Org Mode Compact Guide

Release 9.6

The Org Mode Developers

Table of Contents

1 Introduction

Org is a mode for keeping notes, maintaining TODO lists, and doing project planning with a fast and effective plain-text system. It is also an authoring and publishing system, and it supports working with source code for literal programming and reproducible research.

This document is a much compressed derivative of the `org`. It contains all basic features and commands, along with important hints for customization. It is intended for beginners who would shy back from a 200 pages manual because of sheer size.

Installation

> **Important:** If you are using a version of Org that is part of the Emacs distribution, please skip this section and go directly to [Activation], page 1.

If you have downloaded Org from the web, either as a distribution '`.zip`' or '`.tar`' file, or as a Git archive, it is best to run it directly from the distribution directory. You need to add the '`lisp/`' subdirectories to the Emacs load path. To do this, add the following line to your Emacs init file:

```
(add-to-list 'load-path "~/path/to/orgdir/lisp")
```

If you have been using git or a tar ball to get Org, you need to run the following command to generate autoload information.

```
make autoloads
```

Activation

Add the following lines to your Emacs init file to define *global* keys for three commands that are useful in any Emacs buffer, not just Org buffers. Please choose suitable keys yourself.

```
(global-set-key (kbd "C-c l") #'org-store-link)
(global-set-key (kbd "C-c a") #'org-agenda)
(global-set-key (kbd "C-c c") #'org-capture)
```

Files with extension '`.org`' will be put into Org mode automatically.

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`. For information on how to submit bug reports, see the main manual.

2 Document Structure

Org is an outliner. Outlines allow a document to be organized in a hierarchical structure, which, least for me, is the best representation of notes and thoughts. An overview of this structure is achieved by folding, i.e., 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 and hide functionalities into a single command, `org-cycle`, which is bound to the `TAB` key.

2.1 Headlines

Headlines define the structure of an outline tree. The headlines in Org start on the left margin[1] with one or more stars followed by a space. For example:

```
* Top level headline
** Second level
*** Third level
    some text
*** Third level
    more text
* Another top level headline
```

Note that a headline named after `org-footnote-section`, which defaults to '`Footnotes`', is considered as special. A subtree with this headline will be silently ignored by exporting functions.

Some people find the many stars too noisy and would prefer an outline that has whitespace followed by a single star as headline starters. See Chapter 15 [Miscellaneous], page 38 for a setup to realize this.

2.2 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 --.
'-----------------------------------'
```

When called with a prefix argument (*`C-u TAB`*), or with the Shift key, global cycling is invoked.

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

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

`C-u C-u C-u TAB`
Show all, including drawers.

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

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 a '`STARTUP`' keyword to '`overview`', '`content`', '`showall`', '`showeverything`' or '`show<n>levels`' (n = 2..5) like this:

```
#+STARTUP: content
```

2.3 Motion

The following commands jump to other headlines in the buffer.

C-c C-n Next heading.

C-c C-p Previous heading.

C-c C-f Next heading same level.

C-c C-b Previous heading same level.

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

2.4 Structure Editing

M-RET Insert new heading with same level as current. If point is in a plain list item, a new item is created (see Section 2.6 [Plain Lists], page 4). 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[2].

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

`TAB` in new empty entry
In a new entry with no text yet, `TAB` cycles through reasonable levels.

M-LEFT
M-RIGHT Promote or demote current heading by one level.

M-UP
M-DOWN Move subtree up or down, i.e., swap with previous or next subtree of same level.

C-c C-w Refile entry or region to a different location. See Section 9.2 [Refile and Copy], page 21.

C-x n s
C-x n w Narrow buffer to current subtree and widen it again.

When there is an active region (Transient Mark mode), promotion and demotion work on all headlines in the region.

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

2.5 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[3]. 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. Each match is also highlighted; the highlights disappear by pressing `C-c C-c`.

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

2.6 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 12). Org supports editing such lists, and every exporter (see Chapter 12 [Exporting], page 31) can parse and format them.

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

- *Unordered* list items start with '`-`', '`+`', or '`*`' as bullets.
- *Ordered* list items start with '`1.`', or '`1)`'.
- *Description* list use '`::`' to separate the *term* from the description.

Items belonging to the same list must have the same indentation on the first line. An item ends before the next line that is indented like its bullet/number, or less. A list ends when all items are closed, or before two blank lines. An example:

