Guide to Planner

a day planner and organizer
for GNU Emacs and Xemacs

This manual is for Planner, version 1.0

Table of Contents

1 Preface

This document describes Planner, which was written by John Wiegley and is now maintained by Sacha Chua (see Chapter 11 [Acknowledgements], page 85). This document is available in both the stable and development versions, and should describe the features available in the particular version it is included in.

This document is a work in progress, and your contribution will be greatly appreciated. Please e-mail comments and suggestions to the maintainer, Sacha Chua sacha@free.net.ph . In the subject line of your e-mail, include the word '`Planner`'.

Documentation author: John Sullivan johnsu01@yahoo.com

Maintainer: Sacha Chua sacha@free.net.ph

John Sullivan (johnsu01)
April 25, 2004

2 Introduction

Planner is an organizer and day planner for Emacs. It helps you keep track of your pending and completed tasks, daily schedule, dates to remember, notes and inspirations. It is a powerful tool not only for managing your time and productivity, but also for keeping within easy keystroke reach all of the information you need to be productive. It can even publish reports charting your work for your personal web page, your conscience, or your soon-to-be-impressed boss.

In fact, because it uses as its building blocks simple plain-text files, it is an incredibly modular and flexible tool capable of shaping and handling your personal information in ways whose variety is limited only by your imagination. Because of this, Planner has a very active and generous community who regularly share their innovations with each other. Many of these modules and extensions are included in the archive that you will download. Once you get the basics down, you'll probably want to explore some of them. But as you read this manual and work with Planner, keep in mind that the basic core is actually very simple, and it might be worth spending time with just that before delving into the customizations.

Because they are plain text with very few requirements, the organizer pages kept by Planner can be as basic or as detailed as you like. Your pages can be simple to-do lists with no more additional information than what you would scrawl on a napkin, or they can be a highly technical affair involving hyperlinks, embedded Lisp code, appointment schedules and RSS feeds. As with so much in Emacs, it's all up to you.

To get started with Planner, you first need to download it, and possibly also the packages it depends on. You have your choice between stable and development versions (see Chapter 3 [Installation], page 3).

3 Installation

First, you need to choose which version is right for you. Choose the stable branch if you want to minimize risk and don't feel the need to see just how sharp that edge is. If you are just starting out, you should probably begin with the stable version. You don't need the extra features yet, and there's no reason to risk extra confusion with unforseen bugs or changes that aren't yet documented here.

New features are tested first in the development branch, then added to the stable branch after meeting these conditions:

- Confirmed working by at least one other user
- Documented in the source code, changelog, and info manual

Errors are corrected in the development branch first. Once fixes are confirmed, they are applied to the stable branch. Changes that might break your configuration will be clearly marked with "NOTE:" in the changelog for the branch. The changelog is included in the archive for your convenience. Major changes will also be announced on the emacs-wiki-discuss@nongnu.org mailing list see Chapter 10 [Getting Help], page 84.

3.1 Installing the stable version

Currently, there are three ways to obtain and install the stable version of Planner. You can install Planner from a source archive, Arch repository, or the Debian repository.

- *From a source archive: sacha-stable.tar.gz*

  You can install Planner from the source archive packaged and distributed directly by the author. On Windows, download WinZip or another program capable of opening tar.gz files.

  1. Download and unpack http://sacha.free.net.ph/notebook/emacs/sacha-stable.tar.gz.
  2. Edit your '`~/.emacs`' ('`_emacs`' on Microsoft Windows).

```
;; Add the directories to your load path
(add-to-list 'load-path "/path/to/emacs-wiki")
(add-to-list 'load-path "/path/to/planner")
(add-to-list 'load-path "/path/to/remember")

;; Load planner
(require 'planner)
```

- *From the revision control system*

  Arch allows you to retrieve previous versions and select specific features and bugfixes. Debian users can install arch with *`apt-get install tla`*. Users of other distributions should see http://regexps.srparish.net/www/.

  To get started with Planner using arch, you'll need to run some initial commands to register your local copy of the archive and retrieve the files.

```
# Register the emacs-wiki archive
tla register-archive mwolson@member.fsf.org--2004 http://www.mwolson.org/archives/

# Get the emacs-wiki archive
```

```
tla get mwolson@member.fsf.org--2004/emacs-wiki--main--1.0 emacs-wiki

# Register the archive
tla register-archive sacha@free.net.ph--main http://sacha.free.net.ph/notebook/arch

# Download planner module into the planner/ subdirectory
tla get sacha@free.net.ph--main/planner--stable--1.0 planner
```

Then add the following lines to your `~/.emacs`:

```
;; Add the directories to your load path
(add-to-list 'load-path "/path/to/emacs-wiki")
(add-to-list 'load-path "/path/to/planner")
(add-to-list 'load-path "/path/to/remember")

;; Load planner
(require 'planner)
```

- *From Debian*

  The stable version in the Debian archive is a few versions behind the source archive released by the maintainer. '`planner-el`' is the name of the package, and it is available via apt-get in the Woody, Sarge and Sid distributions from any of the standard Debian repositories. If you want to get stable versions from the unofficial repositories, add the following lines to your '`/etc/apt/sources.list`'

  ```
  deb http://www.mwolson.org/debian/ ./
  deb http://sacha.free.net.ph/notebook/emacs/planner/ ./
  ```

  Then, do the following steps as root:

  ```
  apt-get update
  apt-get install planner-el
  apt-get install remember      # If using the unofficial repository
  ```

Users of the revision control system Arch can browse an arch repository at http://plannerarch.mbobo.org (Updated at 16:00 GMT, Host: Florian Lanthaler).

Point your browser to http://sacha.free.net.ph/notebook/emacs/stable to download individual files from the archive.

Finally, this documentation is available separately in eye-pleasing formats including PDF and HTML at http://sacha.free.net.ph/notebook/doc/stable/planner/.

3.2 Installing the development version

Choose the development branch if you want to use new Planner features, or if you want to participate in testing and hacking new ideas with the Planner community. You can install the development version either by using the Arch revision control system, or by downloading the source archive.

Arch allows you to retrieve previous versions and select specific features and bugfixes. Debian users can install arch with *`apt-get install tla`*. Users of other distributions should see http://regexps.srparish.net/www/.

To get started with Planner using arch, you'll need to run some initial commands to register your local copy of the archive and retrieve the files.

```
# Register the emacs-wiki archive
tla register-archive mwolson@member.fsf.org--2004 \
    http://www.mwolson.org/archives/2004

# Get the emacs-wiki archive
tla get mwolson@member.fsf.org--2004/emacs-wiki--main--1.0 emacs-wiki

# Register the archive
tla register-archive sacha@free.net.ph--main \
    http://sacha.free.net.ph/notebook/arch

# Download planner module into the planner/ subdirectory
tla get sacha@free.net.ph--main/planner--dev--1.0 planner
```

If you don't use arch, you can instead download the source archive from http://sacha.free.net.ph/notebook/emacs/sacha-dev.tar.gz, or the individual files from http://sacha.free.net.ph/notebook/emacs/dev.

Once you have obtained all the files by one of these methods, you will need to open your '`~/.emacs`' ('`_emacs`' on Microsoft Windows), add the '`emacs-wiki/`' and '`planner/`' directories to your load path, and tell Emacs to load Planner when it starts up.

```
(add-to-list 'load-path "/path/to/emacs-wiki")
(add-to-list 'load-path "/path/to/planner")

(require 'planner)
```

Finally, if you are looking for an eye-pleasing document to learn from, this manual is available separately in formats including PDF and HTML at http://sacha.free.net.ph/notebook/doc/dev/planner/.

3.3 Updating Your Version

If you installed Planner from the source archive, download a new version and extract it over the old directory. Don't forget to delete any byte-compiled files ('`*.elc`') in the directories so that the new code will be used.

If you installed Planner from Debian, do *`apt-get update; apt-get upgrade`* to upgrade all packages that can be upgraded, or *`apt-get update; apt-get install planner-el`* to upgrade just planner-el.

To stay up-to-date using arch, here are some commands that might be useful.

To list upstream changes not in local copy:

```
# Change to the source directory you are interested in. Example:
cd emacs-wiki/

# Display the summary of changes
tla missing --summary
```

To update to the latest version:

```
cd emacs-wiki; tla replay
cd ../planner; tla replay
```

```
cd ../remember; tla replay
```

You can browse the arch repository at http://plannerarch.mbobo.org (Updated at 16:00 GMT, Host: Florian Lanthaler).

3.4 Components

Now that you have the archive, let's look at what's in it.

There should be three directories, named '`planner`', '`emacs-wiki`' and '`remember`'.

In the '`planner/`' directory, you'll see many files with names like '`planner-gnus.el`'. These are extra modules and extensions for Planner, which you can use to tailor Planner to fit your desired planning and information management habits.

In the '`emacs-wiki/`' directory, you'll see many files with names like '`emacs-wiki-blosxom.el`'. As in '`planner/`', these are optional modules and extensions.

A minimal working installation includes just '`planner/planner.el`' and '`emacs-wiki/emacs-wiki.el`'.

You need '`planner.el`' because it provides the core functions for handling tasks, notes, and page navigation. You need '`emacs-wiki.el`' because it provides the functions used to display your pages (both in an emacs buffer and as HTML), and for connecting them to each other. More specifically, it enables you to have hyperlinks and formatting in your emacs buffers even though the actual files you are working with are saved in plain text. These abilities are used in Planner to format your planner pages the way you like, to create links from your tasks and notes to the materials and projects they refer to, and to optionally "publish" your pages in different formats, including HTML.

In the '`remember/`' directory are files related to RememberMode. RememberMode does not depend on Planner or EmacsWikiMode, but works best with Planner installed. It is not required in order to use Planner, but it is used by many Planner users to record notes and information to their planner pages.

If you are curious, you can open each file in these directories and read the comments at the top, to get an idea of what each extension is used for. They are also all detailed later in this manual.

3.5 Advanced Installation

Once you decide you want to keep Planner around for a while, there are two additional steps you can take to make using it easier and more efficient. These steps are optional.

- You can install '`remember.el`' and '`remember-planner.el`' so that you can create Planner notes from anywhere. See the documentation at http://sacha.free.net.ph/notebook/doc/dev/remember/ for more information.
- You can make this document, the Planner info file, appear in the index of info files you see when you type `M-x info` or `C-h i` in Emacs. The instructions for doing this vary depending on whether you have permission to edit certain files on your system. Follow the instructions in section "Installing an Info File" in *Texinfo*, using something like:

```
* Planner: (path/to/planner/Planner). Organizer/day planner
```

```
for Emacs.
```

for the new entry in the info '`dir`' file.

- You can byte-compile '`planner.el`', '`emacs-wiki.el`', '`remember.el`', or any of the optional modules you frequently use, in order to improve the speed of their execution. Basically, all you need to do is visit each file in an Emacs buffer and do `M-x byte-compile-file`. To read more detail about byte compilation, see section "Byte Compilation" in *Emacs Lisp Reference Manual*.

4 Overview

Planner is a plain-text hyperlinked personal information manager for Emacs that helps you keep track of tasks, notes, and other information. People use Planner to support different ways of planning one's day, from Franklin-Covey and David Allen's Getting Things Done to home-brew hacks. Planner is even used to manage information not normally handled by a personal information manager, like bugtracking, time tracking, and team data. If you start by using Planner as a basic TODO and notes manager, you might find other ways it can help you improve your process.

You can use Planner to keep track of your tasks, schedule, notes, and other information you want to store in hyperlinkable text files. You can get the most benefit out of a personal information manager if you use it everyday. Most people add `(plan)` to the end of their '`~/.emacs`' so that Planner shows today's schedule and unfinished tasks whenever Emacs starts. If you leave your Emacs running for more than 24 hours, try to get into the habit of running `plan` at least once a day.

Because your time is important, Planner tries to minimize distractions, making it easier for you to jot down tasks and notes without being distracted from your work. People often make tasks based on the current buffer, so Planner tries to create hyperlinks to whatever you're looking at so that you can jump back to it easily. The Chapter 5 [Getting Started], page 14 tutorial will show you how to set that up for both tasks and notes.

The customizability of Planner means you can make your personal information manager truly personal. Planner strives to be as flexible as possible, and we would love to adapt Planner to fit your needs. Browse through our mailing list at `http://lists.nongnu.org/mailman/listinfo/emacs-wiki-discuss` to find out how other people are using Planner, and post your feature requests and bug reports there!

Planner is just a tool. It does not dictate a particular way of planning, although it supports some ways better than it supports others. If you want to take some time thinking about planning, read the following reflections for inspiration and ideas. On the other hand, if you want to hit the ground running, see Chapter 5 [Getting Started], page 14. If you already have a specific way of planning in mind, check out Section 9.4 [Sample Configuration Files], page 69.

4.1 Planning Based on the Franklin-Covey Approach

This is a slightly edited and updated version of an essay by John Wiegley. Read it if you want to get some insights into one way of planning. You can skip this if you want to go straight to planning your day.

What is planning? It can be a nebulous thing to define. In its essence, however, it is very simple: it's how we achieve our dreams.

Our days are filled with time, and hence with actions, whether they be of a mental or physical sort. But there are two kinds of action: reactive and creative. Reactive action is a response to the environment, a reaction to stimulus. Had we enough instincts to ensure survival, we could live according to this kind of action alone. It is a mode of behavior we share with every living species.

The opposite to reactivity is creativity, when we decide upon a course of action that is a wholly a product of personal choice. We then make decisions as to the steps needed to

make this wish a reality. This is planning. Planning is essentially a creative endeavor at every step.

First, create the idea, what you want to achieve. Very short-term ideas do not need much more than thinking about how to do it. But long-term ideas require planning, since the mind cannot contain all of the details.

Second, decide how the idea maps into the circumstances you find yourself in. Some environments will assist your plan, others hinder it. But step by step, identify every barrier to the realization of your idea, and devise a countermeasure to overcome it. Once you've mapped things out from beginning to end, accounting for unknowables as best you can, you now have your plan.

Third is to break the stages of the plan into parts that are not overwhelming in their complexity. It is at during this phase that a plan is turned into task items, each to be accomplished within the span of one day's time. If a task requires several days, break it up further. The smaller it is, the less your mind will recoil from attempting it.

Fourth is to monitor your progress, identifying problems and correcting for them as you go. Some plans start out unachievable, and remain that way indefinitely, due to a simple lack of observation. If nothing is working for you, change it. Otherwise, your plan is merely a well-crafted wish.

Fifth is just to do the work, and be patient. All good plans take a great deal of time, and *cannot* happen immediately. The groundwork must be laid for each step, or else it will rest on an unsecure foundation. If you follow your plan doggedly, applying some time to it each day or week, it *will* happen. Remember the story of the tortoise and the hare. I've even written a short essay on the necessity of gradual accomplishment, which can be found at http://emacswiki.org/johnw/essays/node2.html.

How can this software help? Computers are ideal for manipulating information, since they allow you to change things without erasing or rewriting. And since all plans change quite a bit during their implementation, a planning program can be very helpful.

Start by adding the following to your '`.emacs`' (or '`_emacs`'):

```
(load "planner")
```

Now, conceive your idea. I can't believe there's nothing you want from life. More peace, time to enjoy the world, an end to war? Everyone wants something. Search deeply, and you will find countless unhoped wishes lurking therein. Choose one for now, and think on it for a while.

Then open a file (using `C-x C-f`) within the directory named by `planner-directory`. Emacs will automatically recognize this file as a planner file. Name the file after your plan, such as '`BetterHealth`'.

Choose an idea you really want to accomplish. Struggle to differentiate between the things you want because others want them, and the things you want for yourself. It takes quite an effort, and may require a long time before you notice the difference. Many people want to be more healthy to be more attractive, which is an externally driven goal. Unless *you* really want to accomplish what you envision, the odds are you will fail. Only our own wishes and dreams possess enough personal energy to see themselves to fruition. What happens to many of us is simply that we never become conscious of these dreams: what we love, what we desire most. When I talk to friends, so much of what I hear is things they

want because they feel they should want them. There's just not enough energy there to pursue a good plan, because nearly all of it is negative energy.

Do you know what you really want? Don't worry, many people don't. It's not a question anyone really wants us to pursue, because often we don't want what others do; it doesn't contribute to the social welfare, and all that nonsense. Somehow we always forget that what's good for the social welfare now, was someone else's crazy dream a hundred years ago. The human aversion to fundamental change is always one's greatest enemy, so don't waste any time getting bitter about it.

For the sake of argument I assume you really do want to be healthier, because you've fallen in love with the ideal of purity, or you understand the connection between your physical self and the world around you, and how this can open up your spirit to desiring more. I assume. :)

So you're in a Wiki file called '`BetterHealth`'. Start typing. Type anything related to your idea: what you think about it, your ideas on it, *and especially what the end will look like*. If you can't visualize the end, you can't plan, since planning is about drawing a line between now and then.

When you've typed enough to gain a vision of your goal, start drafting what the possible intermediate steps might be. Then stop, get up, walk around, enjoy life, and come back to it. Taking a long time at the beginning is not a bad idea at all, as long as it's not forever.

As you chew on your idea, it will begin to become more and more concrete. You'll have ideas about the smallest pieces, and ideas about the biggest pieces. Keep going until it starts to take shape before you, and you can see yourself in your mind's eye moving from the present into the future. Write down this progression, and the sorts of things you might encounter along the way.

As you continue, you'll naturally discover discrete phases, or "milestones" as managers love to call them. These are very important, because they let you know you're making progress. I recommend having a big party with friends every time you achieve a milestone. A typical plan might have between three and ten.

Between the milestones are the bigger pieces of your plan. Name these pieces using MixedCase words, and you'll notice that Emacs colors and underlines them for you. Like, FindGoodGym. Hit return on this highlighted word, and you'll find yourself in another, blank file. In this file, start drafting your sub-plan, just as you did with the larger plan. You should find it easier now, since the scope is smaller.

As you break down further, you'll notice simple little things that need to get done. These are your tasks. Every plan is a succession of tasks. The difference from reactivity is that each task is part of the larger plan. This is what it means to be systematic: that everything you do helps further your plan. If you have tasks in your day that contribute to no plan, they are reactive. Of course, life is full of these, but don't let them take up more than 20% of your day. If you allow yourself to be dominated by reactive tasks, you'll regret it at the end of your life. I don't know this personally, but I do know that striving for one's dreams – and seeing them come to fruition – is the greatest joy a man can possess. It is the essence of freedom, of living, of creation. Reactivity is the opposite of this, and serves only to drain our energy and slacken our spirits.

Now that you've thought of a simple task, type `C-c C-t`. This will ask for a brief description of the task, and when you plan to do it. If you hit ⟨RETURN⟩ at the question

'`When`', it assumes you mean today. It will also pop up a three-month calendar at this question, so you can see where your free days are. Make sure you set the variable `mark-diary-entries-in-calendar` to '`t`' in your '`.emacs`' (or '`_emacs`') file. This way, you can see which days your appointments fall on. (Read about the Emacs Calendar and Diary in section "Calendar/Diary" in *GNU Emacs Manual*.)

```
(setq mark-diary-entries-in-calendar t)
```

Once your task is in there, go back to your plan and keep generating more tasks. Generate them all! Fully describe—as tasks—everything necessary to bring your sub-plan to completion. Don't create tasks for the other sub-plans. You may have good idea of what they'll look like, but don't bother rendering them into tasks just yet. Things will change too much between now and then, for that to be a good use of your time.

Is your sub-plan now rendered into all of the tasks necessary to reach your first milestone? Great! That is the purpose of planner.el. The rest is really up to you. If you find that you keep putting things off, and never do them, that's the surest sign you're planning for someone else's dream, and not your own.

Here are some of the things planner.el can do, to help you manage and track your tasks:

At the beginning of every day, type *M-x plan*. This will jump you to the top of the most recent task list before today. If you skipped a bunch of days, you'll have to open up those files on your own.

Probably some of the tasks that day won't be finished – that's OK. Learning to properly estimate time is a magical, mystical art that few have mastered. Put your cursor on those undone tasks, and type *C-c C-c*. This will move them into today's task page. You can jump to today's task page at any time by typing *C-c C-n* (from a Wiki or planning page). I heartily recommend binding *C-c n*, to jump you to this page from anywhere:

```
(define-key mode-specific-map [?n] 'planner-goto-today)
```

As you look at your task sheet each day, the first thing to do is to "clock in" to one of them. This isn't necessary, and is only helpful if you're around your computer a lot. But by typing *C-c C-i* (assuming you have my '`timeclock.el`' on your load-path), it will log the time you spend working on your sub-plan (see section "Time Intervals" in *GNU Emacs Manual*). This is helpful for viewing your progress. Type *C-c C-o* to clock out.

C-M-p and *C-M-n* will move a task up and down in priority. Priority is represented by a letter A through C. 'A' tasks mean they must be done that day, or else your plan is compromised and you will have to replan. 'B' means they should be done that day, to further the plan, otherwise things will be delayed. 'C' means you can put off the task if you need to, although ultimately it will have to be done.

For reactive tasks, the letters mean something different: 'A' means you must do it today, or somebody will roast your chestnuts over an open fire. 'B' means you should do it today, or else someone will be practicing patience at the day's end. 'C' means no one will notice if you don't do it.

Again, reactive tasks are ENEMIES OF PLANNING. Really, until you see them that way, circumstances will push you around and steal your life away. We have only so many years to use, and everyone is greedy to take them. It's insidious, almost invisible. A healthy dislike of reactivity will do wonders for organizing your affairs according to their true priority.

The last word that needs to be said concerns "roles". Every person stands in several positions in his life: husband, employee, manager, etc. These roles will tend to generate tasks not associated with any immediate plan, but necessary to maintain the health and functioning of the role. My suggestion is to keep this the smallest possible number, and fulfill those that remain well. How you decide to apportion your time between pursuing grand designs, and fostering deep relationships, is a personal matter. If you choose well, each will feed the other.

