

```

\documentclass[twoside,a4paper,openright,11pt]book \usepackage[latin1]inputenc
\usepackagepagestyle \usepackageepsfig \usepackagepixmap \inputgnusconfig.tex

\ifx\pdfoutput\undefined \else \usepackage[pdftex,bookmarks,colorlinks=true]hyperref
\usepackagehthmbpdf \pdfcompresslevel=9 \fi

\makeindex \begindocument

\newcommand\gnusversionnameGnus v5.10.6 \newcommand\gnuschaptername \new-
command\gnussectionname

\newcommand\gnusbackslash/

\newcommand\gnusref[1]“#1” on page \pageref#1 \ifx\pdfoutput\undefined \new-
command\gnusuref[1]\gnustt#1 \else \newcommand\gnusuref[1]\href#1\gnustt#1 \fi
\newcommand\gnusxref[1]See “#1” on page \pageref#1 \newcommand\gnuspxref[1]see
“#1” on page \pageref#1

\newcommand\gnusindex[1]\index#1 \newcommand\gnusindex[1]\index#1

\newcommand\gnustt[1]\gnusselectttfont#1 \newcommand\gnuscode[1]\gnustt#1
\newcommand\gnusasis[1]\gnustt#1 \newcommand\gnusurl[1]\gnustt#1 \new-
command\gnuscommand[1]\gnustt#1 \newcommand\gnusenv[1]\gnustt#1
\newcommand\gnussamp[1]“\fontencodingOT1\gnusselectttfont#1” \new-
command\gnuslisp[1]\gnustt#1 \newcommand\gnuskbd[1]‘\gnustt#1’ \new-
command\gnuskey[1]‘\gnustt#1’ \newcommand\gnusfile[1]‘\gnustt#1’ \new-
command\gnusdfn[1]\textit#1 \newcommand\gnusi[1]\textit#1 \newcom-
mand\gnusr[1]\textrm#1 \newcommand\gnusstrong[1]\textbf#1 \newcom-
mand\gnusemph[1]\textit#1 \newcommand\gnusvar[1]\fontsize10pt10\selectfont\textsl\textsf#1
\newcommand\gnussc[1]\textsc#1 \newcommand\gnustitle[1]\huge\textbf#1 \new-
command\gnusversion[1]\small\textit#1 \newcommand\gnusauthor[1]\large\textbf#1
\newcommand\gnusresult[1]\gnustt=> #1 \newcommand\gnusacronym[1]\textsc#1
\newcommand\gnusemail[1]\textit#1

\newcommand\gnusbulet$\bullet$ \newcommand\gnusdollar\$ \newcom-
mand\gnusampersand\& \newcommand\gnuspercent\% \newcommand\gnushash\#
\newcommand\gnushat\symbol"5E \newcommand\gnusunderline\symbol"5F
\newcommand\gnusnot$\neg$ \newcommand\gnustilde\symbol"7E \newcom-
mand\gnusless$<$ \newcommand\gnusgreater$>$ \newcommand\gnusbraceleft$>$
\newcommand\gnusbraceright$>$

\newcommand\gnushead\raisebox-1cm\epsfigfigure=ps/gnus-head,height=1cm
\newcommand\gnusinteresting \marginpar[\mbox\hfill\gnushead]\gnushead

\newcommand\gnuscleardoublepage\ifodd\count0\mbox\clearpage\thispagestyleempty\mbox\clearpage\

\newcommand\gnuspagechapter[1] \mbox

\newdimen\gnusdimen \gnusdimen 0pt

\newcommand\gnuschapter[2] \gnuscleardoublepage \ifdim \gnusdi-
men = 0pt\setcounterpage1\pagestylegnus\pagenumberingarabic \gnus-
dimen 1pt\fi \chapter#2 \renewcommand\gnussectionname \renewcom-
mand\gnuschaptername#2 \thispagestyleempty \hspace*-2cm \beginpicture(500,500)(0,0)
\put(480,350)\makebox(0,0)[tr]#1 \put(40,300)\makebox(500,50)[bl]\Huge\bf#2 \end-
picture \clearpage

```

```

\newcommand\gnusfigure[3] \beginfigure \mbox\ifodd\count0\hspace*-
0.8cm\else\hspace*-3cm\fi\beginpicture(440,#2) #3 \endpicture \caption#1
\endfigure

\newcommand\gnusicon[1] \marginpar[\mbox\hfill\raisebox-1.5cm\epsfigfigure=ps/#1-
up,height=1.5cm]\raisebox-1cm\epsfigfigure=ps/#1-up,height=1cm

\newcommand\gnuspicon[1] \marginindex\epsfigfigure=#1,width=2cm

\newcommand\gnusxface[2] \marginindex\epsfigfigure=#1,width=1cm\epsfigfigure=#2,width=1cm

\newcommand\gnussmiley[2] \marginindex\makebox[2cm]\hfill\epsfigfigure=#1,width=0.5cm\hfill\epsfigfigure=#2,width=0.5cm

\newcommand\gnusitemx[1]\mbox\vspace*-\itemsep\vspace*\parsep\item#1

\newcommand\gnussection[1] \renewcommand\gnussectionname#1 \section#1

\newenvironmentcodelist% \beginlist \endlist
\newenvironmentasislist% \beginlist \endlist
\newenvironmentkbdlist% \beginlist \labelwidth=0cm \endlist
\newenvironmentdfnlist% \beginlist \endlist
\newenvironmentstronglist% \beginlist \endlist
\newenvironmentampllist% \beginlist \endlist
\newenvironmentvarlist% \beginlist \endlist
\newenvironmentemphlist% \beginlist \endlist

\newlength\gnusheadtextwidth \setlength\gnusheadtextwidth\headtextwidth
\addtolength\gnusheadtextwidth1cm

\newpagestylegnus preamble% \ifodd\count0 \hspace*-0.23cm\underline\makebox[\gnusheadtextwidth]\textbf\gnussectionname\hfill\arabicpage\hfill\gnuschaptername
\else \hspace*-3.25cm\underline\makebox[\gnusheadtextwidth]\textbf\romanpage\hfill\mbox\gnusbiglogo,height=1cm \else \raisebox-0.5cm\epsfigfigure=ps/gnus-big-logo,height=1cm \else \raisebox-0.5cm\epsfigfigure=ps/gnus-big-logo,height=1cm \hfill\mbox \fi

\newpagestylegnus index% \ifodd\count0 \hspace*-0.23cm\underline\makebox[\gnusheadtextwidth]\textbf\gnussectionname\hfill\arabicpage\hfill\gnuschaptername
\else \hspace*-3.25cm\underline\makebox[\gnusheadtextwidth]\textbf\arabicpage\hfill\gnuschaptername\hfill\mbox\gnusbiglogo,height=1cm \else \raisebox-0.5cm\epsfigfigure=ps/gnus-big-logo,height=1cm \hfill\mbox \fi

\newpagestylegnus% \ifodd\count0 \makebox[12cm]\hspace*3.1cm\underline\makebox[\gnusheadtextwidth]\textbf\gnussectionname\hfill\arabicpage \else \makebox[12cm]\hspace*-2.95cm\underline\makebox[\gnusheadtextwidth]\textbf\arabicpage\hfill\gnuschaptername\hfill\mbox\gnusbiglogo,height=1cm \else \raisebox-0.5cm\epsfigfigure=ps/gnus-big-logo,height=1cm \hfill\mbox \fi

\pagenumberingroman \pagestylegnus preamble
\begin titlepage

%\addtolength\oddsidemargin-5cm %\addtolength\evensidemargin-5cm
\parindent=0cm \addtolength\textheight2cm

```

```
\gnustitle\gnustitlename\hfill\gnusversion\gnusversionname\\
\fill \hspace*0cm\epsfigfigure=ps/gnus-big-logo,height=15cm \fill \rule15cm1mm\\
\gnusauthorby Lars Magne Ingebrigtsen \newpage
\mbox \vfill
\thispagestyleempty
\newpage \endtitlepage
```

# Gnus Manual

---

by Lars Magne Ingebrigtsen

---

Copyright (c) 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being “A GNU Manual”, and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License” in the Emacs manual.

(a) The FSF’s Back-Cover Text is: “You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.”

This document is part of a collection distributed under the GNU Free Documentation License. If you want to distribute this document separately from the collection, you can do so by adding a copy of the license to the document, as described in section 6 of the license.

# The Gnus Newsreader

\tableofcontents \gnuscleardoublepage

Gnus is the advanced, self-documenting, customizable, extensible unreal-time newsreader for GNU Emacs.

Oops. That sounds oddly familiar, so let's start over again to avoid being accused of plagiarism:

Gnus is a message-reading laboratory. It will let you look at just about anything as if it were a newsgroup. You can read mail with it, you can browse directories with it, you can **ftp** with it—you can even read news with it!

Gnus tries to empower people who read news the same way Emacs empowers people who edit text. Gnus sets no limits to what the user should be allowed to do. Users are encouraged to extend Gnus to make it behave like they want it to behave. A program should not control people; people should be empowered to do what they want by using (or abusing) the program.



# 1 Starting Gnus

If your system administrator has set things up properly, starting Gnus and reading news is extremely easy—you just type *M-x gnus* in your Emacs.

If you want to start Gnus in a different frame, you can use the command *M-x gnus-other-frame* instead.

If things do not go smoothly at startup, you have to twiddle some variables in your ‘`~/.gnus.el`’ file. This file is similar to ‘`~/.emacs`’, but is read when Gnus starts.

If you puzzle at any terms used in this manual, please refer to the terminology section (see [\[Terminology\]](#), page [\[undefined\]](#)).

## 1.1 Finding the News

The `gnus-select-method` variable says where Gnus should look for news. This variable should be a list where the first element says *how* and the second element says *where*. This method is your native method. All groups not fetched with this method are foreign groups.

For instance, if the ‘`news.somewhere.edu`’ NNTP server is where you want to get your daily dosage of news from, you’d say:

```
(setq gnus-select-method '(nntp "news.somewhere.edu"))
```

If you want to read directly from the local spool, say:

```
(setq gnus-select-method '(nnspool ""))
```

If you can use a local spool, you probably should, as it will almost certainly be much faster. But do not use the local spool if your server is running Leafnode (which is a simple, standalone private news server); in this case, use `(nntp "localhost")`.

If this variable is not set, Gnus will take a look at the `NNTPSERVER` environment variable. If that variable isn’t set, Gnus will see whether `gnus-nntpserver-file` (‘`/etc/nntpserver`’ by default) has any opinions on the matter. If that fails as well, Gnus will try to use the machine running Emacs as an NNTP server. That’s a long shot, though.

If `gnus-nntp-server` is set, this variable will override `gnus-select-method`. You should therefore set `gnus-nntp-server` to `nil`, which is what it is by default.

You can also make Gnus prompt you interactively for the name of an NNTP server. If you give a non-numerical prefix to `gnus` (i.e., *C-u M-x gnus*), Gnus will let you choose between the servers in the `gnus-secondary-servers` list (if any). You can also just type in the name of any server you feel like visiting. (Note that this will set `gnus-nntp-server`, which means that if you then *M-x gnus* later in the same Emacs session, Gnus will contact the same server.)

However, if you use one NNTP server regularly and are just interested in a couple of groups from a different server, you would be better served by using the *B* command in the group buffer. It will let you have a look at what groups are available, and you can subscribe to any of the groups you want to. This also makes ‘`.newsrc`’ maintenance much tidier. See [\[Foreign Groups\]](#), page [\[undefined\]](#).

A slightly different approach to foreign groups is to set the `gnus-secondary-select-methods` variable. The select methods listed in this variable are in many ways just as native as the `gnus-select-method` server. They will also be queried for active files during startup

(if that's required), and new newsgroups that appear on these servers will be subscribed (or not) just as native groups are.

For instance, if you use the `nnmbox` back end to read your mail, you would typically set this variable to

```
(setq gnus-secondary-select-methods '((nnmbox "")))
```

## 1.2 The First Time

If no startup files exist (see [\[Startup Files\]](#), page [\[undefined\]](#)), Gnus will try to determine what groups should be subscribed by default.

If the variable `gnus-default-subscribed-newsgroups` is set, Gnus will subscribe you to just those groups in that list, leaving the rest killed. Your system administrator should have set this variable to something useful.

Since she hasn't, Gnus will just subscribe you to a few arbitrarily picked groups (i.e., `*.newusers`). (*Arbitrary* is defined here as *whatever Lars thinks you should read*.)

You'll also be subscribed to the Gnus documentation group, which should help you with most common problems.

If `gnus-default-subscribed-newsgroups` is `t`, Gnus will just use the normal functions for handling new groups, and not do anything special.

## 1.3 The Server is Down

If the default server is down, Gnus will understandably have some problems starting. However, if you have some mail groups in addition to the news groups, you may want to start Gnus anyway.

Gnus, being the trusting sort of program, will ask whether to proceed without a native select method if that server can't be contacted. This will happen whether the server doesn't actually exist (i.e., you have given the wrong address) or the server has just momentarily taken ill for some reason or other. If you decide to continue and have no foreign groups, you'll find it difficult to actually do anything in the group buffer. But, hey, that's your problem. Blllrph!

If you know that the server is definitely down, or you just want to read your mail without bothering with the server at all, you can use the `gnus-no-server` command to start Gnus. That might come in handy if you're in a hurry as well. This command will not attempt to contact your primary server—instead, it will just activate all groups on level 1 and 2. (You should preferably keep no native groups on those two levels.) Also see [\[Group Levels\]](#), page [\[undefined\]](#).

## 1.4 Slave Gnusae

You might want to run more than one Emacs with more than one Gnus at the same time. If you are using different `.newsrsrc` files (e.g., if you are using the two different Gnusae to read from two different servers), that is no problem whatsoever. You just do it.

The problem appears when you want to run two Gnusae that use the same `.newsrsrc` file.



To work around that problem some, we here at the Think-Tank at the Gnus Towers have come up with a new concept: *Masters* and *slaves*. (We have applied for a patent on this concept, and have taken out a copyright on those words. If you wish to use those words in conjunction with each other, you have to send \$1 per usage instance to me. Usage of the patent (*Master/Slave Relationships In Computer Applications*) will be much more expensive, of course.)

Anyway, you start one Gnus up the normal way with *M-x gnus* (or however you do it). Each subsequent slave Gnusae should be started with *M-x gnus-slave*. These slaves won't save normal `.newsrc` files, but instead save *slave files* that contain information only on what groups have been read in the slave session. When a master Gnus starts, it will read (and delete) these slave files, incorporating all information from them. (The slave files will be read in the sequence they were created, so the latest changes will have precedence.)

Information from the slave files has, of course, precedence over the information in the normal (i.e., master) `.newsrc` file.

If the `.newsrc*` files have not been saved in the master when the slave starts, you may be prompted as to whether to read an auto-save file. If you answer “yes”, the unsaved changes to the master will be incorporated into the slave. If you answer “no”, the slave may see some messages as unread that have been read in the master.

## 1.5 New Groups

If you are satisfied that you really never want to see any new groups, you can set `gnus-check-new-newsgroups` to `nil`. This will also save you some time at startup. Even if this variable is `nil`, you can always subscribe to the new groups just by pressing *U* in the group buffer (see [\[Group Maintenance\]](#), page [\[undefined\]](#)). This variable is `ask-server` by default. If you set this variable to `always`, then Gnus will query the back ends for new groups even when you do the *g* command (see [\[Scanning New Messages\]](#), page [\[undefined\]](#)).

### 1.5.1 Checking New Groups

Gnus normally determines whether a group is new or not by comparing the list of groups from the active file(s) with the lists of subscribed and dead groups. This isn't a particularly fast method. If `gnus-check-new-newsgroups` is `ask-server`, Gnus will ask the server for new groups since the last time. This is both faster and cheaper. This also means that you can get rid of the list of killed groups altogether, so you may set `gnus-save-killed-list` to `nil`, which will save time both at startup, at exit, and all over. Saves disk space, too. Why isn't this the default, then? Unfortunately, not all servers support this command.

I bet I know what you're thinking now: How do I find out whether my server supports `ask-server`? No? Good, because I don't have a fail-safe answer. I would suggest just setting this variable to `ask-server` and see whether any new groups appear within the next few days. If any do, then it works. If none do, then it doesn't work. I could write a function to make Gnus guess whether the server supports `ask-server`, but it would just be a guess. So I won't. You could *telnet* to the server and say *HELP* and see whether it lists `'NEWGROUPS'` among the commands it understands. If it does, then it might work. (But there are servers that lists `'NEWGROUPS'` without supporting the function properly.)

This variable can also be a list of select methods. If so, Gnus will issue an `ask-server` command to each of the select methods, and subscribe them (or not) using the normal methods. This might be handy if you are monitoring a few servers for new groups. A side effect is that startup will take much longer, so you can meditate while waiting. Use the mantra “dingnusdingnusdingnus” to achieve permanent bliss.

## 1.5.2 Subscription Methods

What Gnus does when it encounters a new group is determined by the `gnus-subscribe-newsgroup-method` variable.

This variable should contain a function. This function will be called with the name of the new group as the only parameter.

Some handy pre-fab functions are:

### `gnus-subscribe-zombies`

Make all new groups zombies. This is the default. You can browse the zombies later (with `A z`) and either kill them all off properly (with `S z`), or subscribe to them (with `u`).

### `gnus-subscribe-randomly`

Subscribe all new groups in arbitrary order. This really means that all new groups will be added at “the top” of the group buffer.

### `gnus-subscribe-alphabetically`

Subscribe all new groups in alphabetical order.

### `gnus-subscribe-hierarchically`

Subscribe all new groups hierarchically. The difference between this function and `gnus-subscribe-alphabetically` is slight. `gnus-subscribe-alphabetically` will subscribe new groups in a strictly alphabetical fashion, while this function will enter groups into its hierarchy. So if you want to have the ‘rec’ hierarchy before the ‘comp’ hierarchy, this function will not mess that configuration up. Or something like that.

### `gnus-subscribe-interactively`

Subscribe new groups interactively. This means that Gnus will ask you about **all** new groups. The groups you choose to subscribe to will be subscribed hierarchically.

### `gnus-subscribe-killed`

Kill all new groups.

### `gnus-subscribe-topics`

Put the groups into the topic that has a matching `subscribe` topic parameter (see [\[Topic Parameters\]](#), page [\[undefined\]](#)). For instance, a `subscribe` topic parameter that looks like

```
"nnslashdot"
```

will mean that all groups that match that regex will be subscribed under that topic.

If no topics match the groups, the groups will be subscribed in the top-level topic.

A closely related variable is `gnus-subscribe-hierarchical-interactive`. (That's quite a mouthful.) If this variable is non-`nil`, Gnus will ask you in a hierarchical fashion whether to subscribe to new groups or not. Gnus will ask you for each sub-hierarchy whether you want to descend the hierarchy or not.

One common mistake is to set the variable a few paragraphs above (`gnus-subscribe-newsgroup-method`) to `gnus-subscribe-hierarchical-interactive`. This is an error. This will not work. This is ga-ga. So don't do it.

### 1.5.3 Filtering New Groups

A nice and portable way to control which new newsgroups should be subscribed (or ignored) is to put an *options* line at the start of the `‘.newsrc’` file. Here's an example:

```
options -n !alt.all !rec.all sci.all
```

This line obviously belongs to a serious-minded intellectual scientific person (or she may just be plain old boring), because it says that all groups that have names beginning with `‘alt’` and `‘rec’` should be ignored, and all groups with names beginning with `‘sci’` should be subscribed. Gnus will not use the normal subscription method for subscribing these groups. `gnus-subscribe-options-newsgroup-method` is used instead. This variable defaults to `gnus-subscribe-alphabetically`.

If you don't want to mess with your `‘.newsrc’` file, you can just set the two variables `gnus-options-subscribe` and `gnus-options-not-subscribe`. These two variables do exactly the same as the `‘.newsrc’` `‘options -n’` trick. Both are regexps, and if the new group matches the former, it will be unconditionally subscribed, and if it matches the latter, it will be ignored.

Yet another variable that meddles here is `gnus-auto-subscribed-groups`. It works exactly like `gnus-options-subscribe`, and is therefore really superfluous, but I thought it would be nice to have two of these. This variable is more meant for setting some ground rules, while the other variable is used more for user fiddling. By default this variable makes all new groups that come from mail back ends (`nnml`, `nnbabyl`, `nnfolder`, `nnmbox`, `nnmh`, and `nnmaildir`) subscribed. If you don't like that, just set this variable to `nil`.

New groups that match this regexp are subscribed using `gnus-subscribe-options-newsgroup-method`.

## 1.6 Changing Servers

Sometimes it is necessary to move from one NNTP server to another. This happens very rarely, but perhaps you change jobs, or one server is very flaky and you want to use another.

Changing the server is pretty easy, right? You just change `gnus-select-method` to point to the new server?

*Wrong!*

Article numbers are not (in any way) kept synchronized between different NNTP servers, and the only way Gnus keeps track of what articles you have read is by keeping track of article numbers. So when you change `gnus-select-method`, your `‘.newsrc’` file becomes worthless.

Gnus provides a few functions to attempt to translate a `‘.newsrc’` file from one server to another. They all have one thing in common—they take a looong time to run. You don't want to use these functions more than absolutely necessary.