```
* Lord of the Rings
  My favorite scenes are (in this order)
  1. The attack of the Rohirrim
  2. Eowyn's fight with the witch king
     + this was already my favorite scene in the book
     + I really like Miranda Otto.
  Important actors in this film are:
  - Elijah Wood :: He plays Frodo
  - Sean Astin :: He plays Sam, Frodo's friend.
```

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

`TAB` Items can be folded just like headline levels.

`M-RET` Insert new item at current level. With a prefix argument, force a new heading (see Section 2.4 [Structure Editing], page 3).

`M-S-RET` Insert a new item with a checkbox (see Section 5.6 [Checkboxes], page 12).

[3] See also the variable `org-show-context-detail` to decide how much context is shown around each match.

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-LEFT
M-RIGHT Decrease/increase the indentation of an item, leaving children alone.

M-S-LEFT
M-S-RIGHT
Decrease/increase the indentation of the item, including subitems.

C-c C-c If there is a checkbox (see Section 5.6 [Checkboxes], page 12) in the item line, toggle the state of the checkbox. Also verify bullets and indentation consistency in the whole list.

C-c - Cycle the entire list level through the different itemize/enumerate bullets ('`-`', '`+`', '`*`', '`1.`', '`1)`').

3 Tables

Org comes with a fast and intuitive table editor. Spreadsheet-like calculations are supported in connection with the Emacs Calc package (see calc).

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. Even faster would be to type '|Name|Phone|Age' followed by *C-c RET*.

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 point was moved into a new field with* TAB*, S-TAB or* RET, the field is automatically made blank.

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.

If there is no active region, this command creates an empty Org table. But it is easier just to start typing, like *| N a m e | P h o n e | A g e RET | - TAB*.

Re-aligning and field motion

C-c C-c Re-align the table without moving point.

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.

S-UP
S-DOWN
S-LEFT
S-RIGHT Move a cell up, down, left, and right by swapping with adjacent cell.

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 point 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 RET
: Insert a horizontal line below current row, and move the point into the row below that 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.

4 Hyperlinks

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

Org recognizes plain URIs, possibly wrapped within angle brackets, 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, with all brackets present, Org changes the display so that '`DESCRIPTION`' is displayed instead of '`[[LINK][DESCRIPTION]]`' and '`LINK`' is displayed instead of '`[[LINK]]`'. To edit the invisible *LINK* part, use *`C-c C-l`* with the point on the link.

Internal links

If the link does not look like a URL, it is considered to be internal in the current file. The most important case is a link like '`[[#my-custom-id]]`' which links to the entry with the '`CUSTOM_ID`' property '`my-custom-id`'.

Links such as '`[[My Target]]`' or '`[[My Target][Find my target]]`' lead to a text search in the current file for the corresponding target, which looks like '`<<My Target>>`'.

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. Here are some examples:

'`http://www.astro.uva.nl/=dominik`'	on the web
'`file:/home/dominik/images/jupiter.jpg`'	file, absolute path
'`/home/dominik/images/jupiter.jpg`'	same as above
'`file:papers/last.pdf`'	file, relative path
'`./papers/last.pdf`'	same as above
'`file:projects.org`'	another Org file
'`docview:papers/last.pdf::NNN`'	open in DocView mode at page *NNN*
'`id:B7423F4D-2E8A-471B-8810-C40F074717E9`'	link to heading by ID
'`news:comp.emacs`'	Usenet link
'`mailto:adent@galaxy.net`'	mail link
'`mhe:folder#id`'	MH-E message link
'`rmail:folder#id`'	Rmail message link
'`gnus:group#id`'	Gnus article link
'`bbdb:R.*Stallman`'	BBDB link (with regexp)
'`irc:/irc.com/#emacs/bob`'	IRC link
'`info:org#Hyperlinks`'	Info node link

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 colon. Here are a few examples,, together with an explanation:

‘`file:~/code/main.c::255`’	Find line 255
‘`file:~/xx.org::My Target`’	Find ‘`<<My Target>>`’
‘`[[file:~/xx.org::#my-custom-id]]`’	Find entry with a custom ID

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.

The main function is `org-store-link`, called with *`M-x org-store-link`*. Because of its importance, we suggest to bind it to a widely available key (see [Activation], page 1). It stores a link to the current location. The link is stored for later insertion into an Org buffer—see below.

From an Org buffer, the following commands create, navigate or, more generally, act on links.

`C-c C-l` Insert a link. This prompts for a link to be inserted into the buffer. You can just type a link, or use history keys `UP` and `DOWN` to access stored links. You will be prompted for the description part of the link.

When called with a *`C-u`* prefix argument, file name completion is used to link to a file.

`C-c C-l` (with point on existing link)
When point 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.

`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.

5 TODO Items

Org mode does not require TODO lists to live in separate documents. Instead, TODO items are part of a 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 TODO items remain in the context from which they emerged.

Org mode provides methods to give you an overview of all the things that you have to do, collected from many files.

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 agenda buffer with the *t* command key (see Section 10.7 [Agenda Commands], page 25).

S-RIGHT
S-LEFT Select the following/preceding TODO state, similar to cycling.

C-c / t View TODO items in a *sparse tree* (see Section 2.5 [Sparse Trees], page 4). Folds the entire buffer, but shows all TODO items—with not-DONE state—and the headings hierarchy above them.

M-x org-agenda t
Show the global TODO list. Collects the TODO items (with not-DONE states) from all agenda files (see Chapter 10 [Agenda Views], page 23) into a single buffer. See Section 10.4 [Global TODO List], page 24, for more information.

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

Changing a TODO state can also trigger tag changes. See the docstring of the option `org-todo-state-tags-triggers` for details.

5.2 Multi-state Workflow

You can use TODO keywords to indicate *sequential* working progress states:

```
(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 do not provide the separator bar, the last state is used as the 'DONE' state. With this setup, the command *C-c C-t* cycles an entry from 'TODO' to 'FEEDBACK', then to 'VERIFY', and finally to 'DONE' and 'DELEGATED'.

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. Your setup would then look like this:

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

The keywords should all be different, this helps Org mode to keep track of which subsequence should be used for a given entry. The example also shows how to define keys for fast access of a particular state, by adding a letter in parenthesis after each keyword—you will be prompted for the key after *C-c C-t*.

To define TODO keywords that are valid only in a single file, use the following text anywhere in the file.

```
#+TODO: TODO(t) | DONE(d)
#+TODO: REPORT(r) BUG(b) KNOWNCAUSE(k) | FIXED(f)
#+TODO: | CANCELED(c)
```

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.

5.3 Progress Logging

To record a timestamp and a note when changing a TODO state, call the command `org-todo` with a prefix argument.

C-u C-c C-t
: Prompt for a note and record a the time of the TODO state change.

Org mode can also automatically record a timestamp and optionally 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 19.