I mention this to point that reactivity is something not exclusively associated with tasks that have no master plan, because being a father, for example, is something that rarely proceeds according to orderly plans. But the role of father itself is its own plan, whose goal is "to be the best one can", and whose component tasks are spending time on whatever comes up. It is, in a sense, an implicit plan. But reactive tasks follow no plan at all; they are parasites of time that suck the spirit away, whereas properly chose roles actually help fulfill one's own inner needs. At least, this is what I believe.

`plan` *force-days* [Function]

Start your planning for the day, beginning with the last day's tasks.

If `planner-carry-tasks-forward` is non-nil, find the most recent daily page with unfinished tasks and reschedule those tasks to the current day. If *force* is non-nil, examine all past daily pages for unfinished tasks.

If `planner-carry-tasks-forward` is nil, visit the most recent daily page. If a daily page for today exists, visit that instead.

If *force-days* is a positive integer, scan that number of days. If *force-days* is '`t`', scan all days.

4.2 Why Use Planner?

You can skip this essay if you just want to get started, or read it for some insights into why the current maintainer is crazy about it.

Why I Use Planner, by Sacha Chua

I thought about why I liked Planner. Planner as a TODO manager isn't particularly special. Although I can assign tasks to categories and see a breakdown of what projects are taking up my time, Evolution and Microsoft Outlook provide more powerful task support. In other task managers, you can e-mail tasks, assign multiple categories and fill in all sorts of metadata. You can even synchronize your tasks with devices like a phone or PDA. So why use Planner?

I realized that integration into my way of life and automatic context clues are what really make planner tasks worth it for me. I don't have to switch to another application to create a task. I can just hit a keyboard shortcut. Planner uses a minibuffer to get the task description. My windows are not rearranged in any way, and I can look at the data that's relevant to a task. Not only that, tasks automatically pick up context clues, like whom I'm talking to on IRC or the file I'm editing at the moment. This cuts down on the explicit context I need to include and makes it easier for me to bring up the task again.

As a scheduler, Planner is also not particularly distinguished. Sure, it can display my '`~/diary`', but for that matter so can *`M-x diary`*. Evolution and Outlook can give me a more graphical view of my time, sync with my PDA, and coordinate my schedule with other

people. Those applications support detailed schedule entries with powerful cyclic options. On the other hand, Planner gives me a personal, plain text view and (at least the way I use it) requires me to edit a separate file to add new appointments. (I've defined a few shortcut keys to deal with this.) However, it does have one advantage—my schedule is always loaded. I used to use Outlook on Windows, but having my schedule in a separate application meant that I actually looked at it very rarely, as I had turned off reminders because they got annoying.

Planner's notes, however, are what really convinced me. I can hit a keyboard shortcut from anywhere and type my notes into a buffer which automatically keeps context information. After typing the note, I can then categorize it. I think that the critical thing here is that interruptions—fleeting thoughts—don't break my flow. I can just pop up a remember buffer, stow that thought away somewhere, and go back to it whenever I want. In contrast, creating a note in Outlook means switching out of my application, making a couple of keystrokes, typing the note in, and then switching back. The context switches make it hard to keep track of where I am and what I'm supposed to remember. Not only that, I need to enter context by hand. Even though I can color my notes and reorganize them in Outlook, I find the context switch too expensive. I used to keep notes in other knowledge management tools as well. Some applications allowed me to drag-and-drop links into the current note, and that was cool. But that required a manual action, and those applications didn't feel integrated into my way of working. (Note: You'll need remember.el for this.)

I guess that's why I like Planner. Unlike other organizers which don't know anything about the applications I use, Planner tries its best to integrate into the way I work, and it's easy to extend. Fortunately I do almost all my work in Emacs, so I can think of my organizer as integrated into my e-mail client, Internet Relay Chat client, web browser, file editor and even games. It automatically picks up context clues from these applications and allows me to easily jump back to relevant files. It doesn't distract me. It allows me to key in data and then it gets out of my way.

(That said, it's perfectly okay to use Planner even if you don't live in Emacs.)

The processing that happens in the background is a bonus, and publishing my task list and notes online has greatly helped me. It gives other people a way to see what I'm working on and what I've planned for the future. Occasionally people write in with additional resources and helpful tips. (Again, this is purely optional. Many people don't publish their planner pages. Other people use really fine-grained access control.)

I think the greatest feature of Planner, though, is its user community. Because Planner can be easily modified, we can experiment with a lot of new ideas quickly, and we can tailor Planner to fit our needs. I love checking my '`emacs-wiki-discuss`' mail and finding out how people have tweaked Planner or would like to tweak Planner, and I've learned a lot by exchanging reflections on organizing one's life.

I really wasn't an organization freak before I started using Planner. I often forgot to do my homework or answer important mail. I still procrastinate now, but at least it's all being kept track of somewhere! I also really like how Planner lets me to gradually improve how I'm doing things, and I feel I've come a long way.

Please try it out! We'd love to hear how Planner can become *your* personal information manager.

5 Getting Started

At the end of this tutorial, you will be able to use Planner and related modules to keep track of your tasks, schedules and notes, all within the convenience of Emacs.

There are two kinds of pages in a Planner wiki. Day pages show tasks, schedule, and notes for the day, while plan pages organize related tasks and notes into a single page.

If you have not yet added planner to your '`~/.emacs`', add the following lines:

```
(add-to-list 'load-path "/path/to/emacs-wiki")
(add-to-list 'load-path "/path/to/planner")
(add-to-list 'load-path "/path/to/remember")
(require 'planner)
```

This will bring up the most recent day page with unfinished tasks or create a new day page if necessary. By default, planner pages are stored in '`~/Plans`' (`planner-directory`).

5.1 Tasks

Let us start by creating a task labelled

```
Join http://lists.nongnu.org/mailman/listinfo/emacs-wiki-discuss
```

From anywhere (even this info buffer!), call *M-x planner-create-task-from-buffer* to create a new task. Fill in the description and choose a date by:

- typing 1 - 31 to put the task on that day of the month,
- accepting the default (today) by pressing RET,
- selecting the date with mouse-1,
- typing +n (where in is an integer) to schedule the task in n days time, or
- typing nil to make an undated task.

For now, accept the default ('`today`') by pressing ⟨RET⟩.

You will then be prompted for a plan page. Plan pages gather related tasks and notes, giving you an overview of what you've done so far. You can accept the default TaskPool, create your own plan page, or specify nil to make a task that is not associated with a plan page. For now, accept the default ('`TaskPool`') by pressing RET.

You have created your first task. View today's page with *M-x planner-goto-today*. You will see a line of the form

```
#B  _ Join http://lists.nongnu.org/mailman/listinfo/emacs-wiki-discuss (TaskPool)
```

If you created the task from this page, then there will be an additional annotation:

```
#B  _ Join http://lists.nongnu.org/mailman/listinfo/emacs-wiki-discuss : Tasks (TaskPo
```

The URL, '`TaskPool`' and '`Getting Started`' are hyperlinks. You can use TAB and S-TAB to navigate between them and RET to follow the link.

Create more tasks using *M-x planner-create-task-from-buffer*. This is bound to *C-c C-t* in `planner-mode` pages for your convenience. For example, create the following tasks:

- '`Describe my current way of working and how I would like to work`', for three days from now ('`+3`'),

- '`Learn how to schedule a task`', an undated task (*`nil`*) on the TaskPool page,
- '`Browse through the Planner info manual`' for today (`.` or accept the defaults), and
- '`Add (plan) to the end of my [[~/.emacs]]`' for today, but without a plan page (specify *`nil`* at the plan page prompt)

Tip: I bind planner-create-task-from-buffer to "F9 t" so that I can easily call it from anywhere. You can do that with this elisp fragment: `(global-set-key (kbd "<f9> t") 'planner-create-task-from-buffer)`

Next, visit the TaskPool by:

- ⟨TAB⟩-bing or using the cursor and typing ⟨RET⟩ to follow the link,
- *`C-x C-f TaskPool RET`* to use `find-file`, or
- *`C-c C-f TaskPool RET`* to use `emacs-wiki-find-file`

You can see an overview of the tasks as scheduled on different days. Unlike many personal information managers that store all of your data in one file and then perform magic in order to present different views, Planner uses plain text files. The data is duplicated and kept updated by functions. This makes it simpler and easier to modify, because what you see is (almost) what you get. On the other hand, you'll need to get used to either editing both files, or using the built-in functions for editing and updating files. If you prefer not to work with linked tasks, you can configure Planner to use only plan pages or use only day pages.

The TaskPool page should list the tasks you created earlier. Go to the one named Learn how to schedule a task . Type *`C-c C-c`* (`planner-copy-or-move-task`) to schedule the task. Type RET to accept the default (today). Go to the day page by following the link or calling *`M-x planner-goto`* (*`C-c C-j C-d`* or the menu bar); you will see the newly-created task there. You can also use *`C-c C-c`* (*`planner-copy-or-move-task`*) to reschedule a task to an earlier or later date.

Well, that task is done. To mark the task as completed, type *`C-c C-x`* (`planner-task-done`). You can also edit the status manually (change _ to X) as long as you remember to call *`M-x planner-update-task`* to update the link page as well. Updating relies on the task description being the same, so do not edit this manually.

Quick summary of commands:

- Go to today's page: *`M-x plan`* to carry unfinished tasks forward, or *`M-x planner-goto-today`* to just go to today's page.
- Create a task: *`C-c C-t`* (`planner-create-task-from-buffer`), or type a task manually (call M-x planner-update-task if the task is linked)
- Mark a task as done: *`C-c C-x`* (`planner-task-done`), or edit the task and call *`M-x planner-update-task`*
- Edit a task description: *`M-x planner-edit-task-description`*
- Reschedule a task: *`C-c C-c`* (`planner-copy-or-move-task`)
- Reschedule many tasks: Mark a region and use *`M-x planner-copy-or-move-region`*
- Change the plan of a task: *`M-x planner-replan-task`*, or *`C-c C-c`* (`planner-copy-or-move-task`) with a plan page
- Delete a task: *`M-x planner-delete-task`*

- Reorder tasks: ⟨M-p⟩ (`planner-raise-task`) and ⟨M-n⟩ (`planner-lower-task`), or normal editing commands like kill and yank
- Change task priorities ('`#A`' > '`#B`' > '`#C`'): ⟨C-M-p⟩ (`planner-raise-task-priority`) and ⟨C-M-n⟩ (`planner-lower-task-priority`), or edit the task and call *M-x planner-update-task*.

You can save your tasks with *C-x C-s* the same way you save any other file, or Emacs will prompt you to save it when you exit.

5.2 Schedule

This is free-form. You can put anything you want into this, or remove it from `planner-day-page-template` entirely. Some people use it to keep track of their plans for the day with tables like this:

```
hh:mm | hh:mm | activity
hh:mm | hh:mm | activity
hh:mm | hh:mm | activity
```

Remember, Planner files are just plain text. You can add new sections or remove old ones, or use the suggested sections for entirely different activities.

5.3 Notes

You can put anything you want in this section, or remove it from your `planner-day-page-template` entirely.

You may be interested in '`remember-planner.el`', part of the Remember package. `remember-planner.el` makes it easy to create notes from anywhere in Emacs, and it uses the same hyperlink-sensing code that Planner uses. Notes added by remember-planner.el look like this:

```
.#1 Headline 00:00
Body
```

and are outlined at the H2 level in published HTML.

If you include the following in your '`~/.emacs`':

```
(require 'remember-planner)
(setq remember-handler-functions '(remember-planner-append))
(setq remember-annotation-functions planner-annotation-functions)
```

you can easily create context-aware notes. *M-x remember* zwill create a dedicated buffer for you to fill in your note. If Planner recognizes the buffer as one with context then it will include a hyperlink at the bottom. The first line is used as a title, so make it short and meaningful. The rest of the text will be used as the body of the note. Try it now by creating a note, perhaps about things you'd like to remember from this tutorial.

Typing *C-c C-c* after composing will prompt for a plan page to put this on, with autocompletion. By default, notes will go on today's page only. If you specify a plan page, the note will go on today's page and on the plan page. Let's try specifying '`TaskPool`' for the note.

If you look at today's page, you'll find a timestamped note that links to '`TaskPool`'. Likewise, '`TaskPool`' contains a note that links to today's page. Because the text is copied,

changes in one will not be automatically reflected in the other. To update the linked page after editing a note, use *M-x planner-update-note*. To change the plan page of a note, use *planner-replan-note*.

5.4 Hyperlinks

Planner automatically creates context-sensitive hyperlinks for your tasks and notes when you use `planner-create-task-from-buffer` and `remember`.

Blue links indicate URLs and Planner pages that already exist. Red links indicate Planner pages that have not yet been created.

Middle-click or type ⟨RET⟩ on any link to view the link in the current window. Shift-middle-click or type ⟨S-RET⟩ to view the link in another window. ⟨TAB⟩ goes to the next link, while ⟨S-TAB⟩ goes to the previous one.

You can pick up hyperlinks using the `planner-annotation-as-kill` function.

`planner-annotation-as-kill` [Function]
: Create a context-sensitive hyperlink for the current buffer and copy it to the kill ring. When called with a prefix argument, prompt for the link display name.

You can then paste it into any Planner buffer by using *M-x yank* or the keyboard shortcut.

Hyperlinks are a powerful feature of Planner. You can use them to hyperlink to mail, news, Web pages, and even IRC connections. See the section on Chapter 7 [Managing Your Information], page 47 to find out how to enable support for various parts of Emacs. Want to add a new hyperlink scheme? Check out the source code for examples or ask on the mailing list for help.

5.5 Example Page

An example planner file is given below. You'll notice that Planner does not have a well-defined user interface. Rather, it's free-form and open, allowing you to adapt it to your preferences.

```
-----------------------------------------------------------------------
* Tasks

#B  _ Join http://lists.nongnu.org/mailman/listinfo/emacs-wiki-discuss (TaskPool)
#B  _ Browse through the Planner info manual (TaskPool)
#B  _ Add (plan) to the end of my ~/.emacs
#B  X Learn how to schedule a task (TaskPool)

* Schedule

18:00 | 19:00 | Learn how to use Planner

* Notes

Notes are free-form. You can put anything you want into this.
```

```
.#1 This is note number one

Notes on note number one!

.#2 This weird ".#2" syntax is used for allout.el enumerated lists

It makes using allout-mode very handy.
```

5.6 Review

- *Ideas for using planner more effectively:*
  - Add `(plan)` to the end of your '`~/.emacs`' so that you are reminded about your tasks every day.
  - Bind useful functions to shortcut keys and get used to creating tasks and notes from anywhere.
  - Think about how you plan your day and look for ways to improve it. Ask the mailing list for tips.
  - Browse the rest of this manual, the source code, and other resources on the Net for tidbits you can use.
  - Have fun!
- *Useful functions outside planner buffers:*
  - `planner-create-task-from-buffer`
  - `remember`
  - `planner-goto-today`
  - `planner-goto`
  - `plan`
- *Useful functions inside planner buffers:*
  - *C-c C-t* (`planner-create-task-from-buffer`)
  - *C-c C-x* (`planner-task-done`)
  - *M-x planner-edit-task-description*
  - *C-c C-c* (`planner-copy-or-move-task`), *M-x planner-copy-or-move-region*
  - *M-x planner-replan-task*
  - *M-x planner-delete-task*
  - ⟨M-p⟩ (`planner-raise-task`) and ⟨M-n⟩ (`planner-lower-task`)
  - ⟨C-M-p⟩ (`planner-raise-task-priority`) and ⟨C-M-n⟩ (`planner-lower-task-priority`),
  - `planner-replan-note`
  - `planner-update-note`

That's all you need to know in order to use Planner as a basic TODO and notes manager, but there's a whole lot more. Read through this manual and our mailing list archives for lots of wonderful ideas about planning in Emacs!

6 More about Planner

6.1 Basic Configuration

You may want to customize the following variables before you begin using Planner.

`planner-directory` [User Option]
: Plain-text planner files will be stored in this directory. Default: '`~/Plans`'.

`planner-publishing-directory` [User Option]
: Planner HTML files will be published to this directory. If you want to view your tasks, schedule and notes using a web browser, upload the contents of this directory to your website. Default: '`~/WebWiki`'.

If you change these variables after Planner is loaded, please be sure to run `M-: (planner-update-wiki-project)`. It might be a good idea to add `(planner-update-wiki-project)` to the end of your '`~/.emacs`'.

6.2 Starting with Day Pages

`planner-goto-today` opens today's page. Day pages are named '`YYYY.MM.DD`' and contain your notes for the day.

You should see a file that looks like this:

```
* Tasks


* Schedule


* Notes
```

You can type anything you want into this file. You can add or delete sections. When you save, Emacs stores your information in `planner-directory`.

Use the following commands to navigate through day pages:

`plan` [Function]
: Start planning the day. If `planner-carry-tasks-forward` is non-nil, copy the most recent unfinished tasks to today's page, else open the most recent page.

`planner-goto` (*C-c C-j C-d*) [Function]
: Prompt for a date using a calendar pop-up and display the corresponding day page. You can specify dates partially. The current year and month are used if omitted from the input. For example, if today is 2004.05.05, then
  - *+1* is one day from now, or '`2004.05.06`'
  - *-1* is one day before now, or '`2004.05.04`'
  - *12* is equivalent to '`2004.05.12`'
  - *8.12* is equivalent to '`2004.08.12`'
  - *2005.08.12* is a full date specification

  In the calendar buffer, you can also left-click or press ⟨RET⟩ on a date to select it.

`planner-goto-today` (*C-c C-j C-j*) [Function]
Display today's page. Create the page if it does not yet exist.

`planner-goto-tomorrow` (*C-c C-j C-t*) [Function]
Goto the planner page days *after* the currently displayed date. If *days* is nil, go to the day immediately after the currently displayed date. If the current buffer is not a daily planner page, calculate date based on today.

`planner-goto-yesterday` (*C-c C-j C-y*) [Function]
Goto the planner page *days* before the currently displayed date. If *days* is nil, go to the day immediately before the currently displayed date. If the current buffer is not a daily planner page, calculate date based on today.

`planner-goto-most-recent` [Function]
Go to the most recent day with planning info.

`planner-goto-previous-daily-page` [Function]
Goto the last plan page before the current date. The current date is taken from the day page in the current buffer, or today if the current buffer is not a planner page. Do not create pages if they do not yet exist.

`planner-goto-next-daily-page` [Function]
Goto the first plan page after the current date. The current date is taken from the day page in the current buffer, or today if the current buffer is not a planner page. Do not create pages if they do not yet exist.

`planner-goto-plan-page` *page* [Function]
Opens *page* in the the `planner-project` Wiki. Use `planner-goto` if you want fancy calendar completion.

`planner-show` *date* [Function]
Show the plan page for *date* in another window, but don't select it. If no page for *date* exists, return nil.

6.3 More about Tasks

This section is divided into three parts. In the first part, you can read about all the options, strategies and commands to help you efficiently add new tasks to your planner. In the second part, we'll go over all of the aspects of Planner that relate to organizing, editing, rescheduling and viewing the tasks you've already created. Finally, we'll cover some ways to step back and look at various reports and overviews that can be generated from your planner pages.

You may also be interested in tracking time spent on tasks with Section 7.2.3 [Timeclock], page 55 and estimating project completion time with Section 5.2 [Schedule], page 16 (also see see Section 7.2.4 [schedule.el], page 56).

6.3.1 Creating New Tasks

Planner makes it very easy to quickly add something to your list of tasks. Once you get used to the basics of `planner-create-task-from-buffer`, you might want to take a closer look at some things in Planner that can help you create new tasks in a way that fits with your system.

6.3.1.1 Creating a Task

You can create a task from any buffer in Emacs by invoking `M-x planner-create-task-from-buffer`.

This command does more than just add an item to your list of tasks. It also connects that item to some useful context information.

If you create a task while viewing any buffer other than a Planner day page, Planner will associate the task with a hyperlink to that buffer. Try it now by creating a task from this Info buffer.

1. *`M-x planner-create-task-from-buffer`*
2. When prompted for the task name, enter *`Learn how to change a task's status`* and press ⟨RET⟩.
3. When prompted for the date, press ⟨RET⟩ to schedule the task for today.
4. When prompted for the project page, press ⟨RET⟩ to accept the default page of '`TaskPool`'. This is a page for tasks not connected to a larger plan.

Planner prompts you for two pieces of information when you ask it to create a task. First, it asks you when you would like to have the task show up in your planner. If you would like it to be scheduled for today, you can just hit ⟨RET⟩. If you would like it to be scheduled for some day during the current month, you can just enter the date, without the month, like '`16`'. If you would like it to be scheduled for some day in a future month of the current year, you can enter just the month and date, like '`06.16`'. If you would like to schedule something for next year, then enter the full date, like '`06.16.2005`'. If you do not want this task to appear on a day page at all, you can enter '`nil`'.

The second piece of information Planner asks for is the name of the project to associate the task with. In the above example, you associated the task with the project "TaskPool", which means that you did not want to associate the task with a particular project or goal in your life. Another way to do this is to answer the project prompt by entering '`nil`'. But instead, you might enter '`LearnPlanner`' as the project. This creates a new page called "LearnPlanner" in your planner directory and places an entry for the task on that page.

The task then exists in two places: once on your day page, to show how it fits into your daily work; and once on a project page, to show how it fits into your larger projects and goals. In the future you might add related tasks like, "Memorize Planner keybindings". These tasks might be scattered over weeks or months worth of day pages, but as long as you enter the same project name for each, you will have a way to look at them all together on a single project page.

Planner also creates hyperlinks to enable you to easily move back and forth between the day page system and the project page system. Each task on a day page will have a hyperlink to its project page. Each task on a project page will have a hyperlink to its day page.

After using Planner for a while, you may find yourself with quite a few project pages. Keep in mind that completion is enabled at the project prompt when you create a task, so hitting *`SPC`* or *`TAB`* at the prompt will show you a list of your current project pages.