If you have access to both servers, Gnus can request the headers for all the articles you have read and compare `Message-IDs` and map the article numbers of the read articles and article marks. The *M-x* `gnus-change-server` command will do this for all your native groups. It will prompt for the method you want to move to.

You can also move individual groups with the *M-x* `gnus-group-move-group-to-server` command. This is useful if you want to move a (foreign) group from one server to another.

If you don't have access to both the old and new server, all your marks and read ranges have become worthless. You can use the *M-x* `gnus-group-clear-data-on-native-groups` command to clear out all data that you have on your native groups. Use with caution.

Clear the data from the current group only—nix out marks and the list of read articles (`gnus-group-clear-data`).

After changing servers, you **must** move the cache hierarchy away, since the cached articles will have wrong article numbers, which will affect which articles Gnus thinks are read. `gnus-group-clear-data-on-native-groups` will ask you if you want to have it done automatically; for `gnus-group-clear-data`, you can use *M-x* `gnus-cache-move-cache` (but beware, it will move the cache for all groups).

## 1.7 Startup Files

Most common Unix news readers use a shared startup file called `‘.newsrc’`. This file contains all the information about what groups are subscribed, and which articles in these groups have been read.

Things got a bit more complicated with GNUS. In addition to keeping the `‘.newsrc’` file updated, it also used a file called `‘.newsrc.el’` for storing all the information that didn't fit into the `‘.newsrc’` file. (Actually, it also duplicated everything in the `‘.newsrc’` file.) GNUS would read whichever one of these files was the most recently saved, which enabled people to swap between GNUS and other newsreaders.

That was kinda silly, so Gnus went one better: In addition to the `‘.newsrc’` and `‘.newsrc.el’` files, Gnus also has a file called `‘.newsrc.eld’`. It will read whichever of these files that are most recent, but it will never write a `‘.newsrc.el’` file. You should never delete the `‘.newsrc.eld’` file—it contains much information not stored in the `‘.newsrc’` file.

You can turn off writing the `‘.newsrc’` file by setting `gnus-save-newsrc-file` to `nil`, which means you can delete the file and save some space, as well as exiting from Gnus faster. However, this will make it impossible to use other newsreaders than Gnus. But hey, who would want to, right? Similarly, setting `gnus-read-newsrc-file` to `nil` makes Gnus ignore the `‘.newsrc’` file and any `‘.newsrc-SERVER’` files, which can be convenient if you use a different news reader occasionally, and you want to read a different subset of the available groups with that news reader.

If `gnus-save-killed-list` (default `t`) is `nil`, Gnus will not save the list of killed groups to the startup file. This will save both time (when starting and quitting) and space (on disk). It will also mean that Gnus has no record of what groups are new or old, so the automatic new groups subscription methods become meaningless. You should always set `gnus-check-new-newsgroups` to `nil` or `ask-server` if you set this variable to `nil` (see [\(undefined\)](#) [New Groups], page [\(undefined\)](#)). This variable can also be a regular expression. If that's the case, remove all groups that do not match this regexp before saving. This

can be useful in certain obscure situations that involve several servers where not all servers support `ask-server`.

The `gnus-startup-file` variable says where the startup files are. The default value is `'~/newsrsrc'`, with the Gnus (El Dingo) startup file being whatever that one is, with a `'eld'` appended. If you want version control for this file, set `gnus-backup-startup-file`. It respects the same values as the `version-control` variable.

`gnus-save-newsrc-hook` is called before saving any of the newsrc files, while `gnus-save-quick-newsrc-hook` is called just before saving the `'newsrc.eld'` file, and `gnus-save-standard-newsrc-hook` is called just before saving the `'newsrc'` file. The latter two are commonly used to turn version control on or off. Version control is on by default when saving the startup files. If you want to turn backup creation off, say something like:

```
(defun turn-off-backup ()
  (set (make-local-variable 'backup-inhibited) t))

(add-hook 'gnus-save-quick-newsrc-hook 'turn-off-backup)
(add-hook 'gnus-save-standard-newsrc-hook 'turn-off-backup)
```

When Gnus starts, it will read the `gnus-site-init-file` (`'.../site-lisp/gnus'` by default) and `gnus-init-file` (`'~/gnus'` by default) files. These are normal Emacs Lisp files and can be used to avoid cluttering your `'~/emacs'` and `'site-init'` files with Gnus stuff. Gnus will also check for files with the same names as these, but with `'elc'` and `'el'` suffixes. In other words, if you have set `gnus-init-file` to `'~/gnus'`, it will look for `'~/gnus.elc'`, `'~/gnus.el'`, and finally `'~/gnus'` (in this order).

## 1.8 Auto Save

Whenever you do something that changes the Gnus data (reading articles, catching up, killing/subscribing groups), the change is added to a special *dribble buffer*. This buffer is auto-saved the normal Emacs way. If your Emacs should crash before you have saved the `'newsrc'` files, all changes you have made can be recovered from this file.

If Gnus detects this file at startup, it will ask the user whether to read it. The auto save file is deleted whenever the real startup file is saved.

If `gnus-use-dribble-file` is `nil`, Gnus won't create and maintain a dribble buffer. The default is `t`.

Gnus will put the dribble file(s) in `gnus-dribble-directory`. If this variable is `nil`, which it is by default, Gnus will dribble into the directory where the `'newsrc'` file is located. (This is normally the user's home directory.) The dribble file will get the same file permissions as the `'newsrc'` file.

If `gnus-always-read-dribble-file` is non-`nil`, Gnus will read the dribble file on startup without querying the user.

## 1.9 The Active File

When Gnus starts, or indeed whenever it tries to determine whether new articles have arrived, it reads the active file. This is a very large file that lists all the active groups and articles on the server.

Before examining the active file, Gnus deletes all lines that match the regexp `gnus-ignored-newsgroups`. This is done primarily to reject any groups with bogus names, but you can use this variable to make Gnus ignore hierarchies you aren't ever interested in. However, this is not recommended. In fact, it's highly discouraged. Instead, see [\(undefined\)](#) [New Groups], page [\(undefined\)](#) for an overview of other variables that can be used instead.

The active file can be rather Huge, so if you have a slow network, you can set `gnus-read-active-file` to `nil` to prevent Gnus from reading the active file. This variable is `some` by default.

Gnus will try to make do by getting information just on the groups that you actually subscribe to.

Note that if you subscribe to lots and lots of groups, setting this variable to `nil` will probably make Gnus slower, not faster. At present, having this variable `nil` will slow Gnus down considerably, unless you read news over a 2400 baud modem.

This variable can also have the value `some`. Gnus will then attempt to read active info only on the subscribed groups. On some servers this is quite fast (on sparkling, brand new INN servers that support the `LIST ACTIVE group` command), on others this isn't fast at all. In any case, `some` should be faster than `nil`, and is certainly faster than `t` over slow lines.

Some news servers (old versions of Leafnode and old versions of INN, for instance) do not support the `LIST ACTIVE group`. For these servers, `nil` is probably the most efficient value for this variable.

If this variable is `nil`, Gnus will ask for group info in total lock-step, which isn't very fast. If it is `some` and you use an NNTP server, Gnus will pump out commands as fast as it can, and read all the replies in one swoop. This will normally result in better performance, but if the server does not support the aforementioned `LIST ACTIVE group` command, this isn't very nice to the server.

If you think that starting up Gnus takes too long, try all the three different values for this variable and see what works best for you.

In any case, if you use `some` or `nil`, you should definitely kill all groups that you aren't interested in to speed things up.

Note that this variable also affects active file retrieval from secondary select methods.

## 1.10 Startup Variables

### `gnus-load-hook`

A hook run while Gnus is being loaded. Note that this hook will normally be run just once in each Emacs session, no matter how many times you start Gnus.

### `gnus-before-startup-hook`

A hook run after starting up Gnus successfully.

### `gnus-startup-hook`

A hook run as the very last thing after starting up Gnus

### `gnus-started-hook`

A hook that is run as the very last thing after starting up Gnus successfully.

**gnus-setup-news-hook**

A hook that is run after reading the ‘**.newsrc**’ file(s), but before generating the group buffer.

**gnus-check-bogus-newsgroups**

If non-**nil**, Gnus will check for and delete all bogus groups at startup. A *bogus group* is a group that you have in your ‘**.newsrc**’ file, but doesn’t exist on the news server. Checking for bogus groups can take quite a while, so to save time and resources it’s best to leave this option off, and do the checking for bogus groups once in a while from the group buffer instead (see [\[Group Maintenance\]](#), page [\[undefined\]](#)).

**gnus-inhibit-startup-message**

If non-**nil**, the startup message won’t be displayed. That way, your boss might not notice as easily that you are reading news instead of doing your job. Note that this variable is used before ‘**~/.gnus.el**’ is loaded, so it should be set in ‘**.emacs**’ instead.

**gnus-no-groups-message**

Message displayed by Gnus when no groups are available.

**gnus-play-startup-jingle**

If non-**nil**, play the Gnus jingle at startup.

**gnus-startup-jingle**

Jingle to be played if the above variable is non-**nil**. The default is ‘**Tuxedomoon.Jingle4.au**’.





## 2 Group Buffer

The *group buffer* lists all (or parts) of the available groups. It is the first buffer shown when Gnus starts, and will never be killed as long as Gnus is active.

```
\gnusfigureThe Group Buffer320 \put(75,50)\epsfigfigure=ps/group,height=9cm
\put(120,37)\makebox(0,0)[t]Buffer name \put(120,38)\vector(1,2)10 \put(40,60)\makebox(0,0)[r]Mode
line \put(40,58)\vector(1,0)30 \put(200,28)\makebox(0,0)[t]Native select method
\put(200,26)\vector(-1,2)15
```

### 2.1 Group Buffer Format

#### 2.1.1 Group Line Specification

The default format of the group buffer is nice and dull, but you can make it as exciting and ugly as you feel like.

Here's a couple of example group lines:

```
25: news.announce.newusers
* 0: alt.fan.andrea-dworkin
```

Quite simple, huh?

You can see that there are 25 unread articles in ‘`news.announce.newusers`’. There are no unread articles, but some ticked articles, in ‘`alt.fan.andrea-dworkin`’ (see that little asterisk at the beginning of the line?).

You can change that format to whatever you want by fiddling with the `gnus-group-line-format` variable. This variable works along the lines of a `format` specification, which is pretty much the same as a `printf` specifications, for those of you who use (feh!) C. See [\[Formatting Variables\]](#), page [\[undefined\]](#).

‘`%M%S%5y:%B(%g%)\n`’ is the value that produced those lines above.

There should always be a colon on the line; the cursor always moves to the colon after performing an operation. See [\[Positioning Point\]](#), page [\[undefined\]](#). Nothing else is required—not even the group name. All displayed text is just window dressing, and is never examined by Gnus. Gnus stores all real information it needs using text properties.

(Note that if you make a really strange, wonderful, spreadsheet-like layout, everybody will believe you are hard at work with the accounting instead of wasting time reading news.)

Here's a list of all available format characters:

‘M’	An asterisk if the group only has marked articles.
‘S’	Whether the group is subscribed.
‘L’	Level of subscribedness.
‘N’	Number of unread articles.
‘I’	Number of dormant articles.
‘T’	Number of ticked articles.
‘R’	Number of read articles.
‘U’	Number of unseen articles.

- ‘t’ Estimated total number of articles. (This is really *max-number* minus *min-number* plus 1.)  
Gnus uses this estimation because the NNTP protocol provides efficient access to *max-number* and *min-number* but getting the true unread message count is not possible efficiently. For hysterical raisins, even the mail back ends, where the true number of unread messages might be available efficiently, use the same limited interface. To remove this restriction from Gnus means that the back end interface has to be changed, which is not an easy job. If you want to work on this, please contact the Gnus mailing list.
- ‘y’ Number of unread, unticked, non-dormant articles.
- ‘i’ Number of ticked and dormant articles.
- ‘g’ Full group name.
- ‘G’ Group name.
- ‘C’ Group comment (see [\[Group Parameters\]](#), page [\[Group Parameters\]](#)) or group name if there is no comment element in the group parameters.
- ‘D’ Newsgroup description. You need to read the group descriptions before these will appear, and to do that, you either have to set `gnus-read-active-file` or use the group buffer *M-d* command.
- ‘o’ ‘m’ if moderated.
- ‘O’ ‘(m)’ if moderated.
- ‘s’ Select method.
- ‘B’ If the summary buffer for the group is open or not.
- ‘n’ Select from where.
- ‘z’ A string that looks like ‘<%s:%n>’ if a foreign select method is used.
- ‘P’ Indentation based on the level of the topic (see [\[Group Topics\]](#), page [\[Group Topics\]](#)).
- ‘c’ Short (collapsed) group name. The `gnus-group-uncollapsed-levels` variable says how many levels to leave at the end of the group name. The default is 1—this will mean that group names like ‘gnu.emacs.gnus’ will be shortened to ‘g.e.gnus’.
- ‘m’ ‘%’ (`gnus-new-mail-mark`) if there has arrived new mail to the group lately.
- ‘p’ ‘#’ (`gnus-process-mark`) if the group is process marked.
- ‘d’ A string that says when you last read the group (see [\[Group Time-stamp\]](#), page [\[Group Time-stamp\]](#)).
- ‘u’ User defined specifier. The next character in the format string should be a letter. Gnus will call the function `gnus-user-format-function-‘X’`, where ‘X’ is the letter following ‘%u’. The function will be passed a single dummy parameter as argument. The function should return a string, which will be inserted into the buffer just like information from any other specifier.

All the “number-of” specs will be filled with an asterisk (\*) if no info is available—for instance, if it is a non-activated foreign group, or a bogus native group.

### 2.1.2 Group Mode Line Specification

The mode line can be changed by setting `gnus-group-mode-line-format` (see [\[Mode Line Formatting\]](#), page [\[undefined\]](#)). It doesn't understand that many format specifiers:

'S'           The native news server.  
'M'           The native select method.

### 2.1.3 Group Highlighting

Highlighting in the group buffer is controlled by the `gnus-group-highlight` variable. This is an alist with elements that look like *(form . face)*. If *form* evaluates to something non-`nil`, the *face* will be used on the line.

Here's an example value for this variable that might look nice if the background is dark:

```
(cond (window-system
      (setq custom-background-mode 'light)
      (defface my-group-face-1
        '((t (:foreground "Red" :bold t))) "First group face")
      (defface my-group-face-2
        '((t (:foreground "DarkSeaGreen4" :bold t)))
        "Second group face")
      (defface my-group-face-3
        '((t (:foreground "Green4" :bold t))) "Third group face")
      (defface my-group-face-4
        '((t (:foreground "SteelBlue" :bold t))) "Fourth group face")
      (defface my-group-face-5
        '((t (:foreground "Blue" :bold t))) "Fifth group face"))

      (setq gnus-group-highlight
        '(((> unread 200) . my-group-face-1)
          ((and (< level 3) (zerop unread)) . my-group-face-2)
          ((< level 3) . my-group-face-3)
          ((zerop unread) . my-group-face-4)
          (t . my-group-face-5)))
```

Also see [\[undefined\]](#) [\[Faces and Fonts\]](#), page [\[undefined\]](#).

Variables that are dynamically bound when the forms are evaluated include:

`group`       The group name.  
`unread`     The number of unread articles in the group.  
`method`     The select method.  
`mailp`       Whether the group is a mail group.  
`level`       The level of the group.  
`score`       The score of the group.  
`ticked`      The number of ticked articles in the group.

<b>total</b>	The total number of articles in the group. Or rather, <i>max-number</i> minus <i>min-number</i> plus one.
<b>topic</b>	When using the topic minor mode, this variable is bound to the current topic being inserted.

When the forms are `eval`ed, point is at the beginning of the line of the group in question, so you can use many of the normal Gnus functions for snarfing info on the group.

`gnus-group-update-hook` is called when a group line is changed. It will not be called when `gnus-visual` is `nil`. This hook calls `gnus-group-highlight-line` by default.

## 2.2 Group Maneuvering

All movement commands understand the numeric prefix and will behave as expected, hopefully.

<i>n</i>	Go to the next group that has unread articles ( <code>gnus-group-next-unread-group</code> ).
<i>p</i> <i>DEL</i>	Go to the previous group that has unread articles ( <code>gnus-group-prev-unread-group</code> ).
<i>N</i>	Go to the next group ( <code>gnus-group-next-group</code> ).
<i>P</i>	Go to the previous group ( <code>gnus-group-prev-group</code> ).
<i>M-n</i>	Go to the next unread group on the same (or lower) level ( <code>gnus-group-next-unread-group-same-level</code> ).
<i>M-p</i>	Go to the previous unread group on the same (or lower) level ( <code>gnus-group-prev-unread-group-same-level</code> ).

Three commands for jumping to groups:

<i>j</i>	Jump to a group (and make it visible if it isn't already) ( <code>gnus-group-jump-to-group</code> ). Killed groups can be jumped to, just like living groups.
<i>,</i>	Jump to the unread group with the lowest level ( <code>gnus-group-best-unread-group</code> ).
<i>.</i>	Jump to the first group with unread articles ( <code>gnus-group-first-unread-group</code> ).

If `gnus-group-goto-unread` is `nil`, all the movement commands will move to the next group, not the next unread group. Even the commands that say they move to the next unread group. The default is `t`.

## 2.3 Selecting a Group

<i>SPACE</i>	Select the current group, switch to the summary buffer and display the first unread article ( <code>gnus-group-read-group</code> ). If there are no unread articles in the group, or if you give a non-numerical prefix to this command, Gnus will offer to fetch all the old articles in this group from the server. If you give a numerical prefix <i>n</i> , <i>n</i> determines the number of articles Gnus will fetch. If <i>n</i> is
--------------	--

positive, Gnus fetches the *n* newest articles, if *n* is negative, Gnus fetches the `abs(n)` oldest articles.

Thus, *SPC* enters the group normally, *C-u SPC* offers old articles, *C-u 4 2 SPC* fetches the 42 newest articles, and *C-u - 4 2 SPC* fetches the 42 oldest ones.

When you are in the group (in the Summary buffer), you can type *M-g* to fetch new articles, or *C-u M-g* to also show the old ones.

- RET** Select the current group and switch to the summary buffer (`gnus-group-select-group`). Takes the same arguments as `gnus-group-read-group`—the only difference is that this command does not display the first unread article automatically upon group entry.
- M-RET** This does the same as the command above, but tries to do it with the minimum amount of fuzz (`gnus-group-quick-select-group`). No scoring/killing will be performed, there will be no highlights and no expunging. This might be useful if you're in a real hurry and have to enter some humongous group. If you give a 0 prefix to this command (i.e., *0 M-RET*), Gnus won't even generate the summary buffer, which is useful if you want to toggle threading before generating the summary buffer (see [\[Summary Generation Commands\]](#), page [\(undefined\)](#)).
- M-SPACE** This is yet one more command that does the same as the *RET* command, but this one does it without expunging and hiding dormants (`gnus-group-visible-select-group`).
- C-M-RET** Finally, this command selects the current group ephemerally without doing any processing of its contents (`gnus-group-select-group-ephemerally`). Even threading has been turned off. Everything you do in the group after selecting it in this manner will have no permanent effects.

The `gnus-large-newsgroup` variable says what Gnus should consider to be a big group. If it is `nil`, no groups are considered big. The default value is 200. If the group has more (unread and/or ticked) articles than this, Gnus will query the user before entering the group. The user can then specify how many articles should be fetched from the server. If the user specifies a negative number (*-n*), the *n* oldest articles will be fetched. If it is positive, the *n* articles that have arrived most recently will be fetched.

`gnus-large-ephemeral-newsgroup` is the same as `gnus-large-newsgroup`, but is only used for ephemeral newsgroups.

If `gnus-auto-select-first` is non-`nil`, select an article automatically when entering a group with the *SPACE* command. Which article this is is controlled by the `gnus-auto-select-subject` variable. Valid values for this variable is:

- unread** Place point on the subject line of the first unread article.
- first** Place point on the subject line of the first article.
- unseen** Place point on the subject line of the first unseen article.
- unseen-or-unread** Place point on the subject line of the first unseen article, and if there is no such article, place point on the subject line of the first unread article.

**best** Place point on the subject line of the highest-scored unread article.

This variable can also be a function. In that case, that function will be called to place point on a subject line.

If you want to prevent automatic selection in some group (say, in a binary group with Huge articles) you can set the `gnus-auto-select-first` variable to `nil` in `gnus-select-group-hook`, which is called when a group is selected.