Closing items

The most basic logging is to keep track of *when* a certain TODO item was marked as done. This can be 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]`' is inserted just after the headline.

If you want to record a note along with the timestamp, use[2]

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

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

Tracking TODO state changes

You might want to keep track of TODO state changes. You can either record just a time-stamp, or a time-stamped note for a change. These records are inserted after the headline

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

[2] The corresponding in-buffer setting is '`#+STARTUP: logenotedone`'.

as an itemized list. When taking a lot of notes, you might want to get the notes out of the way into a drawer. Customize the variable `org-log-into-drawer` to get this behavior.

For state logging, Org mode expects configuration on a per-keyword basis. This is achieved by adding special markers '`!`' (for a timestamp) and '`@`' (for a note) in parentheses after each keyword. For example:

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

defines TODO keywords and fast access keys, and also request that a time is recorded when the entry is set to '`DONE`', and that a note is recorded when switching to '`WAIT`' or '`CANCELED`'. The same syntax works also when setting `org-todo-keywords`.

5.4 Priorities

If you use Org mode extensively, you may end up with 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
```

Org mode supports three priorities: '`A`', '`B`', and '`C`'. '`A`' is the highest, '`B`' the default if none is given. Priorities make a difference only in the agenda.

C-c , Set the priority of the current headline. Press *A*, *B* or *C* to select a priority, or SPC to remove the cookie.

S-UP (`org-priority-up`)
S-DOWN (`org-priority-down`)
Increase/decrease the priority of the current headline.

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. To keep an overview of the fraction of subtasks that have already been marked as done, insert either '`[/]`' or '`[%]`' anywhere in the headline. These cookies are updated each time the TODO status of a child changes, or when pressing *C-c C-c* on the cookie. For example:

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

5.6 Checkboxes

Every item in a plain list (see Section 2.6 [Plain Lists], page 4) can be made into a checkbox by starting it with the string '`[ ]`'. Checkboxes are not included into the global TODO list, so they are often great to split a task into a number of simple steps.

Here is an example of a checkbox list.

```
* TODO Organize party [2/4]
```

```
- [-] call people [1/2]
  - [ ] Peter
  - [X] Sarah
- [X] order food
```

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

The following commands work with checkboxes:

C-c C-c Toggle checkbox status or—with prefix argument—checkbox presence at point.

M-S-RET Insert a new item with a checkbox. This works only if point is already in a plain list item (see Section 2.6 [Plain Lists], page 4).

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:'. Tags by default are in bold face with the same color as the headline.

Tag inheritance

Tags make use of the hierarchical structure of outline trees. If a heading has a certain tag, all subheadings 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 has 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 just as if these tags were defined in a hypothetical level zero that surrounds the entire file. Use a line like this[1]:

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

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-q Enter new tags for the current headline. Org mode either offers completion or a special single-key interface for setting tags, see below.

C-c C-c When point is in a headline, this does the same as *C-c C-q*.

Org supports 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 using the 'TAGS' keyword, like

```
#+TAGS: @work @home @tennisclub
#+TAGS: laptop car pc sailboat
```

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 init 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:

[1] As with all these in-buffer settings, pressing *C-c C-c* activates any changes in the line.

```
(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' keyword as:

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

Tag groups

A tag can be defined as a *group tag* for a set of other tags. The group tag can be seen as the "broader term" for its set of tags.

You can set group tags by using brackets and inserting a colon between the group tag and its related tags:

```
#+TAGS: [ GTD : Control Persp ]
```

or, if tags in the group should be mutually exclusive:

```
#+TAGS: { Context : @Home @Work }
```

When you search for a group tag, it return matches for all members in the group and its subgroups. In an agenda view, filtering by a group tag displays or hide headlines tagged with at least one of the members of the group or any of its subgroups.

If you want to ignore group tags temporarily, toggle group tags support with `org-toggle-tags-groups`, bound to *C-c C-x q*.

Tag searches

C-c / m or *C-c *

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.

M-x org-agenda m

Create a global list of tag matches from all agenda files. See Section 10.5 [Matching Tags and Properties], page 24.

M-x org-agenda M

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

These commands all prompt for a match string which allows basic Boolean logic like '`+boss+urgent-project1`', to find entries with tags '`boss`' and '`urgent`', but not '`project1`', or '`Kathy|Sally`' to find entries which are tagged, like '`Kathy`' or '`Sally`'. The full syntax of the search string is rich and allows also matching against TODO keywords, entry levels and properties. For a more detailed description with many examples, see Section 10.5 [Matching Tags and Properties], page 24.

7 Properties

Properties are key-value pairs associated with an entry. They live in a special drawer with the name 'PROPERTIES'. Each property is specified on a single line, with the key (surrounded by colons) first, and the value after it:

```
* CD collection
** Classic
*** Goldberg Variations
    :PROPERTIES:
    :Title:     Goldberg Variations
    :Composer:  J.S. Bach
    :Publisher: Deutsche Grammophon
    :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 applies 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 pre-define publishers and the number of disks in a box like this:

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

The following commands help to work with properties:

`C-c C-x p` Set a property. This prompts for a property name and a value.

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

To create sparse trees and special lists with selection based on properties, the same commands are used as for tag searches (see Chapter 6 [Tags], page 14). The syntax for the search string is described in Section 10.5 [Matching Tags and Properties], page 24.

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.

8.1 Timestamps

A timestamp is a specification of a date—possibly with a 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>`'. A timestamp 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 Section 10.3 [Built-in Agenda Views], page 23). We distinguish:

Plain timestamp; Event; Appointment
: A simple timestamp just assigns a date/time to an item. This is just like writing down an appointment or event in a paper agenda.