Once the task is created, you are returned to the buffer you were working in again, Planner gets out of your way, and you can go on about your business. Later on, when you decide to actually work on that "Memorize Planner keybindings" task, you will be able to

follow the hyperlink from that task on your day or project page directly to the relevant node in the Planner info file!

By default, `M-x planner-create-task-from-buffer` creates medium-priority tasks, marked with the letter '`B`'. But you can specify a particular priority or change the default (see Section 6.3.1.2 [Task Priorities], page 22).

You don't have to use `planner-create-task-from-buffer` to create tasks. You can also create new tasks manually by typing them directly on your day or project page in the format Planner expects. You can even still create hyperlinks by using EmacsWiki formatting as you manually type the new task. Keep in mind also that tasks do not have to be linked to any other page.

For convenience, `planner-create-task-from-buffer` is bound to `C-c C-t` in Planner buffers. You can bind `planner-create-task-buffer` to a shortcut key. See the manual for your Emacs distribution to find out more about keybinding.

`planner-create-task-from-buffer` *title date plan-page* [Function]

Create a new task named *title* on *date* based on the current buffer.

With a prefix, associate the task with the current planner page. If you create a task on a date page, you will be prompted for a plan page. If you create a task on a plan page, you will be prompted for a day page. If nil is specified, the task is created only on the current page.

See `planner-create-task` for more information.

The new task is created at the top or bottom of the first block of tasks on the scheduled day page (if any), depending on the value of `planner-add-task-at-end-flag`.

`planner-create-task` *title date annotation plan-page* [Function]

Create a new task named *title* based on the current Wiki page. If *date* is non-nil, makes a daily entry on *date*, else makes an entry in today's planner page. It's assumed that the current Wiki page is the page you're using to plan an activity. Any time accrued to this task will be applied to that page's name in the timelog file, assuming you use timeclock (see section "Time Intervals" in *GNU Emacs Manual*). If *annotation* is non-nil, it will be used for the page annotation. If *plan-page* is non-nil, the task is associated with the given page.

With a prefix, associate the task with the current planner page. If you create a task on a date page, you will be prompted for a plan page. If you create a task on a plan page, you will be prompted for a day page. If nil is specified, the task is created only on the current page.

You probably want to call `planner-create-task-from-buffer` instead.

The new task is created at the top or bottom of the first block of tasks on the scheduled day page (if any), depending on the value of `planner-add-task-at-end-flag`.

6.3.1.2 Task Priorities

You can set the priority of a task when you create it, rather than waiting to adjust it after the fact. In order to do this, call the function corresponding to the priority you want. You probably want to bind these functions to some keys if you intend to use them much.

- `planner-create-high-priority-task-from-buffer` creates a task with priority '`A`'.

- `planner-create-medium-priority-task-from-buffer` creates a task with priority '`B`'.
- `planner-create-low-priority-task-from-buffer` creates a task with priority '`C`'.

Or, you can change the default priority of `planner-create-task-from-buffer` by customizing *planner-default-task-priority*.

You can actually use just one general priority, but using more than one color-codes your tasks and gives you a better overview of your day.

6.3.1.3 Task IDs

After loading '`planner.el`', make sure that '`planner-id.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-id)
```

This module modifies the behavior of '`planner.el`', adding global task IDs so that tasks can be edited and updated. Planner IDs are of the form '`{{Identifier:Number}}`'.

`planner-id-add-task-id-flag` [User Option]
Non-nil means automatically add global task IDs to newly-created tasks. If nil, use `planner-id-add-task-id` to add IDs to existing tasks, or `planner-id-add-task-id-to-all` to add to all tasks on the current page. (Development version 2004.05.05: planner–dev–1.0–patch-81)

`planner-id-update-automatically` [User Option]
Non-nil means automatically update linked tasks whenever a page is saved. If nil, use `planner-update-task` to update the linked task. By default, linked tasks are automatically updated.

`planner-id-tracking-file` [User Option]
File that contains ID tracking data. This file is automatically overwritten.

The following interactive functions are defined in '`planner-id.el`':

`planner-id-jump-to-linked-task` **&optional** *info* [Function]
Display the linked task page. If *info* is specified, follow that task instead.

`planner-id-add-task` [Function]
Add a task ID for the current task if it does not have one yet. Update the linked task page, if any.

`planner-id-update-tasks-on-page` **&optional** *force* [Function]
Update all tasks on this page. Completed or cancelled tasks are not updated. This can be added to `write-file-functions`. If *force* is non-nil, completed and cancelled tasks are also updated.

`planner-id-add-task-id-to-all` [Function]
Add a task ID for all the tasks on the page. Update the linked page, if any.

`planner-id-search-id` *id* [Function]
Search for all occurrences of *id*.

`planner-id-follow-id-at-point` [Function]
Display a list of all pages containing the ID at point.

`planner-id-follow-id-at-mouse` *event* [Function]
Display a list of all pages containing the ID at mouse. *event* is the mouse event.

6.3.1.4 Cyclic Tasks

Make sure that '`planner-cyclic.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-cyclic)
```

Create a diary file named '`~/.diary.cyclic-tasks`' (or the value of `planner-cyclic-diary-file`). Here is an example:

```
Tuesday #B0 _ Study Japanese
Friday #B0 _ Study Japanese (JapaneseStudies)
```

The first will be a plain task, the second will be linked.

By default, planner-cyclic creates multiple tasks if you let tasks build up (that is, the next Tuesday rolls around and you *still* haven't marked the task as done.) To turn off this behavior:

```
(setq planner-cyclic-diary-nag nil)
```

'`planner-cyclic-diary`' includes the following interactive functions:

`planner-cyclic-create-tasks-maybe` [Function]
Maybe create cyclic tasks. This will only create tasks for future dates or today.

'`planner-cyclic.el`' does not define any keybindings.

6.3.1.5 Task Detail

You may find your planner pages getting very full, so that you want to have one broad task entry be linked to a more specific list of sub-tasks. Or, maybe you want to have a number of notes linked to a particular task.

This can be done with targets. You can have a task that is really a meta-task:

```
#A1  _ Do things in RevelleLog#13 {{Tasks:101}} (RevelleLog)
```

'`RevelleLog#13`' could then be a list of sub-tasks in the form of a note, or any kind of note.

Or, instead of pointing to a particular note on '`RevelleLog`', you could have the whole page be tasks that you enter in manually, without linking them to another page. You can just type them in like this:

```
#A1  _ First specific thing to do
```

This way, the tasks will only appear on this specific project page, and not on any daily page, so you only see them when you want to look up all of the specific tasks associated with '`#A1 _ Do things in RevelleLog {{Tasks:101}} (RevelleLog)`'.

As you can see, the ability to manually enter tasks is one of Planner's nicest features. It allows you to create tasks that are not assigned to a specific date (by manually entering them on a project page with no date) or to a specific project (by manually entering them on

a day page with no project). Yet as long as you enter them using the syntax it understands, Planner will continue to recognize them as tasks.

Another way to have a task not be connected to a particular date is to do *C-c C-c* (`planner-copy-or-move-task`) and specify '`nil`' when asked for the date.

If you would like to see a list of all of your unfinished tasks, do *M-x planner-list-unfinished-tasks*. This function only checks day plan pages, not project pages.

6.3.1.6 Deadlines

You can use '`planner-deadline.el`' to automatically recalculate days to a deadline by adding `(require 'planner-deadline)` to your '`~/.emacs`'. With the default setup, make your tasks of the form

```
#A0 _ Some task {{Deadline: 2004.09.12}}
```

(Note: There must be at least one space after the colon.)

After you run `planner-deadline-update` to update task descriptions, the task will be of the form

```
#A0 _ Some task {{Deadline: 2004.09.12 - 2 days}}
```

`planner-deadline-regexp` [User Option]
: Regular expression for deadline data. The special deadline string should be regexp group 1. The date (YYYY.MM.DD) should be regexp group 2.

`planner-deadline-update` [Function]
: Replace the text for all tasks with deadlines. Deadlines are of the form '`{{Deadline: YYYY.MM.DD}}`' by default.

`planner-deadline-change` **&optional** *date* [Function]
: Change the deadline of current task to *date*. If *date* is nil, remove deadline.

6.3.2 Organizing Your Tasks

Okay, now that you've gotten the hang of creating tasks, you're probably facing a really long list of things to do. How can you organize them so that they don't overwhelm you? Planner gives you a number of strategies for dealing with large numbers of tasks.

1. Arrange your tasks in the rough order you're going to do them.
2. Use #A, #B and #C task priorities to differentiate between high-priority, normal and low-priority tasks.
3. Schedule your tasks onto different days.
4. Group your tasks into plan pages.
5. Don't schedule all your tasks.

1. *Task order*

   To remind yourself to do tasks in a certain order, simply edit the lines so that they're in the order you want. You can use normal editing commands like kill, yank and transpose-line to reorder the tasks, or use ⟨M-p⟩ (`planner-raise-task`) and ⟨M-n⟩ (`planner-lower-task`) to rearrange the tasks.

2. *Task priorities*

   By default, tasks are created with medium priority ('`#B`'). You can make a task high-priority ('`#A`') or low-priority ('`#C`') by manually editing the task and calling M-x planner-update-task to update the linked page. Alternatively, you can use ⟨C-M-p⟩ (`planner-raise-task-priority`) and ⟨C-M-n⟩ (`planner-lower-task-priority`) to modify the task and update the linked page.

   You can edit the priority of a task using *M-x planner-edit-task-priority*, or manually edit it and call *M-x planner-update-task* to update tasks on the linked page.

3. *Schedule your tasks on different days*

   You don't have to do everything today. Is this a task you would rather do tomorrow? Schedule it for then instead. You can specify '`+n`' or '`-n`' whenever you are asked for a date, where *n* is the number of days before or after the current file's date or today. Don't over-procrastinate things, though!

4. *Plan pages*

   Plan pages let you group related tasks and notes together for easy reference. For example, you could have a plan page for each major project or goal in your life, like '`GoodHealth`' or '`FurtherStudies`'.

   Although plan pages start by grouping everything under a '`* Tasks`' header, you can organize your plan pages in different ways. For example, you can separate groups of tasks with blank lines, and Planner will sort tasks within each group.

5. *Tasks without dates*

   Plan pages also allow you to have undated tasks or tasks with no particular deadlines. This keeps your daily task list small and manageable while making it easier for you to find things to do if you have free time. Make sure you check your plan pages regularly so that you don't completely forget about them.

   For automated scheduling of the next task on a plan page after you complete a task, see the section in http://sacha.free.net.ph/notebook/emacs/planner-config.el named "Schedule next undated task from same project".

6.3.2.1 Associating Tasks with Multiple Projects

You can use '`planner-multi.el`' to associate a task with more than one project. That way, you can easily keep GTD-style context lists as well as project-related lists.

To use multiple projects, add the following to your '`~/.emacs`':

```
(require 'planner-multi)
```

Under GNU Emacs, you can specify multiple projects by separating them with a single space. For example, you can specify *planner doc* when creating a task to associate the task with those two projects.

Under XEmacs, you can specify multiple projects by typing *RET* after each entry and terminating the list with another *RET*. For example, to specify *planner* and *doc*, you would type *planner RET doc RET RET* at the prompt.

If you want to see an overview of all of your tasks as well as project- or context-specific lists, you can set `planner-multi-copy-tasks-to-page` to your overview page(s). For example, set it to '`TaskPool`' to be able to see an overview of all of your unfinished tasks. You can also set this to multiple pages such as '`[[TasksByProject][p]]`

`[[TasksByContext][c]]`' and use '`planner-trunk.el`' to sort and organize tasks for easy reference. (see Section 6.3.2.7 [Grouping Tasks], page 32)

`planner-multi-copy-tasks-to-page` [User Option]
Automatically copy newly-created tasks to the specified page.

By default, tasks are removed from `planner-multi-copy-tasks-to-page` when you call `planner-task-done` or `planner-task-cancelled`. If you prefer to keep a copy of the task, remove `planner-multi-remove-task-from-pool` from `planner-mark-task-hook`.

If you want to use a different separator instead of spaces, customize the `planner-multi-separator` variable.

`planner-multi-separator` [User Option]
String that separates multiple page references.

For best results, this should be something recognized by `emacs-wiki-link-at-point` so that links are highlighted separately.

6.3.2.2 Viewing tasks

Review the tasks scheduled for today by typing *M-x planner-goto-today*. If you created the task from the previous section in this tutorial, you should see a line that looks like

```
#A0  _ Learn how to change a task's status from Tasks (TaskPool)
```

From left to right, these are what the symbols mean:

- '`A`' - Priority. A (high)
- '`0`' - Priority number. It is calculated whenever you save the file or call `planner-renumber-tasks`. Tasks are numbered in ascending order according to priorities.
- '`_`' - Status. _ (unfinished)

If you click on '`Tasks`' or press ⟨RET⟩ while your cursor is in the link, Emacs will display the previous info page.

If you select '`TaskPool`', Emacs will display the '`TaskPool`' plan page. Plan pages organize your tasks and notes about a project in one file.

You can use `planner-seek-next-unfinished-task` to move to the next unfinished task on the current page.

`planner-list-tasks-with-status` *status* **&optional** *pages* [Function]
Display all tasks that match the STATUS regular expression on all day pages. The PAGES argument limits the pages to be checked in this manner:

- `t`: check all pages
- regexp: search all pages whose filenames match the regexp
- list of page names: limit to those pages
- alist of page/filenames: limit to those pages

Called interactively, this function will search day pages by default. You can specify the start and end dates or leave them as nil to search all days. Calling this function with an interactive prefix will prompt for a regular expression to limit pages. Specify '`.`' or leave this blank to include all pages.

This function could take a long time.

`planner-list-unfinished-tasks` **&optional** *pages* [Function]
Display all unfinished tasks. *pages* follows planner-list-tasks-with-status.

`planner-jump-to-linked-task` *task-info* [Function]
Display the task page linked to by the current task or *task-info*.

6.3.2.3 Changing Tasks

To select a task, move your cursor to the line containing the task.

Change a task's priority ('`A`', '`B`' or '`C`') by editing the line. '`#A`' tasks are important. '`#B`' are medium priority. '`#C`' are low priority. Whenever you save the file or call *M-x planner-fix-tasks*, tasks are sorted and numbered according to priority and status.

Change a task's status by calling one of the following functions:

- planner-task-in-progress '`o`' (*C-c C-z*)
- planner-task-done '`X`' (*C-c C-x*)
- planner-task-cancelled '`C`' (*C-c C-S-x*)
- planner-task-delegated '`>`'
- planner-task-pending '`P`'
- planner-task-open '`_`'

After changing the status using a function, look at the '`TaskPool`' plan page. The task is also updated on the linked page. If you changed the task status manually by replacing the status with another character, you will need to call `planner-update-task` to update the linked page.

To reschedule a task, call `planner-copy-or-move-task` (*C-c C-c*) and choose a new date. You can mark a region and type *M-x planner-copy-or-move-region* to reschedule all the contained tasks to a different date. Enter '`nil`' for the date if you don't want the task or group of tasks to appear on any date page at all anymore. This is a good way to "de-schedule" a task for the time being, but still keep it linked to a plan page for possible future scheduling.

To change the plan page associated with a task, call `planner-replan-task`. Enter '`nil`' for the plan page if you don't want the task to appear on any plan page anymore. If you precede the command with a prefix argument, the text of the original plan page will appear in the prompt for easy editing.

Since the same task may exist on two or more pages, such as a date page and a plan page, it is dangerous to edit the description of the task by hand. You should not do it unless you want to make the exact same changes on all its linked pages.

Instead of doing this by hand, you should use `planner-edit-task-description`. This will prompt you for the changes to the task description and then update all the other pages to which the task is linked. Or, you can just use `planner-delete-task` to remove the task from both pages, and then create it again with the new desired description.

To remind yourself to do tasks in a certain order, simply edit the lines so that they're in the order you want. `planner-raise-task` and `planner-lower-task` update the priorities on linked pages automatically. You can organize tasks into groups by putting a blank line between groups of tasks. Planner will maintain the groupings and only sort the tasks within that group.

`planner-replan-task` *page-name* [Function]
Change or assign the plan page for the current task. *page-name* is the new plan page for the task. Use `planner-copy-or-move-task` if you want to change the date. With a prefix, provide the current link text for editing.

`planner-raise-task-priority` [Function]
Change a low-priority task to a medium-priority task and a medium-priority task to a high-priority task (C to B to A).

`planner-lower-task-priority` [Function]
Change a high-priority task to a medium-priority task and a medium-priority task to a low-priority task (A to B to C).

`planner-raise-task` *arg* [Function]
Move a task up *arg* steps. By default, *arg* is 1.

`planner-lower-task` *arg* [Function]
Move a task down *arg* steps. By default, *arg* is 1.

`planner-edit-task-description` *description* [Function]
Change the description of the current task, updating the linked page if any.

`planner-delete-task` [Function]
Delete this task from the current page and the linked page.

`planner-update-task` [Function]
Update the current task's priority and status on the linked page. Tasks are considered the same if they have the same description. This function allows you to force a task to be recreated if it disappeared from the associated page.

Note that the text of the task must not change. If you want to be able to update the task description, see `planner-edit-task-description` or '`planner-id.el`'.

See `planner-install-extra-task-keybindings` for additional task-related shortcuts.

6.3.2.4 Carrying Over Unfinished Tasks

Sometimes you won't be able to cross off all the tasks on your list. Planner makes it easy for you to keep track of things you still have to do by automatically rescheduling unfinished tasks from the past few days. By default, the `plan` command searches for unfinished tasks from the last three days and reschedules them onto today. If you open Planner every day, this should cover weekends easily.

It's a good idea to start Planner whenever you start Emacs. That way, Planner can help remind you about today's tasks, appointments, and other things. To automatically start Planner whenever you start up Emacs, add the following code to the end of your '`~/.emacs`':

```
(plan)
```

Now, every time you start Emacs (which should be more or less once a day), you'll see today's page. If you don't finish all the tasks today, you'll see them again tomorrow.

It's a good idea to start Planner every time you start Emacs so that you get reminded about your task list. If you prefer to start Planner manually, remember to call *M-x plan*

every so often to make sure that you don't forget any unfinished tasks. Safe in the knowledge that Planner tasks won't slip through the cracks (unlike little slips of paper that will invariably get mislaid), you can then get on with the rest of your life.

If your increased productivity with Planner leads to a well-deserved two-week vacation, then you'll need to tell Planner to search more days for unfinished tasks. By using *M-x plan*, you can automatically bring forward tasks over a given number of days or even scan all the days since time immemorial. *C-u 15 M-x plan* reschedules all unfinished tasks from the last 15 days. *C-u -1 M-x plan* checks all of your past day pages for unfinished tasks.

Like everything else in Planner, you can adapt *M-x plan* to your particular way of life. For example, if you find yourself starting up Emacs and Planner every day–including weekends–because it's just so much fun, you can set the `planner-carry-tasks-forward` to 1. This can make your Emacs startup marginally faster. On the other hand, if you normally start Emacs once a week, you can set it to 7 or 8. If you're worried about tasks dropping off your radar, you can set it to 0. You can set the value of *planner-carry-tasks-forward* either with ⟨M-x customize-variable RET planner-carry-tasks-forward RET⟩, or by putting `(setq planner-carry-tasks-forward 3)` (replacing `3` with the value appropriate for what you want) in your '`~/.emacs`' file.

On the other hand, you might prefer to reschedule tasks yourself. If you set `planner-carry-tasks-forward` to `nil`, then `M-x plan` and `(plan)` will bring you to the most recent page with unfinished tasks if there is no page for today. You can then use `planner-copy-or-move-task` and `planner-copy-or-move-region` to reschedule tasks. This is probably more hassle than it's worth, though, so let Planner take care of this for you.

`planner-carry-tasks-forward` [User Option]
: If non-nil, carry unfinished tasks forward automatically. If a positive integer, scan that number of days in the past. If 0, scan all days for unfinished tasks. If t, scan one day in the past (old behavior). If nil, do not carry unfinished tasks forward.

`plan` **&optional** *force-days* [Function]
: Start your planning for the day, carrying unfinished tasks forward.

  If *force-days* is a positive integer, search that many days in the past for unfinished tasks. If *force-days* is `0` or `t`, scan all days. If *force-days* is `nil`, use the value of `planner-carry-tasks-forward` instead, except t means scan only yesterday

`planner-copy-or-move-task` *date force* [Function]
: Reschedule the task for *date*. If *force* is non-nil, the task is moved regardless of status. It also works for creating tasks from a Note. Use `planner-replan-task` if you want to change the plan page in order to get better completion.

`planner-copy-or-move-region` *beg end date muffle-errors* [Function]
: Move all tasks from *beg* to *end* to *date*. `planner-copy-or-move-region` will copy or move all tasks from the line containing *beg* to the line just before *end*. If *muffle-errors* is non-nil, no errors will be reported.

6.3.2.5 Task Numbering

By default, tasks are numbered according to their position on the page. Task numbers allow you to refer to tasks using emacs-wiki links. For example, the '`#A1`' task in '`2004.08.16`' can be referred to as '`2004.08.16#A1`'.

Note that task numbers change every time you re-sort and re-number tasks with `planner-fix-tasks`. As a result, they are only reliable for references to past tasks.

If you find yourself not using this functionality, you can turn off task numbers by using the following option.

`planner-use-task-numbers` [User Option]

Non-nil means use task numbers when creating tasks. This allows you to refer to past tasks if your tasks are numbered appropriately. If you set this to nil, you can save space in your plan files.

6.3.2.6 Task Ranks

'`planner-rank.el`' models Franklin Covey's Urgency and Importance principle. When you think about a task, there are two aspects in consideration: Urgency and Importance. You may want to do the most urgent things first, like answering an email, or you may want to do the most important things first, like reading this manual. Or much better, balance Urgency and Importance and decide what to do.

'`planner-rank.el`' can help you balance.