## 2.4 Subscription Commands

<i>S t</i>	
<i>u</i>	Toggle subscription to the current group ( <code>gnus-group-unsubscribe-current-group</code> ).
<i>S s</i>	
<i>U</i>	Prompt for a group to subscribe, and then subscribe it. If it was subscribed already, unsubscribe it instead ( <code>gnus-group-unsubscribe-group</code> ).
<i>S k</i>	
<i>C-k</i>	Kill the current group ( <code>gnus-group-kill-group</code> ).
<i>S y</i>	
<i>C-y</i>	Yank the last killed group ( <code>gnus-group-yank-group</code> ).
<i>C-x C-t</i>	Transpose two groups ( <code>gnus-group-transpose-groups</code> ). This isn't really a subscription command, but you can use it instead of a kill-and-yank sequence sometimes.
<i>S w</i>	
<i>C-w</i>	Kill all groups in the region ( <code>gnus-group-kill-region</code> ).
<i>S z</i>	Kill all zombie groups ( <code>gnus-group-kill-all-zombies</code> ).
<i>S C-k</i>	Kill all groups on a certain level ( <code>gnus-group-kill-level</code> ). These groups can't be yanked back after killing, so this command should be used with some caution. The only time where this command comes in really handy is when you have a <code>.newsrc</code> with lots of unsubscribed groups that you want to get rid off. <i>S C-k</i> on level 7 will kill off all unsubscribed groups that do not have message numbers in the <code>.newsrc</code> file.

Also see [\[Group Levels\]](#), page [\[Group Levels\]](#).

## 2.5 Group Data

<i>c</i>	Mark all unticked articles in this group as read ( <code>gnus-group-catchup-current</code> ). <code>gnus-group-catchup-group-hook</code> is called when catching up a group from the group buffer.
<i>C</i>	Mark all articles in this group, even the ticked ones, as read ( <code>gnus-group-catchup-current-all</code> ).
<i>M-c</i>	Clear the data from the current group—nix out marks and the list of read articles ( <code>gnus-group-clear-data</code> ).

*M-x gnus-group-clear-data-on-native-groups*

If you have switched from one NNTP server to another, all your marks and read ranges have become worthless. You can use this command to clear out all data that you have on your native groups. Use with caution.

## 2.6 Group Levels

All groups have a level of *subscribedness*. For instance, if a group is on level 2, it is more subscribed than a group on level 5. You can ask Gnus to just list groups on a given level or lower (see [\[Listing Groups\]](#), page [\[undefined\]](#)), or to just check for new articles in groups on a given level or lower (see [\[Scanning New Messages\]](#), page [\[undefined\]](#)).

Remember: The higher the level of the group, the less important it is.

**S 1** Set the level of the current group. If a numeric prefix is given, the next *n* groups will have their levels set. The user will be prompted for a level.

Gnus considers groups from levels 1 to **gnus-level-subscribed** (inclusive) (default 5) to be subscribed, **gnus-level-subscribed** (exclusive) and **gnus-level-unsubscribed** (inclusive) (default 7) to be unsubscribed, **gnus-level-zombie** to be zombies (walking dead) (default 8) and **gnus-level-killed** to be killed (completely dead) (default 9). Gnus treats subscribed and unsubscribed groups exactly the same, but zombie and killed groups have no information on what articles you have read, etc, stored. This distinction between dead and living groups isn't done because it is nice or clever, it is done purely for reasons of efficiency.

It is recommended that you keep all your mail groups (if any) on quite low levels (e.g. 1 or 2).

Maybe the following description of the default behavior of Gnus helps to understand what these levels are all about. By default, Gnus shows you subscribed nonempty groups, but by hitting **L** you can have it show empty subscribed groups and unsubscribed groups, too. Type **1** to go back to showing nonempty subscribed groups again. Thus, unsubscribed groups are hidden, in a way.

Zombie and killed groups are similar to unsubscribed groups in that they are hidden by default. But they are different from subscribed and unsubscribed groups in that Gnus doesn't ask the news server for information (number of messages, number of unread messages) on zombie and killed groups. Normally, you use **C-k** to kill the groups you aren't interested in. If most groups are killed, Gnus is faster.

Why does Gnus distinguish between zombie and killed groups? Well, when a new group arrives on the server, Gnus by default makes it a zombie group. This means that you are normally not bothered with new groups, but you can type **A z** to get a list of all new groups. Subscribe the ones you like and kill the ones you don't want. (**A k** shows a list of killed groups.)

If you want to play with the level variables, you should show some care. Set them once, and don't touch them ever again. Better yet, don't touch them at all unless you know exactly what you're doing.

Two closely related variables are **gnus-level-default-subscribed** (default 3) and **gnus-level-default-unsubscribed** (default 6), which are the levels that new groups will



be put on if they are (un)subscribed. These two variables should, of course, be inside the relevant valid ranges.

If `gnus-keep-same-level` is non-`nil`, some movement commands will only move to groups of the same level (or lower). In particular, going from the last article in one group to the next group will go to the next group of the same level (or lower). This might be handy if you want to read the most important groups before you read the rest.

If this variable is `best`, Gnus will make the next newsgroup the one with the best level.

All groups with a level less than or equal to `gnus-group-default-list-level` will be listed in the group buffer by default.

If `gnus-group-list-inactive-groups` is non-`nil`, non-active groups will be listed along with the unread groups. This variable is `t` by default. If it is `nil`, inactive groups won't be listed.

If `gnus-group-use-permanent-levels` is non-`nil`, once you give a level prefix to `g` or `l`, all subsequent commands will use this level as the “work” level.

Gnus will normally just activate (i. e., query the server about) groups on level `gnus-activate-level` or less. If you don't want to activate unsubscribed groups, for instance, you might set this variable to 5. The default is 6.

## 2.7 Group Score

You would normally keep important groups on high levels, but that scheme is somewhat restrictive. Don't you wish you could have Gnus sort the group buffer according to how often you read groups, perhaps? Within reason?

This is what *group score* is for. You can have Gnus assign a score to each group through the mechanism described below. You can then sort the group buffer based on this score. Alternatively, you can sort on score and then level. (Taken together, the level and the score is called the *rank* of the group. A group that is on level 4 and has a score of 1 has a higher rank than a group on level 5 that has a score of 300. (The level is the most significant part and the score is the least significant part.))

If you want groups you read often to get higher scores than groups you read seldom you can add the `gnus-summary-bubble-group` function to the `gnus-summary-exit-hook` hook. This will result (after sorting) in a bubbling sort of action. If you want to see that in action after each summary exit, you can add `gnus-group-sort-groups-by-rank` or `gnus-group-sort-groups-by-score` to the same hook, but that will slow things down somewhat.

## 2.8 Marking Groups

If you want to perform some command on several groups, and they appear subsequently in the group buffer, you would normally just give a numerical prefix to the command. Most group commands will then do your bidding on those groups.

However, if the groups are not in sequential order, you can still perform a command on several groups. You simply mark the groups first with the process mark and then execute the command.

**#**

*M m*           Set the mark on the current group (`gnus-group-mark-group`).



<i>M-#</i>	
<i>M u</i>	Remove the mark from the current group ( <code>gnus-group-unmark-group</code> ).
<i>M U</i>	Remove the mark from all groups ( <code>gnus-group-unmark-all-groups</code> ).
<i>M w</i>	Mark all groups between point and mark ( <code>gnus-group-mark-region</code> ).
<i>M b</i>	Mark all groups in the buffer ( <code>gnus-group-mark-buffer</code> ).
<i>M r</i>	Mark all groups that match some regular expression ( <code>gnus-group-mark-regexp</code> ).

Also see [\[Process/Prefix\]](#), page [\[Process/Prefix\]](#).

If you want to execute some command on all groups that have been marked with the process mark, you can use the *M-@* (`gnus-group-universal-argument`) command. It will prompt you for the command to be executed.

## 2.9 Foreign Groups

Below are some group mode commands for making and editing general foreign groups, as well as commands to ease the creation of a few special-purpose groups. All these commands insert the newly created groups under point—`gnus-subscribe-newsgroup-method` is not consulted.

<i>G m</i>	Make a new group ( <code>gnus-group-make-group</code> ). Gnus will prompt you for a name, a method and possibly an <i>address</i> . For an easier way to subscribe to NNTP groups (see <a href="#">[Browse Foreign Server]</a> , page <a href="#">[Browse Foreign Server]</a> ).
<i>G M</i>	Make an ephemeral group ( <code>gnus-group-read-ephemeral-group</code> ). Gnus will prompt you for a name, a method and an <i>address</i> .
<i>G r</i>	Rename the current group to something else ( <code>gnus-group-rename-group</code> ). This is valid only on some groups—mail groups mostly. This command might very well be quite slow on some back ends.
<i>G c</i>	Customize the group parameters ( <code>gnus-group-customize</code> ).
<i>G e</i>	Enter a buffer where you can edit the select method of the current group ( <code>gnus-group-edit-group-method</code> ).
<i>G p</i>	Enter a buffer where you can edit the group parameters ( <code>gnus-group-edit-group-parameters</code> ).
<i>G E</i>	Enter a buffer where you can edit the group info ( <code>gnus-group-edit-group</code> ).
<i>G d</i>	Make a directory group (see <a href="#">[Directory Groups]</a> , page <a href="#">[Directory Groups]</a> ). You will be prompted for a directory name ( <code>gnus-group-make-directory-group</code> ).
<i>G h</i>	Make the Gnus help group ( <code>gnus-group-make-help-group</code> ).
<i>G a</i>	Make a Gnus archive group ( <code>gnus-group-make-archive-group</code> ). By default a group pointing to the most recent articles will be created ( <code>gnus-group-recent-archive-directory</code> ), but given a prefix, a full group will be created from <code>gnus-group-archive-directory</code> .

- G k** Make a kiboze group. You will be prompted for a name, for a regexp to match groups to be “included” in the kiboze group, and a series of strings to match on headers (`gnus-group-make-kiboze-group`). See [\[Kibozed Groups\]](#), page [\[undefined\]](#).
- G D** Read an arbitrary directory as if it were a newsgroup with the `nneething` back end (`gnus-group-enter-directory`). See [\[Anything Groups\]](#), page [\[undefined\]](#).
- G f** Make a group based on some file or other (`gnus-group-make-doc-group`). If you give a prefix to this command, you will be prompted for a file name and a file type. Currently supported types are `mbox`, `babyl`, `digest`, `news`, `rnews`, `mmdf`, `forward`, `rfc934`, `rfc822-forward`, `mime-parts`, `standard-digest`, `slack-digest`, `clari-briefs`, `nsmail`, `outlook`, `oe-dbx`, and `mailman`. If you run this command without a prefix, Gnus will guess at the file type. See [\[Document Groups\]](#), page [\[undefined\]](#).
- G u** Create one of the groups mentioned in `gnus-useful-groups` (`gnus-group-make-useful-group`).
- G w** Make an ephemeral group based on a web search (`gnus-group-make-web-group`). If you give a prefix to this command, make a solid group instead. You will be prompted for the search engine type and the search string. Valid search engine types include `google`, `dejanews`, and `gmane`. See [\[Web Searches\]](#), page [\[undefined\]](#).
- If you use the `google` search engine, you can limit the search to a particular group by using a match string like `‘shaving group:alt.sysadmin.recovery’`.
- G R** Make a group based on an RSS feed (`gnus-group-make-rss-group`). You will be prompted for an URL. See [\[RSS\]](#), page [\[undefined\]](#).
- G DEL** This function will delete the current group (`gnus-group-delete-group`). If given a prefix, this function will actually delete all the articles in the group, and forcibly remove the group itself from the face of the Earth. Use a prefix only if you are absolutely sure of what you are doing. This command can’t be used on read-only groups (like `nnntp` groups), though.
- G V** Make a new, fresh, empty `nnvirtual` group (`gnus-group-make-empty-virtual`). See [\[Virtual Groups\]](#), page [\[undefined\]](#).
- G v** Add the current group to an `nnvirtual` group (`gnus-group-add-to-virtual`). Uses the process/prefix convention.

See [\[Select Methods\]](#), page [\[undefined\]](#), for more information on the various select methods.

If `gnus-activate-foreign-newsgroups` is a positive number, Gnus will check all foreign groups with this level or lower at startup. This might take quite a while, especially if you subscribe to lots of groups from different NNTP servers. Also see [\[Group Levels\]](#), page [\[undefined\]](#); `gnus-activate-level` also affects activation of foreign newsgroups.

## 2.10 Group Parameters

The group parameters store information local to a particular group. Here's an example group parameter list:

```
((to-address . "ding@gnus.org")
 (auto-expire . t))
```

We see that each element consists of a “dotted pair”—the thing before the dot is the key, while the thing after the dot is the value. All the parameters have this form *except* local variable specs, which are not dotted pairs, but proper lists.

Some parameters have correspondent customizable variables, each of which is an alist of regexps and values.

The following group parameters can be used:

### to-address

Address used by when doing followups and new posts.

```
(to-address . "some@where.com")
```

This is primarily useful in mail groups that represent closed mailing lists—mailing lists where it's expected that everybody that writes to the mailing list is subscribed to it. Since using this parameter ensures that the mail only goes to the mailing list itself, it means that members won't receive two copies of your followups.

Using **to-address** will actually work whether the group is foreign or not. Let's say there's a group on the server that is called 'fa.4ad-1'. This is a real newsgroup, but the server has gotten the articles from a mail-to-news gateway. Posting directly to this group is therefore impossible—you have to send mail to the mailing list address instead.

See also **gnus-parameter-to-address-alist**.

### to-list

Address used when doing a in that group.

```
(to-list . "some@where.com")
```

It is totally ignored when doing a followup—except that if it is present in a news group, you'll get mail group semantics when doing **f**.

If you do an **a** command in a mail group and you have neither a **to-list** group parameter nor a **to-address** group parameter, then a **to-list** group parameter will be added automatically upon sending the message if **gnus-add-to-list** is set to **t**.

If this variable is set, **gnus-mailing-list-mode** is turned on when entering summary buffer.

See also **gnus-parameter-to-list-alist**.

### subscribed

If this parameter is set to **t**, Gnus will consider the **to-address** and **to-list** parameters for this group as addresses of mailing lists you are subscribed to. Giving Gnus this information is (only) a first step in getting it to generate correct Mail-Followup-To headers for your posts to these lists. The second step is to put the following in your '.gnus.el'

```
(setq message-subscribed-address-functions
      '(gnus-find-subscribed-addresses))
```

See [section “Mailing Lists” in \*The Message Manual\*](#), for a complete treatment of available MFT support.

**visible** If the group parameter list has the element `(visible . t)`, that group will always be visible in the Group buffer, regardless of whether it has any unread articles.

#### **broken-reply-to**

Elements like `(broken-reply-to . t)` signals that `Reply-To` headers in this group are to be ignored, and for the header to be hidden if `reply-to` is part of `gnus-boring-article-headers`. This can be useful if you’re reading a mailing list group where the listserv has inserted `Reply-To` headers that point back to the listserv itself. That is broken behavior. So there!

**to-group** Elements like `(to-group . "some.group.name")` means that all posts in that group will be sent to `some.group.name`.

#### **newsgroup**

If you have `(newsgroup . t)` in the group parameter list, Gnus will treat all responses as if they were responses to news articles. This can be useful if you have a mail group that’s really a mirror of a news group.

**gcc-self** If `(gcc-self . t)` is present in the group parameter list, newly composed messages will be `Gcc`’d to the current group. If `(gcc-self . none)` is present, no `Gcc`: header will be generated, if `(gcc-self . "string")` is present, this string will be inserted literally as a `gcc` header. This parameter takes precedence over any default `Gcc` rules as described later (see [\(undefined\)](#) [\[Archived Messages\]](#), page [\(undefined\)](#)).

**Caveat:** Adding `(gcc-self . t)` to the parameter list of `nnntp` groups (or the like) isn’t valid. An `nnntp` server doesn’t accept articles.

#### **auto-expire**

If the group parameter has an element that looks like `(auto-expire . t)`, all articles read will be marked as expirable. For an alternative approach, see [\(undefined\)](#) [\[Expiring Mail\]](#), page [\(undefined\)](#).

See also `gnus-auto-expirable-newsgroups`.

#### **total-expire**

If the group parameter has an element that looks like `(total-expire . t)`, all read articles will be put through the expiry process, even if they are not marked as expirable. Use with caution. Unread, ticked and dormant articles are not eligible for expiry.

See also `gnus-total-expirable-newsgroups`.

#### **expiry-wait**

If the group parameter has an element that looks like `(expiry-wait . 10)`, this value will override any `nnmail-expiry-wait` and `nnmail-expiry-wait-function` (see [\(undefined\)](#) [\[Expiring Mail\]](#), page [\(undefined\)](#)) when expiring

expirable messages. The value can either be a number of days (not necessarily an integer) or the symbols **never** or **immediate**.

#### **expiry-target**

Where expired messages end up. This parameter overrides **nnmail-expiry-target**.

#### **score-file**

Elements that look like (**score-file** . "file") will make 'file' into the current score file for the group in question. All interactive score entries will be put into this file.

#### **adapt-file**

Elements that look like (**adapt-file** . "file") will make 'file' into the current adaptive file for the group in question. All adaptive score entries will be put into this file.

#### **admin-address**

When unsubscribing from a mailing list you should never send the unsubscription notice to the mailing list itself. Instead, you'd send messages to the administrative address. This parameter allows you to put the admin address somewhere convenient.

**display** Elements that look like (**display** . MODE) say which articles to display on entering the group. Valid values are:

**all** Display all articles, both read and unread.

#### **an integer**

Display the last *integer* articles in the group. This is the same as entering the group with **C-u integer**.

**default** Display the default visible articles, which normally includes unread and ticked articles.

**an array** Display articles that satisfy a predicate.

Here are some examples:

**[unread]** Display only unread articles.

**[not expire]**

Display everything except expirable articles.

**[and (not reply) (not expire)]**

Display everything except expirable and articles you've already responded to.

The available operators are **not**, **and** and **or**. Predicates include **tick**, **unsent**, **undownload**, **unread**, **dormant**, **expire**, **reply**, **killed**, **bookmark**, **score**, **save**, **cache**, **forward**, **unseen** and **recent**.

The **display** parameter works by limiting the summary buffer to the subset specified. You can pop the limit by using the **/w** command (see [\[Limiting\]](#), page [\[undefined\]](#)).

**comment** Elements that look like `(comment . "This is a comment")` are arbitrary comments on the group. You can display comments in the group line (see [\[Group Line Specification\]](#), page [\[undefined\]](#)).

**charset** Elements that look like `(charset . iso-8859-1)` will make `iso-8859-1` the default charset; that is, the charset that will be used for all articles that do not specify a charset.  
See also `gnus-group-charset-alist`.

**ignored-charsets**  
Elements that look like `(ignored-charsets x-unknown iso-8859-1)` will make `iso-8859-1` and `x-unknown` ignored; that is, the default charset will be used for decoding articles.  
See also `gnus-group-ignored-charsets-alist`.

**posting-style**  
You can store additional posting style information for this group here (see [\[Posting Styles\]](#), page [\[undefined\]](#)). The format is that of an entry in the `gnus-posting-styles` alist, except that there's no regexp matching the group name (of course). Style elements in this group parameter will take precedence over the ones found in `gnus-posting-styles`.  
For instance, if you want a funky name and signature in this group only, instead of hacking `gnus-posting-styles`, you could put something like this in the group parameters:

```
(posting-style
 (name "Funky Name")
 ("X-My-Header" "Funky Value")
 (signature "Funky Signature"))
```

**post-method**  
If it is set, the value is used as the method for posting message instead of `gnus-post-method`.

**banner** An item like `(banner . regexp)` causes any part of an article that matches the regular expression *regexp* to be stripped. Instead of *regexp*, you can also use the symbol `signature` which strips the last signature or any of the elements of the alist `gnus-article-banner-alist`.

**sieve** This parameter contains a Sieve test that should match incoming mail that should be placed in this group. From this group parameter, a Sieve 'IF' control structure is generated, having the test as the condition and `'fileinto "group.name";'` as the body.

For example, if the `'INBOX.list.sieve'` group has the `(sieve address "sender" "sieve-admin@extundo.com")` group parameter, when translating the group parameter into a Sieve script (see [\[Sieve Commands\]](#), page [\[undefined\]](#)) the following Sieve code is generated:

```
if address "\"sender\" \"sieve-admin@extundo.com\" {
    fileinto \"INBOX.list.sieve\";
}
```

The Sieve language is described in RFC 3028. See [section “Top” in \*Emacs Sieve\*](#).

#### (agent parameters)

If the agent has been enabled, you can set any of the its parameters to control the behavior of the agent in individual groups. See Agent Parameters in [\(undefined\)](#) [Category Syntax], page [\(undefined\)](#). Most users will choose to set agent parameters in either an agent category or group topic to minimize the configuration effort.

#### (variable form)

You can use the group parameters to set variables local to the group you are entering. If you want to turn threading off in ‘news.answers’, you could put `(gnus-show-threads nil)` in the group parameters of that group. `gnus-show-threads` will be made into a local variable in the summary buffer you enter, and the form `nil` will be `eval`ed there.

Note that this feature sets the variable locally to the summary buffer. But some variables are evaluated in the article buffer, or in the message buffer (of a reply or followup or otherwise newly created message). As a workaround, it might help to add the variable in question to `gnus-newsgroup-variables`. See [\(undefined\)](#) [Various Summary Stuff], page [\(undefined\)](#). So if you want to set `message-from-style` via the group parameters, then you may need the following statement elsewhere in your ‘`~/.gnus`’ file:

```
(add-to-list 'gnus-newsgroup-variables 'message-from-style)
```

A use for this feature is to remove a mailing list identifier tag in the subject fields of articles. E.g. if the news group

```
nnntp+news.gnus.org:gmame.text.docbook.apps
```

has the tag ‘DOC-BOOK-APPS:’ in the subject of all articles, this tag can be removed from the article subjects in the summary buffer for the group by putting `(gnus-list-identifiers "DOCBOOK-APPS:")` into the group parameters for the group.

