserting mantissa and exponent, for example `"%s\\times10^{%s}"`. The default is `"%s\\,(%s)"`. This may also be a property list with column numbers and formats, for example `:efmt (2 "$%s\\times10^{%s}$" 4 "$%s\\cdot10^{%s}$")`. After `efmt` has been applied to a value, `fmt` will also be applied. Similar to `fmt`, functions of two arguments can be supplied instead of strings.

B.2.3 Translator functions

Orgtbl mode has several translator functions built-in: `orgtbl-to-csv` (comma-separated values), `orgtbl-to-tsv` (TAB-separated values) `orgtbl-to-latex`, `orgtbl-to-html`, and `orgtbl-to-texinfo`. Except for `orgtbl-to-html`[3], these all use a generic translator, `orgtbl-to-generic`. For example, `orgtbl-to-latex` itself is a very short function that computes the column definitions for the `tabular` environment, defines a few field and line separators and then hands over to the generic translator. Here is the entire code:

```
(defun orgtbl-to-latex (table params)
  "Convert the Orgtbl mode TABLE to LaTeX."
  (let* ((alignment (mapconcat (lambda (x) (if x "r" "l"))
                               org-table-last-alignment ""))
         (params2
          (list
           :tstart (concat "\\begin{tabular}{" alignment "}")
           :tend "\\end{tabular}"
           :lstart "" :lend " \\\\" :sep " & "
           :efmt "%s\\,(%s)" :hline "\\hline")))
    (orgtbl-to-generic table (org-combine-plists params2 params))))
```

As you can see, the properties passed into the function (variable *PARAMS*) are combined with the ones newly defined in the function (variable *PARAMS2*). The ones passed into the function (i.e. the ones set by the 'ORGTBL SEND' line) take precedence. So if you would

---

[3] The HTML translator uses the same code that produces tables during HTML export.

like to use the LaTeX translator, but wanted the line endings to be '\\[2mm]' instead of the default '\\', you could just overrule the default with

```
#+ORGTBL: SEND test orgtbl-to-latex :lend " \\\\[2mm]"
```

For a new language, you can either write your own converter function in analogy with the LaTeX translator, or you can use the generic function directly. For example, if you have a language where a table is started with '!BTBL!', ended with '!ETBL!', and where table lines are started with '!BL!', ended with '!EL!' and where the field separator is a TAB, you could call the generic translator like this (on a single line!):

```
#+ORGTBL: SEND test orgtbl-to-generic :tstart "!BTBL!" :tend "!ETBL!"
                              :lstart "!BL! " :lend " !EL!" :sep "\t"
```

Please check the documentation string of the function `orgtbl-to-generic` for a full list of parameters understood by that function and remember that you can pass each of them into `orgtbl-to-latex`, `orgtbl-to-texinfo`, and any other function using the generic function.

Of course you can also write a completely new function doing complicated things the generic translator cannot do. A translator function takes two arguments. The first argument is the table, a list of lines, each line either the symbol `hline` or a list of fields. The second argument is the property list containing all parameters specified in the '#+ORGTBL: SEND' line. The function must return a single string containing the formatted table. If you write a generally useful translator, please post it on `emacs-orgmode@gnu.org` so that others can benefit from your work.

B.2.4 Radio lists

Sending and receiving radio lists works exactly the same way than sending and receiving radio tables (see Section B.2.1 [Radio tables], page 118)[4]. As for radio tables, you can insert radio lists templates in HTML, LaTeX and Texinfo modes by calling `org-list-insert-radio-list`.

Here are the differences with radio tables:

- Use `ORGLST` instead of `ORGTBL`.
- The available translation functions for radio lists don't take parameters.
- 'C-c C-c' will work when pressed on the first item of the list.

Here is a LaTeX example. Let's say that you have this in your LaTeX file:

```
% BEGIN RECEIVE ORGLST to-buy
% END RECEIVE ORGLST to-buy
\begin{comment}
#+ORGLIST: SEND to-buy orgtbl-to-latex
- a new house
- a new computer
  + a new keyboard
  + a new mouse
- a new life
```

---

[4] You need to load the `org-export-latex.el` package to use radio lists since the relevant code is there for now.

```
\end{comment}
```

Pressing 'C-c C-c' on `a new house` and will insert the converted LaTeX list between the two marker lines.