Urgency and Importance are both measured by scores from 0-9. The higher the score, the more you want to do it first. 9 stands for "I should have done this" and 0 stands for "I can forget this".

If you are using the planner Section 6.3.1.6 [Deadlines], page 25 feature, the Urgency score is automatically calculated from how many days are left to meet the deadline. By default, it will score 9 if the task is overdue and 0 if the deadline is years away. Please refer to the docstring of `planner-rank-deadline-urgency-map-list` for detail.

The task rank is calculated from Urgency and Importance scores. As different people balance urgency and importance differently, a number of `planner-rank-calculate-rank-*` functions are provided. The algorithms vary from a simple average to something like a weighted root mean square deviation.

The aggressive versions of these functions (`planner-rank-calculate-rank-*-aggressive`) will make sure if one of Urgency and Importance is high, the resulting rank will be high as well. `planner-rank-calculate-rank-weighted-*` functions weigh the Urgency and Important scores depending on `planner-rank-importance-vs-urgency-factor`.

Call `planner-rank-test-algorithm` on each of the functions and check the result tables to see which one you like most, and set it to `planner-rank-rank-calculate-function`. Alternatively, accept the defaults and tweak them when you get a better feel for rranking.

Once the Rank is calculated, the Section 6.3.1.2 [Task Priorities], page 22 will be automatically reset. If the Rank is greater than or equal to `planner-rank-priority-A-valve`, the task priority will be '`A`', if the Rank is between `planner-rank-priority-A-valve` and `planner-rank-priority-B-valve`, the priority will be '`B`', else it will be '`C`'.

After setting the task importance and deadline, you can leave it as is. As the deadline approaches, the task priority will automatically be raised and the task re-colored to catch your eyes.

If you are using `planner-sort-tasks` (see see Section 6.5 [Making Files Pretty], page 38), you can set `planner-sort-tasks-key-function` to one of `planner-sort-tasks-by-rank`, `planner-sort-tasks-by-importance`, and `planner-sort-tasks-by-urgency`.

`planner-rank-change-hook` [User Option]
Functions to run after `planner-rank-change`.

`planner-rank-priority-A-valve` [User Option]
Tasks with rank greater than or equal to this value will be set to priority '`A`'.

`planner-rank-priority-B-valve` [User Option]
Tasks with rank greater than or equal to this value and less than `planner-rank-priority-A-valve` will be set to priority '`B`'. Tasks with rank less than this value will be set to priority '`C`'.

`planner-rank-deadline-urgency-map-list` [User Option]
Defines how to calculate the Urgency score according to how many days are left to meet the deadline.

`planner-rank-default-importance` [User Option]
Default importance value for newly added rank.

`planner-rank-default-urgency` [User Option]
Default urgency value for newly added rank.

`planner-rank-importance-vs-urgency-factor` [User Option]
How much do you think importance is more "important" than urgency. This will be used in `planner-rank-calculate-rank-weighted-*` functions.

`planner-rank-rank-calculate-function` [User Option]
Define the function called to calculate rank.

`planner-rank-change` **&optional** *importance urgency* [Function]
Set the Importance and Urgency of the current task.

`planner-rank-update-current-task` [Function]
Recalculate rank for the current task.

`planner-rank-update-all` [Function]
Recalculate rank for all tasks in the current page

`planner-rank-update-all` [Function]
Recalculate rank for all tasks in the current page

6.3.2.7 Grouping Tasks with planner-trunk.el

'`planner-trunk.el`' helps you automatically group tasks according to a set of rules. It sorts and splits your tasks, adding a blank line between groups of tasks. For example, if today's page looks like this:

```
* Tasks

#B   _ Buy milk (GroceryShopping)
#B   _ Learn how to use planner-trunk (PlannerMode)
#B   _ Buy a notebook (Bookstore)
#B   _ Buy cereal (GroceryShopping)
```

```
#B   _ Set up my own planner-trunk rules (PlannerMode)
#B   _ Customize my stylesheet (EmacsWikiMode)
#B   _ Go for a health checkup (BetterHealth)
```

then you might want to group the tasks into: planner and emacs-wiki, shopping list, and other items. If you set up the appropriate rules by customizing `planner-trunk-rule-list`, '`planner-trunk.el`' can automatically rewrite that section line this:

```
* Tasks

#B   _ Learn how to use planner-trunk (PlannerMode)
#B   _ Set up my own planner-trunk rules (PlannerMode)
#B   _ Customize my stylesheet (EmacsWikiMode)

#B   _ Buy milk (GroceryShopping)
#B   _ Buy a notebook (BookstoreShopping)
#B   _ Buy cereal (GroceryShopping)

#B   _ Go for a health checkup
```

In this case, you would set `planner-trunk-rule-list` to `(("." nil ("PlannerMode\\|EmacsWikiMode" "Shopping")))`.

You can load '`planner-trunk`' with *M-x load-library RET planner-trunk RET* or add `(require 'planner-trunk)`. If you're not yet comfortable with Emacs Lisp, you can use *M-x customize-variable RET planner-trunk-rule-list RET* to edit this rule using an easy-to-use interface.

WARNING: Do not keep non-task information in the Tasks section. planner-trunk will delete **all** non-task lines from the Tasks section of your plan page in the process of grouping the tasks.

After you set up `planner-trunk-rule-list`, use `M-x planner-trunk-tasks` to try out your rules until you're satistfied.

If you want to do this automatically, you can use `(add-hook 'planner-mode-hook 'planner-trunk-tasks)` to trigger it automatically whenever you open a Planner page.

6.3.3 Task Reports and Overviews

Planner provides a number of different ways to generate different presentations of your tasks.

6.3.3.1 Generating Daily Accomplishment Reports

You can use '`planner-accomplishments.el`' to get a summary of your task activity for a particular day. The report is grouped by status and plan (on day pages) or date (on plan pages). An example report follows:

```
Link        | Unfinished | In progress | Delegated | Completed | Total
nil         | 1          | 0           | 0         | 6         | 7
TaskPool    | 1          | 1           | 0         | 3         | 5
Planner     | 0          | 0           | 1         | 1         | 2
SchoolWork  | 0          | 0           | 0         | 1         | 1
Total       | 2          | 1           | 1         | 11        | 15
```

This lets you see how you balance your time between your projects.

There are currently two ways to use '`planner-accomplishments.el`'.

- Displaying a temporary buffer

  You can call `planner-accomplishments-show` to display a buffer containing the current page's accomplishment report.

- Rewriting sections of your planner

  Choose this approach if you want accomplishment reports to be in their own section and you would like them to be readable in your plain text files even outside Emacs. Caveat: The accomplishment section should already exist in your template and will be rewritten when updated.

  To use, set `planner-accomplishments-section` to the name of the section to rewrite (default: '`Accomplishments`'). If you want rewriting to be automatically performed, call `planner-accomplishments-insinuate`. The accomplishments will be rewritten whenever you save a planner page. If you want rewriting to be manual, call `planner-accomplishments-update`.

`planner-accomplishments-section` [User Option]
: Header for the accomplishments section in a plan page. Used by `planner-accomplishments-update`.

`planner-accomplishments-status-display` [User Option]
: Alist of status-label maps also defining the order of display. Used by `planner-accomplishments-format-table`.

`planner-accomplishments-insinuate` () [Function]
: Automatically call `planner-accomplishments-update` when saving tasks in `planner-mode` buffers.

`planner-accomplishments-update` () [Function]
: Rewrite `planner-accomplishments-section` with a summary of tasks on this page.

`planner-accomplishments-show` () [Function]
: Display a buffer with the current page's accomplishment report.

6.3.3.2 Seeing an Overview of Tasks

You can see a list of tasks with '`planner-tasks-overview.el`'. Seeing how you've scheduled tasks over the next few days can help you decide when to schedule another task. Table entries will be of the form

date | *link* | *priority status* | *task-description*

To display the tasks between a set of day pages, use

`planner-tasks-overview` *start end* [Function]
: Display an overview of your tasks from *start* to *end*. If *start* is nil, start from the very first day page. If *end* is nil, include the very last day page. You can use `planner-expand-name` shortcuts here, like `+1` or `-1`. Pressing ⟨RET⟩ at the prompt will use today.

  Once in a `planner-tasks-overview` buffer, you can use the keyboard shortcut ⟨o⟩ to change the overview period.

You can sort the task display with the following functions:

`planner-tasks-overview-sort-by-date` [Function]
Sort the tasks by date. Keyboard shortcut: ⟨1⟩

`planner-tasks-overview-sort-by-plan` [Function]
Sort the tasks by associated plan page. Keyboard shortcut: ⟨2⟩

`planner-tasks-overview-sort-by-priority` [Function]
Sort the tasks by priority. Keyboard shortcut: ⟨3⟩

`planner-tasks-overview-sort-by-status` [Function]
Sort the tasks by status. Keyboard shortcut: ⟨4⟩

You can jump to linked tasks with

`planner-tasks-overview-jump` *other-window* [Function]
Display the current task. If a prefix argument is supplied, show the task in another window. Keyboard shortcut: ⟨j⟩

`planner-tasks-overview-jump-other-window` [Function]
Display the current task in another window. Keyboard shortcut: *C-u j*

You can view a summary of the tasks in your plan pages with

`planner-tasks-overview-show-summary` **&optional** *file-list* [Function]
Count unscheduled, scheduled, and completed tasks for FILE-LIST. If called with an interactive prefix, prompt for the plan page(s) to display. Load '`planner-multi.el`' to be able to specify multiple pages.

⟨TAB⟩, *SHIFT-TAB* and ⟨RET⟩ navigate links in the usual fashion.

6.3.3.3 <tasks> tag

'`<tasks>`' is replaced by a report of tasks over all day pages in published pages. (see Section 6.8 [Publishing], page 41)

```
All incomplete tasks

<tasks status="^X">

All completed tasks

<tasks status="X">

All tasks

<tasks>
```

Warning: this function can be slow, as it checks all the day pages!

6.4 More about Notes

You can put anything in this section. Notes added by '`remember-planner.el`' look like this:

```
.#1 Headline
Body
```

and are outlined at the `H3` level. If you want to take notes conveniently, check out '`remember-planner.el`'.

Planner by default organizes the notes on a planner page so that the most recent note is first. Each note is numbered, with the oldest note labeled '`.#1`'. If you would like to reverse this behavior, look at *`C-h v planner-reverse-chronological-notes`*.

Notes are associated with day pages, but can also be associated with plan pages when they are created. A linked note looks like this:

```
.#1 Headline (LinkedNote#1)
Text
```

You can jump to the linked note with `planner-jump-to-linked-note`.

Deleting a note can be dangerous, as the notes are automatically numbered. Removing a note could break links in other pages.

`planner-create-note` *page* [Function]
: Create a note to be remembered in *page* (today if *page* is nil). If `planner-reverse-chronological-notes` is non-nil, create the note at the beginning of the notes section; otherwise, add it to the end. Position point after the anchor.

`planner-create-note-from-task` [Function]
: Create a note based on the current task and update the current task to link to the note.

`planner-renumber-notes` [Function]
: Update note numbering.

`planner-jump-to-linked-note` *note-info* [Function]
: Display the note linked to by the current note or *note-info* if non-nil.

`planner-search-notes` *regexp limit* [Function]
: Return a buffer with all the notes returned by the query for *regexp*. If called with a prefix argument, prompt for *limit* and search days on or after *limit*.

The following sections include instructions for displaying, manipulating, and navigating your notes efficiently.

6.4.1 Using Allout Mode

The format of the notes in Planner works well with Allout mode, which provides helpful commands for navigating and formatting outlines. You can, for example, hide the bodies of all the notes on a page so you can see just their headlines. You can also jump easily from headline to headline, skipping over the bodies in between.

The commands for using Allout mode vary depending on which Emacs version you are using. In either case, type *`M-x load-library`* ⟨RET⟩ *`allout`* ⟨RET⟩ to start. If you are using

CVS Emacs, type *M-x allout-mode* ⟨RET⟩. If you are using an earlier version of Emacs, type *M-x outline-mode* ⟨RET⟩.

The exact commands then available to you differ depending on your Emacs version, but you can view the commands and their keybindings by typing *C-h m*. In CVS Emacs, they will start with `allout-`, while in previous versions, they will start with `outline-`.

6.4.2 <notes>

'`<notes>`' is replaced by a list of note headlines when the page is published. For example, the notes tag in the following example will be replaced by the two headlines when published. (see Section 6.8 [Publishing], page 41)

```
<notes>

* Notes

.#1 This is a headline

and this is body text

.#2 This is another headline

and this is more body text
```

'`<notes>`' is useful if you want to provide a quick summary of blog entries at the top of your page. Just add it to your `planner-day-page-template`.

6.4.3 <past-notes>

'`<past-notes>`' is replaced by an index of note headlines. If *start* is specified, it lists notes starting from that date. If *directory* is specified, you can index notes in another planner directory.

```
All the notes I've taken in 2004:

<past-notes start="2004.01.01" end="2004.12.31">
```

6.4.4 Note Indices

Make sure that '`planner-notes-index.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-notes-index)
```

Then you can use tags of the form:

```
<planner-notes-index page="2004.03.02">
<planner-notes-index from="2004.03.01" to="2004.03.31">
<planner-notes-index limit="10">
<planner-notes-index page="PlanPage">
<planner-notes-index-month-table month="2004.03" limit="5">
<planner-notes-index-month-table month="2004.03">
```

You can also use the following interactive functions:

`planner-notes-index` `planner-notes-index-days` `planner-notes-index-weeks` `planner-notes-index-months` `planner-notes-index-years` (wow!)

These work based on the current date (date of current buffer, or today).

If a single page is specified, this page is scanned for headlines of the form:

```
.#1 Headline
```

The results are presented as a bulleted list.

If *from* and *to* are specified, all date pages between them (inclusive) are scanned. If *from* is omitted, it is assumed to be the earliest entry. If *to* is omitted, it is assumed to be the latest entry.

If *recent* is specified, the list includes only that many recent headlines. and the results are presented as a bulleted list.

To customize presentation, you can write a function that generates the appropriate `<planner-notes-index>` tags. You can also use `planner-extract-note-headlines` in your own functions.

The following interactive functions are defined in '`planner-notes-index.el`':

`planner-notes-index` **&optional** *from to limit* [Function]
: Display a clickable notes index. If called from a Lisp program, display only dates between *from* and *to*. With a prefix arg *limit*, display only that number of entries.

`planner-notes-index-days` *days* [Function]
: Display an index of notes posted over the past few *days*. The list ends with the day of the current buffer or `planner-today`.

`planner-notes-index-weeks` *weeks* [Function]
: Display an index of notes posted over the past few *weeks*. The list ends with the week of the current buffer or `planner-today`. Weeks start from Sunday.

`planner-notes-index-months` *months* [Function]
: Display an index of notes posted over the past few *months*. The list ends with the month of the current buffer or `planner-today`.

`planner-notes-index-years` *years* [Function]
: Display an index of notes posted over the past few *years*. The current year is included.

'`planner-notes-index.el`' does not define any keybindings.

6.5 Making Files Pretty

By default, planner does a little bit of fancy reformatting when you save a file. Tasks are sorted by priority (ABC) and status (_oP>XC) on day pages. On plan pages, tasks are sorted by priority (ABC), status (XC), and date. Undated tasks are sorted after dated tasks.

`planner-sort-tasks` [Function]
: Sort tasks according to planner-sort-tasks-key-function. By default, sort tasks according to priority and position on day pages, and according to status, priority, date, and position on plan pages.

`planner-renumber-tasks` [Function]
Update task numbering.

`planner-align-tasks` [Function]
Align tasks neatly. You can add this to `write-file-functions` to have the tasks automatically lined up whenever you save. For best results, ensure `planner-align-tasks` is run after `planner-renumber-tasks`.

`planner-fix-tasks` [Function]
Sort, renumber and align tasks.

`planner-sort-tasks-key-function` [User Option]
Control task sorting. Some options include `planner-sort-tasks-default-key`, `planner-sort-tasks-basic`, `planner-sort-tasks-by-date`, and `planner-sort-tasks-by-link`.

`planner-sort-undated-tasks-equivalent` [User Option]
This option controls the behavior of task sorting on plan pages. By default, the value '`9999.99.99`' causes dated tasks to be listed before undated tasks. To sort undated tasks before dated tasks, set this to a blank string.

`planner-sort-tasks-automatically` [User Option]
Non-nil means sort tasks whenever a planner file is saved. On day pages, tasks are sorted by status. On plan pages, they are sorted by status and date. Sorting can take a while.

`planner-renumber-tasks-automatically` [User Option]
Non-nil means renumber tasks from 1 to N whenever a planner file is saved. This allows you to refer to tasks in previous day pages using anchors like '`2003.08.12#A1`'. If you use this function, make sure `planner-use-task-numbers` is non-nil so that new tasks are created with task numbers.

`planner-align-tasks-automatically` [User Option]
Non-nil means align tasks whenever a planner file is saved. This causes the status to line up in a neat column if you have less than 100 tasks.

`planner-renumber-notes-automatically` [User Option]
Non-nil means renumber the notes. If most of your notes are only on one page, you might like seeing the notes renumbered if you delete a note in the middle. However, if you use a lot of cross-referencing, note renumbering will break those links.

6.6 Annotations

The context included when you create a task and the context included when you create a note are gained the same way. It sounds like black magic, but it turns out not to be.

All that happens is, Planner has a list of functions, `planner-annotation-functions`. When you create a task from a buffer, or remember a note from a buffer, Planner goes through this list from top to bottom. The first one that returns true is the one it uses.

For example, if you're in Wanderlust, and you hit the key you've bound to `planner-create-task-from-buffer`, it looks at this list and does something like this. Is it an ERC

buffer? No. Is it a BBDB buffer? No. Are we in w3m? No. Are we in Wanderlust? Yes. So this function succeeds. It stops searching and runs the annotation function for Wanderlust, which in this case finds out who the message is from and what the message ID of the message is. It then takes those and constructs a link back to that message, with a link title something like '`Email from Joe Blogs`'.

So, you've read the email from Joe Blogs. He's asked you to do something and you've hit your key to add that task to your list of things to do. So what you end up with is a description of the task, and a link back to what made you create the task in the first place.

The same happens with remembering notes, except that it ends up in the '`* Notes`' section of your page instead.

By default, '`planner.el`' can annotate tasks and notes based on the current filename. To change the behavior of annotations, customize the following options:

`planner-annotation-functions` [User Option]
: A list of functions tried in order by `planner-create-task-from-buffer` and other functions that pick up context. The first non-nil value returned is used as the annotation. To cause an item to **not** be annotated, return the empty string '`""`'.

`planner-annotation-strip-directory` [User Option]
: File links are usually generated with the full path to the file so that you can easily tell apart files with the same base name. If `planner-annotation-strip-directory` is non-nil, though, only the base name of the file will be displayed. For example, a link to '`/foo/bar/baz`' will be displayed as '`baz`' and hyperlinked to the file.

`planner-annotation-use-relative-file` [User Option]
: If t, always use relative filenames. `planner-annotation-use-relative-file` can also be a function that takes the filename and returns non-nil if the link should be relative. Filenames are resolved relative to `planner-directory`. That is, the created link will be of the form '`../../somefile`' instead of '`/absolute/path/to/file`'. This can be helpful if you publish your planner files to the Net and your directory structure ensures that relative links resolve the same from `planner-directory` and `planner-publishing-directory`.

6.7 Interactive Lisp with planner-lisp.el

You can include interactive Lisp functions in your planner pages.

First, you need '`planner-lisp.el`'. Put this in your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-lisp)
```

Then, add a link to the Lisp function to your page, like:

```
[[lisp:/plan][Plan]]
```

This will be rendered as '`Plan`'. Selecting the link will run the `plan` function interactively.

You can also execute other Lisp expressions. For example:

```
[[lisp:/(planner-goto (planner-expand-name "+7"))][Next week]]
```

'`planner-lisp.el`' does not provide any interactive functions or keybindings.

6.8 Publishing

You can publish your planner files to HTML and put them on a normal web server–no special server support required. This gives you an easy way to keep other people up to date on your tasks, keep a web log, or simply organize information.

Publish your planner wiki with *C-c C-p* (`emacs-wiki-publish`). You can publish individual files with `emacs-wiki-publish-this-file`. To automatically publish files when you save them, add the following code to your '`~/.emacs`':

```
(defun sacha/emacs-wiki-auto-publish ()
  (when (planner-derived-mode-p 'emacs-wiki-mode)
    (unless emacs-wiki-publishing-p
      (let ((emacs-wiki-publishing-p t)
            (emacs-wiki-after-wiki-publish-hook nil))
(emacs-wiki-publish-this-page)))))
(add-hook 'emacs-wiki-mode-hook
          (lambda ()
           (add-hook 'after-save-hook
                     'sacha/emacs-wiki-auto-publish nil t)))
```

Published files are stored in `planner-publishing-directory`. Just copy the contents of this directory to your webserver, and you're all set! Of course, publishing is completely optional.

Here are some more features related to publishing:

6.8.1 Publishing Calendars

Here is an example Emacs Lisp snippet that automatically publishes calendars in your day pages.

```
(require 'planner-calendar)
(add-hook
 'planner-mode-hook
 (lambda ()
   "Add the relevant hooks for `planner-calendar' to work."
   (add-hook 'emacs-wiki-before-markup-hook
             'planner-calendar-insert-calendar-maybe nil t)
   (add-hook 'emacs-wiki-after-file-publish-hook
             'planner-calendar-create-today-link nil t)
   (add-hook 'emacs-wiki-after-markup-hook
             'planner-calendar-move-calendar-to-top-of-page-maybe
             nil t)))
```

`planner-calendar-prev-month-button` [User Option]
HTML text used for previous month buttons.

`planner-calendar-next-month-button` [User Option]
HTML text used for next month buttons.

`planner-calendar-day-header-chars` [User Option]
Number of characters to use for day column header names.