This can also be used as a group-specific hook function. If you want to hear a beep when you enter a group, you could put something like `(dummy-variable (ding))` in the parameters of that group. `dummy-variable` will be set to the (meaningless) result of the `(ding)` form.

Alternatively, since the VARIABLE becomes local to the group, this pattern can be used to temporarily change a hook. For example, if the following is added to a group parameter

```
(gnus-summary-prepared-hook
 '(lambda nil (local-set-key "d" (local-key-binding "n"))))
```

when the group is entered, the ‘d’ key will not mark the article as expired.

Use the `G p` or the `G c` command to edit group parameters of a group. (`G p` presents you with a Lisp-based interface, `G c` presents you with a Customize-like interface. The latter helps avoid silly Lisp errors.) You might also be interested in reading about topic parameters (see [\(undefined\)](#) [Topic Parameters], page [\(undefined\)](#)).



Group parameters can be set via the `gnus-parameters` variable too. But some variables, such as `visible`, have no effect. For example:

```
(setq gnus-parameters
      '(("mail\\.*"
         (gnus-show-threads nil)
         (gnus-use-scoring nil)
         (gnus-summary-line-format
          "%U%R%z%I%(%[d:%ub%-23,23f%]%) %s\n")
         (gcc-self . t)
         (display . all))

        ("^nnimap:\\(foo.bar\\)$"
         (to-group . "\\1"))

        ("mail\\.me"
         (gnus-use-scoring t))

        ("list\\.*"
         (total-expire . t)
         (broken-reply-to . t))))
```

String value of parameters will be subjected to regexp substitution, as the `to-group` example shows.

## 2.11 Listing Groups

These commands all list various slices of the groups available.

- l**
- A s** List all groups that have unread articles (`gnus-group-list-groups`). If the numeric prefix is used, this command will list only groups of level ARG and lower. By default, it only lists groups of level five (i.e., `gnus-group-default-list-level`) or lower (i.e., just subscribed groups).
- L**
- A u** List all groups, whether they have unread articles or not (`gnus-group-list-all-groups`). If the numeric prefix is used, this command will list only groups of level ARG and lower. By default, it lists groups of level seven or lower (i.e., just subscribed and unsubscribed groups).
- A l** List all unread groups on a specific level (`gnus-group-list-level`). If given a prefix, also list the groups with no unread articles.
- A k** List all killed groups (`gnus-group-list-killed`). If given a prefix argument, really list all groups that are available, but aren't currently (un)subscribed. This could entail reading the active file from the server.
- A z** List all zombie groups (`gnus-group-list-zombies`).
- A m** List all unread, subscribed groups with names that match a regexp (`gnus-group-list-matching`).



- A M** List groups that match a regexp (**gnus-group-list-all-matching**).
- A A** List absolutely all groups in the active file(s) of the server(s) you are connected to (**gnus-group-list-active**). This might very well take quite a while. It might actually be a better idea to do a **A M** to list all matching, and just give ‘.’ as the thing to match on. Also note that this command may list groups that don’t exist (yet)—these will be listed as if they were killed groups. Take the output with some grains of salt.
- A a** List all groups that have names that match a regexp (**gnus-group-apropos**).
- A d** List all groups that have names or descriptions that match a regexp (**gnus-group-description-apropos**).
- A c** List all groups with cached articles (**gnus-group-list-cached**).
- A ?** List all groups with dormant articles (**gnus-group-list-dormant**).
- A /** List groups limited within the current selection (**gnus-group-list-limit**).
- A f** Flush groups from the current selection (**gnus-group-list-flush**).
- A p** List groups plus the current selection (**gnus-group-list-plus**).

Groups that match the **gnus-permanently-visible-groups** regexp will always be shown, whether they have unread articles or not. You can also add the **visible** element to the group parameters in question to get the same effect.

Groups that have just ticked articles in it are normally listed in the group buffer. If **gnus-list-groups-with-ticked-articles** is **nil**, these groups will be treated just like totally empty groups. It is **t** by default.

## 2.12 Sorting Groups

The **C-c C-s** (**gnus-group-sort-groups**) command sorts the group buffer according to the function(s) given by the **gnus-group-sort-function** variable. Available sorting functions include:

**gnus-group-sort-by-alphabet**

Sort the group names alphabetically. This is the default.

**gnus-group-sort-by-real-name**

Sort the group alphabetically on the real (unprefixed) group names.

**gnus-group-sort-by-level**

Sort by group level.

**gnus-group-sort-by-score**

Sort by group score. See [\[Group Score\]](#), page [\[Group Score\]](#).

**gnus-group-sort-by-rank**

Sort by group score and then the group level. The level and the score are, when taken together, the group’s *rank*. See [\[Group Score\]](#), page [\[Group Score\]](#).

**gnus-group-sort-by-unread**

Sort by number of unread articles.

**gnus-group-sort-by-method**

Sort alphabetically on the select method.

**gnus-group-sort-by-server**

Sort alphabetically on the Gnus server name.

**gnus-group-sort-function** can also be a list of sorting functions. In that case, the most significant sort key function must be the last one.

There are also a number of commands for sorting directly according to some sorting criteria:

- G S a**      Sort the group buffer alphabetically by group name (**gnus-group-sort-groups-by-alphabet**).
- G S u**      Sort the group buffer by the number of unread articles (**gnus-group-sort-groups-by-unread**).
- G S l**      Sort the group buffer by group level (**gnus-group-sort-groups-by-level**).
- G S v**      Sort the group buffer by group score (**gnus-group-sort-groups-by-score**). See [\(undefined\) \[Group Score\]](#), page [\(undefined\)](#).
- G S r**      Sort the group buffer by group rank (**gnus-group-sort-groups-by-rank**). See [\(undefined\) \[Group Score\]](#), page [\(undefined\)](#).
- G S m**      Sort the group buffer alphabetically by back end name (**gnus-group-sort-groups-by-method**).
- G S n**      Sort the group buffer alphabetically by real (unprefixed) group name (**gnus-group-sort-groups-by-real-name**).

All the commands below obey the process/prefix convention (see [\(undefined\) \[Process/Prefix\]](#), page [\(undefined\)](#)).

When given a symbolic prefix (see [\(undefined\) \[Symbolic Prefixes\]](#), page [\(undefined\)](#)), all these commands will sort in reverse order.

You can also sort a subset of the groups:

- G P a**      Sort the groups alphabetically by group name (**gnus-group-sort-selected-groups-by-alphabet**).
- G P u**      Sort the groups by the number of unread articles (**gnus-group-sort-selected-groups-by-unread**).
- G P l**      Sort the groups by group level (**gnus-group-sort-selected-groups-by-level**).
- G P v**      Sort the groups by group score (**gnus-group-sort-selected-groups-by-score**). See [\(undefined\) \[Group Score\]](#), page [\(undefined\)](#).
- G P r**      Sort the groups by group rank (**gnus-group-sort-selected-groups-by-rank**). See [\(undefined\) \[Group Score\]](#), page [\(undefined\)](#).
- G P m**      Sort the groups alphabetically by back end name (**gnus-group-sort-selected-groups-by-method**).

*G P n* Sort the groups alphabetically by real (unprefixed) group name (`gnus-group-sort-selected-groups-by-real-name`).

*G P s* Sort the groups according to `gnus-group-sort-function`.

And finally, note that you can use *C-k* and *C-y* to manually move groups around.

## 2.13 Group Maintenance

*b* Find bogus groups and delete them (`gnus-group-check-bogus-groups`).

*F* Find new groups and process them (`gnus-group-find-new-groups`). With 1 *C-u*, use the `ask-server` method to query the server for new groups. With 2 *C-u*'s, use most complete method possible to query the server for new groups, and subscribe the new groups as zombies.

*C-c C-x* Run all expirable articles in the current group through the expiry process (if any) (`gnus-group-expire-articles`). That is, delete all expirable articles in the group that have been around for a while. (see [\(undefined\)](#) [Expiring Mail], page [\(undefined\)](#)).

*C-c C-M-x* Run all expirable articles in all groups through the expiry process (`gnus-group-expire-all-groups`).

## 2.14 Browse Foreign Server

*B* You will be queried for a select method and a server name. Gnus will then attempt to contact this server and let you browse the groups there (`gnus-group-browse-foreign-server`).

A new buffer with a list of available groups will appear. This buffer will use the `gnus-browse-mode`. This buffer looks a bit (well, a lot) like a normal group buffer.

Here's a list of keystrokes available in the browse mode:

*n* Go to the next group (`gnus-group-next-group`).

*p* Go to the previous group (`gnus-group-prev-group`).

*SPACE* Enter the current group and display the first article (`gnus-browse-read-group`).

*RET* Enter the current group (`gnus-browse-select-group`).

*u* Unsubscribe to the current group, or, as will be the case here, subscribe to it (`gnus-browse-unsubscribe-current-group`).

*l*

*q* Exit browse mode (`gnus-browse-exit`).

*d* Describe the current group (`gnus-browse-describe-group`).

*?* Describe browse mode briefly (well, there's not much to describe, is there) (`gnus-browse-describe-briefly`).

## 2.15 Exiting Gnus

Yes, Gnus is ex(c)iting.

- z** Suspend Gnus (`gnus-group-suspend`). This doesn't really exit Gnus, but it kills all buffers except the Group buffer. I'm not sure why this is a gain, but then who am I to judge?
- q** Quit Gnus (`gnus-group-exit`).
- Q** Quit Gnus without saving the `.newsrc` files (`gnus-group-quit`). The dribble file will be saved, though (see [\[Auto Save\]](#), page [\[undefined\]](#)).

`gnus-suspend-gnus-hook` is called when you suspend Gnus and `gnus-exit-gnus-hook` is called when you quit Gnus, while `gnus-after-exiting-gnus-hook` is called as the final item when exiting Gnus.

Note:

Miss Lisa Cannifax, while sitting in English class, felt her feet go numbly heavy and herself fall into a hazy trance as the boy sitting behind her drew repeated lines with his pencil across the back of her plastic chair.

## 2.16 Group Topics

If you read lots and lots of groups, it might be convenient to group them hierarchically according to topics. You put your Emacs groups over here, your sex groups over there, and the rest (what, two groups or so?) you put in some misc section that you never bother with anyway. You can even group the Emacs sex groups as a sub-topic to either the Emacs groups or the sex groups—or both! Go wild!

`\gnusfigureGroup Topics400 \put(75,50)\epsfigfigure=ps/group-topic,height=9cm`

Here's an example:

```
Gnus
  Emacs -- I wuw it!
    3: comp.emacs
    2: alt.religion.emacs
  Naughty Emacs
    452: alt.sex.emacs
    0: comp.talk.emacs.recovery
  Misc
    8: comp.binaries.fractals
    13: comp.sources.unix
```

To get this *fab* functionality you simply turn on (ooh!) the `gnus-topic` minor mode—type `t` in the group buffer. (This is a toggling command.)

Go ahead, just try it. I'll still be here when you get back. La de dum... Nice tune, that... la la la... What, you're back? Yes, and now press `l`. There. All your groups are now listed under 'misc'. Doesn't that make you feel all warm and fuzzy? Hot and bothered?

If you want this permanently enabled, you should add that minor mode to the hook for the group mode. Put the following line in your `'~/gnus.el'` file:

```
(add-hook 'gnus-group-mode-hook 'gnus-topic-mode)
```

### 2.16.1 Topic Commands

When the topic minor mode is turned on, a new *T* submap will be available. In addition, a few of the standard keys change their definitions slightly.

In general, the following kinds of operations are possible on topics. First of all, you want to create topics. Secondly, you want to put groups in topics and to move them around until you have an order you like. The third kind of operation is to show/hide parts of the whole shebang. You might want to hide a topic including its subtopics and groups, to get a better overview of the other groups.

Here is a list of the basic keys that you might need to set up topics the way you like.

<i>T n</i>	Prompt for a new topic name and create it ( <code>gnus-topic-create-topic</code> ).
<i>T TAB</i>	
<i>TAB</i>	“Indent” the current topic so that it becomes a sub-topic of the previous topic ( <code>gnus-topic-indent</code> ). If given a prefix, “un-indent” the topic instead.
<i>M-TAB</i>	“Un-indent” the current topic so that it becomes a sub-topic of the parent of its current parent ( <code>gnus-topic-unindent</code> ).

The following two keys can be used to move groups and topics around. They work like the well-known cut and paste. *C-k* is like cut and *C-y* is like paste. Of course, this being Emacs, we use the terms kill and yank rather than cut and paste.

<i>C-k</i>	Kill a group or topic ( <code>gnus-topic-kill-group</code> ). All groups in the topic will be removed along with the topic.
<i>C-y</i>	Yank the previously killed group or topic ( <code>gnus-topic-yank-group</code> ). Note that all topics will be yanked before all groups.

So, to move a topic to the beginning of the list of topics, just hit *C-k* on it. This is like the “cut” part of cut and paste. Then, move the cursor to the beginning of the buffer (just below the “Gnus” topic) and hit *C-y*. This is like the “paste” part of cut and paste. Like I said – E-Z.

You can use *C-k* and *C-y* on groups as well as on topics. So you can move topics around as well as groups.

After setting up the topics the way you like them, you might wish to hide a topic, or to show it again. That’s why we have the following key.

<i>RET</i>	
<i>SPACE</i>	Either select a group or fold a topic ( <code>gnus-topic-select-group</code> ). When you perform this command on a group, you’ll enter the group, as usual. When done on a topic line, the topic will be folded (if it was visible) or unfolded (if it was folded already). So it’s basically a toggling command on topics. In addition, if you give a numerical prefix, group on that level (and lower) will be displayed.

Now for a list of other commands, in no particular order.

<i>T m</i>	Move the current group to some other topic ( <code>gnus-topic-move-group</code> ). This command uses the process/prefix convention (see <a href="#">[Process/Prefix]</a> , <a href="#">page [undefined]</a> ).
<i>T j</i>	Go to a topic ( <code>gnus-topic-jump-to-topic</code> ).

<i>T c</i>	Copy the current group to some other topic ( <code>gnus-topic-copy-group</code> ). This command uses the process/prefix convention (see <a href="#">(undefined)</a> [Process/Prefix], page <a href="#">(undefined)</a> ).
<i>T h</i>	Hide the current topic ( <code>gnus-topic-hide-topic</code> ). If given a prefix, hide the topic permanently.
<i>T s</i>	Show the current topic ( <code>gnus-topic-show-topic</code> ). If given a prefix, show the topic permanently.
<i>T D</i>	Remove a group from the current topic ( <code>gnus-topic-remove-group</code> ). This command is mainly useful if you have the same group in several topics and wish to remove it from one of the topics. You may also remove a group from all topics, but in that case, Gnus will add it to the root topic the next time you start Gnus. In fact, all new groups (which, naturally, don't belong to any topic) will show up in the root topic.  This command uses the process/prefix convention (see <a href="#">(undefined)</a> [Process/Prefix], page <a href="#">(undefined)</a> ).
<i>T M</i>	Move all groups that match some regular expression to a topic ( <code>gnus-topic-move-matching</code> ).
<i>T C</i>	Copy all groups that match some regular expression to a topic ( <code>gnus-topic-copy-matching</code> ).
<i>T H</i>	Toggle hiding empty topics ( <code>gnus-topic-toggle-display-empty-topics</code> ).
<i>T #</i>	Mark all groups in the current topic with the process mark ( <code>gnus-topic-mark-topic</code> ). This command works recursively on sub-topics unless given a prefix.
<i>T M-#</i>	Remove the process mark from all groups in the current topic ( <code>gnus-topic-unmark-topic</code> ). This command works recursively on sub-topics unless given a prefix.
<i>C-c C-x</i>	Run all expirable articles in the current group or topic through the expiry process (if any) ( <code>gnus-topic-expire-articles</code> ). (see <a href="#">(undefined)</a> [Expiring Mail], page <a href="#">(undefined)</a> ).
<i>T r</i>	Rename a topic ( <code>gnus-topic-rename</code> ).
<i>T DEL</i>	Delete an empty topic ( <code>gnus-topic-delete</code> ).
<i>A T</i>	List all groups that Gnus knows about in a topics-ified way ( <code>gnus-topic-list-active</code> ).
<i>T M-n</i>	Go to the next topic ( <code>gnus-topic-goto-next-topic</code> ).
<i>T M-p</i>	Go to the next topic ( <code>gnus-topic-goto-previous-topic</code> ).
<i>G p</i>	Edit the topic parameters ( <code>gnus-topic-edit-parameters</code> ). See <a href="#">(undefined)</a> [Topic Parameters], page <a href="#">(undefined)</a> .

### 2.16.2 Topic Variables

The previous section told you how to tell Gnus which topics to display. This section explains how to tell Gnus what to display about each topic.

The topic lines themselves are created according to the `gnus-topic-line-format` variable (see [\[Formatting Variables\]](#), page [\[undefined\]](#)). Valid elements are:

<code>'i'</code>	Indentation.
<code>'n'</code>	Topic name.
<code>'v'</code>	Visibility.
<code>'l'</code>	Level.
<code>'g'</code>	Number of groups in the topic.
<code>'a'</code>	Number of unread articles in the topic.
<code>'A'</code>	Number of unread articles in the topic and all its subtopics.

Each sub-topic (and the groups in the sub-topics) will be indented with `gnus-topic-indent-level` times the topic level number of spaces. The default is 2.

`gnus-topic-mode-hook` is called in topic minor mode buffers.

The `gnus-topic-display-empty-topics` says whether to display even topics that have no unread articles in them. The default is `t`.

### 2.16.3 Topic Sorting

You can sort the groups in each topic individually with the following commands:

<code>T S a</code>	Sort the current topic alphabetically by group name ( <code>gnus-topic-sort-groups-by-alphabet</code> ).
<code>T S u</code>	Sort the current topic by the number of unread articles ( <code>gnus-topic-sort-groups-by-unread</code> ).
<code>T S l</code>	Sort the current topic by group level ( <code>gnus-topic-sort-groups-by-level</code> ).
<code>T S v</code>	Sort the current topic by group score ( <code>gnus-topic-sort-groups-by-score</code> ). See <a href="#">[Group Score]</a> , page <a href="#">[undefined]</a> .
<code>T S r</code>	Sort the current topic by group rank ( <code>gnus-topic-sort-groups-by-rank</code> ). See <a href="#">[Group Score]</a> , page <a href="#">[undefined]</a> .
<code>T S m</code>	Sort the current topic alphabetically by back end name ( <code>gnus-topic-sort-groups-by-method</code> ).
<code>T S e</code>	Sort the current topic alphabetically by server name ( <code>gnus-topic-sort-groups-by-server</code> ).
<code>T S s</code>	Sort the current topic according to the function(s) given by the <code>gnus-group-sort-function</code> variable ( <code>gnus-topic-sort-groups</code> ).

When given a prefix argument, all these commands will sort in reverse order. See [\[Sorting Groups\]](#), page [\[undefined\]](#), for more information about group sorting.

### 2.16.4 Topic Topology

So, let's have a look at an example group buffer:

```
Gnus
  Emacs -- I wuw it!
    3: comp.emacs
    2: alt.religion.emacs
  Naughty Emacs
    452: alt.sex.emacs
    0: comp.talk.emacs.recovery
  Misc
    8: comp.binaries.fractals
    13: comp.sources.unix
```

So, here we have one top-level topic ('Gnus'), two topics under that, and one sub-topic under one of the sub-topics. (There is always just one (1) top-level topic). This topology can be expressed as follows:

```
((("Gnus" visible)
  ((("Emacs -- I wuw it!" visible)
    ((("Naughty Emacs" visible))))
  ((("Misc" visible)))))
```

This is in fact how the variable `gnus-topic-topology` would look for the display above. That variable is saved in the `.newsrc.eld` file, and shouldn't be messed with manually—unless you really want to. Since this variable is read from the `.newsrc.eld` file, setting it in any other startup files will have no effect.

This topology shows what topics are sub-topics of what topics (right), and which topics are visible. Two settings are currently allowed—`visible` and `invisible`.

### 2.16.5 Topic Parameters

All groups in a topic will inherit group parameters from the parent (and ancestor) topic parameters. All valid group parameters are valid topic parameters (see [\[Group Parameters\]](#), page [\[undefined\]](#)). When the agent is enabled, all agent parameters (See Agent Parameters in [\[Category Syntax\]](#), page [\[undefined\]](#)) are also valid topic parameters.

In addition, the following parameters are only valid as topic parameters:

#### `subscribe`

When subscribing new groups by topic (see [\[Subscription Methods\]](#), page [\[undefined\]](#)), the `subscribe` topic parameter says what groups go in what topic. Its value should be a regexp to match the groups that should go in that topic.

#### `subscribe-level`

When subscribing new groups by topic (see the `subscribe` parameter), the group will be subscribed with the level specified in the `subscribe-level` instead of `gnus-level-default-subscribed`.