B.3 Dynamic blocks

Org 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).

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:dynamic 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. If you want to use the original content in the writer function, you can use the extra parameter `: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`.

B.4 Special agenda views

Org provides a special hook that can be used to narrow down the selection made by any of the agenda views. You may specify a function that is used at each match to verify if the match should indeed be part of the agenda view, and if not, how much should be skipped.

Let's say you want to produce a list of projects that contain a WAITING tag anywhere in the project tree. Let's further assume that you have marked all tree headings that define a project with the TODO keyword PROJECT. In this case you would run a TODO search for the keyword PROJECT, but skip the match unless there is a WAITING tag anywhere in the subtree belonging to the project line.

To achieve this, you must write a function that searches the subtree for the tag. If the tag is found, the function must return `nil` to indicate that this match should not be skipped. If there is no such tag, return the location of the end of the subtree, to indicate that search should continue from there.

```
(defun my-skip-unless-waiting ()
  "Skip trees that are not waiting"
  (let ((subtree-end (save-excursion (org-end-of-subtree t))))
    (if (re-search-forward ":waiting:" subtree-end t)
        nil          ; tag found, do not skip
      subtree-end))) ; tag not found, continue after end of subtree
```

Now you may use this function in an agenda custom command, for example like this:

```
(org-add-agenda-custom-command
 '("b" todo "PROJECT"
   ((org-agenda-skip-function 'my-org-waiting-projects)
    (org-agenda-overriding-header "Projects waiting for something: "))))
```

Note that this also binds `org-agenda-overriding-header` to get a meaningful header in the agenda view.

A general way to create custom searches is to base them on a search for entries with a certain level limit. If you want to study all entries with your custom search function, simply do a search for 'LEVEL>0', and then use `org-agenda-skip-function` to select the entries you really want to have.

You may also put a Lisp form into `org-agenda-skip-function`. In particular, you may use the functions `org-agenda-skip-entry-if` and `org-agenda-skip-subtree-if` in this form, for example:

`'(org-agenda-skip-entry-if 'scheduled)`
> Skip current entry if it has been scheduled.

`'(org-agenda-skip-entry-if 'notscheduled)`
> Skip current entry if it has not been scheduled.

`'(org-agenda-skip-entry-if 'deadline)`
> Skip current entry if it has a deadline.

`'(org-agenda-skip-entry-if 'scheduled 'deadline)`
> Skip current entry if it has a deadline, or if it is scheduled.

```
'(org-agenda-skip-entry 'regexp "regular expression")
```
          Skip current entry if the regular expression matches in the entry.

```
'(org-agenda-skip-entry 'notregexp "regular expression")
```
          Skip current entry unless the regular expression matches.

```
'(org-agenda-skip-subtree-if 'regexp "regular expression")
```
          Same as above, but check and skip the entire subtree.

Therefore we could also have written the search for WAITING projects like this, even without defining a special function:

```
(org-add-agenda-custom-command
 '("b" todo "PROJECT"
   ((org-agenda-skip-function '(org-agenda-skip-subtree-if
                                'regexp ":waiting:"))
    (org-agenda-overriding-header "Projects waiting for something: "))))
```

B.5 Using the property API

Here is a description of the functions that can be used to work with properties.

org-entry-properties &optional *pom which*   [Function]
          Get all properties of the entry at point-or-marker POM. This includes the TODO keyword, the tags, time strings for deadline, scheduled, and clocking, and any additional properties defined in the entry. The return value is an alist, keys may occur multiple times if the property key was used several times. POM may also be nil, in which case the current entry is used. If WHICH is nil or 'all', get all properties. If WHICH is 'special' or 'standard', only get that subclass.

org-entry-get *pom property* &optional *inherit*   [Function]
          Get value of PROPERTY for entry at point-or-marker POM. By default, this only looks at properties defined locally in the entry. If INHERIT is non-nil and the entry does not have the property, then also check higher levels of the hierarchy. If INHERIT is the symbol `selective`, use inheritance if and only if the setting of `org-use-property-inheritance` selects PROPERTY for inheritance.

org-entry-delete *pom property*   [Function]
          Delete the property PROPERTY from entry at point-or-marker POM.

org-entry-put *pom property value*   [Function]
          Set PROPERTY to VALUE for entry at point-or-marker POM.