`planner-calendar-today-page-name` [User Option]
: Default Emacs-Wiki base name for published today page link.

`planner-calendar-insert-calendar-maybe` [Function]
: Add this function to `emacs-wiki-before-markup-hook` in `planner-mode` buffers in order to automatically add a day page during the calendar. This cannot be done from the page header because the inserted text still needs to be processed.

`planner-calendar-create-today-link` [Function]
: Add this function to `emacs-wiki-after-file-publish-hook` to create a "today" soft-link to the newest published planner day page, on operating systems that support POSIX `ln`.

`planner-calendar-move-calendar-to-top-of-page-maybe` [Function]
: Add this function to `emacs-wiki-after-markup-hook` if you want the calendar inserted by `planner-calendar-insert-calendar-maybe` to be moved to the beginning of the page. This work-around is necessary because some functions used by '`planner-calendar.el`' only work during markup.

6.8.2 Authz Access Restriction

'`planner-authz.el`' was written by Andrew J. Korty in order to allow the easy restriction of portions of published pages. It uses the HTML::Mason module available on CPAN (http://www.cpan.org). Setting up HTML::Mason is outside the scope of this document. Make sure that it works before trying out '`planner-authz.el`'.

'`planner-authz.el`' modifies the behavior of `emacs-wiki-publish` so that published pages follow access modifiers.

This library lets you publish your planner pages while controlling access to certain portions of them to users you specify. When you load this library, you gain access to two additional markup directives to use in your planner pages. The '`<authz>`' tag lets you restrict access to arbitrary content as follows:

```
Here is a sentence everyone should see.  This sentence also
contains no sensitive data whatsoever.  <authz users="ajk">This
sentence, however, talks about my predilection for that French
vanilla instant coffee that comes in the little tin, and I'm
embarrassed for anyone else to know about that.</authz> And
here's some more perfectly innocuous content.
```

You can use '`<authz>`' tags to mark up entire paragraphs, tasks, notes, and anything else. The tags are replaced with Mason code in the published pages.

The '`#authz`' directive restricts access to an entire page. A Mason call is added to this page to generate a 403 error when someone not listed tries to access it. Any notes or tasks on a '`#authz`'-protected page are also wrapped in Mason code on linked pages. To add a '`#authz`' directive to an emacs-wiki page, place '`#authz`' followed by a space-delimited list of users on one line. For example:

```
#authz ajk sacha
```

`planner-authz-project-default` [User Option]
: Default access list for project pages (not day pages). If a given project page doesn't contain a '`#authz`' tag, it will receive the access list defined here. If this variable is

nil, all users will be allowed to view the page. No corresponding variable is provided for day pages because it doesn't seem like you'd ever want to control access based on what day it was. (But I will accept patches. :) Notes and tasks referencing pages without '`#authz`' tags will also be restricted to the users listed here.

`planner-authz-day-note-default` [User Option]
Default access list for notes on day pages not associated with any project. There is way to set a default for notes on project pages for the reason above; they would only be associated with date pages anyway.

`planner-authz-day-task-default` [User Option]
Same as *`planner-authz-day-note-default`*, but for tasks.

`planner-authz-publish-index` [Function]
Publish an index for the planner marked up with Mason code. Only those links to pages which the remote user is authorized to access will be shown.

6.8.3 RSS Publication

'`planner-rss.el`' allows you to publish your notes in the RSS 2.0 XML format for blog syndication. You will also want to customize the following variables:

`planner-rss-base-url` [User Option]
Base absolute URL for published blog entries. Should include trailing '`/`'.

`planner-rss-category-feeds` [User Option]
Rules for automatic categorization of posts and publishing to RSS files. A blog entry is matched against each condition. If it matches the regular expression or the function returns a non-nil value, the blog entry is copied into the specified file.

`planner-rss-feed-limits` [User Option]
A list of regular expressions that match feed filenames and the size and item limits for feeds that match. For example, you can use '`(("." nil 10))`' to ensure that all feeds are limited to the 10 most recent items.

To manually add the current note to all the matching RSS feeds, invoke `planner-rss-add-note`. You can specify a filename with the universal prefix, like this: *`C-u M-x planner-rss-add-note`*.

If you use the '`remember-planner.el`' module to create notes, you can automatically publish new notes to RSS feeds by adding the following code to your '`.emacs`' (or '`_emacs`').

```
(add-to-list 'remember-planner-append-hook 'planner-rss-add-note t)
```

'`planner-rss.el`' defines the following interactive functions:

`planner-rss-add-note` *feed* [Function]
Export the current note using `planner-add-item`. If *feed* is specified, add the entry to the specified file. Else, add the entry to all matching RSS feeds specified by `planner-rss-category-feeds`.

6.8.4 iCal Publication

iCal is an Internet Engineering Task Force (IETF) standard for calendaring and scheduling. http://www.ietf.org/rfc/rfc2445.txt

You can export your tasks to the iCal format using 'planner-ical'. Add `(require 'planner-ical)` to your '~/.emacs'. Then you can use *M-x planner-ical-export-page* to display the iCal equivalent of tasks on a page, which you can then save to a file.

To export today's tasks to a file in your publishing directory, add the following to your '~/.emacs'.

```
(planner-ical-export-file
 (planner-today)
 (expand-file-name "tasks.ics"
                   planner-publishing-directory))
```

`planner-ical-export-page` *page* **&optional** *file* [Function]
Export PAGE's tasks in the iCal format. If FILE is non-nil, results are saved to that file. If FILE is nil, results are displayed in a 'planner-ical-export-buffer'.

`planner-ical-export-this-page` [Function]
Display the tasks on the current page in iCal format.

6.8.5 RDF Publication

Put planner-rdf.el in a directory that is in your Emacs load-path and the following into your ~/.emacs file:

```
(require 'planner-rdf)
(add-hook 'emacs-wiki-after-file-publish-hook
          'planner-rdf-publish-file)
(add-hook 'emacs-wiki-after-wiki-publish-hook
          'planner-rdf-publish-index)
```

6.8.5.1 Publishing with planner-rdf

Planner-rdf is now included in the normal Planner publishing process. Pressing ⟨C-p⟩ will create a .owl and a .rdf file for every planner file. Additionally it creates an index, 'index.rdf'.

By default all generated files will be stored in the normal Planner publishing directory, where the HTML files end up. If you would ike to change that, set the variable `planner-rdf-directory` to the desired location.

The generated files:

- 'index.rdf' - a collection with pointers to the '<plan-page>.rdf' files.
- '<plan-page>.rdf' - contains Dublin Core metadata about the files related to the current Planner page. Currently it contains metadata about the source file, the Emacs plan page, the generated HTML page, and the generated OWL file.
- '<plan-page>.owl' - contains task and note data from the Planner file. Task information is stored completely. For notes, the note headline is stored.

6.8.5.2 planner-rdf Tags

Besides the factual information, e.g. the task status or description, planner-rdf extracts links (in the format '`[[...][...]]`' or '`[[...]]`') and tags ('`{{...:...}}`') from tasks and notes (including the notes text). Links and tags provide context for the plan elements and so are stored and linked with the containing elements.

Links point to locations that can be used to enrich the information in the Planner pages (e.g, by retrieving data from them and adding it), tags – like the one for the task ids '`{{Tasks:198}}`' – can be used to express abstract qualities. Some examples:

- an '`{{audience:myteam}}`' tag, can be used to restrict publishing of items to a certain user group;
- a '`{{lang:de}}`' tag, signifying the language of the text;
- a '`{{location:Hamburg}}`' tag, can be used to make geographic reference to an entity that is not stored in your address book, bbdb.

What tags to use is up to the user. Planner-rdf makes no assumptions about them, it just extracts and stores them. Only the applications using the data know what to do with them.

6.8.5.3 Usage Examples

Report generation with OpenOffice

The Perl file '`this-week.pl`' (http://www.rainervolz.de/planner-rdf/this-week.pl) creates a simple report for the current week. The script extracts task and note information from the generated OWL files and inserts it into a simple OpenOffice Writer document. Nothing fancy, just a proof of concept, to show how planner-rdf can be used to integrate Planner Mode with other applications.

Besides Perl and OpenOffice you'll need the Redland RDF Application Framework (http://www.redland.opensource.ac.uk/). It is used to process the RDF data. Redland is small, but powerful, and available for many platforms and languages.

As an example the application loads the RDF data each time it is run. In the real world you probably would use Redland to store the Planner data in a database, to save the loading step each time you access the data.

Importing Planner data into Protege

Protege is a popular ontology editor and knowledge management application. A simple way to import data into it, is to provide a OWL file that contains the data as well as the schema. To do this:

- Use '`planner2protege.pl`' (http://www.rainervolz.de/planner-rdf/planner2protege.pl) to combine all OWL files into a single one.
- Use CWM (http://www.w3.org/2000/10/swap/doc/cwm.html) to combine schema and data, with `python cmw --rdf planner-rdf.owl planner-data.owl --think --rdf >planner2.owl`

Not the most straightforward process, but it works. The resulting file, here planner2.owl, can then be loaded into Protege.

6.9 Experimental Functions

These functions are experimental. This means that they may not do exactly what you expect them to do, so keep backups, be careful, and don't blame us.

To use these functions, make sure that '`planner-experimental.el`' is in your load path, and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-experimental)
```

'`planner-experimental.el`' defines the following interactive functions:

`planner-search-notes-next-match` [Function]
: Jump to the next matching entry. Call after `planner-search-notes`.

`planner-search-notes-previous-match` [Function]
: Jump to the previous matching entry. Call after `planner-search-notes`.

`planner-remove-duplicates` [Function]
: Remove duplicate tasks.

'`planner-experimental.el`' does not define any keybindings.

7 Managing Your Information

Planner can be integrated with other Emacs and even some non-Emacs programs by loading additional modules. You can pick and choose from these modules, choosing those that work with programs you use and that produce information you want to have included in your Planner pages.

7.1 E-mail

Planner can work together with several different Emacs e-mail clients. If you load the appropriate module for the e-mail client you use, then your notes and tasks can be annotated with information pointing to the specific e-mail message you were reading when you created that note or task. When you are looking at the note or task, you will be able to jump straight to that message.

7.1.1 Unix mail

This module supports links from any kind of Unix mailbox (mbox). To use this module, make sure that '`planner-unix-mail.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-unix-mail)
```

Unix mail URLs are of the form:

```
;; mail://PATH/TO/INBOX/message-id
```

Annotations will be of the form:

```
[[mail://PATH/TO/INBOX/E1AyTpt-0000JR-LU%40sacha.ateneo.edu][E-mail from Sacha Chua]]
```

'`planner-unix-mail.el`' does not define any interactive functions or create any new keybindings.

7.1.2 Gnus

To use this module, make sure that it is in your load path and put this in your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-gnus)
(planner-gnus-insinuate)
```

With this module loaded, when you create tasks from Gnus summary or message buffers, the tasks will be annotated with information from the message you were looking at when you created each task. A link will also be created on your planner page that you can select in order to return to the message.

So, the created task will look something like this:

```
#A34 _ Send writing to publishme.com from
[[gnus://alt.books.beatgeneration/<Ifo5c.24632$F9.9567@nwrddc01.gnilink.net>][E-Mail
from editor@verizon.net]] {{Tasks:71}} ([[Writing]])
```

This module also binds `C-c C-t` in the Gnus summary and article views to the command to create a new task.

'`planner-gnus.el`' does not define any interactive functions.

For more information about Gnus, see See Info file '`gnus`', node '`Top`'.

7.1.3 VM

To use this module, make sure that '`planner-vm.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-vm)
```

VM URLs are of the form:

```
vm://path/to/inbox/message-id
```

Annotations will be of the form:

```
[[vm://home/test/INBOX/<E1AyTpt-0000JR-LU@sacha.ateneo.edu>][E-mail from Sacha Chua]]
```

'`planner-vm.el`' does not define any interactive functions or keybindings.

7.1.4 Wanderlust

To use this module, make sure that '`planner-wl.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-wl)
```

Then, when you call *M-x planner-create-task-from-buffer* from Wanderlust summary or message buffers, the task will be created with the correct annotation.

'`planner-wl`' does not define any interactive functions.

To add keybindings to Wanderlust, call:

```
(planner-wl-insinuate)
```

This binds *F* to `planner-create-task-from-buffer`.

7.1.5 MH-E

To use this module, make sure that '`planner-mhe.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-mhe)
```

Then, when you call *M-x planner-create-task-from-buffer* from MH-E summary or message buffers, the task will be created with the correct annotation.

'`planner-mhe`' does not define any interactive functions.

7.1.6 Rmail

To use this module, make sure that '`planner-rmail.el`' is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-rmail)
```

Rmail URLs are of the form:

```
rmail://message-id
```

Annotations will be of the form:

```
[[rmail://<E1AyTpt-0000JR-LU@sacha.ateneo.edu>][E-mail from Sacha Chua]]
```

'`planner-rmail.el`' does not define any interactive functions or create any new keybindings.

For more information about Rmail, see See Info file '`Emacs`', node '`Rmail`'.

7.2 Scheduling and Time

7.2.1 Diary

If you use Emacs's diary feature, Planner-Diary could be helpful for you. It puts all diary entries for the current day in the '`* Diary`' section of your day plan page. This section is updated every time you display the file in Emacs. By default the diary section of past pages is not updated; it's pretty unlikely that you want to add new diary entries for the past. (see section "Diary" in *GNU Emacs Manual*)

If you want to use '`planner-diary.el`', make sure the file is in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-diary)
```

'`planner-diary.el`' needs `fancy-diary-display`. To use `fancy-diary-display`, add this to your '`.emacs`' (or '`_emacs`'):

```
(add-hook 'diary-display-hook 'fancy-diary-display)
```

You can use Planner-Diary in two different ways:

1. If you want the saved files to contain your entries and not just a line of Lisp, add the following lines to your '`.emacs`' (or '`_emacs`'):

   ```
   (setq planner-diary-use-diary t)
   (planner-diary-insinuate)
   ```

   You should also customize or set `planner-day-page-template` to include a '`* Diary`':

   ```
   (setq planner-day-page-template
    "* Tasks\n\n\n* Schedule\n\n\n* Diary\n\n\n* Notes")
   ```

2. (GNU EMACS ONLY) You can put the following line of Lisp code in your day plan pages to display your diary entries:

   ```
   <lisp>(planner-diary-entries-here)</lisp>
   ```

   You can do this automatically for all day plan pages:

   ```
   (setq planner-day-page-template
     "* Tasks\n\n\n* Diary\n\n<lisp>(planner-diary-entries-here)</lisp>\n\n* Notes")
   ```

   When you open a day plan page outside Emacs, you will see the line of Lisp code and not your diary entries.

C-c C-e updates the diary sections. *C-u C-c C-e* forces an update—it inserts the diary section for the day, even if the day is in the past or if there is no '`Diary`' section in the buffer.

If you want to see your diary entries for more than just one day, set `planner-diary-number-of-days` accordingly. This works for either of the two approaches.

If you want to use the Cal-Desk package, simply follow the instructions in '`cal-desk.el`'. If you get the Cal-Desk layout from the Calendar buffer, you get it in the day plan buffer, too.

If you use Planner-Diary, you might consider using the Calendar support of Planner. (see section "Calendar/Diary" in *GNU Emacs Manual*) To get Calendar integration, add this to your '`.emacs`' (or '`_emacs`'):

```
(planner-insinuate-calendar)
```

If `planner-diary-create-section-flag` is non-nil, sections are always inserted if necessary.

`planner-diary-create-section-flag` [User Option]
: Non-nil means create the requested diary sections if they do not exist. By default, planner-diary tries to create the section. If you want more control over your pages, you can set this to nil. Trying to write a diary section to a page that does not have it yet will then result in an error.

By default, planner-diary lists only the appointments you have on that day. If you want the date headers included even when showing the diary entries for a single day, set planner-diary-include-all-output to non-nil.

`planner-diary-include-all-output-flag` [User Option]
: Non-nil means don't omit any data when copying diary entries into day pages.

'`planner-diary.el`' defines the following interactive functions:

`planner-diary-add-entry` *date time text* [Function]
: Prompt for a diary entry to add to `diary-file` on *date*. Uses `planner-annotation-functions` to make hyperlinks. *time* and *text* are used in the description."

`planner-diary-insert-diary` *force* [Function]
: Insert the fancy diary for the day into the day plan file. If *force* is non-nil, insert a diary section even if there is no *planner-diary-string* in the buffer.

`planner-diary-insert-diary-maybe` *force* [Function]
: Maybe insert the fancy diary for the day into the day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a diary section even if there is no `planner-diary-string` in the buffer.

`planner-diary-insert-appts` *force* [Function]
: Insert the diary appointments for the day into the day plan file. If *force* is non-nil, insert a diary appointments section even if there is no `planner-diary-appts-string` in the buffer.

`planner-diary-insert-appts-maybe` *force* [Function]
: Maybe insert the diary appointments for the day into the day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a diary appointments section even if there is no `planner-diary-appts-string` in the buffer.

`planner-diary-insert-cal-desk` *force* [Function]
: Insert the cal-desk diary for the day into the day plan file. If *force* is non-nil, insert a cal-desk diary section even if there is no `planner-diary-cal-desk-string` in the buffer.

`planner-diary-insert-cal-desk-maybe` *force* [Function]
: Maybe insert the cal-desk diary for the day into the day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a cal-desk appointments section even if there is no `planner-diary-cal-desk-string` in the buffer.

`planner-diary-insert-public` *force* [Function]
Insert the public diary for the day into the day plan file. If *force* is non-nil, insert a public diary section even if there is no `planner-diary-public-string` in the buffer.

`planner-diary-insert-public-maybe` *force* [Function]
Maybe insert the public diary for the day into the day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a public appointments section even if there is no `planner-diary-public-string` in the buffer.

`planner-diary-insert-private` *force* [Function]
Insert the private diary for the day into the day plan file. If *force* is non-nil, insert a private diary section even if there is no `planner-diary-private-string` in the buffer.

`planner-diary-insert-private-maybe` *force* [Function]
Maybe insert the private diary for the day into the day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a private appointments section even if there is no `planner-diary-private-string` in the buffer.

`planner-diary-insert-all-diaries` *force* [Function]
Update all diary sections in a day plan file. If *force* is non-nil, insert a diary section even if there is no section header. It only inserts diaries if the corresponding `planner-diary-use-*` variable is '`t`'.

`planner-diary-insert-all-diaries-maybe` *force* [Function]
Update all diary sections in a day plan file. If the current day is in the past and *force* is nil, don't do anything. If *force* is non-nil, insert a diary section even if there is no section header. It only inserts diaries if the corresponding `planner-diary-use-*` variable is '`t`'.

`planner-diary-show-day-plan-or-diary` [Function]
Show the day plan or diary entries for the date under point in calendar. Add this to `calendar-move-hook` if you want to use it. In that case, you should also `remove-hook` '`planner-calendar-show`' from `calendar-move-hook`.

'`planner-diary.el`' adds the following keybinding to Planner, if `planner-diary-insinuate` is in your '`.emacs`' (or '`_emacs`'):

- *C-c C-e* updates the diary sections.

7.2.1.1 Planner-Diary Advanced Features

The features described here are part of the development version. They are subject to change without notice. They may be buggy. The documentation may be inaccurate. Use at your own risk.

There is a lot of code redundancy in the development version. This is intentional and makes it easier to change the code for one type of diary section without breaking others.

Currently Planner-Diary supports six different views of your diary entries:

1. Ordinary fancy diary display (what you get by pressing *d* in the calendar buffer with `fancy-diary-display` switched on)

2. Schedule/Appointments (all entries from 1 that have a time assigned to them
3. Diary without appts (1 without 2)
4. cal-desk display (appts on top, non appts entries at bottom)
5. A private diary (same as 1, but uses a different diary-file)
6. A public diary (same as 1, but uses a different diary-file)

Put the following line of Lisp code in your day plan pages to display your diary entries:

```
<lisp>(planner-diary-entries-here)</lisp>
```

The function `planner-diary-entries-here` takes two optional arguments—the diary file you want to use and the number of days you want to display.

7.2.2 Appointments

If you would like to use planner for your appointment alerts instead of using the diary system, you might like to try '`planner-appt`'.

According to your preferences, you may choose from two different approaches. Appointments in task descriptions on today's plan page are like this:

```
#A   _ @12:45 Do something (TaskPool)
```

and appointments in today's schedule section are like this:

```
* Schedule

9:00 | 12:00 | Read Simmel's Philosophy of Money
@12:45 |       | Do Something Else
@13:00 | 14:00 | lunch
```

You can even use both at the same time if you like.

Usage

In the file where you configure planner:

```
(require 'planner-appt)
```

then one of the following:

- for task-based appointments: `(planner-appt-use-tasks)`
- for schedule-based appointments: `(planner-appt-use-schedule)`
- for both task- and schedule-based appointments: `(planner-appt-use-tasks-and-schedule)`

and finally if you want everything to be updated automatically add:

```
(planner-appt-insinuate)
```

If you don't want to do the insinuation then you can call:

```
M-x planner-appt-update
```

after editing appointments on the page (note that this is not necessary if you use tasks for the appointments and you don't edit the task descriptions outside of `planner-edit-task-description`).

Try both methods; if you find that you prefer one over the other, use one of the specific `planner-appt-use-` functions, as there are some performance gains when using one method exclusively.

7.2.2.2 Task-based Appointments

A task has an appointment if it looks like this:

```
#A   _ @12:45 Do something (TaskPool)
```

i.e., if it has `@` followed by a time at the beginning. This means the task is a regular appointment, and will not be carried forward at the start of a new day.

Alternatively, it may have a !, like this:

```
#A   _ !12:45 Do something else (TaskPool)
```

This makes it a "nagging" appointment, which *will* be carried forward. It will, however, lose the appointment time in the process.