Group parameters (of course) override topic parameters, and topic parameters in sub-topics override topic parameters in super-topics. You know. Normal inheritance rules. (*Rules* is here a noun, not a verb, although you may feel free to disagree with me here.)

```
Gnus
  Emacs
    3: comp.emacs
    2: alt.religion.emacs
  452: alt.sex.emacs
  Relief
    452: alt.sex.emacs
    0: comp.talk.emacs.recovery
  Misc
    8: comp.binaries.fractals
    13: comp.sources.unix
    452: alt.sex.emacs
```

The ‘Emacs’ topic has the topic parameter (`score-file . "emacs.SCORE"`); the ‘Relief’ topic has the topic parameter (`score-file . "relief.SCORE"`); and the ‘Misc’ topic has the topic parameter (`score-file . "emacs.SCORE"`). In addition, ‘alt.religion.emacs’ has the group parameter (`score-file . "religion.SCORE"`).

Now, when you enter ‘alt.sex.emacs’ in the ‘Relief’ topic, you will get the ‘relief.SCORE’ home score file. If you enter the same group in the ‘Emacs’ topic, you’ll get the ‘emacs.SCORE’ home score file. If you enter the group ‘alt.religion.emacs’, you’ll get the ‘religion.SCORE’ home score file.

This seems rather simple and self-evident, doesn’t it? Well, yes. But there are some problems, especially with the `total-expiry` parameter. Say you have a mail group in two topics; one with `total-expiry` and one without. What happens when you do *M-x gnus-expire-all-expirable-groups*? Gnus has no way of telling which one of these topics you mean to expire articles from, so anything may happen. In fact, I hereby declare that it is *undefined* what happens. You just have to be careful if you do stuff like that.

## 2.17 Misc Group Stuff

- ~ Enter the server buffer (`gnus-group-enter-server-mode`). See [\[Server Buffer\]](#), page [\[undefined\]](#).
- a Start composing a message (a news by default) (`gnus-group-post-news`). If given a prefix, post to the group under the point. If the prefix is 1, prompt for a group to post to. Contrary to what the name of this function suggests, the prepared article might be a mail instead of a news, if a mail group is specified with the prefix argument. See [\[Composing Messages\]](#), page [\[undefined\]](#).
- m Mail a message somewhere (`gnus-group-mail`). If given a prefix, use the posting style of the group under the point. If the prefix is 1, prompt for a group name to find the posting style. See [\[Composing Messages\]](#), page [\[undefined\]](#).

- i* Start composing a news (**gnus-group-news**). If given a prefix, post to the group under the point. If the prefix is 1, prompt for group to post to. See [\[Composing Messages\]](#), page [\[undefined\]](#).

This function actually prepares a news even when using mail groups. This is useful for “posting” messages to mail groups without actually sending them over the network: they’re just saved directly to the group in question. The corresponding back end must have a request-post method for this to work though.

Variables for the group buffer:

**gnus-group-mode-hook**

is called after the group buffer has been created.

**gnus-group-prepare-hook**

is called after the group buffer is generated. It may be used to modify the buffer in some strange, unnatural way.

**gnus-group-prepared-hook**

is called as the very last thing after the group buffer has been generated. It may be used to move point around, for instance.

**gnus-permanently-visible-groups**

Groups matching this regexp will always be listed in the group buffer, whether they are empty or not.

**gnus-group-name-charset-method-alist**

An alist of method and the charset for group names. It is used to show non-ASCII group names.

For example:

```
(setq gnus-group-name-charset-method-alist
      '(((nntp "news.com.cn") . cn-gb-2312)))
```

**gnus-group-name-charset-group-alist**

An alist of regexp of group name and the charset for group names. It is used to show non-ASCII group names. `(("." utf-8))` is the default value if UTF-8 is supported, otherwise the default is `nil`.

For example:

```
(setq gnus-group-name-charset-group-alist
      '(("\\.com\\.cn:" . cn-gb-2312)))
```

### 2.17.1 Scanning New Messages

- g* Check the server(s) for new articles. If the numerical prefix is used, this command will check only groups of level *arg* and lower (**gnus-group-get-new-news**). If given a non-numerical prefix, this command will force a total re-reading of the active file(s) from the back end(s).

- M-g* Check whether new articles have arrived in the current group (**gnus-group-get-new-news-this-group**). **gnus-goto-next-group-when-activating** says whether this command is to move point to the next group or not. It is `t` by default.

- C-c M-g*    Activate absolutely all groups (`gnus-activate-all-groups`).
- R*            Restart Gnus (`gnus-group-restart`). This saves the `.newsrc` file(s), closes the connection to all servers, clears up all run-time Gnus variables, and then starts Gnus all over again.

`gnus-get-new-news-hook` is run just before checking for new news.

`gnus-after-getting-new-news-hook` is run after checking for new news.

## 2.17.2 Group Information

- H f*            Try to fetch the FAQ for the current group (`gnus-group-fetch-faq`). Gnus will try to get the FAQ from `gnus-group-faq-directory`, which is usually a directory on a remote machine. This variable can also be a list of directories. In that case, giving a prefix to this command will allow you to choose between the various sites. `ange-ftp` (or `efs`) will be used for fetching the file.
- If fetching from the first site is unsuccessful, Gnus will attempt to go through `gnus-group-faq-directory` and try to open them one by one.
- H c*            Try to open the charter for the current group in a web browser (`gnus-group-fetch-charter`). Query for a group if given a prefix argument.
- Gnus will use `gnus-group-charter-alist` to find the location of the charter. If no location is known, Gnus will fetch the control messages for the group, which in some cases includes the charter.
- H C*            Fetch the control messages for the group from the archive at `ftp.isc.org` (`gnus-group-fetch-control`). Query for a group if given a prefix argument.
- If `gnus-group-fetch-control-use-browse-url` is non-nil, Gnus will open the control messages in a browser using `browse-url`. Otherwise they are fetched using `ange-ftp` and displayed in an ephemeral group.
- Note that the control messages are compressed. To use this command you need to turn on `auto-compression-mode` (see [section “Compressed Files” in \*The Emacs Manual\*](#)).
- H d*
- C-c C-d*       Describe the current group (`gnus-group-describe-group`). If given a prefix, force Gnus to re-read the description from the server.
- M-d*           Describe all groups (`gnus-group-describe-all-groups`). If given a prefix, force Gnus to re-read the description file from the server.
- H v*
- V*            Display current Gnus version numbers (`gnus-version`).
- ?*            Give a very short help message (`gnus-group-describe-briefly`).
- C-c C-i*       Go to the Gnus info node (`gnus-info-find-node`).

## 2.17.3 Group Timestamp

It can be convenient to let Gnus keep track of when you last read a group. To set the ball rolling, you should add `gnus-group-set-timestamp` to `gnus-select-group-hook`:

```
(add-hook 'gnus-select-group-hook 'gnus-group-set-timestamp)
```

After doing this, each time you enter a group, it'll be recorded.

This information can be displayed in various ways—the easiest is to use the ‘%d’ spec in the group line format:

```
(setq gnus-group-line-format
      "%M%S%p\P%5y: %(-40,40g%) %d\n")
```

This will result in lines looking like:

```
*      0: mail.ding                      19961002T012943
      0: custom                          19961002T012713
```

As you can see, the date is displayed in compact ISO 8601 format. This may be a bit too much, so to just display the date, you could say something like:

```
(setq gnus-group-line-format
      "%M%S%p\P%5y: %(-40,40g%) %6,6~(cut 2)d\n")
```

If you would like greater control of the time format, you can use a user-defined format spec. Something like the following should do the trick:

```
(setq gnus-group-line-format
      "%M%S%p\P%5y: %(-40,40g%) %ud\n")
(defun gnus-user-format-function-d (headers)
  (let ((time (gnus-group-timestamp gnus-tmp-group)))
    (if time
        (format-time-string "%b %d %H:%M" time)
        "")))
```

## 2.17.4 File Commands

- r** Re-read the init file (`gnus-init-file`, which defaults to ‘~/`.gnus.el`’) (`gnus-group-read-init-file`).
- s** Save the ‘`.newsrsrc.eld`’ file (and ‘`.newsrsrc`’ if wanted) (`gnus-group-save-newsrc`). If given a prefix, force saving the file(s) whether Gnus thinks it is necessary or not.

## 2.17.5 Sieve Commands

Sieve is a server-side mail filtering language. In Gnus you can use the `sieve` group parameter (see [\[Group Parameters\]](#), page [\[Group Parameters\]](#)) to specify sieve rules that should apply to each group. Gnus provides two commands to translate all these group parameters into a proper Sieve script that can be transferred to the server somehow.

The generated Sieve script is placed in `gnus-sieve-file` (by default ‘~/`.sieve`’). The Sieve code that Gnus generate is placed between two delimiters, `gnus-sieve-region-start` and `gnus-sieve-region-end`, so you may write additional Sieve code outside these delimiters that will not be removed the next time you regenerate the Sieve script.

The variable `gnus-sieve-crosspost` controls how the Sieve script is generated. If it is non-`nil` (the default) articles is placed in all groups that have matching rules, otherwise the article is only placed in the group with the first matching rule. For example, the group parameter ‘(`sieve address "sender" "owner-ding@hpc.uh.edu"`)’ will generate the following piece of Sieve code if `gnus-sieve-crosspost` is `nil`. (When `gnus-sieve-crosspost` is non-`nil`, it looks the same except that the line containing the call to `stop` is removed.)

```
if address "sender" "owner-ding@hpc.uh.edu" {  
    fileinto "INBOX.ding";  
    stop;  
}
```

See [section “Top” in \*Emacs Sieve\*](#).

- D g*      Regenerate a Sieve script from the `sieve` group parameters and put you into the `gnus-sieve-file` without saving it.
- D u*      Regenerates the Gnus managed part of `gnus-sieve-file` using the `sieve` group parameters, save the file and upload it to the server using the `sieveshell` program.



## 3 Summary Buffer

A line for each article is displayed in the summary buffer. You can move around, read articles, post articles and reply to articles.

The most common way to a summary buffer is to select a group from the group buffer (see [\[Selecting a Group\]](#), page [\[undefined\]](#)).

You can have as many summary buffers open as you wish.

### 3.1 Summary Buffer Format

`\gnusfigureThe Summary Buffer180 \put(0,0)\epsfigfigure=ps/summary,width=7.5cm  
\put(445,0)\makebox(0,0)[br]\epsfigfigure=ps/summary-article,width=7.5cm`

Gnus will use the value of the `gnus-extract-address-components` variable as a function for getting the name and address parts of a `From` header. Two pre-defined functions exist: `gnus-extract-address-components`, which is the default, quite fast, and too simplistic solution; and `mail-extract-address-components`, which works very nicely, but is slower. The default function will return the wrong answer in 5% of the cases. If this is unacceptable to you, use the other function instead:

```
(setq gnus-extract-address-components
      'mail-extract-address-components)
```

`gnus-summary-same-subject` is a string indicating that the current article has the same subject as the previous. This string will be used with those specs that require it. The default is "".

#### 3.1.1 Summary Buffer Lines

You can change the format of the lines in the summary buffer by changing the `gnus-summary-line-format` variable. It works along the same lines as a normal `format` string, with some extensions (see [\[Formatting Variables\]](#), page [\[undefined\]](#)).

There should always be a colon or a point position marker on the line; the cursor always moves to the point position marker or the colon after performing an operation. (Of course, Gnus wouldn't be Gnus if it wasn't possible to change this. Just write a new function `gnus-goto-colon` which does whatever you like with the cursor.) See [\[Positioning Point\]](#), page [\[undefined\]](#).

The default string is `'%U%R%Z%I%(("[%4L: %-23,23f%])%) %s\n'`.

The following format specification characters and extended format specification(s) are understood:

- 'N'      Article number.
- 'S'      Subject string. List identifiers stripped, `gnus-list-identifies`. See [\[undefined\]](#) [\[Article Hiding\]](#), page [\[undefined\]](#).
- 's'      Subject if the article is the root of the thread or the previous article had a different subject, `gnus-summary-same-subject` otherwise. (`gnus-summary-same-subject` defaults to "").
- 'F'      Full `From` header.

- ‘n’ The name (from the **From** header).
- ‘f’ The name, **To** header or the **Newsgroups** header (see [\[To From Newsgroups\]](#), page [\[undefined\]](#)).
- ‘a’ The name (from the **From** header). This differs from the **n** spec in that it uses the function designated by the `gnus-extract-address-components` variable, which is slower, but may be more thorough.
- ‘A’ The address (from the **From** header). This works the same way as the **a** spec.
- ‘L’ Number of lines in the article.
- ‘c’ Number of characters in the article. This specifier is not supported in some methods (like `nnfolder`).
- ‘k’ Pretty-printed version of the number of characters in the article; for example, ‘1.2k’ or ‘0.4M’.
- ‘I’ Indentation based on thread level (see [\[Customizing Threading\]](#), page [\[undefined\]](#)).
- ‘B’ A complex trn-style thread tree, showing response-connecting trace lines. A thread could be drawn like this:

```

>
+-->
| +-->
| | \->
| |   \->
| \->
+-->
\->

```

You can customize the appearance with the following options. Note that it is possible to make the thread display look really neat by replacing the default ASCII characters with graphic line-drawing glyphs.

`gnus-sum-thread-tree-root`

Used for the root of a thread. If `nil`, use subject instead. The default is ‘>’.

`gnus-sum-thread-tree-false-root`

Used for the false root of a thread (see [\[Loose Threads\]](#), page [\[undefined\]](#)). If `nil`, use subject instead. The default is ‘>’.

`gnus-sum-thread-tree-single-indent`

Used for a thread with just one message. If `nil`, use subject instead. The default is ‘.’.

`gnus-sum-thread-tree-vertical`

Used for drawing a vertical line. The default is ‘|’.

`gnus-sum-thread-tree-indent`

Used for indenting. The default is ‘ ’.



`gnus-sum-thread-tree-leaf-with-other`

Used for a leaf with brothers. The default is ‘+->’.

`gnus-sum-thread-tree-single-leaf`

Used for a leaf without brothers. The default is ‘\->’.

- ‘T’ Nothing if the article is a root and lots of spaces if it isn’t (it pushes everything after it off the screen).
- ‘[’ Opening bracket, which is normally ‘[’, but can also be ‘<’ for adopted articles (see [\(undefined\)](#) [Customizing Threading], page [\(undefined\)](#)).
- ‘]’ Closing bracket, which is normally ‘]’, but can also be ‘>’ for adopted articles.
- ‘>’ One space for each thread level.
- ‘<’ Twenty minus thread level spaces.
- ‘U’ Unread. See [\(undefined\)](#) [Read Articles], page [\(undefined\)](#).
- ‘R’ This misleadingly named specifier is the *secondary mark*. This mark will say whether the article has been replied to, has been cached, or has been saved. See [\(undefined\)](#) [Other Marks], page [\(undefined\)](#).
- ‘i’ Score as a number (see [\(undefined\)](#) [Scoring], page [\(undefined\)](#)).
- ‘z’ Zcore, ‘+’ if above the default level and ‘-’ if below the default level. If the difference between `gnus-summary-default-score` and the score is less than `gnus-summary-zcore-fuzz`, this spec will not be used.
- ‘V’ Total thread score.
- ‘x’ Xref.
- ‘D’ Date.
- ‘d’ The Date in DD-MMM format.
- ‘o’ The Date in YYYYMMDDTHHMMSS format.
- ‘M’ Message-ID.
- ‘r’ References.
- ‘t’ Number of articles in the current sub-thread. Using this spec will slow down summary buffer generation somewhat.
- ‘e’ An ‘=’ (`gnus-not-empty-thread-mark`) will be displayed if the article has any children.
- ‘P’ The line number.
- ‘O’ Download mark.
- ‘&user-date;’  
Age sensitive date format. Various date format is defined in `gnus-user-date-format-alist`.

‘u’ User defined specifier. The next character in the format string should be a letter. Gnus will call the function `gnus-user-format-function-x`, where `x` is the letter following ‘u’. The function will be passed the current header as argument. The function should return a string, which will be inserted into the summary just like information from any other summary specifier.

Text between ‘(’ and ‘)’ will be highlighted with `gnus-mouse-face` when the mouse point is placed inside the area. There can only be one such area.

The ‘%U’ (status), ‘%R’ (replied) and ‘%Z’ (zcore) specs have to be handled with care. For reasons of efficiency, Gnus will compute what column these characters will end up in, and “hard-code” that. This means that it is invalid to have these specs after a variable-length spec. Well, you might not be arrested, but your summary buffer will look strange, which is bad enough.

The smart choice is to have these specs as far to the left as possible. (Isn’t that the case with everything, though? But I digress.)

This restriction may disappear in later versions of Gnus.

### 3.1.2 To From Newsgroups

In some groups (particularly in archive groups), the **From** header isn’t very interesting, since all the articles there are written by you. To display the information in the **To** or **Newsgroups** headers instead, you need to decide three things: What information to gather; where to display it; and when to display it.

1. The reading of extra header information is controlled by the `gnus-extra-headers`. This is a list of header symbols. For instance:

```
(setq gnus-extra-headers
      '(To Newsgroups X-Newsreader))
```

This will result in Gnus trying to obtain these three headers, and storing it in header structures for later easy retrieval.

2. The value of these extra headers can be accessed via the `gnus-extra-header` function. Here’s a format line spec that will access the **X-Newsreader** header:

```
"%(form (gnus-extra-header 'X-Newsreader))@"
```

3. The `gnus-ignored-from-addresses` variable says when the ‘%f’ summary line spec returns the **To**, **Newsreader** or **From** header. If this regexp matches the contents of the **From** header, the value of the **To** or **Newsreader** headers are used instead.

A related variable is `nnmail-extra-headers`, which controls when to include extra headers when generating overview (NOV) files. If you have old overview files, you should regenerate them after changing this variable, by entering the server buffer using `^`, and then `g` on the appropriate mail server (e.g. `nnml`) to cause regeneration.

You also have to instruct Gnus to display the data by changing the `%n` spec to the `%f` spec in the `gnus-summary-line-format` variable.

In summary, you’d typically put something like the following in ‘`~/gnus.el`’:

```
(setq gnus-extra-headers
      '(To Newsgroups))
(setq nnmail-extra-headers gnus-extra-headers)
```

```
(setq gnus-summary-line-format
      "%U%R%z%I%([%4L: %-23,23f%]) %s\n")
(setq gnus-ignored-from-addresses
      "Your Name Here")
```

(The values listed above are the default values in Gnus. Alter them to fit your needs.)

A note for news server administrators, or for users who wish to try to convince their news server administrator to provide some additional support:

The above is mostly useful for mail groups, where you have control over the NOV files that are created. However, if you can persuade your nntp admin to add (in the usual implementation, notably INN):

```
Newsgroups:full
```

to the end of her `overview.fmt` file, then you can use that just as you would the extra headers from the mail groups.

### 3.1.3 Summary Buffer Mode Line

You can also change the format of the summary mode bar (see [\[Mode Line Formatting\]](#), page [\[undefined\]](#)). Set `gnus-summary-mode-line-format` to whatever you like. The default is `'Gnus: %%b [%A] %Z'`.

Here are the elements you can play with:

'G'	Group name.
'p'	Unprefixed group name.
'A'	Current article number.
'z'	Current article score.
'V'	Gnus version.
'U'	Number of unread articles in this group.
'e'	Number of unread articles in this group that aren't displayed in the summary buffer.
'Z'	A string with the number of unread and unselected articles represented either as <code>&lt;%U(+%e) more&gt;</code> if there are both unread and unselected articles, and just as <code>&lt;%U more&gt;</code> if there are just unread articles and no unselected ones.
'g'	Shortish group name. For instance, <code>'rec.arts.anime'</code> will be shortened to <code>'r.a.anime'</code> .
'S'	Subject of the current article.
'u'	User-defined spec (see <a href="#">[User-Defined Specs]</a> , page <a href="#">[undefined]</a> ).
's'	Name of the current score file (see <a href="#">[Scoring]</a> , page <a href="#">[undefined]</a> ).
'd'	Number of dormant articles (see <a href="#">[Unread Articles]</a> , page <a href="#">[undefined]</a> ).
't'	Number of ticked articles (see <a href="#">[Unread Articles]</a> , page <a href="#">[undefined]</a> ).
'r'	Number of articles that have been marked as read in this session.
'E'	Number of articles expunged by the score files.

### 3.1.4 Summary Highlighting

#### `gnus-visual-mark-article-hook`

This hook is run after selecting an article. It is meant to be used for highlighting the article in some way. It is not run if `gnus-visual` is `nil`.

#### `gnus-summary-update-hook`

This hook is called when a summary line is changed. It is not run if `gnus-visual` is `nil`.

#### `gnus-summary-selected-face`

This is the face (or *font* as some people call it) used to highlight the current article in the summary buffer.

#### `gnus-summary-highlight`

Summary lines are highlighted according to this variable, which is a list where the elements are of the format `(form . face)`. If you would, for instance, like ticked articles to be italic and high-scored articles to be bold, you could set this variable to something like

```
((eq mark gnus-ticked-mark) . italic)
(> score default) . bold))
```

As you may have guessed, if *form* returns a non-`nil` value, *face* will be applied to the line.