```
* Meet Peter at the movies
  <2006-11-01 Wed 19:15>
* Discussion on climate change
  <2006-11-02 Thu 20:00-22:00>
```

Timestamp with repeater interval
: A timestamp 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 hours (h), days (d), weeks (w), months (m), or years (y). The following shows up in the agenda every Wednesday:

```
* Pick up Sam at school
  <2007-05-16 Wed 12:30 +1w>
```

Diary-style expression entries
: For more complex date specifications, Org mode supports using the special expression diary entries implemented in the Emacs Calendar package. For example, with optional time:

```
* 22:00-23:00 The nerd meeting on every 2nd Thursday of the month
  <%%(diary-float t 4 2)>
```

Time/Date range
: Two timestamps connected by '`--`' denote a range.

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

Inactive timestamp
: Just like a plain timestamp, but with square brackets instead of angular ones. These timestamps 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 timestamps, they need to be in the specific format. All commands listed below produce timestamps in the correct format.

C-c . Prompt for a date and insert a corresponding timestamp. When point is at an existing timestamp in the buffer, the command is used to modify this timestamp instead of inserting a new one. When this command is used twice in succession, a time range is inserted. With a prefix argument, it also adds the current time.

C-c ! Like *C-c .*, but insert an inactive timestamp that does not cause an agenda entry.

S-LEFT
S-RIGHT Change date at point by one day.

S-UP
S-DOWN On the beginning or enclosing bracket of a timestamp, change its type. Within a timestamp, change the item under point. Point can be on a year, month, day, hour or minute. When the timestamp contains a time range like '`15:30-16:30`', modifying the first time also shifts the second, shifting the time block with constant length. To change the length, modify the second time.

When Org mode prompts for a date/time, it accepts any string containing some date and/or time information, and intelligently interprets the string, deriving defaults for unspecified information from the current date and time. You can also select a date in the pop-up calendar. See the manual for more information on how exactly the date/time prompt works.

8.3 Deadlines and Scheduling

A timestamp may be preceded by special keywords to facilitate planning:

C-c C-d Insert '`DEADLINE`' keyword along with a time stamp, in the line following the headline.

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 is listed in the agenda. In addition, the agenda for *today* carries a warning about the approaching or missed deadline, starting `org-deadline-warning-days` before the due date, and continuing until the entry is marked as done. An example:

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

C-c C-s Insert '`SCHEDULED`' keyword along with a stamp, in the line following the headline.

Meaning: you are planning to start working on that task on the given date[1].

[1] This is quite different from what is normally understood by *scheduling a meeting*, which is done in Org by just inserting a time stamp without keyword.

The headline is listed under the given date[2]. In addition, a reminder that the scheduled date has passed is present in the compilation for *today*, until the entry is marked as done, i.e., the task is automatically forwarded until completed.

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

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 timestamps. 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.

8.4 Clocking Work Time

Org mode allows you to clock the time you spend on specific tasks in 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. When called with a *C-u* prefix argument, select the task from a list of recently clocked tasks.

C-c C-x C-o
: Stop the clock (clock-out). This 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`'.

C-c C-x C-e
: Update the effort estimate for the current clock task.

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

C-c C-x C-j
: Jump to the headline of the currently clocked in task. With a *C-u* prefix argument, select the target task from a list of recently clocked tasks.

The *l* key may be used in the agenda (see Section 10.3 [Built-in Agenda Views], page 23) to show which tasks have been worked on or closed during a day.

[2] It will still be listed on that date after it has been marked as done. If you do not like this, set the variable `org-agenda-skip-scheduled-if-done`.

9 Capture, Refile, Archive

An important part of any organization system is the ability to quickly capture new ideas and tasks, and to associate reference material with them. Org does this using a process called *capture*. It also can store files related to a task (*attachments*) in a special directory. Once in the system, tasks and projects need to be moved around. Moving completed project trees to an archive file keeps the system compact and fast.

9.1 Capture

Capture lets you quickly store notes with little interruption of your work flow. You can define templates for new entries and associate them with different targets for storing notes.

Setting up capture

The following customization sets a default target[1] file for notes.

```
(setq org-default-notes-file (concat org-directory "/notes.org"))
```

You may also define a global key for capturing new material (see [Activation], page 1).

Using capture

M-x org-capture
: Start a capture process, placing you into a narrowed indirect buffer to edit.

C-c C-c
: Once you have finished entering information into the capture buffer, *C-c C-c* returns you to the window configuration before the capture process, so that you can resume your work without further distraction.

C-c C-w
: Finalize the capture process by refiling the note to a different place (see Section 9.2 [Refile and Copy], page 21).

C-c C-k
: Abort the capture process and return to the previous state.

Capture templates

You can use templates for different types of capture items, and for different target locations. Say you would like to use one template to create general TODO entries, and you want to put these entries under the heading '`Tasks`' in your file '`~/org/gtd.org`'. Also, a date tree in the file '`journal.org`' should capture journal entries. A possible configuration would look like:

```
(setq org-capture-templates
      '(("t" "Todo" entry (file+headline "~/org/gtd.org" "Tasks")
         "* TODO %?\n  %i\n  %a")
        ("j" "Journal" entry (file+datetree "~/org/journal.org")
         "* %?\nEntered on %U\n  %i\n  %a")))
```

If you then press *t* from the capture menu, Org will prepare the template for you like this:

[1] Using capture templates, you get finer control over capture locations. See [Capture templates], page 20.