This may seem like a strange feature, but here is Henrik's reasoning:

> Sometimes I have a task that I want to do at a certain time, so I make it an appointment. If I don't get around to doing it anyway, I want it to be carried forward. Basically, I sometimes use appointments on tasks to annoy me until I get them done. :)

You can edit, move and delete tasks with the usual functions, and appointments will be updated automatically.

You can update all task appointments on your page with

```
M-x planner-appt-update
```

Cyclic Entries

If you have '`planner-cyclic`' loaded, entries in your cyclical tasks file such as

```
Friday #A _ @12:45 staff meeting
```

will appear every Friday and there will be an appointment alert set up (see Section 6.3.1.4 [Cyclic Tasks], page 24).

Appointments Section

You can have all task-based appointments copied to a separate section, providing an overview of your appointments.

To do this, add

```
(setq planner-appt-task-use-appointments-section-flag t)
```

to your configuration, or use *`M-x customize-variable`*.

The variable `planner-appt-task-appointments-section` is the name of the section where the appointments will be copied. By default, it is set to `"Schedule"`, which means that task appointments will be intermingled with schedule entries.

It is also a good idea to add the section you wish to use to `planner-day-page-template` in order to control where that section will appear on the page (otherwise it will appear at the top).

The format of the appointments follows that of a schedule; if you don't like the way it looks, you can write something different and set `planner-appt-format-appt-section-line-function` appropriately. See the documentation for `planner-appt-format-appt-section-line-function` for details. It should be fairly easy to see what needs to be done

if you look at the source for the default function (which is `planner-appt-format-appt-section-line`).

If the section specified in `planner-appt-task-appointments-section` is the same as the schedule section specified in `planner-appt-schedule-section` (by default `"Schedule"`), the default formatting function adds a `#` to the description so that one can visually distinguish appointments from the task list from those that have been added to the schedule.

7.2.2.5 Schedule-Based Appointments

Some scheduled tasks require reminders, others don't. In this schedule:

```
* Schedule

9:00   | 12:00 | Read Simmel's Philosophy of Money
@12:45          Do Something Else
@13:00 | 14:00 | lunch
@14:30 |       | Meet jrs to discuss his dissertation
@16:00           Test Society seminar
18:00            go home
```

those that have an `@` prefix will be added to the appointment reminder list; the others will not. The formats that are recognized are fairly flexible, as you can see from the example.

If you change your schedule, you can update the appointment list with

```
M-x planner-appt-update
```

You can also have the schedule sorted as part of the update, if you have this in your configuration:

```
(setq planner-appt-sort-schedule-on-update-flag t)
```

Cyclical Entries

You can also have cyclical schedule entries if you add

```
(planner-appt-schedule-cyclic-insinuate)
```

to your configuration.

If you put an entry in your cyclical task file like this

```
Friday @12:45 | 13:45 | Staff Meeting
```

then it will appear in your schedule every Friday, and an appointment alert will be set up.

7.2.2.7 Appt Notes

Update on Save:

If you have

```
(setq planner-appt-update-appts-on-save-flag t)
```

in your configuration, then the appointment reminders will be updated whenever you save today's plan page.

Show Current Alerts:

```
M-x planner-appt-show-alerts
```

will display all alerts currently scheduled.

Appt Calendar Integration

Not strictly part of appointment handling, but if one isn't using the diary, marking dates with plan pages seems to make sense. If you want this, add this to your configuration:

```
(planner-appt-calendar-insinuate)
```

Removing planner-appt

Since planner-appt is not fully tested, things may go wrong; if this happens and it interferes with other planner actions, you may remove planner-appt with the command

```
M-x planner-appt-de-insinuate
```

This will remove planner-appt from all hooks.

7.2.3 Timeclock

This module allows you to clock in and clock out of your projects (see section "Time Intervals" in *GNU Emacs Manual*). You can also generate reports with the `<timeclock-report>` tag. You may want to read the Section 9.1 [Keeping Track of Time], page 65 page to see how you can use planner-timeclock to produce detailed reports.

With '`planner-timeclock.el`' loaded, `planner-task-in-progress` clocks in a task. To clock out, use `planner-task-done` or `timeclock-out`.

'`planner-timeclock.el`' defines the following keybindings:

- *C-c C-i*: `planner-task-in-progress`.
- *C-c C-o*: `timeclock-out`.
- *C-c C-T C-i*: `planner-timeclock-in`. (XEmacs)
- *C-c C-T C-o*: `timeclock-out`. (XEmacs)
- *C-c C-S-t C-i*: `planner-timeclock-in`. (GNU Emacs)
- *C-c C-S-t C-o*: `timeclock-out`. (GNU Emacs)

If you use `timeclock` a lot, you may also be interested in Dryice Liu's '`planner-timeclock-summary.el`', which produces timeclock reports for planner files.

Here is a sample report:

```
Project               |     Time| Ratio| Task
PlannerMaintenance    |  0:03:58|  1.1%| Merge doc patch
                      |  0:17:09|  5.0%| Track down subdirectories
                      |  0:18:11|  5.3%| Merge planner-timeclock-summary-proj.el
                Total: |  0:39:18| 11.4%|
JapanCaseStudy        |  2:37:56| 45.6%| Prototype search result page
                      |  0:31:50|  9.2%| Update design documents
                Total: |  3:09:46| 54.8%|
ChristmasLetter       |  1:46:37| 30.8%| Write and send my Christmas letters
                Total: |  1:46:37| 30.8%|
LinuxPeople           |  0:10:29|  3.0%| Send questions for Linux in Education
                Total: |  0:10:29|  3.0%|
```

If you add `(require 'planner-timeclock-summary)` to your '`~/.emacs`', you can then use it in two ways.

- Display a temporary buffer

  Call `planner-timeclock-summary-show` and Emacs will ask you which day's summary do you want. Choose the date as anywhere else of Emacs planner, and a tempory buffer will be displayed with the timeclock summary of that day.

  To review tasks over a date range, use `planner-timeclock-summary-show-range`. You can use a regexp or a function to filter tasks by calling `planner-timeclock-summary-show-range-filter`.

- Rewrite sections of your planner

  Choose this approach if you want timeclock summary to be in their own section and you would like them to be readable in your plain text files even outside Emacs. Caveat: The timeclock summary section should already exist in your template and will be rewritten when updated. Tip: Add `planner-timeclock-summary-section` (default: '`"Timeclock"`') to your `planner-day-page-template`.

  To use, call `planner-timeclock-summary-update` in the planner day page to update the section. If you want rewriting to be automatically performed, call `planner-timeclock-summary-insinuate` in your '`.emacs`' file.

'`planner-timeclock-summary-proj.el`' produces task / time breakdowns on plan pages. Reports are of the form:

```
TASK 0 | duration
TASK 1 | duration
 TOTAL | duration.
```

To use, add `(require 'planner-timeclock-summary)` to your '`~/.emacs`'. Call `planner-timeclock-summary-proj-current` from a project page. The report is inserted at the current position in the buffer. The function `planner-timeclock-summary-proj-section` does the same but the report is inserted inside a section called '`* Report`'.

7.2.4 'schedule.el'

'`planner-schedule.el`' allows you to project task completion time. Tasks should be of the form:

```
#A0 _ 2h Do something
#B0 _ 1h30m Do something
#B0 _ 2d Do something
#B0 _ 2w Do something
#B0 _ 10s Do something

s: seconds, m: minutes, h: hours, d: days, w: weeks
```

You can get '`schedule.el`' from http:// www.newartisans.com/ johnw/ Emacs/ schedule.el

'`planner-schedule.el`' defines the following interactive functions:

`planner-schedule-show-end-project` [Function]

Display the estimated project completion time.

‘planner-schedule.el’ defines the following keybindings:

C-c RET is bound to planner-schedule-show-end-project. *C-c C-m* is also bound to planner-schedule-show-end-project.

In Xemacs, planner-schedule-show-end-project is bound to *C-c C-T c-e* and *C-c C-S-t C-e*.

‘schedule.el’ provides a single Lisp function, schedule-completion-time. It takes an Emacs time object and a quantity of seconds. It returns an Emacs time object that represents when the given number of seconds will be completed, assuming that work can only be done during work hours.

The available work hours are affected by several factors:

1. If ‘timeclock.el’ is being used, the amount of time left in the current work day (timeclock-workday-remaining) (see section “Time Intervals” in *GNU Emacs Manual*)
2. The amount of time in each work day (schedule-workday)
3. The definition of a work week (schedule-week)
4. Any holidays defined in the Emacs calendar (see section “Holidays” in *GNU Emacs Manual*)
5. Any appointments in the Emacs diary (see section “Appointments” in *GNU Emacs Manual*)

Taking all of the “block out” periods into account, schedule-completion-time will compute when the given number of seconds will be done, based on your current definitions of time available.

As an example, here’s a function which, given a list of durations in seconds, will return a list of completion times starting from the current moment:

```
(defun compute-completion-times (&rest durations)
  ‘‘Compute completion times for a list of DURATIONS (in seconds).’’
  (let ((now (current-time)))
    (mapcar
     (function
      (lambda (dura)
        (setq now (schedule-completion-time now dura))))
     durations)))
```

To call this function, do:

```
(compute-completion-times 3600 7200 3600)
```

‘schedule.el’ does not define any interactive functions, or keybindings.

7.3 Finances

Currently, Planner provides one module dedicated to tracking your finances. This module works with a program called Ledger.

7.3.1 Ledger

'planner-ledger.el' provides integration between planner and John Wiegley's ledger accounting program, which can be found at http://newartisans.com/johnw/ledger.tar.gz.

To use planner-ledger to insert a ledger balance overview and a list of pending transactions into planner day pages, make sure that your day page includes sections that match planner-ledger-balance-regexp and planner-ledger-pending-regexp. Example:

```
* Tasks

* Ledger

** Pending transactions

* Notes
```

You can manually update ledger sections with the following command:

planner-ledger-insert-maybe [Function]
: Update any ledger sections on the current page.

You can also automatically update ledger sections with the following hook:

```
(add-hook 'planner-goto-hook 'planner-ledger-insert-maybe)
```

You can create ledger entries from specially-formatted tasks using planner-ledger-add-entry-from-task. Tasks should be of the form 'payment due: payee, amount [comment]'. Example:

```
#A1 _ payment due: foobar, $1000.00 some comment here
#A2 _ payment due: baz, 1000.00
```

planner-ledger-add-entry-from-task [Function]
: Create a ledger entry based on the task at point. Task should match planner-ledger-payment-task-regexp.

planner-ledger-balance-accounts [User Option]
: List of accounts to be included or excluded from the balance overview. '+' includes all matching accounts, and '-' excludes matching accounts. See the documentation for ledger-run-ledger for more details.

planner-ledger-balance-regexp [User Option]
: Regular expression matching section for ledger balance. Do not store other data in this section, as it will be overwritten.

planner-ledger-pending-regexp [User Option]
: Regular expression matching section for ledger balance. Do not store other data in this section, as it will be overwritten.

planner-ledger-payment-task-regexp [User Option]
: Regular expression matching special ledger tasks.

7.4 Contacts and Conversations

Planner has two modules available for keeping track of contact and conversation information. The first uses the Big Brother Database (BBDB), and the second uses Emacs Relay Chat (ERC). BBDB is a full contact database. ERC is a client for chatting online on Internet Relay Chat (IRC) networks. The ERC module for Planner will help you keep track of online conversations you have if you use ERC for those conversations, but does not by itself store contact information other than the time you had the conversation, the network and channel you were on when you had it, and maybe who you had it with. If you are looking for a way to manage your full address book, then '`planner-bbdb.el`' in combination with BBDB is what you are looking for.

7.4.1 BBDB

'`planner-bbdb.el`' allows you to refer to your contacts easily from within a planner page. See Info file '`bbdb`', node '`Top`'.

'`[[bbdb://Sacha.*Chua][Sacha]]`', for example, will be linked to the blog, web or net fields of the first matching BBDB record.

'`planner-bbdb.el`' does not define any interactive functions, or keybindings.

7.4.2 Emacs Relay Chat

To use planner-erc, place '`planner-erc.el`' in your load path and add this to your '`.emacs`' (or '`_emacs`'):

```
(require 'planner-erc)
```

ERC URLs are of the form, '`erc://server/nick/channel`', '`erc://server/nick`' or '`erc://server/nick`'.

Annotations will be of the form:

```
[[erc://server/nick/#channel][Chat with nick on server#channel]]
[[erc://server/nick][Chat with nick on server]]
[[erc://server][Chat on server]]
```

'`planner-erc.el`' does not define any interactive functions, or keybindings.

7.5 Tracking Research and Resources

Planner provides three modules for keeping track of information involving three specific tools: w3m, BibTeX, and '`bookmark.el`'.

7.5.1 W3m

This module allows you to create tasks from a w3m buffer.

'`planner-w3m.el`' does not define any interactive functions, or keybindings.

7.5.2 BibTeX

BibTeX URLs are of the form '`bibtex://file/name:key`'.

'`planner-bibtex.el`' does not define any interactive functions.

7.5.3 Bookmark

'`planner-bookmark.el`' uses the '`remember`' package to create a note whenever you create a bookmark (see See Info file '`Emacs`', node '`Bookmarks`'). For more information about '`remember`', please check out

- http://www.emacswiki.org/cgi-bin/wiki/RememberMode - EmacsWiki page
- http://sacha.free.net.ph/notebook/doc/dev/remember/remember.html - Online info documentation

Configure remember according to the instructions in '`remember-planner.el`' so that notes are saved to your planner pages.

`planner-bookmark-take-note-after-set-bookmark-flag` [User Option]
Non-nil means show a `remember` buffer after setting a new bookmark.

When you create a bookmark, Emacs will open a buffer for your notes. *C-c C-c* saves the buffer to today's page. If you don't want to save a note, you can kill the buffer.

Bookmark URLs are of the form '`bookmark://bookmark-name`'.

'`planner-bookmark.el`' does not define any interactive functions.

7.6 Tracking Development

Planner has three modules geared toward programmers. Two modules deal with version control and integrating information from those projects into the planner page. One module deals with the Gnats bug-tracking system.

7.6.1 Log Edit

This module allows you to automatically record CVS (and VC) commits in today's page.

You can load the module with `(require 'planner-log-edit)`. When you load the module, `planner-log-edit-add-note` will be added to `log-edit-done-hook`. A note containing the text of the commit and optionally a list of modified files will be added to today's page if you use the the Emacs version control interface. (see section "Version Control" in *GNU Emacs Manual*)

`planner-log-edit-include-files-flag` [User Option]
Non-nil means include a list of committed files in the note.

`planner-log-edit-notice-commit-function` [User Option]
Non-nil means include a list of committed files in the note. If you want to enable this feature for some projects but not for others, set this to a function that returns t only for the commits you want to note.

'`planner-log-edit.el`' does not define any interactive functions.

7.6.2 XTLA

This module allows you to refer to changesets in Tom Lord's Arch (tla) version control system. You can load the module with `(require 'planner-xtla)`. When you load the module, you can create tasks from XTLA windows by positioning point on a revision.

XTLA URLs are of the form '`xtla://miles@gnu.org--gnu-2005/emacs--cvs-trunk--0--patch-19`'

'`planner-xtla.el`' does not define any interactive functions.

7.6.3 Gnats

'`planner-gnats.el`' provides support for the GNU problem report management system Gnats. This module allows you to refer to problem reports using hyperlinks.

Configure your Emacs for Gnats access, then add '`(require 'planner-gnats)`' to your '`.emacs`'. You can then create tasks from Gnats edit or view buffers.

To add keybindings to Gnats, use '`(planner-gnats-insinuate)`'.

Gnats URLs are of the form '`gnats:pr-number`'.

'`planner-gnats.el`' does not define any interactive functions.

8 Advanced Configuration

8.1 Customizing Your Day Pages

With the variable `planner-day-page-template`, you can define how you want any newly created day planner pages to look.

You might want to include a section for your diary entries. For how to do this, see Section 7.2.1 [Diary], page 49.

You can add interactive Lisp buttons with the '`planner-lisp.el`' module. (see Section 6.7 [Interactive Lisp], page 40)

Your `planner-day-page-template` can also include any Emacs-Wiki tags.

For more complex day pages, you can set `planner-day-page-template` to a function that will be called from an empty day page buffer. The function should initialize the contents of the day page.

8.2 Variables to Customize

If you want to change `planner-directory` and some other variables, either use Customize (*M-x customize-group RET planner RET*) or use `planner-option-customized`. For example:

```
(planner-option-customized 'planner-directory "~/Plans")
  (planner-option-customized 'planner-publishing-directory
                             "~/public_html/plans")
```

If you want to modify other Emacs-Wiki variables, do:

```
(add-to-list 'planner-custom-variables
             '(some-emacs-wiki-variable . "some-emacs-wiki-value"))
(planner-option-customized 'planner-custom-variables
                           planner-custom-variables)
```

See `emacs-wiki-update-project` and `planner-custom-variables` for more details.

Other user options are:

`planner-use-day-pages` [User Option]
: If you really don't like day pages, you can set this variable to nil and you won't be prompted for dates for tasks (and notes, if using '`remember-planner`').

`planner-use-plan-pages` [User Option]
: If you really don't like plan pages, you can set this variable to nil and you won't be prompted for plan pages for tasks (and notes, if using '`remember-planner`').

`planner-mode-hook` [User Option]
: List of functions to run after `planner-mode` is initialized.

`planner-tasks-file-behavior` [User Option]
: This variable controls what happens to files Planner opens by itself. If your tasks are associated with plan pages, the plan pages are updated whenever a task is rescheduled. This could lead to a lot of open buffers as Planner applies updates to all linked files. By default, Planner is configured to do nothing. A value of '`save`' means save but do not close buffers, and a value of '`nil`' means do not save any of the buffers.

`planner-add-task-at-end-flag` [User Option]
: This variable controls where new tasks are created. Non-nil means create tasks at the bottom of the first task block. If you set this to non-nil, new tasks will be listed in order of creation (oldest). Tasks carried over from previous days will appear at the bottom of the list.

  Nil means create tasks at the top of the first task block. Carried-over tasks and newly created tasks are prominently placed on top of the list of tasks for the day.

`planner-default-page` [User Option]
: Default page for created tasks. This is used as the initial value for tasks. After you create a task, it will be set to the previous page used.

`planner-hide-task-status-when-highlighting` [User Option]
: Font-locking for tasks may be enough for you to determine status and priority. Set this to non-nil if you want to hide the status marker and rely on font-locking instead.

`planner-create-task-hook` [User Option]
: Functions run after creating a task. `planner-id` hooks into this.

`planner-expand-name-favor-future-p` [User Option]
: If non-nil, partial dates (ex: *2* or *5.2*) are completed to dates in the future instead of using the current year and month.

`planner-task-dates-favor-future-p` [User Option]
: Like `planner-expand-name-favor-future-p`, but only for tasks.

`planner-publish-dates-first-p` [User Option]
: Non-nil means list day pages first in the planner index.

8.3 Ideas for Other Keybindings

By default and for backward compatibility, the following operations do not have keybindings, and are only accessible from the Planner menu:

- `planner-copy-or-move-region`
- `planner-delete-task`
- `planner-task-delegated`
- `planner-task-pending`
- `planner-task-open`
- `planner-renumber-tasks`

You may find it easier to install keybindings for those operations by inserting the following in your '`.emacs`' (or '`_emacs`'). Note: This changes some of the default keybindings for Planner.

```
(planner-install-extra-task-keybindings)
```

If you install the extra task keybindings, your keybindings will include:

- *C-c C-t* will be unbound from the default and will serve as the prefix for the other task keybindings.
- *C-c C-t C-t*: `planner-create-task-from-buffer`.

- *C-c C-t C-k*: planner-delete-task.
- *C-c C-t C-u*: planner-update-task.
- *C-c C-t C-c*: planner-copy-or-move-task.
- *C-c C-t C-S-c*: planner-copy-or-move-region.
- *C-c C-t C-x*: planner-task-done.
- *C-c C-t C-S-x*: planner-task-cancelled.
- *C-c C-t C-d*: planner-task-delegated.
- *C-c C-t C-p*: planner-task-pending.
- *C-c C-t C-o*: planner-task-in-progress.
- *C-c C-t C-r*: planner-raise-task.
- *C-c C-t C-l*: planner-lower-task.
- *C-c C-t C-n*: planner-renumber-tasks.

Other keybindings can be configured by adding this to your '.emacs' (or '_emacs'):

```
(planner-install-extra-context-keybindings)
```

This will set up the following keybindings:

- *shift up* planner-move-up
- *shift down* planner-move-down
- *shift right* planner-jump-to-link

9 Reference Material

9.1 Keeping Track of Time

One of the coolest things you can do with Planner is keep track of how much time you spend not only on projects but even on particular tasks. ‘`planner-timeclock.el`’ makes it as easy and natural as marking a task as in progress, postponed, or done. This can help you determine just how much time you spend working each day. If you add estimates to your task descriptions, you’ll also be able to use this information to improve your time estimation skills.

Here’s how you can keep track of the time you

Then the fun began. I wanted to see if I could match my estimates. Before I started working on a task, I used *C-c TAB* to mark it `in progress` and start the clock. If I decided to work on something else, I used *C-c TAB* to clock out of the previous task and clock into the new one.