## 3.2 Summary Maneuvering

All the straight movement commands understand the numeric prefix and behave pretty much as you'd expect.

None of these commands select articles.

#### *G M-n*

*M-n* Go to the next summary line of an unread article (`gnus-summary-next-unread-subject`).

#### *G M-p*

*M-p* Go to the previous summary line of an unread article (`gnus-summary-prev-unread-subject`).

#### *G g*

Ask for an article number and then go to the summary line of that article without displaying the article (`gnus-summary-goto-subject`).

If Gnus asks you to press a key to confirm going to the next group, you can use the *C-n* and *C-p* keys to move around the group buffer, searching for the next group to read without actually returning to the group buffer.

Variables related to summary movement:

#### `gnus-auto-select-next`

If you issue one of the movement commands (like *n*) and there are no more unread articles after the current one, Gnus will offer to go to the next group. If this variable is `t` and the next group is empty, Gnus will exit summary mode and return to the group buffer. If this variable is neither `t` nor `nil`, Gnus will select the next group with unread articles. As a special case, if this variable is

quietly, Gnus will select the next group without asking for confirmation. If this variable is `almost-quietly`, the same will happen only if you are located on the last article in the group. Finally, if this variable is `slightly-quietly`, the `Z n` command will go to the next group without confirmation. Also see [\[Group Levels\]](#), page [\[undefined\]](#).

#### `gnus-auto-select-same`

If non-`nil`, all the movement commands will try to go to the next article with the same subject as the current. (*Same* here might mean *roughly equal*. See `gnus-summary-gather-subject-limit` for details (see [\[Customizing Threading\]](#), page [\[undefined\]](#).) If there are no more articles with the same subject, go to the first unread article.

This variable is not particularly useful if you use a threaded display.

#### `gnus-summary-check-current`

If non-`nil`, all the “unread” movement commands will not proceed to the next (or previous) article if the current article is unread. Instead, they will choose the current article.

#### `gnus-auto-center-summary`

If non-`nil`, Gnus will keep the point in the summary buffer centered at all times. This makes things quite tidy, but if you have a slow network connection, or simply do not like this un-Emacsism, you can set this variable to `nil` to get the normal Emacs scrolling action. This will also inhibit horizontal re-centering of the summary buffer, which might make it more inconvenient to read extremely long threads.

This variable can also be a number. In that case, center the window at the given number of lines from the top.

## 3.3 Choosing Articles

### 3.3.1 Choosing Commands

None of the following movement commands understand the numeric prefix, and they all select and display an article.

If you want to fetch new articles or redisplay the group, see [\[Exiting the Summary Buffer\]](#), page [\[undefined\]](#).

**SPACE**      Select the current article, or, if that one’s read already, the next unread article (`gnus-summary-next-page`).

If you have an article window open already and you press **SPACE** again, the article will be scrolled. This lets you conveniently **SPACE** through an entire newsgroup. See [\[Paging the Article\]](#), page [\[undefined\]](#).

**G n**

**n**      Go to next unread article (`gnus-summary-next-unread-article`).

**G p**

**p**      Go to previous unread article (`gnus-summary-prev-unread-article`).

<i>G N</i>	
<i>N</i>	Go to the next article ( <code>gnus-summary-next-article</code> ).
<i>G P</i>	
<i>P</i>	Go to the previous article ( <code>gnus-summary-prev-article</code> ).
<i>G C-n</i>	Go to the next article with the same subject ( <code>gnus-summary-next-same-subject</code> ).
<i>G C-p</i>	Go to the previous article with the same subject ( <code>gnus-summary-prev-same-subject</code> ).
<i>G f</i>	
<i>.</i>	Go to the first unread article ( <code>gnus-summary-first-unread-article</code> ).
<i>G b</i>	
<i>,</i>	Go to the unread article with the highest score ( <code>gnus-summary-best-unread-article</code> ). If given a prefix argument, go to the first unread article that has a score over the default score.
<i>G l</i>	
<i>l</i>	Go to the previous article read ( <code>gnus-summary-goto-last-article</code> ).
<i>G o</i>	Pop an article off the summary history and go to this article ( <code>gnus-summary-pop-article</code> ). This command differs from the command above in that you can pop as many previous articles off the history as you like, while <i>l</i> toggles the two last read articles. For a somewhat related issue (if you use these commands a lot), see <a href="#">[Article Backlog]</a> , page <a href="#">[undefined]</a> .
<i>G j</i>	
<i>j</i>	Ask for an article number or Message-ID, and then go to that article ( <code>gnus-summary-goto-article</code> ).

### 3.3.2 Choosing Variables

Some variables relevant for moving and selecting articles:

#### `gnus-auto-extend-newsgroup`

All the movement commands will try to go to the previous (or next) article, even if that article isn't displayed in the Summary buffer if this variable is non-`nil`. Gnus will then fetch the article from the server and display it in the article buffer.

#### `gnus-select-article-hook`

This hook is called whenever an article is selected. By default it exposes any threads hidden under the selected article. If you would like each article to be saved in the Agent as you read it, putting `gnus-agent-fetch-selected-article` on this hook will do so.

#### `gnus-mark-article-hook`

This hook is called whenever an article is selected. It is intended to be used for marking articles as read. The default value is `gnus-summary-mark-read-and-unread-as-read`, and will change the mark of almost any article you read to `gnus-unread-mark`. The only articles not affected by this function are ticked,

dormant, and expirable articles. If you'd instead like to just have unread articles marked as read, you can use `gnus-summary-mark-unread-as-read` instead. It will leave marks like `gnus-low-score-mark`, `gnus-del-mark` (and so on) alone.

## 3.4 Scrolling the Article

**SPACE** Pressing *SPACE* will scroll the current article forward one page, or, if you have come to the end of the current article, will choose the next article (`gnus-summary-next-page`).

If `gnus-article-skip-boring` is non-`nil` and the rest of the article consists only of citations and signature, then it will be skipped; the next article will be shown instead. You can customize what is considered uninteresting with `gnus-article-boring-faces`. You can manually view the article's pages, no matter how boring, using *C-M-v*.

**DEL** Scroll the current article back one page (`gnus-summary-prev-page`).

**RET** Scroll the current article one line forward (`gnus-summary-scroll-up`).

**M-RET** Scroll the current article one line backward (`gnus-summary-scroll-down`).

**A g**

*g* (Re)fetch the current article (`gnus-summary-show-article`). If given a prefix, fetch the current article, but don't run any of the article treatment functions. This will give you a "raw" article, just the way it came from the server.

If given a numerical prefix, you can do semi-manual charset stuff. *C-u 0 g cn-gb-2312 RET* will decode the message as if it were encoded in the `cn-gb-2312` charset. If you have

```
(setq gnus-summary-show-article-charset-alist
      '((1 . cn-gb-2312)
        (2 . big5)))
```

then you can say *C-u 1 g* to get the same effect.

**A <**

*<* Scroll to the beginning of the article (`gnus-summary-beginning-of-article`).

**A >**

*>* Scroll to the end of the article (`gnus-summary-end-of-article`).

**A s**

*s* Perform an isearch in the article buffer (`gnus-summary-isearch-article`).

**h**

Select the article buffer (`gnus-summary-select-article-buffer`).

## 3.5 Reply, Followup and Post

### 3.5.1 Summary Mail Commands

Commands for composing a mail message:

**S r**

*r* Mail a reply to the author of the current article (`gnus-summary-reply`).

- S R*  
*R* Mail a reply to the author of the current article and include the original message (`gnus-summary-reply-with-original`). This command uses the process/prefix convention.
- S w* Mail a wide reply to the author of the current article (`gnus-summary-wide-reply`). A *wide reply* is a reply that goes out to all people listed in the To, From (or Reply-to) and Cc headers. If Mail-Followup-To is present, that's used instead.
- S W* Mail a wide reply to the current article and include the original message (`gnus-summary-wide-reply-with-original`). This command uses the process/prefix convention.
- S v* Mail a very wide reply to the author of the current article (`gnus-summary-wide-reply`). A *very wide reply* is a reply that goes out to all people listed in the To, From (or Reply-to) and Cc headers in all the process/prefixed articles. This command uses the process/prefix convention.
- S V* Mail a very wide reply to the author of the current article and include the original message (`gnus-summary-very-wide-reply-with-original`). This command uses the process/prefix convention.
- S B r* Mail a reply to the author of the current article but ignore the Reply-To field (`gnus-summary-reply-broken-reply-to`). If you need this because a mailing list incorrectly sets a Reply-To header pointing to the list, you probably want to set the `broken-reply-to` group parameter instead, so things will work correctly. See [\[Group Parameters\]](#), page [\[Group Parameters\]](#).
- S B R* Mail a reply to the author of the current article and include the original message but ignore the Reply-To field (`gnus-summary-reply-broken-reply-to-with-original`).
- S o m*  
*C-c C-f* Forward the current article to some other person (`gnus-summary-mail-forward`). If no prefix is given, the message is forwarded according to the value of (`message-forward-as-mime`) and (`message-forward-show-mml`); if the prefix is 1, decode the message and forward directly inline; if the prefix is 2, forward message as an rfc822 MIME section; if the prefix is 3, decode message and forward as an rfc822 MIME section; if the prefix is 4, forward message directly inline; otherwise, the message is forwarded as no prefix given but use the flipped value of (`message-forward-as-mime`). By default, the message is decoded and forwarded as an rfc822 MIME section.
- S m*  
*m* Prepare a mail (`gnus-summary-mail-other-window`). By default, use the posting style of the current group. If given a prefix, disable that. If the prefix is 1, prompt for a group name to find the posting style.
- S i*  
*i* Prepare a news (`gnus-summary-news-other-window`). By default, post to the current group. If given a prefix, disable that. If the prefix is 1, prompt for a group to post to.



This function actually prepares a news even when using mail groups. This is useful for “posting” messages to mail groups without actually sending them over the network: they’re just saved directly to the group in question. The corresponding back end must have a request-post method for this to work though.

***S D b*** If you have sent a mail, but the mail was bounced back to you for some reason (wrong address, transient failure), you can use this command to resend that bounced mail (`gnus-summary-resend-bounced-mail`). You will be popped into a mail buffer where you can edit the headers before sending the mail off again. If you give a prefix to this command, and the bounced mail is a reply to some other mail, Gnus will try to fetch that mail and display it for easy perusal of its headers. This might very well fail, though.

***S D r*** Not to be confused with the previous command, `gnus-summary-resend-message` will prompt you for an address to send the current message off to, and then send it to that place. The headers of the message won’t be altered—but lots of headers that say **Resent-To**, **Resent-From** and so on will be added. This means that you actually send a mail to someone that has a **To** header that (probably) points to yourself. This will confuse people. So, natchery you’ll only do that if you’re really eVil.

This command is mainly used if you have several accounts and want to ship a mail to a different account of yours. (If you’re both `root` and `postmaster` and get a mail for `postmaster` to the `root` account, you may want to resend it to `postmaster`. Ordnung muss sein!

This command understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\[Process/Prefix\]](#)).

***S D e***

Like the previous command, but will allow you to edit the message as if it were a new message before resending.

***S O m*** Digest the current series (see [\[Decoding Articles\]](#), page [\[Decoding Articles\]](#)) and forward the result using mail (`gnus-uu-digest-mail-forward`). This command uses the process/prefix convention (see [\[Process/Prefix\]](#), page [\[Process/Prefix\]](#)).

***S M-c*** Send a complaint about excessive crossposting to the author of the current article (`gnus-summary-mail-crosspost-complaint`).

This command is provided as a way to fight back against the current crossposting pandemic that’s sweeping Usenet. It will compose a reply using the `gnus-crosspost-complaint` variable as a preamble. This command understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\[Process/Prefix\]](#)) and will prompt you before sending each mail.

Also See [section “Header Commands”](#) in *The Message Manual*, for more information.

### 3.5.2 Summary Post Commands

Commands for posting a news article:

<i>S p</i> <i>a</i>	Prepare for posting an article ( <code>gnus-summary-post-news</code> ). By default, post to the current group. If given a prefix, disable that. If the prefix is 1, prompt for another group instead.
<i>S f</i> <i>f</i>	Post a followup to the current article ( <code>gnus-summary-followup</code> ).
<i>S F</i> <i>F</i>	Post a followup to the current article and include the original message ( <code>gnus-summary-followup-with-original</code> ). This command uses the process/prefix convention.
<i>S n</i>	Post a followup to the current article via news, even if you got the message through mail ( <code>gnus-summary-followup-to-mail</code> ).
<i>S N</i>	Post a followup to the current article via news, even if you got the message through mail and include the original message ( <code>gnus-summary-followup-to-mail-with-original</code> ). This command uses the process/prefix convention.
<i>S o p</i>	Forward the current article to a newsgroup ( <code>gnus-summary-post-forward</code> ). If no prefix is given, the message is forwarded according to the value of ( <code>message-forward-as-mime</code> ) and ( <code>message-forward-show-mml</code> ); if the prefix is 1, decode the message and forward directly inline; if the prefix is 2, forward message as an rfc822 MIME section; if the prefix is 3, decode message and forward as an rfc822 MIME section; if the prefix is 4, forward message directly inline; otherwise, the message is forwarded as no prefix given but use the flipped value of ( <code>message-forward-as-mime</code> ). By default, the message is decoded and forwarded as an rfc822 MIME section.
<i>S O p</i>	Digest the current series and forward the result to a newsgroup ( <code>gnus-uu-digest-mail-forward</code> ). This command uses the process/prefix convention.
<i>S u</i>	Uencode a file, split it into parts, and post it as a series ( <code>gnus-uu-post-news</code> ). (see <a href="#">[Uencoding and Posting]</a> , page <a href="#">[undefined]</a> ).

Also See [section “Header Commands” in \*The Message Manual\*](#), for more information.

### 3.5.3 Summary Message Commands

<i>S y</i>	Yank the current article into an already existing Message composition buffer ( <code>gnus-summary-yank-message</code> ). This command prompts for what message buffer you want to yank into, and understands the process/prefix convention (see <a href="#">[Process/Prefix]</a> , page <a href="#">[undefined]</a> ).
------------	--

### 3.5.4 Canceling Articles

Have you ever written something, and then decided that you really, really, really wish you hadn't posted that?

Well, you can't cancel mail, but you can cancel posts.

Find the article you wish to cancel (you can only cancel your own articles, so don't try any funny stuff). Then press *C* or *S c* (`gnus-summary-cancel-article`). Your article will

be canceled—machines all over the world will be deleting your article. This command uses the process/prefix convention (see [\[Process/Prefix\]](#), page [\[undefined\]](#)).

Be aware, however, that not all sites honor cancels, so your article may live on here and there, while most sites will delete the article in question.

Gnus will use the “current” select method when canceling. If you want to use the standard posting method, use the ‘a’ symbolic prefix (see [\[Symbolic Prefixes\]](#), page [\[undefined\]](#)).

Gnus ensures that only you can cancel your own messages using a **Cancel-Lock** header (see [section “Canceling News” in Message Manual](#)).

If you discover that you have made some mistakes and want to do some corrections, you can post a *superseding* article that will replace your original article.

Go to the original article and press **S s** (**gnus-summary-supersede-article**). You will be put in a buffer where you can edit the article all you want before sending it off the usual way.

The same goes for superseding as for canceling, only more so: Some sites do not honor superseding. On those sites, it will appear that you have posted almost the same article twice.

If you have just posted the article, and change your mind right away, there is a trick you can use to cancel/supersede the article without waiting for the article to appear on your site first. You simply return to the post buffer (which is called **\*sent ...\***). There you will find the article you just posted, with all the headers intact. Change the **Message-ID** header to a **Cancel** or **Supersedes** header by substituting one of those words for the word **Message-ID**. Then just press **C-c C-c** to send the article as you would do normally. The previous article will be canceled/superseded.

Just remember, kids: There is no ‘c’ in ‘supersede’.

## 3.6 Delayed Articles

Sometimes, you might wish to delay the sending of a message. For example, you might wish to arrange for a message to turn up just in time to remind you about the birthday of your Significant Other. For this, there is the **gnus-delay** package. Setup is simple:

```
(gnus-delay-initialize)
```

Normally, to send a message you use the **C-c C-c** command from Message mode. To delay a message, use **C-c C-j** (**gnus-delay-article**) instead. This will ask you for how long the message should be delayed. Possible answers are:

- A time span. Consists of an integer and a letter. For example, **42d** means to delay for 42 days. Available letters are **m** (minutes), **h** (hours), **d** (days), **w** (weeks), **M** (months) and **Y** (years).
- A specific date. Looks like **YYYY-MM-DD**. The message will be delayed until that day, at a specific time (eight o’clock by default). See also **gnus-delay-default-hour**.
- A specific time of day. Given in **hh:mm** format, 24h, no am/pm stuff. The deadline will be at that time today, except if that time has already passed, then it’s at the given time tomorrow. So if it’s ten o’clock in the morning and you specify **11:15**, then the deadline is one hour and fifteen minutes hence. But if you specify **9:20**, that means a time tomorrow.

The action of the `gnus-delay-article` command is influenced by a couple of variables:

**gnus-delay-default-hour**

When you specify a specific date, the message will be due on that hour on the given date. Possible values are integers 0 through 23.

**gnus-delay-default-delay**

This is a string and gives the default delay. It can be of any of the formats described above.

**gnus-delay-group**

Delayed articles will be kept in this group on the drafts server until they are due. You probably don't need to change this. The default value is `"delayed"`.

**gnus-delay-header**

The deadline for each article will be stored in a header. This variable is a string and gives the header name. You probably don't need to change this. The default value is `"X-Gnus-Delayed"`.

The way delaying works is like this: when you use the `gnus-delay-article` command, you give a certain delay. Gnus calculates the deadline of the message and stores it in the `X-Gnus-Delayed` header and puts the message in the `nn draft:delayed` group.

And whenever you get new news, Gnus looks through the group for articles which are due and sends them. It uses the `gnus-delay-send-queue` function for this. By default, this function is added to the hook `gnus-get-new-news-hook`. But of course, you can change this. Maybe you want to use the demon to send drafts? Just tell the demon to execute the `gnus-delay-send-queue` function.

**gnus-delay-initialize**

By default, this function installs `gnus-delay-send-queue` in `gnus-get-new-news-hook`. But it accepts the optional second argument `no-check`. If it is non-`nil`, `gnus-get-new-news-hook` is not changed. The optional first argument is ignored.

For example, `(gnus-delay-initialize nil t)` means to do nothing. Presumably, you want to use the demon for sending due delayed articles. Just don't forget to set that up :-)

## 3.7 Marking Articles

There are several marks you can set on an article.

You have marks that decide the *readedness* (whoo, neato-keano neologism ohoy!) of the article. Alphabetic marks generally mean *read*, while non-alphabetic characters generally mean *unread*.

In addition, you also have marks that do not affect readedness.

### 3.7.1 Unread Articles

The following marks mark articles as (kinda) unread, in one form or other.

‘!’           Marked as ticked (`gnus-ticked-mark`).

*Ticked articles* are articles that will remain visible always. If you see an article that you find interesting, or you want to put off reading it, or replying to it, until sometime later, you'd typically tick it. However, articles can be expired (from news servers by the news server software, Gnus itself never expires ticked messages), so if you want to keep an article forever, you'll have to make it persistent (see [\[Persistent Articles\]](#), page [\[undefined\]](#)).

'?' Marked as dormant (`gnus-dormant-mark`).

*Dormant articles* will only appear in the summary buffer if there are followups to it. If you want to see them even if they don't have followups, you can use the `/D` command (see [\[Limiting\]](#), page [\[undefined\]](#)). Otherwise (except for the visibility issue), they are just like ticked messages.

'SPACE' Marked as unread (`gnus-unread-mark`).

*Unread articles* are articles that haven't been read at all yet.

### 3.7.2 Read Articles

All the following marks mark articles as read.

'r' These are articles that the user has marked as read with the `d` command manually, more or less (`gnus-del-mark`).

'R' Articles that have actually been read (`gnus-read-mark`).

'O' Articles that were marked as read in previous sessions and are now *old* (`gnus-ancient-mark`).

'K' Marked as killed (`gnus-killed-mark`).

'X' Marked as killed by kill files (`gnus-kill-file-mark`).

'Y' Marked as read by having too low a score (`gnus-low-score-mark`).

'C' Marked as read by a catchup (`gnus-catchup-mark`).

'G' Canceled article (`gnus-canceled-mark`)

'F' SOUPed article (`gnus-souped-mark`). See [\[SOUP\]](#), page [\[undefined\]](#).

'Q' Sparsely reffed article (`gnus-sparse-mark`). See [\[Customizing Threading\]](#), page [\[undefined\]](#).

'M' Article marked as read by duplicate suppression (`gnus-duplicate-mark`). See [\[Duplicate Suppression\]](#), page [\[undefined\]](#).

All these marks just mean that the article is marked as read, really. They are interpreted differently when doing adaptive scoring, though.

One more special mark, though:

'E' Marked as expirable (`gnus-expirable-mark`).