org-buffer-property-keys &optional *include-specials*   [Function]
          Get all property keys in the current buffer.

org-insert-property-drawer   [Function]
          Insert a property drawer at point.

org-entry-add-to-multivalued-property *pom property value*   [Function]
          Treat the value of the property PROPERTY as a whitespace-separated list of values and make sure that VALUE is in this list.

org-entry-remove-from-multivalued-property *pom property*   [Function]
        value
        Treat the value of the property PROPERTY as a whitespace-separated list of values
        and make sure that VALUE is *not* in this list.

org-entry-member-in-multivalued-property *pom property value*   [Function]
        Treat the value of the property PROPERTY as a whitespace-separated list of values
        and check if VALUE is in this list.

B.6 Using the mapping API

Org has sophisticated mapping capabilities to find all entries satisfying certain criteria.
Internally, this functionality is used to produce agenda views, but there is also an API that
can be used to execute arbitrary functions for each or selected entries. The main entry
point for this API is:

org-map-entries *func* &optional *match scope* &rest *skip*   [Function]
        Call FUNC at each headline selected by MATCH in SCOPE.

        FUNC is a function or a lisp form. The function will be called without arguments,
        with the cursor positioned at the beginning of the headline. The return values of all
        calls to the function will be collected and returned as a list.

        MATCH is a tags/property/todo match as it is used in the agenda tags view. Only
        headlines that are matched by this query will be considered during the iteration.
        When MATCH is nil or t, all headlines will be visited by the iteration.

        SCOPE determines the scope of this command. It can be any of:

        `nil`   the current buffer, respecting the restriction if any
        `tree`   the subtree started with the entry at point
        `file`   the current buffer, without restriction
        `file-with-archives`
                    the current buffer, and any archives associated with it
        `agenda`   all agenda files
        `agenda-with-archives`
                    all agenda files with any archive files associated with them
        `(file1 file2 ...)`
                    if this is a list, all files in the list will be scanned

        The remaining args are treated as settings for the skipping facilities of the scanner.
        The following items can be given here:

        `archive`   skip trees with the archive tag
        `comment`   skip trees with the COMMENT keyword
        `function or Lisp form`
                    will be used as value for `org-agenda-skip-function`,
                    so whenever the the function returns t, FUNC
                    will not be called for that entry and search will
                    continue from the point where the function leaves it

The function given to that mapping routine can really do anything you like. It can uce the property API (see Section B.5 [Using the property API], page 125) to gather more information about the entry, or in order to change metadate in the entry. Here are a couple of functions that might be handy:

org-todo &optional *arg*   [Function]
> Change the TODO state of the entry, see the docstring of the functions for the many possible values for the argument ARG.

org-priority &optional *action*   [Function]
> Change the priority of the entry, see the docstring of this function for the possible values for ACTION.

org-toggle-tag *tag* &optional *onoff*   [Function]
> Toggle the tag TAG in the current entry. Setting ONOFF to either `on` or `off` will not toggle tag, but ensure that it is either on or off.

org-promote   [Function]
> Promote the current entry.

org-demote   [Function]
> Demote the current entry.

Here is a simple example that will turn all entries in the current file with a tag `TOMORROW` into TODO entries with the keyword `UPCOMING`. Entries in comment trees and in archive trees will be ignored.

```
(org-map-entries
   '(org-todo "UPCOMING")
   "+TOMORROW" 'file 'archive 'comment)
```

The following example counts the number of entries with TODO keyword `WAITING`, in all agenda files.

```
(length (org-map-entries t "/+WAITING" nil 'agenda))
```

Appendix C History and Acknowledgments

Org was borne in 2003, out of frustration over the user interface of the Emacs Outline mode. I was trying to organize my notes and projects, and using Emacs seemed to be the natural way to go. However, having to remember eleven different commands with two or three keys per command, only to hide and show parts of the outline tree, that seemed entirely unacceptable to me. Also, when using outlines to take notes, I constantly want to restructure the tree, organizing it parallel to my thoughts and plans. *Visibility cycling* and *structure editing* were originally implemented in the package '`outline-magic.el`', but quickly moved to the more general '`org.el`'. As this environment became comfortable for project planning, the next step was adding *TODO entries*, basic *time stamps*, and *table support*. These areas highlight the two main goals that Org 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.