```
* TODO
  [[file:LINK TO WHERE YOU INITIATED CAPTURE]]
```

During expansion of the template, special %-escapes[2] allow dynamic insertion of content. Here is a small selection of the possibilities, consult the manual for more.

'`%a`'	annotation, normally the link created with `org-store-link`
'`%i`'	initial content, the region when capture is called with *C-u*
'`%t`', '`%T`'	timestamp, date only, or date and time
'`%u`', '`%U`'	like above, but inactive timestamps
'`%?`'	after completing the template, position point here

9.2 Refile and Copy

When reviewing the captured data, you may want to refile or to copy 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 or region at point. This command offers possible locations for refiling the entry and lets you select one with completion. The item (or all items in the region) is filed below the target heading as a 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.

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.

C-c M-w Copying works like refiling, except that the original note is not deleted.

9.3 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. Archiving is important to keep your working files compact and global searches like the construction of agenda views fast.

The most common archiving action is to move a project tree to another file, the archive file.

C-c C-x C-a
Archive the current entry using the command specified in the variable `org-archive-default-command`.

C-c C-x C-s or short *C-c $*
Archive the subtree starting at point position to the location given by `org-archive-location`.

[2] If you need one of these sequences literally, escape the '`%`' with a backslash.

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. You can also choose what heading to file archived items under, with the possibility to add them to a datetree in a file. For information and examples on how to specify the file and the heading, see the documentation string of the variable `org-archive-location`.

There is also an in-buffer option for setting this variable, for example:

```
#+ARCHIVE: %s_done::
```

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.

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. Remote editing from the agenda buffer means, for example, that you can change the dates of deadlines and appointments from the agenda buffer. For commands available in the Agenda buffer, see Section 10.7 [Agenda Commands], page 25.

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`.

`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.

10.2 The Agenda Dispatcher

The views are created through a dispatcher, accessible with `M-x org-agenda`, or, better, bound to a global key (see [Activation], page 1). It displays a menu from which 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 [Built-in Agenda Views], page 23).

`t`
`T` Create a list of all TODO items (see Section 10.4 [Global TODO List], page 24).

`m`
`M` Create a list of headlines matching a given expression (see Section 10.5 [Matching Tags and Properties], page 24).

`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.

10.3 The Weekly/Daily Agenda

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

M-x org-agenda a
: Compile an agenda for the current week from a list of Org files. The agenda shows the entries for each day.

Org mode understands the syntax of the diary and allows you to use diary expression entries directly in Org files:

```
* Holidays
  :PROPERTIES:
  :CATEGORY: Holiday
  :END:
%%(org-calendar-holiday)   ; special function for holiday names

* Birthdays
  :PROPERTIES:
  :CATEGORY: Ann
  :END:
%%(org-anniversary 1956  5 14) Arthur Dent is %d years old
%%(org-anniversary 1869 10  2) Mahatma Gandhi would be %d years old
```

Org can interact with Emacs appointments notification facility. To add the appointments of your agenda files, use the command `org-agenda-to-appt`.

10.4 The Global TODO List

The global TODO list contains all unfinished TODO items formatted and collected into a single place. Remote editing of TODO items lets you can change the state of a TODO entry with a single key press. For commands available in the TODO list, see Section 10.7 [Agenda Commands], page 25.

M-x org-agenda t
: Show the global TODO list. This collects the TODO items from all agenda files (see Chapter 10 [Agenda Views], page 23) into a single buffer.

M-x org-agenda T
: Like the above, but allows selection of a specific TODO keyword.

10.5 Matching Tags and Properties

If headlines in the agenda files are marked with *tags* (see Chapter 6 [Tags], page 14), or have properties (see Chapter 7 [Properties], page 16), you can select headlines based on this metadata and collect them into an agenda buffer. The match syntax described here also applies when creating sparse trees with *C-c / m*.

M-x org-agenda 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 14). If you often need a specific search, define a custom command for it (see Section 10.2 [Agenda Dispatcher], page 23).

M-x org-agenda M
: Like *m*, but only select headlines that are also TODO items.

A search string can use Boolean operators '`&`' for AND and '`|`' for OR. '`&`' binds more strongly than '`|`'. Parentheses are currently not implemented. Each element in the search is either a tag, a regular expression matching tags, or an expression like '`PROPERTY OPERATOR VALUE`' with a comparison operator, accessing a property value. Each element may be preceded by '`-`' to select against it, and '`+`' is syntactic sugar for positive selection. The AND operator '`&`' is optional when '`+`' or '`-`' is present. Here are some examples, using only tags.

'`+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`'.

You may also test for properties at the same time as matching tags, see the manual for more information.

10.6 Search View

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

`M-x org-agenda s` (`org-search-view`)
: This is a special search that lets you select entries by matching a substring or specific words using a boolean logic.

For example, the search string '`computer equipment`' matches entries that contain '`computer equipment`' as a substring.

Search view can also search for specific keywords in the entry, using Boolean logic. The search string '`+computer +wifi -ethernet -{8\.11[bg]}`' matches 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 also searches the files listed in `org-agenda-text-search-extra-files`.

10.7 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. This is just a selection of the many commands, explore the agenda menu and the manual for a complete list.

Motion

`n` Next line (same as `DOWN` and *`C-n`*).

`p` Previous line (same as `UP` and *`C-p`*).

View/Go to Org file

`SPC` Display the original location of the item in another window. With a prefix argument, make sure that drawers stay folded.

`TAB` Go to the original location of the item in another window.

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

Change display

`o` Delete other windows.

`v d` or short `d`
Switch to day view.

`v w` or short `w`
Switch to week view.

`f` Go forward in time to display the span following the current one. For example, if the display covers a week, switch to the following week.

`b` Go backward in time to display earlier dates.

`.` Go to today.

`j` Prompt for a date and go there.

`v l` or `v L` or short `l`
Toggle Logbook mode. In Logbook mode, entries that were marked as done while logging was on (see the variable `org-log-done`) are shown in the agenda, as are entries that have been clocked on that day. When called with a `C-u` prefix argument, show all possible logbook entries, including state changes.

`r`
`g` Recreate the agenda buffer, for example to reflect the changes after modification of the timestamps of items.

`s` Save all Org buffers in the current Emacs session, and also the locations of IDs.

Remote editing

`0--9` Digit argument.

`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.

`C-c C-w` Refile the entry at point.

`a` Archive the subtree corresponding to the entry at point using the default archiving command set in `org-archive-default-command`.

`$` Archive the subtree corresponding to the current headline.

`C-c C-s` Schedule this item. With a prefix argument, remove the scheduling timestamp

`C-c C-d` Set a deadline for this item. With a prefix argument, remove the deadline.

`S-RIGHT` Change the timestamp associated with the current line by one day into the future.

`S-LEFT` Change the timestamp associated with the current line by one day into the past.

`I` Start the clock on the current item.

`O` Stop the previously started clock.

`X` Cancel the currently running clock.

`J` Jump to the running clock in another window.

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.

10.8 Custom Agenda Views

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` from the agenda dispatcher (see Section 10.2 [Agenda Dispatcher], page 23). You can also directly set it with Emacs Lisp in the Emacs init file. The following example contains all valid agenda views:

```
(setq org-agenda-custom-commands
      '(("w" todo "WAITING")
        ("u" tags "+boss-urgent")
        ("v" tags-todo "+boss-urgent")))
```

The initial string in each entry defines the keys you have to press after the dispatcher command in order to access the command. Usually this is just a single character. 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:

`w` as a global search for TODO entries with '`WAITING`' as the TODO keyword.

`u` as a global tags search for headlines tagged '`boss`' but not '`urgent`'.

`v` The same search, but limiting it to headlines that are also TODO items.

11 Markup for Rich Contents

Org is primarily about organizing and searching through your plain-text notes. However, it also provides a lightweight yet robust markup language for rich text formatting and more. Used in conjunction with the export framework (see Chapter 12 [Exporting], page 31), you can author beautiful documents in Org.

11.1 Paragraphs

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 preserve the line breaks, indentation and blank lines in a region, but otherwise use normal formatting, you can use this construct, which can also be used to format poetry.

```
#+BEGIN_VERSE
 Great clouds overhead
 Tiny black birds rise and fall
 Snow covers Emacs

    ---AlexSchroeder
#+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 documents like this:

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

If you would like to center some text, do it like this:

```
#+BEGIN_CENTER
Everything should be made as simple as possible, \\
but not any simpler
#+END_CENTER
```

11.2 Emphasis and Monospace

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

11.3 Embedded LaTeX

For scientific notes which need to be able to contain mathematical symbols and the occasional formula, Org mode supports embedding LaTeX code into its files. You can directly use TeX-like syntax for special symbols, enter formulas and entire LaTeX environments.

```
The radius of the sun is R_sun = 6.96 x 10^8 m.  On the other hand,
the radius of Alpha Centauri is R_{Alpha Centauri} = 1.28 x R_{sun}.
```

```
\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 \[ a=-\sqrt{2} \].
```

11.4 Literal examples

You can include literal examples that should not be subjected to markup. Such examples are 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 followed by a space. There may also be additional whitespace before the colon:

```
Here is an example
   : 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.

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

To edit the example in a special buffer supporting this language, use `C-c '` to both enter and leave the editing buffer.

11.5 Images

An image is a link to an image file that does not have a description part, for example

```
./img/cat.jpg
```

If you wish to define a caption for the image and maybe a label for internal cross references (see Chapter 4 [Hyperlinks], page 8), make sure that the link is on a line by itself and precede it with '`CAPTION`' and '`NAME`' keywords as follows:

```
#+CAPTION: This is the caption for the next figure link (or table)
#+NAME:   fig:SED-HR4049
[[./img/a.jpg]]
```

11.6 Creating Footnotes

A footnote is defined in a paragraph that is started by a footnote marker in square brackets in column 0, no indentation allowed. The footnote reference is simply the marker in square brackets, inside text. For example:

```
The Org website[fn:1] now looks a lot better than it used to.
...
[fn:1] The link is: https://orgmode.org
```

The following commands handle footnotes:

`C-c C-x f` The footnote action command. When point is on a footnote reference, jump to the definition. When it is at a definition, jump to the (first) reference. Otherwise, create a new footnote. When this command is called with a prefix argument, a menu of additional options including renumbering is offered.

`C-c C-c` Jump between definition and reference.

12 Exporting

Org can convert and export documents to a variety of other formats while retaining as much structure (see Chapter 2 [Document Structure], page 2) and markup (see Chapter 11 [Markup], page 28) as possible.

12.1 The Export Dispatcher

The export dispatcher is the main interface for Org's exports. A hierarchical menu presents the currently configured export formats. Options are shown as easy toggle switches on the same screen.

`C-c C-e` Invokes the export dispatcher interface.

Org exports the entire buffer by default. If the Org buffer has an active region, then Org exports just that region.

12.2 Export Settings

The exporter recognizes special lines in the buffer which provide additional information. These lines may be put anywhere in the file:

```
#+TITLE: I'm in the Mood for Org
```

Most proeminent export options include:

'`TITLE`' the title to be shown
'`AUTHOR`' the author (default taken from `user-full-name`)
'`DATE`' a date, fixed, or an Org timestamp
'`EMAIL`' email address (default from `user-mail-address`)
'`LANGUAGE`' language code, e.g., '`en`'

Option keyword sets can be inserted from the export dispatcher (see Section 12.1 [The Export Dispatcher], page 31) using the '`Insert template`' command by pressing `#`.

12.3 Table of Contents

The table of contents includes all headlines in the document. Its depth is therefore the same as the headline levels in the file. If you need to use a different depth, or turn it off entirely, set the `org-export-with-toc` variable accordingly. You can achieve the same on a per file basis, using the following '`toc`' item in '`OPTIONS`' keyword:

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

Org normally inserts the table of contents directly before the first headline of the file.

12.4 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 first parameter is the file name to include. The optional second parameter specifies the block type: '`example`', '`export`' or '`src`'. The optional third parameter specifies the

source code language to use for formatting the contents. This is relevant to both '`export`' and '`src`' block types.

You can visit the included file with *C-c '*.

12.5 Comment Lines

Lines starting with zero or more whitespace characters followed by one '`#`' and a whitespace are treated as comments and, as such, are not exported.

Likewise, regions surrounded by '`#+BEGIN_COMMENT`' ... '`#+END_COMMENT`' are not exported.

Finally, a '`COMMENT`' keyword at the beginning of an entry, but after any other keyword or priority cookie, comments out the entire subtree. The command below helps changing the comment status of a headline.

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

12.6 ASCII/UTF-8 Export

ASCII export produces an output file containing only plain ASCII characters. This is the simplest and most direct text output. It does not contain any Org markup. UTF-8 export uses additional characters and symbols available in this encoding standards.

C-c C-e t a
C-c C-e t u
: Export as an ASCII file with a '`.txt`' extension. For '`myfile.org`', Org exports to '`myfile.txt`', overwriting without warning. For '`myfile.txt`', Org exports to '`myfile.txt.txt`' in order to prevent data loss.

12.7 HTML Export

Org mode contains an HTML exporter with extensive HTML formatting compatible with XHTML 1.0 strict standard.

C-c C-e h h
: Export as HTML file with a '`.html`' extension. For '`myfile.org`', Org exports to '`myfile.html`', overwriting without warning. *C-c C-e h o* exports to HTML and opens it in a web browser.

The HTML export back-end transforms '`<`' and '`>`' to '`<`' and '`>`'. To include raw HTML code in the Org file so the HTML export back-end can insert that HTML code in the output, use this inline syntax: '`@@html:...@@`'. For example:

```
@@html:<b>@@bold text@@html:</b>@@
```

For larger raw HTML code blocks, use these HTML export code blocks:

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

#+BEGIN_EXPORT html
  All lines between these markers are exported literally
#+END_EXPORT
```

12.8 LaTeX Export

The LaTeX export back-end can handle complex documents, incorporate standard or custom LaTeX document classes, generate documents using alternate LaTeX engines, and produce fully linked PDF files with indexes, bibliographies, and tables of contents, destined for interactive online viewing or high-quality print publication.

By default, the LaTeX output uses the *article* class. You can change this by adding an option like '`#+LATEX_CLASS: myclass`' in your file. The class must be listed in `org-latex-classes`.

C-c C-e l l
: Export to a LaTeX file with a '`.tex`' extension. For '`myfile.org`', Org exports to '`myfile.tex`', overwriting without warning.

C-c C-e l p
: Export as LaTeX file and convert it to PDF file.

C-c C-e l o
: Export as LaTeX file and convert it to PDF, then open the PDF using the default viewer.

The LaTeX export back-end can insert any arbitrary LaTeX code, see Section 11.3 [Embedded LaTeX], page 28. There are three ways to embed such code in the Org file and they all use different quoting syntax.

Inserting in-line quoted with @ symbols:

```
Code embedded in-line @@latex:any arbitrary LaTeX code@@ in a paragraph.
```

Inserting as one or more keyword lines in the Org file:

```
#+LATEX: any arbitrary LaTeX code
```

Inserting as an export block in the Org file, where the back-end exports any code between begin and end markers:

```
#+BEGIN_EXPORT latex
  any arbitrary LaTeX code
#+END_EXPORT
```

12.9 iCalendar Export

A large part of Org mode's interoperability success is its ability to easily export to or import from external applications. The iCalendar export back-end takes calendar data from Org files and exports to the standard iCalendar format.

C-c C-e c f
: Create iCalendar entries from the current Org buffer and store them in the same directory, using a file extension '`.ics`'.

C-c C-e c c
: Create a combined iCalendar file from Org files in `org-agenda-files` and write it to `org-icalendar-combined-agenda-file` file name.

13 Publishing

Org includes a publishing management system that allows you to configure automatic HTML conversion of *projects* composed of interlinked Org files. You can also configure Org to automatically upload your exported HTML pages and related attachments, such as images and source code files, to a web server.

You can also use Org to convert files into PDF, or even combine HTML and PDF conversion so that files are available in both formats on the server.

For detailed instructions about setup, see the manual. Here is an example:

```
(setq org-publish-project-alist
      '(("org"
         :base-directory "~/org/"
         :publishing-function org-html-publish-to-html
         :publishing-directory "~/public_html"
         :section-numbers nil
         :with-toc nil
         :html-head "<link rel=\"stylesheet\"
                     href=\"../other/mystyle.css\"
                     type=\"text/css\"/>")))
```

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

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

C-c C-e P f
: Publish only the current file.

C-c C-e P a
: Publish every project.

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 to any of the commands above.

14 Working with Source Code

Org mode provides a number of features for working with source code, including editing of code blocks in their native major mode, evaluation of code blocks, tangling of code blocks, and exporting code blocks and their results in several formats.