Marking articles as *expirable* (or have them marked as such automatically) doesn't make much sense in normal groups—a user doesn't control expiring of news articles, but in mail groups, for instance, articles marked as *expirable* can be deleted by Gnus at any time.

### 3.7.3 Other Marks

There are some marks that have nothing to do with whether the article is read or not.

- You can set a bookmark in the current article. Say you are reading a long thesis on cats' urinary tracts, and have to go home for dinner before you've finished reading the thesis. You can then set a bookmark in the article, and Gnus will jump to this bookmark the next time it encounters the article. See [\[Setting Marks\]](#), page [\[undefined\]](#).
- All articles that you have replied to or made a followup to (i.e., have answered) will be marked with an 'A' in the second column (`gnus-replied-mark`).
- All articles that you have forwarded will be marked with an 'F' in the second column (`gnus-forwarded-mark`).
- Articles stored in the article cache will be marked with an '\*' in the second column (`gnus-cached-mark`). See [\[Article Caching\]](#), page [\[undefined\]](#).
- Articles "saved" (in some manner or other; not necessarily religiously) are marked with an 'S' in the second column (`gnus-saved-mark`).
- Articles that according to the server haven't been shown to the user before are marked with a 'N' in the second column (`gnus-recent-mark`). Note that not all servers support this mark, in which case it simply never appears. Compare with `gnus-unseen-mark`.
- Articles that haven't been seen before in Gnus by the user are marked with a '.' in the second column (`gnus-unseen-mark`). Compare with `gnus-recent-mark`.
- When using the Gnus agent (see [\[Agent Basics\]](#), page [\[undefined\]](#)), articles may be downloaded for unplugged (offline) viewing. If you are using the '%O' spec, these articles get the '+' mark in that spec. (The variable `gnus-downloaded-mark` controls which character to use.)
- When using the Gnus agent (see [\[Agent Basics\]](#), page [\[undefined\]](#)), some articles might not have been downloaded. Such articles cannot be viewed while you are unplugged (offline). If you are using the '%O' spec, these articles get the '-' mark in that spec. (The variable `gnus-undownloaded-mark` controls which character to use.)
- The Gnus agent (see [\[Agent Basics\]](#), page [\[undefined\]](#)) downloads some articles automatically, but it is also possible to explicitly mark articles for download, even if they would not be downloaded automatically. Such explicitly-marked articles get the '%' mark in the first column. (The variable `gnus-downloadable-mark` controls which character to use.)
- If the '%e' spec is used, the presence of threads or not will be marked with `gnus-not-empty-thread-mark` and `gnus-empty-thread-mark` in the third column, respectively.
- Finally we have the *process mark* (`gnus-process-mark`). A variety of commands react to the presence of the process mark. For instance, `X u` (`gnus-uu-decode-uu`) will uudecode and view all articles that have been marked with the process mark. Articles marked with the process mark have a '#' in the second column.

You might have noticed that most of these "non-readedness" marks appear in the second column by default. So if you have a cached, saved, replied article that you have process-marked, what will that look like?

Nothing much. The precedence rules go as follows: process -> cache -> replied -> saved. So if the article is in the cache and is replied, you'll only see the cache mark and not the replied mark.

### 3.7.4 Setting Marks

All the marking commands understand the numeric prefix.

<i>M c</i>	
<i>M-u</i>	Clear all readedness-marks from the current article ( <code>gnus-summary-clear-mark-forward</code> ). In other words, mark the article as unread.
<i>M t</i>	
<i>!</i>	Tick the current article ( <code>gnus-summary-tick-article-forward</code> ). See <a href="#">(undefined)</a> [Article Caching], page <a href="#">(undefined)</a> .
<i>M ?</i>	
<i>?</i>	Mark the current article as dormant ( <code>gnus-summary-mark-as-dormant</code> ). See <a href="#">(undefined)</a> [Article Caching], page <a href="#">(undefined)</a> .
<i>M d</i>	
<i>d</i>	Mark the current article as read ( <code>gnus-summary-mark-as-read-forward</code> ).
<i>D</i>	Mark the current article as read and move point to the previous line ( <code>gnus-summary-mark-as-read-backward</code> ).
<i>M k</i>	
<i>k</i>	Mark all articles that have the same subject as the current one as read, and then select the next unread article ( <code>gnus-summary-kill-same-subject-and-select</code> ).
<i>M K</i>	
<i>C-k</i>	Mark all articles that have the same subject as the current one as read ( <code>gnus-summary-kill-same-subject</code> ).
<i>M C</i>	Mark all unread articles as read ( <code>gnus-summary-catchup</code> ).
<i>M C-c</i>	Mark all articles in the group as read—even the ticked and dormant articles ( <code>gnus-summary-catchup-all</code> ).
<i>M H</i>	Catchup the current group to point (before the point) ( <code>gnus-summary-catchup-to-here</code> ).
<i>M h</i>	Catchup the current group from point (after the point) ( <code>gnus-summary-catchup-from-here</code> ).
<i>C-w</i>	Mark all articles between point and mark as read ( <code>gnus-summary-mark-region-as-read</code> ).
<i>M V k</i>	Kill all articles with scores below the default score (or below the numeric prefix) ( <code>gnus-summary-kill-below</code> ).
<i>M e</i>	
<i>E</i>	Mark the current article as expirable ( <code>gnus-summary-mark-as-expirable</code> ).
<i>M b</i>	Set a bookmark in the current article ( <code>gnus-summary-set-bookmark</code> ).



<i>M B</i>	Remove the bookmark from the current article ( <code>gnus-summary-remove-bookmark</code> ).
<i>M V c</i>	Clear all marks from articles with scores over the default score (or over the numeric prefix) ( <code>gnus-summary-clear-above</code> ).
<i>M V u</i>	Tick all articles with scores over the default score (or over the numeric prefix) ( <code>gnus-summary-tick-above</code> ).
<i>M V m</i>	Prompt for a mark, and mark all articles with scores over the default score (or over the numeric prefix) with this mark ( <code>gnus-summary-clear-above</code> ).

The `gnus-summary-goto-unread` variable controls what action should be taken after setting a mark. If non-`nil`, point will move to the next/previous unread article. If `nil`, point will just move one line up or down. As a special case, if this variable is `never`, all the marking commands as well as other commands (like `SPACE`) will move to the next article, whether it is unread or not. The default is `t`.

### 3.7.5 Generic Marking Commands

Some people would like the command that ticks an article (!) go to the next article. Others would like it to go to the next unread article. Yet others would like it to stay on the current article. And even though I haven't heard of anybody wanting it to go to the previous (unread) article, I'm sure there are people that want that as well.

Multiply these five behaviors with five different marking commands, and you get a potentially complex set of variable to control what each command should do.

To sidestep that mess, Gnus provides commands that do all these different things. They can be found on the *MM* map in the summary buffer. Type *MM C-h* to see them all—there are too many of them to list in this manual.

While you can use these commands directly, most users would prefer altering the summary mode keymap. For instance, if you would like the `!` command to go to the next article instead of the next unread article, you could say something like:

```
(add-hook 'gnus-summary-mode-hook 'my-alter-summary-map)
(defun my-alter-summary-map ()
  (local-set-key "!" 'gnus-summary-put-mark-as-ticked-next))
```

or

```
(defun my-alter-summary-map ()
  (local-set-key "!" "MM!n"))
```

### 3.7.6 Setting Process Marks

Process marks are displayed as `#` in the summary buffer, and are used for marking articles in such a way that other commands will process these articles. For instance, if you process mark four articles and then use the `*` command, Gnus will enter these four commands into the cache. For more information, see [\[Process/Prefix\]](#), page [\[undefined\]](#).

<i>M P p</i>	
<i>#</i>	Mark the current article with the process mark ( <code>gnus-summary-mark-as-processable</code> ).



<i>M P u</i>	
<i>M-#</i>	Remove the process mark, if any, from the current article ( <code>gnus-summary-unmark-as-processable</code> ).
<i>M P U</i>	Remove the process mark from all articles ( <code>gnus-summary-unmark-all-processable</code> ).
<i>M P i</i>	Invert the list of process marked articles ( <code>gnus-uu-invert-processable</code> ).
<i>M P R</i>	Mark articles that have a <b>Subject</b> header that matches a regular expression ( <code>gnus-uu-mark-by-regexp</code> ).
<i>M P G</i>	Unmark articles that have a <b>Subject</b> header that matches a regular expression ( <code>gnus-uu-unmark-by-regexp</code> ).
<i>M P r</i>	Mark articles in region ( <code>gnus-uu-mark-region</code> ).
<i>M P g</i>	Unmark articles in region ( <code>gnus-uu-unmark-region</code> ).
<i>M P t</i>	Mark all articles in the current (sub)thread ( <code>gnus-uu-mark-thread</code> ).
<i>M P T</i>	Unmark all articles in the current (sub)thread ( <code>gnus-uu-unmark-thread</code> ).
<i>M P v</i>	Mark all articles that have a score above the prefix argument ( <code>gnus-uu-mark-over</code> ).
<i>M P s</i>	Mark all articles in the current series ( <code>gnus-uu-mark-series</code> ).
<i>M P S</i>	Mark all series that have already had some articles marked ( <code>gnus-uu-mark-sparse</code> ).
<i>M P a</i>	Mark all articles in series order ( <code>gnus-uu-mark-all</code> ).
<i>M P b</i>	Mark all articles in the buffer in the order they appear ( <code>gnus-uu-mark-buffer</code> ).
<i>M P k</i>	Push the current process mark set onto the stack and unmark all articles ( <code>gnus-summary-kill-process-mark</code> ).
<i>M P y</i>	Pop the previous process mark set from the stack and restore it ( <code>gnus-summary-yank-process-mark</code> ).
<i>M P w</i>	Push the current process mark set onto the stack ( <code>gnus-summary-save-process-mark</code> ).

Also see the `&` command in [\[Searching for Articles\]](#), page [\[undefined\]](#), for how to set process marks based on article body contents.

### 3.8 Limiting

It can be convenient to limit the summary buffer to just show some subset of the articles currently in the group. The effect most limit commands have is to remove a few (or many) articles from the summary buffer.

All limiting commands work on subsets of the articles already fetched from the servers. None of these commands query the server for additional articles.

<i>//</i>	
<i>/ s</i>	Limit the summary buffer to articles that match some subject ( <code>gnus-summary-limit-to-subject</code> ). If given a prefix, exclude matching articles.

- `/ a` Limit the summary buffer to articles that match some author (`gnus-summary-limit-to-author`). If given a prefix, exclude matching articles.
- `/ x` Limit the summary buffer to articles that match one of the “extra” headers (see [<undefined> \[To From Newsgroups\], page <undefined>](#)) (`gnus-summary-limit-to-extra`). If given a prefix, exclude matching articles.
- `/ u`  
`x` Limit the summary buffer to articles not marked as read (`gnus-summary-limit-to-unread`). If given a prefix, limit the buffer to articles strictly unread. This means that ticked and dormant articles will also be excluded.
- `/ m` Ask for a mark and then limit to all articles that have been marked with that mark (`gnus-summary-limit-to-marks`).
- `/ t` Ask for a number and then limit the summary buffer to articles older than (or equal to) that number of days (`gnus-summary-limit-to-age`). If given a prefix, limit to articles younger than that number of days.
- `/ n` Limit the summary buffer to the current article (`gnus-summary-limit-to-articles`). Uses the process/prefix convention (see [<undefined> \[Process/Prefix\], page <undefined>](#)).
- `/ w` Pop the previous limit off the stack and restore it (`gnus-summary-pop-limit`). If given a prefix, pop all limits off the stack.
- `/ .` Limit the summary buffer to the unseen articles (`gnus-summary-limit-to-unseen`).
- `/ v` Limit the summary buffer to articles that have a score at or above some score (`gnus-summary-limit-to-score`).
- `/ p` Limit the summary buffer to articles that satisfy the `display` group parameter predicate (`gnus-summary-limit-to-display-predicate`). See [<undefined> \[Group Parameters\], page <undefined>](#), for more on this predicate.
- `/ E`  
`M S` Include all expunged articles in the limit (`gnus-summary-limit-include-expunged`).
- `/ D` Include all dormant articles in the limit (`gnus-summary-limit-include-dormant`).
- `/ *` Include all cached articles in the limit (`gnus-summary-limit-include-cached`).
- `/ d` Exclude all dormant articles from the limit (`gnus-summary-limit-exclude-dormant`).
- `/ M` Exclude all marked articles (`gnus-summary-limit-exclude-marks`).
- `/ T` Include all the articles in the current thread in the limit.
- `/ c` Exclude all dormant articles that have no children from the limit (`gnus-summary-limit-exclude-childless-dormant`).

<code>/ C</code>	Mark all excluded unread articles as read ( <code>gnus-summary-limit-mark-excluded-as-read</code> ). If given a prefix, also mark excluded ticked and dormant articles as read.
<code>/ N</code>	Insert all new articles in the summary buffer. It scans for new emails if <code>back-end-get-new-mail</code> is non- <code>nil</code> .
<code>/ o</code>	Insert all old articles in the summary buffer. If given a numbered prefix, fetch this number of articles.

## 3.9 Threading

Gnus threads articles by default. *To thread* is to put responses to articles directly after the articles they respond to—in a hierarchical fashion.

Threading is done by looking at the **References** headers of the articles. In a perfect world, this would be enough to build pretty trees, but unfortunately, the **References** header is often broken or simply missing. Weird news propagation exacerbates the problem, so one has to employ other heuristics to get pleasing results. A plethora of approaches exists, as detailed in horrible detail in [\[Customizing Threading\]](#), page [\[undefined\]](#).

First, a quick overview of the concepts:

<i>root</i>	The top-most article in a thread; the first article in the thread.
<i>thread</i>	A tree-like article structure.
<i>sub-thread</i>	A small(er) section of this tree-like structure.
<i>loose threads</i>	Threads often lose their roots due to article expiry, or due to the root already having been read in a previous session, and not displayed in the summary buffer. We then typically have many sub-threads that really belong to one thread, but are without connecting roots. These are called loose threads.
<i>thread gathering</i>	An attempt to gather loose threads into bigger threads.
<i>sparse threads</i>	A thread where the missing articles have been “guessed” at, and are displayed as empty lines in the summary buffer.

### 3.9.1 Customizing Threading

#### 3.9.1.1 Loose Threads

##### `gnus-summary-make-false-root`

If non-`nil`, Gnus will gather all loose subtrees into one big tree and create a dummy root at the top. (Wait a minute. Root at the top? Yup.) Loose subtrees occur when the real root has expired, or you’ve read or killed the root in a previous session.

When there is no real root of a thread, Gnus will have to fudge something. This variable says what fudging method Gnus should use. There are four possible values:

```

\gnusfigureThe      Summary      Buffer390      \put(0,0)\epsfigfigure=ps/summary-
adopt,width=7.5cm    \put(445,0)\makebox(0,0)[br]\epsfigfigure=ps/summary-
empty,width=7.5cm    \put(0,400)\makebox(0,0)[tl]\epsfigfigure=ps/summary-
none,width=7.5cm     \put(445,400)\makebox(0,0)[tr]\epsfigfigure=ps/summary-
dummy,width=7.5cm

```

- adopt** Gnus will make the first of the orphaned articles the parent. This parent will adopt all the other articles. The adopted articles will be marked as such by pointy brackets (`<>`) instead of the standard square brackets (`[]`). This is the default method.
- dummy** Gnus will create a dummy summary line that will pretend to be the parent. This dummy line does not correspond to any real article, so selecting it will just select the first real article after the dummy article. `gnus-summary-dummy-line-format` is used to specify the format of the dummy roots. It accepts only one format spec: `'S'`, which is the subject of the article. See [\[Formatting Variables\]](#), page [\[undefined\]](#). If you want all threads to have a dummy root, even the non-gathered ones, set `gnus-summary-make-false-root-always` to `t`.
- empty** Gnus won't actually make any article the parent, but simply leave the subject field of all orphans except the first empty. (Actually, it will use `gnus-summary-same-subject` as the subject (see [\[Summary Buffer Format\]](#), page [\[undefined\]](#))).
- none** Don't make any article parent at all. Just gather the threads and display them after one another.
- nil** Don't gather loose threads.

`gnus-summary-gather-subject-limit` Loose threads are gathered by comparing subjects of articles. If this variable is `nil`, Gnus requires an exact match between the subjects of the loose threads before gathering them into one big super-thread. This might be too strict a requirement, what with the presence of stupid newsreaders that chop off long subject lines. If you think so, set this variable to, say, 20 to require that only the first 20 characters of the subjects have to match. If you set this variable to a really low number, you'll find that Gnus will gather everything in sight into one thread, which isn't very helpful.

If you set this variable to the special value `fuzzy`, Gnus will use a fuzzy string comparison algorithm on the subjects (see [\[Fuzzy Matching\]](#), page [\[undefined\]](#)).

`gnus-simplify-subject-fuzzy-regexp` This can either be a regular expression or list of regular expressions that match strings that will be removed from subjects if fuzzy subject simplification is used.

`gnus-simplify-ignored-prefixes` If you set `gnus-summary-gather-subject-limit` to something as low as 10, you might consider setting this variable to something sensible:

```

(setq gnus-simplify-ignored-prefixes
  (concat
    "\\['\\[?\\(\"
    (mapconcat
      'identity

```

```

'("looking"
  "wanted" "followup" "summary\\( of\\)?"
  "help" "query" "problem" "question"
  "answer" "reference" "announce"
  "How can I" "How to" "Comparison of"
  ;; ...
)
"\\|")
"\\)\\s *\\("
(mapconcat 'identity
  '("for" "for reference" "with" "about")
  "\\|")
"\\)?\\[\\]?[ \\t]*"))

```

All words that match this regexp will be removed before comparing two subjects.

**gnus-simplify-subject-functions** If non-*nil*, this variable overrides **gnus-summary-gather-subject-limit**. This variable should be a list of functions to apply to the **Subject** string iteratively to arrive at the simplified version of the string.

Useful functions to put in this list include:

```

gnus-simplify-subject-re
  Strip the leading 'Re:'.

gnus-simplify-subject-fuzzy
  Simplify fuzzily.

gnus-simplify-whitespace
  Remove excessive whitespace.

gnus-simplify-all-whitespace
  Remove all whitespace.

```

You may also write your own functions, of course.

**gnus-summary-gather-exclude-subject** Since loose thread gathering is done on subjects only, that might lead to many false hits, especially with certain common subjects like “” and “(none)”. To make the situation slightly better, you can use the regexp **gnus-summary-gather-exclude-subject** to say what subjects should be excluded from the gathering process.

The default is “`^ *$\\|^(none)$`”.

**gnus-summary-thread-gathering-function** Gnus gathers threads by looking at **Subject** headers. This means that totally unrelated articles may end up in the same “thread”, which is confusing. An alternate approach is to look at all the **Message-IDs** in all the **References** headers to find matches. This will ensure that no gathered threads ever include unrelated articles, but it also means that people who have posted with broken newsreaders won’t be gathered properly. The choice is yours—plague or cholera:

```

gnus-gather-threads-by-subject
  This function is the default gathering function and looks at Subjects exclusively.

```

**gnus-gather-threads-by-references**

This function looks at **References** headers exclusively.

If you want to test gathering by **References**, you could say something like:

```
(setq gnus-summary-thread-gathering-function
      'gnus-gather-threads-by-references)
```

**3.9.1.2 Filling In Threads****gnus-fetch-old-headers**

If non-**nil**, Gnus will attempt to build old threads by fetching more old headers—headers to articles marked as read. If you would like to display as few summary lines as possible, but still connect as many loose threads as possible, you should set this variable to **some** or a number. If you set it to a number, no more than that number of extra old headers will be fetched. In either case, fetching old headers only works if the back end you are using carries overview files—this would normally be **nntp**, **nnsPOOL**, **nnml**, and **nnmaildir**. Also remember that if the root of the thread has been expired by the server, there's not much Gnus can do about that.

This variable can also be set to **invisible**. This won't have any visible effects, but is useful if you use the **A T** command a lot (see [\[Finding the Parent\]](#), page [\[undefined\]](#)).

**gnus-fetch-old-ephemeral-headers**

Same as **gnus-fetch-old-headers**, but only used for ephemeral newsgroups.

**gnus-build-sparse-threads**

Fetching old headers can be slow. A low-rent similar effect can be gotten by setting this variable to **some**. Gnus will then look at the complete **References** headers of all articles and try to string together articles that belong in the same thread. This will leave *gaps* in the threading display where Gnus guesses that an article is missing from the thread. (These gaps appear like normal summary lines. If you select a gap, Gnus will try to fetch the article in question.) If this variable is **t**, Gnus will display all these “gaps” without regard for whether they are useful for completing the thread or not. Finally, if this variable is **more**, Gnus won't cut off sparse leaf nodes that don't lead anywhere. This variable is **nil** by default.

**gnus-read-all-available-headers**

This is a rather obscure variable that few will find useful. It's intended for those non-news newsgroups where the back end has to fetch quite a lot to present the summary buffer, and where it's impossible to go back to parents of articles. This is mostly the case in the web-based groups, like the **nnultimate** groups.

If you don't use those, then it's safe to leave this as the default **nil**. If you want to use this variable, it should be a regexp that matches the group name, or **t** for all groups.