When I finished it, I used *C-c C-x* (`planner-task-done`) to mark it completed and automatically clock out. This is not yet done for cancelled tasks, so I clocked out of those manually with *C-c C-o* (`timeclock-out`). I also clocked out whenever I caught myself being distracted so that the totals wouldn’t include the time I spent chatting on #emacs or checking out del.icio.us links. =) At the end of the day, I used `planner-timeclock-summary-show-range-filter` to show me the time elapsed for all of the tasks I’d worked on over the past two days. Here’s the report for that project, edited to reflect how it looks on my screen and annotated with comments:

```
Timeclock summary report for 2004.12.28 - 2004.12.29

Project     |     Time| Ratio| Task
JapanProject|  0:23:17|  3.6%| Translate javadoc comment for Messages.java
            |  0:33:48|  5.3%| Translate javadoc comment for LoginAction.java
            |  1:54:07| 17.8%| Study Struts in Japanese
            |  0:46:08|  7.2%| Add javadoc tags for input, output and forwards
            |  1:03:48|  9.9%| Help review code
            |  0:04:14|  0.7%| Import todo list
            |  0:00:37|  0.1%| 2min       Fix Menu Action's unnecessary code - delegat
            |  0:01:01|  0.2%| 2min       Remove unnecessary list in UserRemoveSetupAc
            |  0:02:10|  0.3%| 2min       Remove hard-coded database path from MenuAct
            |  0:02:46|  0.4%| 30min      Create a superclass for our action classes t
            |  0:07:32|  1.2%| 5min       Add a method that returns the validity of a
            |  0:08:28|  1.3%| 5min       Fix indentation
            |  0:03:52|  0.6%| 10min      Fix UserPeer so that it doesn't get null poi
            |  0:04:34|  0.7%| 5min       Add current password field in user_modify pa
            |  0:21:56|  3.4%| 15min      Make a super class for our service classes t
            |  0:06:05|  0.9%| 10min      Remove hard-coded constants from the Logic c
            |  0:10:55|  1.7%| 10min      Move logic from UserBean.checkPassword to Us
            |  0:01:20|  0.2%| 20min      Guard against null pointer exceptions in pee
            |  0:04:57|  0.8%| 10min      Instead of displaying uneditable data with b
            |  0:25:03|  3.9%| 10min      Deploy 10:00 version
```

```
              |  0:04:46|  0.7%| 5min       Separate the configuration file of database
              |  2:09:48| 20.2%| 1h         Decide on a naming convention for localized
              |  0:00:07|  0.0%| 20min      Explain what is happening in UserModifyActio
              |  1:50:23| 17.2%| 2h         Write Javadoc comments in English and Japane
              |  0:04:19|  0.7%| 2h         Write Javadoc comments in English and Japane
              |  0:05:40|  0.9%| 20min      Make a factory class for the database - pend
        Total: | 10:41:41|100.0%|

    Day began: 13:03:58, Day ended: 20:51:46
    Time elapsed: 31:47:48, Time clocked: 10:41:41
    Time clocked ratio: 33.6%
```

The time record isn't perfect. I cancelled some tasks after thinking about them a little and did some tasks simultaneously. Sometimes I didn't notice that I was getting distracted, too. Still, having all of that time information neatly summarized made me realize a number of things.

First, I goof off much less when I have a nice, broken-down task list in front of me. There's just something about knowing there's a five- or ten-minute hack you can get out of the way. I found myself looking forward to getting to the next task just to see if I could make my estimate. That said, seeing a five-minute task stretch and stretch due to unforeseen problems did make me a little nervous. I should probably just make generous estimates so that I don't end up with bugs because of haste.

Second, I don't goof off as much as I thought I did, although there's still room for improvement. Yesterday's workday was 9:00 - 12:00, 1:00 - 5:30–7.5 hours. Today was the last day of work, so cleaning and celebration interrupted my hacking at around 3:00–5 hours of work. According to my task list, 10:41/12:30 was productive work. Hmm. 1:49 hours unclocked time when I was thinking or goofing off. planner-timeclock-summary-show for today reveals that I actually clocked 5:30 today, which means the goofing off happened yesterday. That makes sense; I remember a pretty long unclocked segment recuperating from Japanese overload. (This was before we came up with the task list.)

Third, keeping track of time is way, way cool even if you don't bill anyone for your time.

Like the idea? It's easy to try out. Just add

```
(require 'planner-timeclock)
(require 'planner-timeclock-summary)
```

to your ~/.emacs. If you want to try it out now, eval those statements in your Emacs session. After that, simply use *C-c TAB* to "clock in" a task before you start working on it and *C-c C-x* (`planner-task-done`) to mark it completed. *M-x planner-task-pending* also clocks out the current task if it was clocked in. To see a summary of how you spent your day, check out the different functions in '`planner-timeclock-summary`'.

See Section 7.2.3 [Timeclock], page 55 for more details.

Happy hacking!

9.2 Other Interactive Functions

With '`planner.el`' loaded, you can use any of the functions in this section by typing *M-x* followed by the name of the function. Many of these functions are also bound to keys.

For a list of Planner keybindings, see Section 9.3 [Planner Keybindings], page 68.

They are listed in no particular order.

'`planner.el`' defines the following interactive functions:

`planner-create-high-priority-task-from-buffer` [Function]
Create a high priority task based on this buffer. Do not use this in LISP programs. Instead, set the value of *planner-default-task-priority* and call `planner-create-task` or `planner-create-task-from-buffer`.

`defun` *planner-create-medium-priority-task-from-buffer* [Function]
Create a medium-priority task based on this buffer. Do not use this in LISP programs. Instead, set the value of *planner-default-task-priority* and call `planner-create-task` or `planner-create-task-from-buffer`.

`planner-create-low-priority-task-from-buffer` [Function]
Create a high-priority task based on this buffer. Do not use this in LISP programs. Instead, set the value of *planner-default-task-priority* and call `planner-create-task` or `planner-create-task-from-buffer`.

`planner-install-extra-context-keybindings` [Function]
Install extra context-sensitive keybindings. These keybindings conflict with '`windmove.el`', but might be useful.

`planner-narrow-to-section` *section* **&optional** *create* [Function]
Widen to the whole page and narrow to the section labelled *section*. If *create* is non-nil and the section is not found, the section is created. Return non-nil if *section* was found or created.

`planner-save-buffers` [Function]
Save all planner-mode buffers.

`planner-seek-to-first` *section* [Function]
Positions the point at the specified *section*, or '`Tasks`' if not specified.

`planner-save-buffers` [Function]
Save all planner buffers.

`planner-calendar-insinuate` [Function]
This hooks Planner into Emacs Calendar (see section "Calendar/Diary" in *GNU Emacs Manual*).

It adds special planner key bindings to `calendar-mode-map`. After this function is evaluated, you can use the following planner-related keybindings in `calendar-mode-map`:

n Jump to the planner page for the current day.

N Display the planner page for the current day.

`planner-kill-calendar-files` [Function]
Remove planner files shown from Calendar (see section "Calendar/Diary" in *GNU Emacs Manual*).

`planner-calendar-goto` [Function]
Goto the plan page corresponding to the calendar date (see section "Calendar/Diary" in *GNU Emacs Manual*).

`planner-calendar-show` [Function]
Show the plan page for the calendar date under point in another window (see section "Calendar/Diary" in *GNU Emacs Manual*).

`planner-calendar-select` [Function]
Return to `planner-read-date` with the date currently selected (see section "Calendar/Diary" in *GNU Emacs Manual*).

`planner-jump-to-link` [Function]
Jump to the item linked to by the current item.

`planner-move-up` [Function]
Move a task up. You can use this to indicate that you will do a task before another one. On a note, go to the previous note. On a headline, go to the previous headline of the same depth.

`planner-move-down` [Function]
Move down. You can use this to indicate that you will do a task after another one. On a note, go to the next note. On a headline, go to the next headline of the same depth.

9.3 Planner Keybindings

In order to refresh and renumber all of your tasks according to their actual order in the buffer, simply save the file or call *`M-x planner-renumber-tasks`*.

Here is a summary of the keystrokes available:

`M-x plan` Begin your planning session. This goes to the last day for which there is any planning info (or today if none), allowing you to review, and create/move tasks from that day.

`C-c C-u` Move a task up.

`C-c C-d` Move a task down.

`C-c C-s` Mark the task as in progress or delegated.

`C-c C-x` Mark the task as finished.

`C-c C-t` Create a task associated with the current Wiki page. If you are on the opening line of a Note entry, it is assumed that the note itself is the origin of the task.

`C-c C-c` Move or copy the current task to another date. If the current task is an original (meaning you are in the buffer where's defined, hopefully a planning page) then it will be copied, and the original task will also now point to the copy. If the current task is a copy, it will just be moved to the new day, and the original task's link will be updated.

`C-c C-n` Jump to today's task page. If you call `(planner-calendar-insinuate)`, typing `n` in the Emacs calendar (see section "Calendar/Diary" in *GNU Emacs Manual*) will jump to today's task page.

`C-c C-x` `planner-task-done`

`C-c C-z` `planner-task-in-progress`

`C-c C-d` `planner-lower-task`

`C-c C-u` `planner-raise-task`

`C-c C-c` `planner-copy-or-move-task`

`C-c C-t` `planner-create-task-from-buffer`

`C-c C-j` This is a prefix command.

`C-c C-n` `planner-goto-today`

`C-c C-j C-r`
`planner-goto-most-recent`

`C-c C-j C-t`
`planner-goto-tomorrow`

`C-c C-j C-y`
`planner-goto-yesterday`

`C-c C-j C-j`
`planner-goto-today`

`C-c C-j C-n`
`planner-goto-next-daily-page`

`C-c C-j C-p`
`planner-goto-previous-daily-page`

`C-c C-j C-d`
`planner-goto`

9.4 Sample Configuration Files

This section includes some sample configuration files. This way, once you've got the hang of the basics, you can see some different, more advanced, setups.

There is no One True Way to plan. Every person is different. We hope you'll find a good starting point among the example configurations below. If what you want to do does not perfectly fit under one of these examples, please post a description of the way you plan to the emacs-wiki-discuss mailing list. We look forward to helping you customizing planner to fit your needs.

9.4.1 File Organization

- **Tasks, schedule and notes on day pages.**

  By default, tasks, schedule entries and notes are filed on day pages. This makes it easy for you to see all the entries relevant to a single day without becoming overwhelmed

with information. Unfinished tasks are carried over to the next day when you use *M-x plan*, so it's always kept up to date. Completed tasks are left on the day page you finished them on, which helps when reviewing one's progress and writing accomplishment reports.

- **Cross-referenced with plan pages.**

  You can associate your tasks with projects either when you create the task or later, with *M-x planner-replan-task*. This makes it easy for you to see all the information associated with a particular project. If you use RememberMode to create notes, you will also be able to associate notes with a plan page.

- **Just plan pages.**

  If your tasks don't usually have dates, you can turn day pages off by customizing `planner-use-day-pages`. If so, then all of your tasks and notes will be stored on the WelcomePage and/or a plan page.

9.4.2 Bare-Bones Planning

You can keep all of your tasks, notes and schedules in a single file: WelcomePage. This is good for people who are used to storing all of their information in a flat text file. By storing your information in planner, you'll be able to take advantage of automatic hyperlinking to files and other resources. You can also sort your tasks by priority and status.

To set your system up for bare-bones planning, set the `planner-use-day-pages` variable to nil before loading planner. For example, you can put this in your '`~/.emacs`' (or '`_emacs`'):

```
(setq planner-use-day-pages nil)
(setq planner-default-page nil)
(require 'planner)
```

When you create a task or note, planner will not prompt you for a date. If you press ⟨RET⟩ when prompted for a plan page, it will accept the default of nil, so no other plan pages will be used. All of your data will be kept in one file, which can then be easily backed up.

You can use commands like `planner-create-task-from-buffer` to create tasks, or you can type tasks in manually. You can edit or delete anything in the page without having to update other files.

9.4.3 Bare-Bones Planning with Plan Pages

When you create a task or note, Planner.el can copy this to a plan page. Plan pages allow you to see an overview of all the data for a project.

For convenience, the `planner-create-task-from-buffer` command prompts you for a plan page when you create a task.

9.4.4 Tasks on Plan Pages with Some Day Pages

If most of your tasks are associated with plan pages but you want to schedule some tasks on day pages, you can leave day pages on (default) and then write a function that turns off day pages. For example, the following code snippet turns off day pages for task creation from buffers.

```
(require 'planner)
```

```
(defun my-planner-create-task-from-buffer ()
  "Call 'planner-create-task-from-buffer', but without dates."
  (interactive)
  (let ((planner-use-day-pages nil))
    (call-interactively 'planner-create-task-from-buffer)))
```

9.4.5 Hierarchical Tasks

You can use '`allout.el`' or other modes for outlining to support hierarchical tasks in plan pages. No special support is needed.

Tasks created by `planner-create-task-from-buffer` and `planner-create-task` are created in the '`* Tasks`' section. If `planner-add-task-at-end-flag` is non-nil, tasks are added to the end of the first task block, else they are added to the beginning. You can then copy and paste tasks into your preferred hierarchy. Blank lines delimit blocks of tasks upon which automatic sorting is performed.

You can also type in tasks manually. You may find this approach faster when you are comfortable with planner.

For example, a '`LearnPlanner`' plan page might contain the following lines:

```
* Learn how to use planner.el

** Installation

#C0 _ Decide whether you want stable or devel
#C0 _ Download the archives

** Configuration

*** Load path

#C0 _ Figure out how to add things to my load path
#C0 _ Actually add it to my load path

...
```

If you create tasks for the finest level of detail available at the moment, you can schedule them onto day pages with *C-c C-c* (`planner-copy-or-move-task`). Then you can use `planner-jump-to-link` to switch between the day page and the plan page link.