A source code block conforms to this structure:

```
#+NAME: <name>
#+BEGIN_SRC <language> <switches> <header arguments>
  <body>
#+END_SRC
```

where:

- '`<name>`' is a string used to uniquely name the code block,
- '`<language>`' specifies the language of the code block, e.g., '`emacs-lisp`', '`shell`', '`R`', '`python`', etc.,
- '`<switches>`' can be used to control export of the code block,
- '`<header arguments>`' can be used to control many aspects of code block behavior as demonstrated below,
- '`<body>`' contains the actual source code.

Use `C-c '` to edit the current code block. It opens a new major mode edit buffer containing the body of the source code block, ready for any edits. Use `C-c '` again to close the buffer and return to the Org buffer.

Using header arguments

A header argument is specified with an initial colon followed by the argument's name in lowercase.

Header arguments can be set in several ways; Org prioritizes them in case of overlaps or conflicts by giving local settings a higher priority.

System-wide header arguments
: Those are specified by customizing `org-babel-default-header-args` variable, or, for a specific language *LANG* `org-babel-default-header-args:LANG`.

Header arguments in properties
: You can set them using '`header-args`' property (see Chapter 7 [Properties], page 16)—or '`header-args:LANG`' for language *LANG*. Header arguments set through properties drawers apply at the sub-tree level on down.

Header arguments in code blocks
: Header arguments are most commonly set at the source code block level, on the '`BEGIN_SRC`' line:

```
#+NAME: factorial
#+BEGIN_SRC haskell :results silent :exports code :var n=0
  fac 0 = 1
  fac n = n * fac (n-1)
#+END_SRC
```

Code block header arguments can span multiple lines using '`HEADER`' keyword on each line.

Evaluating code blocks

Use `C-c C-c` to evaluate the current code block and insert its results in the Org document. By default, evaluation is only turned on for '`emacs-lisp`' code blocks, however support exists for evaluating blocks in many languages. For a complete list of supported languages see the manual. The following shows a code block and its results.

```
#+BEGIN_SRC emacs-lisp
  (+ 1 2 3 4)
#+END_SRC

#+RESULTS:
: 10
```

The following syntax is used to pass arguments to code blocks using the '`var`' header argument.

```
:var NAME=ASSIGN
```

NAME is the name of the variable bound in the code block body. *ASSIGN* is a literal value, such as a string, a number, a reference to a table, a list, a literal example, another code block—with or without arguments—or the results of evaluating a code block.

Results of evaluation

How Org handles results of a code block execution depends on many header arguments working together. The primary determinant, however, is the '`results`' header argument. It controls the *collection*, *type*, *format*, and *handling* of code block results.

Collection How the results should be collected from the code block. You may choose either '`output`' or '`value`' (the default).

Type What result types to expect from the execution of the code block. You may choose among '`table`', '`list`', '`scalar`', and '`file`'. Org tries to guess it if you do not provide it.

Format How Org processes results. Some possible values are '`code`', '`drawer`', '`html`', '`latex`', '`link`', and '`raw`'.

Handling How to insert the results once properly formatted. Allowed values are '`silent`', '`replace`' (the default), '`append`', or '`prepend`'.

Code blocks which output results to files—e.g.: graphs, diagrams and figures—can accept a '`:file FILENAME`' header argument, in which case the results are saved to the named file, and a link to the file is inserted into the buffer.

Exporting code blocks

It is possible to export the *code* of code blocks, the *results* of code block evaluation, *both* the code and the results of code block evaluation, or *none*. Org defaults to exporting *code* for most languages.

The ‘`exports`’ header argument is to specify if that part of the Org file is exported to, say, HTML or LaTeX formats. It can be set to either ‘`code`’, ‘`results`’, ‘`both`’ or ‘`none`’.

Extracting source code

Use `C-c C-v t` to create pure source code files by extracting code from source blocks in the current buffer. This is referred to as “tangling”—a term adopted from the literate programming community. During tangling of code blocks their bodies are expanded using `org-babel-expand-src-block`, which can expand both variable and “Noweb” style references. In order to tangle a code block it must have a ‘`tangle`’ header argument, see the manual for details.

15 Miscellaneous

Completion

Org has in-buffer completions with *M-TAB*. No minibuffer is involved. Type one or more letters and invoke the hot key to complete the text in-place.

For example, this command will complete TEX symbols after '\', TODO keywords at the beginning of a headline, and tags after ':' in a headline.

Structure Templates

To quickly insert empty structural blocks, such as '`#+BEGIN_SRC`' ... '`#+END_SRC`', or to wrap existing text in such a block, use

C-c C-, Prompt for a type of block structure, and insert the block at point. If the region is active, it is wrapped in the block.

Clean view

Org's default outline with stars and no indents can become too cluttered for short documents. For *book-like* long documents, the effect is not as noticeable. Org provides an alternate stars and indentation scheme, as shown on the right in the following table. It uses only one star and indents text to line with the heading:

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

This kind of view can be achieved dynamically at display time using Org Indent mode (*M-x org-indent-mode RET*), which prepends intangible space to each line. You can turn on Org Indent mode for all files by customizing the variable `org-startup-indented`, or you can turn it on for individual files using

```
#+STARTUP: indent
```

If you want the indentation to be hard space characters so that the plain text file looks as similar as possible to the Emacs display, Org supports you by helping to indent (with `TAB`) text below each headline, by hiding leading stars, and by only using levels 1, 3, etc to get two characters indentation for each level. To get this support in a file, use

```
#+STARTUP: hidestars odd
```