A special thanks goes to *Bastien Guerry* who has not only writen a large number of extensions to Org (most of them integrated into the core by now), but has also helped the development and maintenance of Org so much that he should be considered co-author of this package.

Since the first release, literally thousands of emails to me or on `emacs-orgmode@gnu.org` have provided a constant stream of bug reports, feedback, new ideas, and sometimes 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. The list may not be complete, if I have forgotten someone, please accept my apologies and let me know.

- *Russel Adams* came up with the idea for drawers.
- *Thomas Baumann* wrote '`org-bbdb.el`' and '`org-mhe.el`'.
- *Christophe Bataillon* created the great unicorn logo that we use on the Org-mode website.
- *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. He also came up with the idea of properties, and that there should be an API for them.
- *Kees Dullemond* used to edit projects lists directly in HTML and so inspired some of the early development, including HTML export. He also asked for a way to narrow wide table columns.
- *Christian Egli* converted the documentation into Texinfo format, patched CSS formatting into the HTML exporter, and inspired the agenda.
- *David Emery* provided a patch for custom CSS support in exported HTML agendas.
- *Nic Ferrier* contributed mailcap and XOXO support.

- *Miguel A. Figueroa-Villanueva* implemented hierarchical checkboxes.
- *John Foerch* figured out how to make incremental search show context around a match in a hidden outline tree.
- *Niels Giesen* had the idea to automatically archive DONE trees.
- *Bastien Guerry* wrote the LaTeX exporter and '`org-bibtex.el`', and has been prolific with patches, ideas, and bug reports.
- *Kai Grossjohann* pointed out key-binding conflicts with other packages.
- *Bernt Hansen* has driven much of the support for auto-repeating tasks, task state change logging, and the clocktable. His clear explanations have been critical when we started to adopt the GIT version control system.
- *Phil Jackson* wrote '`org-irc.el`'.
- *Scott Jaderholm* proposed footnotes, control over whitespace between folded entries, and column view for properties.
- *Tokuya Kameshima* wrote '`org-wl.el`' and '`org-mew.el`'.
- *Shidai Liu* ("Leo") asked for embedded LaTeX and tested it. He also provided frequent feedback and some patches.
- *Jason F. McBrayer* suggested agenda export to CSV format.
- *Max Mikhanosha* came up with the idea of refiling.
- *Dmitri Minaev* sent a patch to set priority limits on a per-file basis.
- *Stefan Monnier* provided a patch to keep the Emacs-Lisp compiler happy.
- *Rick Moynihan* proposed to allow multiple TODO sequences in a file and to be able to quickly restrict the agenda to a subtree.
- *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.
- *Takeshi Okano* translated the manual and David O'Toole's tutorial into Japanese.
- *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.
- *Sebastian Rose* wrote '`org-info.js`', a Java script for displaying webpages derived from Org using an Info-like, or a folding interface with single key navigation.
- *Frank Ruell* solved the mystery of the `keymapp nil` bug, a conflict with '`allout.el`'.
- *Jason Riedy* generalized the send-receive mechanism for orgtbl tables with extensive patches.
- *Philip Rooke* created the Org reference card, provided lots of feedback, developed and applied standards to the Org documentation.
- *Christian Schlauer* proposed angular brackets around links, among other things.

- Linking to VM/BBDB/Gnus was first inspired by *Tom Shannon*'s '`organizer-mode.el`'.
- *Ilya Shlyakhter* proposed the Archive Sibling.
- *Daniel Sinder* came up with the idea of internal archiving by locking subtrees.
- *Dale Smith* proposed link abbreviations.
- *Adam Spiers* asked for global linking commands, inspired the link extension system, added support for mairix, and proposed the mapping API.
- *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`', '`planner.el`', and '`muse.el`', which have similar goals as Org. Initially the development of Org was fully independent because I was not aware of the existence of these packages. But with time I have accasionally looked at John's code and learned a lot from it. John has also contributed a number of great ideas and patches directly to Org, including the file `org-mac-message.el`'
- *Carsten Wimmer* suggested some changes and helped fix a bug in linking to Gnus.
- *Roland Winkler* requested additional key bindings to make Org work on a tty.
- *Piotr Zielinski* wrote '`org-mouse.el`', proposed agenda blocks and contributed various ideas and code snippets.

The Main Index

Key Index

D

E