9.4.6 Sacha Chua's Configuration

```
;;; Sacha's configuration for planner.el
;; Sacha Chua <sacha@free.net.ph>

;;;_+ Loading

;; This directory contains all the latest emacs-wiki and planner files.
(add-to-list 'load-path "~/notebook/emacs/dev/planner")
(add-to-list 'load-path "~/notebook/emacs/dev/emacs-wiki")
(add-to-list 'load-path "~/notebook/emacs/dev/remember")
```

```
;; Separate file to make this config less scary
(require 'emacs-wiki-config)
(require 'remember-config)

;; Stock files
(require 'planner)
(require 'planner-accomplishments)
;;(require 'planner-appt)
;;(require 'planner-authz)
(require 'planner-bbdb)
;;(require 'planner-bibtex)
(require 'planner-bookmark)
;;(require 'planner-calendar)
(require 'planner-cyclic)
(require 'planner-deadline)
(require 'planner-diary)
(require 'planner-erc)
(require 'planner-experimental)
;;(require 'planner-export-diary)
(require 'planner-gnus)
;;(require 'planner-id)
;;(require 'planner-ledger)
(require 'planner-lisp)
;;(require 'planner-log-edit)
;;(require 'planner-mhe)
(require 'planner-multi)
(require 'planner-notes-index)
(require 'planner-report)
;;(require 'planner-rmail)
(require 'planner-rss)
(require 'planner-schedule)
;;(require 'planner-tasks-overview)
(require 'planner-timeclock)
(require 'planner-timeclock-summary)
;;(require 'planner-timeclock-summary-proj)
(require 'planner-trunk)
;;(require 'planner-unix-mail)
;;(require 'planner-vm)
(require 'planner-w3m)
;;(require 'planner-wl)
;;(require 'planner-xtla)

;;_+ Keybindings

;; This reminds me what I'm working on. C-u F9 F9 jumps to the task, too.
(global-set-key (kbd "<f9> <f9>") 'sacha/planner-what-am-i-supposed-to-be-doing)
(global-set-key (kbd "<f9> p SPC") 'planner-goto-today)
(global-set-key (kbd "<f9> P SPC") 'planner-goto)
(global-set-key (kbd "<f9> r SPC") 'remember)
(global-set-key (kbd "<f9> R SPC") 'remember-region)
(global-set-key (kbd "<f9> t SPC") 'planner-create-task-from-buffer)
(global-set-key (kbd "<f9> T SPC") 'planner-create-task)
;; I use F9 p to go to today's page, anyway.
(define-key planner-mode-map (kbd "C-c C-n") 'planner-create-note-from-task)
(define-key planner-mode-map (kbd "C-c C-e") 'planner-edit-task-description)
;; I use an after-save-hook to publish, so I can remap C-c C-p
(define-key planner-mode-map (kbd "C-c C-p") 'planner-task-pending)
```

```
;;;_+ Basic setup

(setq planner-directory "/home/sacha/notebook/plans")
(setq planner-publishing-directory "/home/sacha/public_html/notebook/wiki")
(setq planner-carry-tasks-forward t)
(setq planner-expand-name-favor-future-p nil)
(setq planner-task-dates-favor-future-p t)
(setq planner-default-task-priority "B")
(setq planner-expand-name-default ".")

;; I don't need my tasks renumbered.
(setq planner-renumber-tasks-automatically nil)
(setq planner-align-tasks-automatically nil)
(setq planner-renumber-notes-automatically nil)

;; Do not automatically add task IDs. I used to set this to non-nil,
;; but realized that I didn't edit my task descriptions that often. If
;; I want to edit a task, I can just add the task ID _before_ editing.
(setq planner-id-add-task-id-flag nil)

(setq planner-custom-variables
      '((emacs-wiki-publishing-header . "<lisp>(my-publishing-header)</lisp>")
        (emacs-wiki-publishing-footer . "<lisp>(my-publishing-footer)</lisp>")
        (emacs-wiki-publishing-file-suffix . ".php")))

;;;_+ planner-rss configuration

(setq planner-rss-base-url "http://sacha.free.net.ph/notebook/wiki/")
(setq planner-rss-category-feeds
      '(("ShortStories\\|flash" "/home/sacha/notebook/wiki/flash.rdf" "")
        ("planner" "/home/sacha/notebook/wiki/planner.rdf" "")
        ("education" "/home/sacha/notebook/wiki/education.rdf" "")
        ("cook" "/home/sacha/notebook/wiki/cook.rdf" "")
        ("emacs\\|planner" "/home/sacha/notebook/wiki/emacs.rdf" "")
        ("." "/home/sacha/public_html/notebook/wiki/blog-burn.rdf"
         "<?xml version=\"1.0\"?><rss version=\"2.0\"><channel>
<title>sachachua's blog</title>
<link>http://sacha.free.net.ph/notebook/wiki/today.php</link>
<description>Random notes</description>
</channel></rss>
")))
(setq planner-rss-feed-limits '(("." 20000 nil)))

;;;_+ Chronological notes on day pages and reverse-chronological on plan pages

(defun sacha/planner-twiddle-chronological-notes ()
  "Use chronological notes on day pages and reverse-chronological notes on plan pages.
People visit my site once a day, so chronologically-ordered notes
are easier for them to understand. People visit plan pages less
often, so new things should be closer to the top."
  (set (make-variable-buffer-local 'planner-reverse-chronological-notes)
       (not (string-match planner-date-regexp
                          (or (planner-page-name) "")))))
(add-hook 'planner-mode-hook 'sacha/planner-twiddle-chronological-notes)

;;;_+ Allowing =) smileys

;; Remove the =...= verbatim highlighter
```

```
(emacs-wiki-configure-highlighting 'emacs-wiki-highlight-markup
                                   (delete '("=[^[:blank:]=]" 61 emacs-wiki-highlight-
verbatim)
   emacs-wiki-highlight-markup))
(defadvice emacs-wiki-highlight-verbatim-tag (around sacha activate)
  "Do not do verbatim at all.")

;;;_+ Don't automatically highlight words

;; I like being able to use * and _ without worring about how they're
;; marked up in the published page
(setq emacs-wiki-publishing-markup
      (delq    ;; emphasized or literal text
       '["\\(^\\|[-[[:space:]<('`\"]\\)\\(=[^=[:space:]]\\|_[^_[:space:]]\\|\\*+[^*[:space:]]\\)"
         2 emacs-wiki-markup-word]
       emacs-wiki-publishing-markup))
(defadvice emacs-wiki-markup-word (around sacha activate)
  ;; Do nothing
  )

;;;_+ Compatibility for old pages or old code

;;; Compatibility, purely for old pages I'm too lazy to change.
;;; planner-diary is so much cooler.
(defun sacha/planner-get-diary-entries (date)
  "For DATE (yyyy.mm.dd), return a list of diary entries as a string."
  (require 'diary-lib)
  (when (string-match planner-date-regexp date)
    (let* ((diary-display-hook 'ignore)
           (entries (list-diary-entries
                     (list (string-to-number (match-string 2 date)) ; month
                           (string-to-number (match-string 3 date)) ; day
                           (string-to-number (match-string 1 date))) ; year
                     1))) ; Get entries for one day
      (if entries
          (mapconcat (lambda (item) (nth 1 item)) entries "\n")
        nil))))

(fset 'planner-get-diary-entries 'sacha/planner-get-diary-entries)

;;;_+ planner-diary

;;; Here we use planner-diary.
;(setq planner-diary-string "* ~/.diary schedule")
;(setq planner-diary-use-diary t)
;(planner-diary-insinuate)

;; Just in case?
;;(defadvice plan (after sacha)
;;  "Call `planner-diary-insert-diary'."
;;  (planner-diary-insert-diary))

(defun sacha/planner-diary-schedule-task (start end)
  "Add a diary entry for the current task from START to END."
  (interactive "MStart: \nMEnd: ")
  (save-window-excursion
    (save-excursion
      (save-restriction
```

```
          (let* ((info (planner-current-task-info))
                 (original (planner-task-description info))
                 main
                 description)
            ;; TODO: Mark the task as scheduled for a particular time
            (setq description
                  (cond
                   ((string-match "^\\(.+\\)\\s-+{{Schedule:\\([^-]+\\)-\\([^}]+\\)}}\\(.*\\)" original)
                    (setq main (match-string 1 original))
                    (save-excursion
                      (save-match-data
                        (goto-char (point-min))
                        (when (re-search-forward
                               (concat (match-string 2 original)
                                       " | "
                                       (match-string 3 original)
                                       " | "
                                       (match-string 1 original))
                               nil t)
                          (sacha/planner-diary-unschedule-entry))))
                    (concat (match-string 1 original)
                            " {{Schedule:"
                            start
                            "-"
                            end
                            "}}"
                            (match-string 4 original)))
                   ((string-match "\\(.*\\)\\(\\s-*\\)$" original)
                    (setq main (match-string 1 original))
                    (replace-match (concat " {{Schedule:" start "-" end "}}")
                                   t t original 2))))
            (planner-edit-task-description description)
            ;; Add the diary entry
            (sacha/planner-diary-add-entry
             (planner-task-date info)
             (concat start " | " end " | " main)))))))

(defun sacha/planner-diary-add-entry (date text &optional annotation)
  "Prompt for a diary entry to add to ‘diary-file’."
  (interactive
   (list
    (if (or current-prefix-arg
            (not (string-match planner-date-regexp (planner-page-name))))
        (planner-read-date)
      (planner-page-name))
    (read-string
     "Diary entry: ")))
  (save-excursion
    (save-window-excursion
      (let ((inhibit-read-only t))
        (make-diary-entry
         (concat
          (let ((cal-date (planner-filename-to-calendar-date date)))
            (calendar-date-string cal-date t t))
          " " text
          (or annotation
              (let ((annotation (run-hook-with-args-until-success
                                 'planner-annotation-functions)))
```

```
                (if annotation
                    (concat " " annotation)
                  "")))))
      (planner-goto date)
      (planner-diary-insert-diary-maybe))))

(defun sacha/planner-diary-unschedule-entry ()
  "Unschedule the current entry."
  (interactive)
  (goto-char (planner-line-beginning-position))
  (let ((id
         (if (re-search-forward "{{Tasks:\\([^}]+\\)}}"
                                (planner-line-end-position) t)
             (match-string 0)
           nil)))
    (sacha/planner-diary-delete-entry)
    (when id
      (planner-seek-to-first "Tasks")
      (re-search-forward id nil t))))

(defun sacha/planner-diary-delete-entry ()
  "Delete the current entry from `diary-file'."
  (interactive)
  (let ((cal-date (planner-filename-to-calendar-date (planner-page-name)))
        (text (buffer-substring (planner-line-beginning-position)
                                (planner-line-end-position)))
        (case-fold-search nil))
    (save-excursion
      (save-window-excursion
        (let ((inhibit-read-only t))
          (find-file diary-file)
          (save-excursion
            (save-restriction
              (widen)
              (goto-char (point-max))
              (when (re-search-backward
                     (concat "^"
                             (regexp-quote
                              (concat (calendar-date-string cal-date t t)
                                      " " text))))
                (delete-region (planner-line-beginning-position)
                               (min (1+ (planner-line-end-position))
                                    (point-max))))
              (save-buffer))))
        (planner-diary-insert-diary-maybe t)))))

(define-key planner-mode-map (kbd "C-c C-s") 'sacha/planner-diary-schedule-task)
(define-key planner-mode-map (kbd "C-c C-S-s") 'sacha/planner-diary-unschedule-entry)

;;;_+ Header and footer

;;(defvar planner-day-header-file "/home/sacha/notebook/wiki/.day.header"
;;  "The header file to include for planner day pages (ex: 2003.03.17)")
;;(defvar planner-day-footer-file "/home/sacha/notebook/wiki/.day.footer"
;;  "The footer file to include for planner day pages (ex: 2003.03.17)")
(defvar planner-header-file "/home/sacha/notebook/wiki/.header"
  "The header file to include for normal planner pages (ex: WelcomePage)")
(defvar planner-footer-file "/home/sacha/notebook/wiki/.footer"
```

```
  "The footer file to include for normal planner pages (ex: WelcomePage)")

(defvar sacha/planner-no-header-or-footer '("SideBar")
  "Pages that do not have header or footer.")

(defun my-publishing-header ()
  "Insert the header only if this file should have it."
  (cond
   ((member (emacs-wiki-page-name) sacha/planner-no-header-or-footer) "")
   ((file-readable-p planner-header-file) (ignore (insert-file-contents planner-header-
file)))
   (t "this is the default header text, if the file can't be found\n")))

(defun my-publishing-footer ()
  "Insert the footer only if this file should have it."
  (cond
   ((member (emacs-wiki-page-name) sacha/planner-no-header-or-footer) "")
   ((file-readable-p planner-footer-file)
    (ignore (insert-file-contents planner-footer-file)))
   (t "this is the default footer text, if the file can't be found\n")))

;;;_+ Emacspeak

(defadvice emacs-wiki-next-reference (after emacspeak pre act comp)
  "Provide additional feedback"
  (message "%s" (match-string 0)))
(defadvice emacs-wiki-previous-reference (after emacspeak pre act comp)
  "Provide additional feedback"
  (message "%s" (match-string 0)))

;;;_+ RSS blogging
(add-to-list 'remember-planner-append-hook 'planner-rss-add-note t)

(defadvice planner-rss-add-note (around sacha/absolute-urls activate)
  "Publish absolute URLs."
  (let ((sacha/emacs-wiki-use-absolute-url-flag t))
    (setq ad-return-value ad-do-it)))

(defadvice planner-rss-add-note (around sacha/norss activate)
  "Do not publish if note includes \"norss\""
  (save-restriction
    (when (planner-narrow-to-note)
      (goto-char (point-min))
      (unless (search-forward "norss" nil t)
        ad-do-it))))

(defun sacha/rss-delete-item ()
  (interactive)
  (delete-region
   (if (looking-at "<item>")
       (point)
     (when (re-search-backward "<item>" nil t)
       (match-beginning 0)))
   (when (re-search-forward "</item>" nil t)
     (match-end 0))))

(defun sacha/planner-rss-undo-this-note ()
  "Delete the current entry from the RDFs it matched."
```

```
  (interactive)
  (save-excursion
    (save-restriction
      (planner-narrow-to-note)
      (let* ((feeds planner-rss-category-feeds)
             (info (planner-current-note-info))
             (page
              (concat "<link>"
                      planner-rss-base-url
                      (emacs-wiki-published-name (planner-note-page info))
                      "#"
                      (planner-note-anchor info)
                      "</link>"))
             files)
        (while feeds
          (goto-char (point-min))
          (let ((criterion (car (car feeds)))
                (file (car (cdr (car feeds)))))
            (if (if (functionp criterion)
                    (funcall criterion)
                  (re-search-forward criterion nil t))
                (add-to-list 'files file))
            (setq feeds (cdr feeds))))
        (while files
          (with-current-buffer (find-file-noselect (car files))
            (goto-char (point-min))
            (when (re-search-forward page nil t)
              (sacha/rss-delete-item)
              (save-buffer)))
          (setq files (cdr files)))))))


;;;_+ Misc

(defun sacha/planner-replan-region (beg end &optional page)
  "Replan all tasks from BEG to END to PAGE."
  (interactive (list (point) (mark)
                     (planner-read-name (planner-file-alist) "Replan to: ")))
  (let ((start (if (< beg end) beg end))
        (finish (if (< beg end) end beg)))
    ;; Invoke planner-copy-or-move-task on each line in reverse
    (save-excursion
      (save-restriction
        (narrow-to-region
         (and (goto-char start) (planner-line-beginning-position))
         (and (goto-char (1- finish)) (min (point-max)
                                           (1+ (planner-line-end-position)))))
        (goto-char (point-min))
        (while (not (eobp))
  (save-excursion (save-restriction (planner-replan-task page)))
  (forward-line 1))))))

;;;_+ 20040504: Relative annotations
(setq planner-annotation-use-relative-file
      (lambda (filename)
        "Use relative filename if FILENAME is under my home directory."
        (save-match-data
          (or (string-match "^/home/sacha" filename)
```

```
              (string-match "^/mnt/data/home/sacha" filename)
              (string-match "^/mnt/media/home/sacha" filename)))))

;;;_+ Permalinks

;; I want notes preceded by a number so I know how to link to them.
(defun sacha/planner-markup-note ()
  "Replace note with marked-up span."
  (let ((id (match-string 1))
        (val (match-string 1)))
    (replace-match
     (save-match-data
       (format "#%s\n** %s " id
               (planner-make-link
                (concat (emacs-wiki-page-name)
                        "#" val)
                (concat val "."))))))))
(defalias 'planner-markup-note 'sacha/planner-markup-note)

;;_+ Schedule next undated task from same project
;; For Jody Klymak
(defun sacha/planner-seek-next-unfinished-and-undated-task ()
  "Move point to the next unfinished task on this page.
Return nil if not found, the task if otherwise."
  (interactive)
  (let (task-info)
    (while (and (not task-info)
                (re-search-forward "^#[A-C][0-9]*\\s-+[^CX]\\s-+" nil t))
      (setq task-info (planner-current-task-info))
      (when (planner-task-date task-info) (setq task-info nil)))
    task-info))

(defun sacha/planner-queue-next-task (&optional task-info)
  "Schedule the next task for TASK-INFO or the current task for today."
  (interactive)
  (save-window-excursion
    (save-excursion
      (setq task-info (or task-info (planner-current-task-info)))
      (when (and task-info (planner-task-plan task-info))
        (planner-find-file (planner-task-plan task-info))
        (goto-char (point-min))
        (if (sacha/planner-seek-next-unfinished-and-undated-task)
            (planner-copy-or-move-task (planner-today))
          (message "No more unscheduled tasks for %s."
                   (planner-task-plan task-info)))))))

(defadvice planner-task-done (after sacha activate)
  "Schedule next task if there are no other unfinished tasks for this project."
  (let ((task-info (planner-current-task-info))
        (not-seen t))
    (when (and task-info
               (planner-task-plan task-info)
               (planner-task-date task-info))
    (save-window-excursion
      (save-excursion
        (when (string= (planner-task-plan task-info)
                       (planner-task-page task-info))
          (planner-jump-to-linked-task))
```

```
        (goto-char (point-min))
        (while (and not-seen
                    (re-search-forward "^#[A-C][0-9]*\\s-+[^CX]\\s-+" nil t))
          (let ((current (planner-current-task-info)))
            (when (string= (planner-task-plan task-info)
                           (planner-task-plan current))
              (setq not-seen nil))))))
    (when not-seen
      (sacha/planner-queue-next-task task-info)))))

;;;_+ Keep track of what I'm supposed to be doing

;; I've bound sacha/planner-what-am-i-supposed-to-be-doing to F9 F9. I
;; start out by clocking into the task (use planner-timeclock.el and
;; C-c TAB to mark a task as in progress). Then, when I find myself
;; getting distracted, I hit F9 F9 to see my current task in the
;; minibuffer. C-u F9 F9 jumps back to the task so that I can either
;; mark it as postponed. M-x planner-task-pending (bound to C-c C-p in
;; my local config) and M-x planner-task-done (C-c C-x) both clock out
;; of the task. If I want to jump back to the previous window
;; configuration from that planner page, I can just hit F9 F9 again.

(defvar sacha/window-register "w"
  "Register for jumping back and forth between planner and wherever I am.")
(defvar sacha/planner-current-task nil
  "Current task info.")
(defadvice planner-task-in-progress (after sacha activate)
  "Keep track of the task info."
  (setq sacha/planner-current-task (planner-current-task-info)))

(defun sacha/planner-what-am-i-supposed-to-be-doing (&optional prefix)
  "Make it easy to keep track of what I'm supposed to be working on.
If PREFIX is non-nil, jump to the current task, else display it
in a message. If called from the plan page, jump back to whatever
I was looking at."
  (interactive "P")
  (if planner-timeclock-current-task
      (if (string= (planner-task-page sacha/planner-current-task)
                   (planner-page-name))
          (jump-to-register sacha/window-register)
        (if (null prefix)
            (message "%s" planner-timeclock-current-task)
          (frame-configuration-to-register sacha/window-register)
          (planner-find-file (planner-task-page sacha/planner-current-task))
          (planner-find-task sacha/planner-current-task)))
    (if prefix
        (planner-goto-today)
      (message "No current task. HEY!"))))

;;;_+ Removing task numbers

(defun sacha/planner-strip-task-numbers ()
  (interactive)
  (while (re-search-forward "^#.\\([0-9]+\\)\\s-+.\\s-+" nil t)
    (replace-match "" t t nil 1))
  (planner-align-tasks))

;;;_+ Marking up IDs as images
```

```
(defun planner-id-image (id)
  "Return the image to mark up ID as, or nil if none."
  (save-match-data (when (string-match "Tasks" id) "~/notebook/pics/screen/id-small.png")))

(defun planner-id-highlight-images (beg end &optional verbose)
  "Highlight IDs as pictures from BEG to END.
VERBOSE is ignored."
  (goto-char beg)
  (while (re-search-forward "{{[^}]+}}" end t)
    (let ((image (planner-id-image (match-string 0))))
      (when image
        (emacs-wiki-inline-image (match-beginning 0)
                                 (match-end 0)
                                 image
                                 (match-string 0))))))

(add-hook 'planner-mode-hook
          (lambda () (add-hook 'emacs-wiki-highlight-buffer-hook
                               'planner-id-highlight-images)))

;;;_+ Fancy task sorting: idea and base code from johnsu01 on 2005.02.18.

;; This code allows you to sort your tasks based on regular expressions.
;; Try it out with
;;
;;    C-u M-x sacha/planner-score-sort-tasks RET some-regexp-matching-tasks-to-be-raised RET
;;
;; If you like the effects and want to keep a whole bunch of sorting
;; rules so that you can call M-x sacha/planner-score-sort-tasks
;; without any arguments, modify the sacha/planner-score-rules
;; variable.
;;
;; If you want this to become your default sorting algorithm,
;; (setq planner-sort-tasks-key-function 'sacha/planner-score-tasks-key)
;;
;; If you want it to trigger only on some pages but not on others, see
;; the `planner-sort-tasks-basic' function for inspiration.
;;
;; I hope this code shows how easy it is to tweak task sorting. =)
;; It's also handy for quickly pulling up certain tasks, as the regular
;; M-x planner-sort-tasks will leave some semblance of the old order in.

(defvar sacha/planner-score-rules '(("patch" . 100)
                                    ("bug" . 100))
  "*Alist of planner scoring rules of the form (regexp . score-value).
Tasks with higher scores are listed first.")

(defun sacha/planner-score-tasks-key ()
  "Sort tasks by the rules in `sacha/planner-score-rules'."
  (let ((score 0)
        (case-fold-search t)
        (line (buffer-substring-no-properties (planner-line-beginning-position)
                                              (planner-line-end-position))))
    (mapc
     (lambda (item)
       (when (string-match (car item) line)
         (setq score (- score (cdr item)))))
```

```
     sacha/planner-score-rules)
    score))

(defun sacha/planner-score-sort-tasks (&optional new-rule)
  "Sort tasks by ‘sacha/planner-score-rules’ or NEW-RULE.
If called interactively, prompt for NEW-RULE. If NEW-RULE is
non-nil, tasks matching that regexp are raised. If not, tasks are
sorted according to ‘sacha/planner-score-rules’."
  (interactive (list (if current-prefix-arg (read-string "Task regexp: "))))
  (let ((planner-sort-tasks-key-function 'sacha/planner-score-tasks-key)
        (sacha/planner-score-rules
         (if new-rule
             (list (cons new-rule 1))
           sacha/planner-score-rules)))
    (planner-sort-tasks)))

;;;_+ 2005.03.14 Don't resolve e-mail addresses

(defun sacha/planner-bbdb-resolve-url (id)
  "Replace ID with the blog, web or e-mail address of the BBDB record."
  (save-match-data
    (when (string-match "^bbdb:/+" id)
      (setq id (replace-match "" t t id)))
    (let ((record (car (bbdb-search (bbdb-records) id id id))))
      (and record
           (or (bbdb-record-getprop record 'blog)
               (bbdb-record-getprop record 'web))))))

(defalias 'planner-bbdb-resolve-url 'sacha/planner-bbdb-resolve-url)

;;;_+ 2005.03.24 Random fortunes

(defvar sacha/fortune-file "/usr/share/games/fortunes/linuxcookie" "*Base file for fortune.")
(defvar sacha/fortune-command "/usr/games/fortune" "The fortune executable.")

(defun sacha/fortune (&optional file)
  "Return a fortune as a string."
  (interactive)
  (let ((line
         (shell-command-to-string
          (concat sacha/fortune-command " " (or file sacha/fortune-file)))))
    (kill-new line)
    (message line)
    line))

(setq planner-day-page-template
      (lambda ()
        (insert
         "<notes>

* Tasks


* Notes

* Fortune

<example>
```

```
" (sacha/fortune) "
</example>
")))

;;;_+ No more line-breaking for tasks. Thanks to Keith Amidon

(add-hook 'planner-mode-hook
          (lambda ()
            (setq auto-fill-inhibit-regexp "^#[ABC] +[_oX].*")
            (setq truncate-lines t)))

;;;_+ End

(provide 'planner-config)

;;;_* Local emacs vars.
;;;Local variables:
;;;allout-layout: (* 0 : )
;;;End:

;;; planner-config.el ends here
```

10 Getting Help

After you have read this guide, if you still have questions about Planner, or if you have bugs to report, there are several places you can go.

You can join the mailing list at emacs-wiki-discuss@nongnu.org using the subscription form at http:// mail.nongnu.org/ mailman/ listinfo/ emacs-wiki-discuss. This mailing list is also available via Gmane (http://gmane.org/). The group is called 'gmane.emacs.wiki.general'.

You can also contact the maintainer of Planner, Sacha Chua, at sacha@free.net.ph.

You can explore the relevant sections of the EmacsWiki:

- http://www.emacswiki.org/cgi-bin/wiki/PlannerMode
- http://www.emacswiki.org/cgi-bin/wiki/EmacsWikiMode
- http://www.emacswiki.org/cgi-bin/wiki/RememberMode

You can visit the IRC Freenode channel '#emacs'. Many of the contributors are frequently around and willing to answer your questions.

There is an Orkut community called PlannerMode.

For issues relating to this documentation, please contact John Sullivan at johnsu01@yahoo.com.

11 Acknowledgements

- Maintainers
  - 2004

    Damien Elmes handed EmacsWikiMode to Mark Triggs for a short period of time. Mark Triggs deferred to Sacha Chua as official maintainer of Planner. Sacha Chua volunteered to maintain RememberMode. Michael Olson became EmacsWikiMode maintainer.
  - 2003

    Sacha Chua volunteered to maintain Planner. Damien Elmes volunteered to maintain EmacsWikiMode.
  - 2001

    John Wiegley wrote EmacsWikiMode and Planner.
- Contributors

  This is an incomplete list of people who have helped out with Planner:
  - John Wiegley (johnw AT gnu.org)
  - Sacha Chua (sacha@free.net.ph)
  - Eric Belpaire (belpaire AT yahoo DOT com)
  - Damien Elmes (resolve AT repose DOT cx)
  - Jody Klymak (jklymak AT coas DOT oregonstate DOT edu): some documentation for '`planner-diary.el`', parts of planner-bibtex
  - Thomas Gehrlein (Thomas DOT Gehrlein AT t-online DOT de): planner-diary
  - David Forrest (drf5n AT mug DOT sys DOT virginia DOT edu)
  - Quasi (quasi AT kc4 DOT so-net DOT ne DOT jp)
  - Jason Lewis (jason AT dickson DOT st)
  - Oliver Krause (oik AT gmx DOT net)
  - Bob Newell (bnewell AT chungkuo DOT org)
  - Brent Goodrick (bgoodrick AT ipns DOT com)
  - Yvonne Thomson (yvonne AT thewatch DOT net): planner-wl
  - Chris Hall (hall DOT cj AT verizon DOT net): minor bugfix to the planner-publishing-markup
  - Markus Hoenicka (markus DOT hoenicka AT mhoenicka DOT de): '`tasklist.pl`'
  - David Smith (davidsmith AT acm DOT org): Lots of patches
  - Ephrem Christopher Walborn (christopherw AT bonitabaygroup DOT com)
  - Martin Morgan (mtmorgan AT moscow DOT com): Better fontlocking
  - Frederik Fouvry (fouvry AT coli DOT uni-sb DOT de): planner-unix-mail, info links and bugfix
  - John Sullivan (johnsu01 AT yahoo.com): texinfo documentation
  - Dryice Dong Liu (dryice AT liu DOT com DOT cn): planner-trunk, planner-bookmark, planner-log-edit, planner-rank, planner-timeclock-summary

- Keith Amidon (camalot AT picnicpark dot org): planner-trunk
- Henrik S. Hansen (hsh AT freecode DOT dk): planner-appt
- Jim Ottaway (j DOT ottaway AT lse DOT ac DOT uk): planner-appt
- Andrew J. Korty (ajk AT iu DOT edu): planner-authz, planner-report
- James Clarke (james AT jamesclarke DOT info): planner-bibtex
- Gary V. Vaughan (gary AT gnu DOT org): planner-calendar
- Travis B. Hartwell (nafai AT travishartwell DOT net): planner-diary
- Xin Wei Hu (huxw AT knight DOT 6test DOT edu DOT cn): planner-export-diary
- Sven Kloppenburg (kloppenburg AT informatik.tu-darmstadt.de): planner-gnus
- Magnus Henoch (mange AT freemail.hu): planner-gnus
- Mario Domgrgen (kanaldrache AT gmx.de): planner-gnus
- Will Glozer (will AT glozer DOT net): planner-ledger
- Simon Winwood (sjw AT cse DOT unsw DOT edu DOT au): planner-log-edit
- Christophe Garion (garion AT supaero DOT fr): planner-mhe
- Ranier Volz (http://www.raniervolz.de): planner-rdf
- Pascal Quesseveur (quesseveur AT abaksystemes DOT fr): planner-timeclock-summary-proj
- Jrgen Doser: planner-vm
- Stefan Reichr (stefan AT xsteve DOT at): planner-psvn, planner-xtla

These are people whose work contributed specifically to Emacs-Wiki Mode:

- Alex Schroeder (alex AT gnu DOT org), current author of '`wiki.el`'. His latest version is here: `http://www.geocities.com/kensanata/wiki/WikiMode.html`.
- Frank Gerhardt (Frank.Gerhardt AT web DOT de), author of the original Wiki-Mode. His latest version is here: `http://www.s.netic.de/fg/wiki-mode/wiki.el`.
- Thomas Link (t.link AT gmx DOT at).

Appendix A GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

A.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

A.3 Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index

N

O

P

R

S

T

U

V

W

X