### 3.9.1.3 More Threading

#### `gnus-show-threads`

If this variable is `nil`, no threading will be done, and all of the rest of the variables here will have no effect. Turning threading off will speed group selection up a bit, but it is sure to make reading slower and more awkward.

#### `gnus-thread-hide-subtree`

If non-`nil`, all threads will be hidden when the summary buffer is generated.

This can also be a predicate specifier (see [\[Predicate Specifiers\]](#), page [\[undefined\]](#)). Available predicates are `gnus-article-unread-p` and `gnus-article-unseen-p`.

Here's an example:

```
(setq gnus-thread-hide-subtree
      '(or gnus-article-unread-p
           gnus-article-unseen-p))
```

(It's a pretty nonsensical example, since all unseen articles are also unread, but you get my drift.)

#### `gnus-thread-expunge-below`

All threads that have a total score (as defined by `gnus-thread-score-function`) less than this number will be expunged. This variable is `nil` by default, which means that no threads are expunged.

#### `gnus-thread-hide-killed`

if you kill a thread and this variable is non-`nil`, the subtree will be hidden.

#### `gnus-thread-ignore-subject`

Sometimes somebody changes the subject in the middle of a thread. If this variable is non-`nil`, which is the default, the subject change is ignored. If it is `nil`, a change in the subject will result in a new thread.

#### `gnus-thread-indent-level`

This is a number that says how much each sub-thread should be indented. The default is 4.

#### `gnus-sort-gathered-threads-function`

Sometimes, particularly with mailing lists, the order in which mails arrive locally is not necessarily the same as the order in which they arrived on the mailing list. Consequently, when sorting sub-threads using the default `gnus-thread-sort-by-number`, responses can end up appearing before the article to which they are responding to. Setting this variable to an alternate value (e.g. `gnus-thread-sort-by-date`), in a group's parameters or in an appropriate hook (e.g. `gnus-summary-generate-hook`) can produce a more logical sub-thread ordering in such instances.

### 3.9.1.4 Low-Level Threading

#### `gnus-parse-headers-hook`

Hook run before parsing any headers.

**gnus-alter-header-function**

If non-nil, this function will be called to allow alteration of article header structures. The function is called with one parameter, the article header vector, which it may alter in any way. For instance, if you have a mail-to-news gateway which alters the `Message-IDs` in systematic ways (by adding prefixes and such), you can use this variable to un-scramble the `Message-IDs` so that they are more meaningful. Here's one example:

```
(setq gnus-alter-header-function 'my-alter-message-id)

(defun my-alter-message-id (header)
  (let ((id (mail-header-id header)))
    (when (string-match
           "\\(<[^>@]*\\)\\.?cygnus\\.\\.\\.*@\\([^\>@]*>\\)" id)
      (mail-header-set-id
       (concat (match-string 1 id) "@" (match-string 2 id))
       header))))
```

**3.9.2 Thread Commands***T k*

*C-M-k* Mark all articles in the current (sub-)thread as read (`gnus-summary-kill-thread`). If the prefix argument is positive, remove all marks instead. If the prefix argument is negative, tick articles instead.

*T l*

*C-M-l* Lower the score of the current (sub-)thread (`gnus-summary-lower-thread`).

*T i*

Increase the score of the current (sub-)thread (`gnus-summary-raise-thread`).

*T #*

Set the process mark on the current (sub-)thread (`gnus-uu-mark-thread`).

*T M-#*

Remove the process mark from the current (sub-)thread (`gnus-uu-unmark-thread`).

*T T*

Toggle threading (`gnus-summary-toggle-threads`).

*T s*

Expose the (sub-)thread hidden under the current article, if any (`gnus-summary-show-thread`).

*T h*

Hide the current (sub-)thread (`gnus-summary-hide-thread`).

*T S*

Expose all hidden threads (`gnus-summary-show-all-threads`).

*T H*

Hide all threads (`gnus-summary-hide-all-threads`).

*T t*

Re-thread the current article's thread (`gnus-summary-rethread-current`). This works even when the summary buffer is otherwise unthreaded.

*T ^*

Make the current article the child of the marked (or previous) article (`gnus-summary-reparent-thread`).

The following commands are thread movement commands. They all understand the numeric prefix.



*T n*  
*C-M-f*  
*M-down*      Go to the next thread (`gnus-summary-next-thread`).  
  
*T p*  
*C-M-b*  
*M-up*        Go to the previous thread (`gnus-summary-prev-thread`).  
  
*T d*         Descend the thread (`gnus-summary-down-thread`).  
  
*T u*         Ascend the thread (`gnus-summary-up-thread`).  
  
*T o*         Go to the top of the thread (`gnus-summary-top-thread`).

If you ignore subject while threading, you'll naturally end up with threads that have several different subjects in them. If you then issue a command like *T k* (`gnus-summary-kill-thread`) you might not wish to kill the entire thread, but just those parts of the thread that have the same subject as the current article. If you like this idea, you can fiddle with `gnus-thread-operation-ignore-subject`. If it is non-`nil` (which it is by default), subjects will be ignored when doing thread commands. If this variable is `nil`, articles in the same thread with different subjects will not be included in the operation in question. If this variable is `fuzzy`, only articles that have subjects fuzzily equal will be included (see [\[Fuzzy Matching\]](#), page [\[Fuzzy Matching\]](#)).

### 3.10 Sorting the Summary Buffer

If you are using a threaded summary display, you can sort the threads by setting `gnus-thread-sort-functions`, which can be either a single function, a list of functions, or a list containing functions and (not some-function) elements.

By default, sorting is done on article numbers. Ready-made sorting predicate functions include `gnus-thread-sort-by-number`, `gnus-thread-sort-by-author`, `gnus-thread-sort-by-subject`, `gnus-thread-sort-by-date`, `gnus-thread-sort-by-score`, `gnus-thread-sort-by-most-recent-number`, `gnus-thread-sort-by-most-recent-date`, `gnus-thread-sort-by-random` and `gnus-thread-sort-by-total-score`.

Each function takes two threads and returns non-`nil` if the first thread should be sorted before the other. Note that sorting really is normally done by looking only at the roots of each thread.

If you use more than one function, the primary sort key should be the last function in the list. You should probably always include `gnus-thread-sort-by-number` in the list of sorting functions—preferably first. This will ensure that threads that are equal with respect to the other sort criteria will be displayed in ascending article order.

If you would like to sort by reverse score, then by subject, and finally by number, you could do something like:

```
(setq gnus-thread-sort-functions
      '(gnus-thread-sort-by-number
        gnus-thread-sort-by-subject
        (not gnus-thread-sort-by-total-score)))
```

The threads that have highest score will be displayed first in the summary buffer. When threads have the same score, they will be sorted alphabetically. The threads that have the

same score and the same subject will be sorted by number, which is (normally) the sequence in which the articles arrived.

If you want to sort by score and then reverse arrival order, you could say something like:

```
(setq gnus-thread-sort-functions
      '((lambda (t1 t2)
          (not (gnus-thread-sort-by-number t1 t2)))
        gnus-thread-sort-by-score))
```

The function in the `gnus-thread-score-function` variable (default `+`) is used for calculating the total score of a thread. Useful functions might be `max`, `min`, or squared means, or whatever tickles your fancy.

If you are using an unthreaded display for some strange reason or other, you have to fiddle with the `gnus-article-sort-functions` variable. It is very similar to the `gnus-thread-sort-functions`, except that it uses slightly different functions for article comparison. Available sorting predicate functions are `gnus-article-sort-by-number`, `gnus-article-sort-by-author`, `gnus-article-sort-by-subject`, `gnus-article-sort-by-date`, `gnus-article-sort-by-random`, and `gnus-article-sort-by-score`.

If you want to sort an unthreaded summary display by subject, you could say something like:

```
(setq gnus-article-sort-functions
      '(gnus-article-sort-by-number
        gnus-article-sort-by-subject))
```

### 3.11 Asynchronous Article Fetching

If you read your news from an NNTP server that's far away, the network latencies may make reading articles a chore. You have to wait for a while after pressing `n` to go to the next article before the article appears. Why can't Gnus just go ahead and fetch the article while you are reading the previous one? Why not, indeed.

First, some caveats. There are some pitfalls to using asynchronous article fetching, especially the way Gnus does it.

Let's say you are reading article 1, which is short, and article 2 is quite long, and you are not interested in reading that. Gnus does not know this, so it goes ahead and fetches article 2. You decide to read article 3, but since Gnus is in the process of fetching article 2, the connection is blocked.

To avoid these situations, Gnus will open two (count 'em two) connections to the server. Some people may think this isn't a very nice thing to do, but I don't see any real alternatives. Setting up that extra connection takes some time, so Gnus startup will be slower.

Gnus will fetch more articles than you will read. This will mean that the link between your machine and the NNTP server will become more loaded than if you didn't use article pre-fetch. The server itself will also become more loaded—both with the extra article requests, and the extra connection.

Ok, so now you know that you shouldn't really use this thing. . . unless you really want to.

Here's how: Set `gnus-asynchronous` to `t`. The rest should happen automatically.

You can control how many articles are to be pre-fetched by setting `gnus-use-article-prefetch`. This is 30 by default, which means that when you read an article in the group, the back end will pre-fetch the next 30 articles. If this variable is `t`, the back end will pre-fetch all the articles it can without bound. If it is `nil`, no pre-fetching will be done.

There are probably some articles that you don't want to pre-fetch—read articles, for instance. The `gnus-async-prefetch-article-p` variable controls whether an article is to be pre-fetched. This function should return non-`nil` when the article in question is to be pre-fetched. The default is `gnus-async-read-p`, which returns `nil` on read articles. The function is called with an article data structure as the only parameter.

If, for instance, you wish to pre-fetch only unread articles shorter than 100 lines, you could say something like:

```
(defun my-async-short-unread-p (data)
  "Return non-nil for short, unread articles."
  (and (gnus-data-unread-p data)
        (< (mail-header-lines (gnus-data-header data))
            100)))

(setq gnus-async-prefetch-article-p 'my-async-short-unread-p)
```

These functions will be called many, many times, so they should preferably be short and sweet to avoid slowing down Gnus too much. It's probably a good idea to byte-compile things like this.

Articles have to be removed from the asynch buffer sooner or later. The `gnus-prefetched-article-deletion-strategy` says when to remove articles. This is a list that may contain the following elements:

`read`        Remove articles when they are read.  
`exit`        Remove articles when exiting the group.

The default value is `(read exit)`.

### 3.12 Article Caching

If you have an *extremely* slow NNTP connection, you may consider turning article caching on. Each article will then be stored locally under your home directory. As you may surmise, this could potentially use *huge* amounts of disk space, as well as eat up all your inodes so fast it will make your head swim. In vodka.

Used carefully, though, it could be just an easier way to save articles.

To turn caching on, set `gnus-use-cache` to `t`. By default, all articles ticked or marked as dormant will then be copied over to your local cache (`gnus-cache-directory`). Whether this cache is flat or hierarchical is controlled by the `gnus-use-long-file-name` variable, as usual.

When re-selecting a ticked or dormant article, it will be fetched from the cache instead of from the server. As articles in your cache will never expire, this might serve as a method of saving articles while still keeping them where they belong. Just mark all articles you want to save as dormant, and don't worry.

When an article is marked as read, is it removed from the cache.

The entering/removal of articles from the cache is controlled by the `gnus-cache-enter-articles` and `gnus-cache-remove-articles` variables. Both are lists of symbols. The first is (`ticked dormant`) by default, meaning that ticked and dormant articles will be put in the cache. The latter is (`read`) by default, meaning that articles marked as read are removed from the cache. Possibly symbols in these two lists are `ticked`, `dormant`, `unread` and `read`.

So where does the massive article-fetching and storing come into the picture? The `gnus-jog-cache` command will go through all subscribed newsgroups, request all unread articles, score them, and store them in the cache. You should only ever, ever ever ever, use this command if 1) your connection to the NNTP server is really, really, really slow and 2) you have a really, really, really huge disk. Seriously. One way to cut down on the number of articles downloaded is to score unwanted articles down and have them marked as read. They will not then be downloaded by this command.

It is likely that you do not want caching on all groups. For instance, if your `nnml` mail is located under your home directory, it makes no sense to cache it somewhere else under your home directory. Unless you feel that it's neat to use twice as much space.

To limit the caching, you could set `gnus-cacheable-groups` to a regexp of groups to cache, `'^nntp'` for instance, or set the `gnus-uncacheable-groups` regexp to `'^nnml'`, for instance. Both variables are `nil` by default. If a group matches both variables, the group is not cached.

The cache stores information on what articles it contains in its active file (`gnus-cache-active-file`). If this file (or any other parts of the cache) becomes all messed up for some reason or other, Gnus offers two functions that will try to set things right. `M-x gnus-cache-generate-nov-databases` will (re)build all the NOV files, and `gnus-cache-generate-active` will (re)generate the active file.

`gnus-cache-move-cache` will move your whole `gnus-cache-directory` to some other location. You get asked to where, isn't that cool?

### 3.13 Persistent Articles

Closely related to article caching, we have *persistent articles*. In fact, it's just a different way of looking at caching, and much more useful in my opinion.

Say you're reading a newsgroup, and you happen on to some valuable gem that you want to keep and treasure forever. You'd normally just save it (using one of the many saving commands) in some file. The problem with that is that it's just, well, yucky. Ideally you'd prefer just having the article remain in the group where you found it forever; untouched by the expiry going on at the news server.

This is what a *persistent article* is—an article that just won't be deleted. It's implemented using the normal cache functions, but you use two explicit commands for managing persistent articles:

- \*           Make the current article persistent (`gnus-cache-enter-article`).
- M-\*       Remove the current article from the persistent articles (`gnus-cache-remove-article`). This will normally delete the article.

Both these commands understand the process/prefix convention.

To avoid having all ticked articles (and stuff) entered into the cache, you should set `gnus-use-cache` to `passive` if you're just interested in persistent articles:

```
(setq gnus-use-cache 'passive)
```

### 3.14 Article Backlog

If you have a slow connection, but the idea of using caching seems unappealing to you (and it is, really), you can help the situation some by switching on the *backlog*. This is where Gnus will buffer already read articles so that it doesn't have to re-fetch articles you've already read. This only helps if you are in the habit of re-selecting articles you've recently read, of course. If you never do that, turning the backlog on will slow Gnus down a little bit, and increase memory usage some.

If you set `gnus-keep-backlog` to a number *n*, Gnus will store at most *n* old articles in a buffer for later re-fetching. If this variable is non-`nil` and is not a number, Gnus will store *all* read articles, which means that your Emacs will grow without bound before exploding and taking your machine down with you. I put that in there just to keep y'all on your toes.

The default value is 20.

### 3.15 Saving Articles

Gnus can save articles in a number of ways. Below is the documentation for saving articles in a fairly straight-forward fashion (i.e., little processing of the article is done before it is saved). For a different approach (uudecoding, unsharing) you should use `gnus-uu` (see [\(undefined\) \[Decoding Articles\]](#), page [\(undefined\)](#)).

For the commands listed here, the target is a file. If you want to save to a group, see the *B c* (`gnus-summary-copy-article`) command (see [\(undefined\) \[Mail Group Commands\]](#), page [\(undefined\)](#)).

If `gnus-save-all-headers` is non-`nil`, Gnus will not delete unwanted headers before saving the article.

If the preceding variable is `nil`, all headers that match the `gnus-saved-headers` regexp will be kept, while the rest will be deleted before saving.

- O o*
- o*            Save the current article using the default article saver (`gnus-summary-save-article`).
- O m*            Save the current article in mail format (`gnus-summary-save-article-mail`).
- O r*            Save the current article in Rmail format (`gnus-summary-save-article-rmail`).
- O f*            Save the current article in plain file format (`gnus-summary-save-article-file`).
- O F*            Write the current article in plain file format, overwriting any previous file contents (`gnus-summary-write-article-file`).
- O b*            Save the current article body in plain file format (`gnus-summary-save-article-body-file`).
- O h*            Save the current article in mh folder format (`gnus-summary-save-article-folder`).

- `O v` Save the current article in a VM folder (`gnus-summary-save-article-vm`).
- `O p`  
| Save the current article in a pipe. Uhm, like, what I mean is—Pipe the current article to a process (`gnus-summary-pipe-output`). If given a symbolic prefix (see [\[Symbolic Prefixes\]](#), page [\[Symbolic Prefixes\]](#)), include the complete headers in the piped output.
- `O P` Save the current article into muttprint. That is, print it using the external program **Muttprint**. The program name and options to use is controlled by the variable `gnus-summary-muttprint-program`. (`gnus-summary-muttprint`).

All these commands use the process/prefix convention (see [\[Process/Prefix\]](#), page [\[Process/Prefix\]](#)). If you save bunches of articles using these functions, you might get tired of being prompted for files to save each and every article in. The prompting action is controlled by the `gnus-prompt-before-saving` variable, which is **always** by default, giving you that excessive prompting action you know and loathe. If you set this variable to `t` instead, you'll be prompted just once for each series of articles you save. If you like to really have Gnus do all your thinking for you, you can even set this variable to `nil`, which means that you will never be prompted for files to save articles in. Gnus will simply save all the articles in the default files.

You can customize the `gnus-default-article-saver` variable to make Gnus do what you want it to. You can use any of the six ready-made functions below, or you can create your own.

#### `gnus-summary-save-in-rmail`

This is the default format, *Babyl*. Uses the function in the `gnus-rmail-save-name` variable to get a file name to save the article in. The default is `gnus-plain-save-name`.

#### `gnus-summary-save-in-mail`

Save in a Unix mail (mbox) file. Uses the function in the `gnus-mail-save-name` variable to get a file name to save the article in. The default is `gnus-plain-save-name`.

#### `gnus-summary-save-in-file`

Append the article straight to an ordinary file. Uses the function in the `gnus-file-save-name` variable to get a file name to save the article in. The default is `gnus-numeric-save-name`.

#### `gnus-summary-write-to-file`

Write the article straight to an ordinary file. The file is overwritten if it exists. Uses the function in the `gnus-file-save-name` variable to get a file name to save the article in. The default is `gnus-numeric-save-name`.

#### `gnus-summary-save-body-in-file`

Append the article body to an ordinary file. Uses the function in the `gnus-file-save-name` variable to get a file name to save the article in. The default is `gnus-numeric-save-name`.

#### `gnus-summary-save-in-folder`

Save the article to an MH folder using `rcvstore` from the MH library. Uses the function in the `gnus-folder-save-name` variable to get a file name to save

the article in. The default is `gnus-folder-save-name`, but you can also use `gnus-Folder-save-name`, which creates capitalized names.

`gnus-summary-save-in-vm`

Save the article in a VM folder. You have to have the VM mail reader to use this setting.

All of these functions, except for the last one, will save the article in the `gnus-article-save-directory`, which is initialized from the `SAVEDIR` environment variable. This is `~/News/` by default.

As you can see above, the functions use different functions to find a suitable name of a file to save the article in. Below is a list of available functions that generate names:

`gnus-Numeric-save-name`

File names like `~/News/Alt.andrea-dworkin/45`.

`gnus-numeric-save-name`

File names like `~/News/alt.andrea-dworkin/45`.

`gnus-Plain-save-name`

File names like `~/News/Alt.andrea-dworkin`.

`gnus-plain-save-name`

File names like `~/News/alt.andrea-dworkin`.

`gnus-sender-save-name`

File names like `~/News/larsi`.

You can have Gnus suggest where to save articles by plonking a regexp into the `gnus-split-methods` alist. For instance, if you would like to save articles related to Gnus in the file `gnus-stuff`, and articles related to VM in `vm-stuff`, you could set this variable to something like:

```
((("^Subject:.*gnus\\|^Newsgroups:.*gnus" "gnus-stuff")
  ("^Subject:.*vm\\|^Xref:.*vm" "vm-stuff")
  (my-choosing-function "../other-dir/my-stuff")
  ((equal gnus-newsgroup-name "mail.misc") "mail-stuff")))
```

We see that this is a list where each element is a list that has two elements—the *match* and the *file*. The match can either be a string (in which case it is used as a regexp to match on the article head); it can be a symbol (which will be called as a function with the group name as a parameter); or it can be a list (which will be `eval`ed). If any of these actions have a non-`nil` result, the *file* will be used as a default prompt. In addition, the result of the operation itself will be used if the function or form called returns a string or a list of strings.

You basically end up with a list of file names that might be used when saving the current article. (All “matches” will be used.) You will then be prompted for what you really want to use as a name, with file name completion over the results from applying this variable.

This variable is `((gnus-article-archive-name))` by default, which means that Gnus will look at the articles it saves for an `Archive-name` line and use that as a suggestion for the file name.



Here’s an example function to clean up file names somewhat. If you have lots of mail groups called things like ‘nnml:mail.whatever’, you may want to chop off the beginning of these group names before creating the file name to save to. The following will do just that:

```
(defun my-save-name (group)
  (when (string-match "^nnml:mail." group)
    (substring group (match-end 0))))

(setq gnus-split-methods
  '((gnus-article-archive-name)
    (my-save-name)))
```

Finally, you have the `gnus-use-long-file-name` variable. If it is `nil`, all the preceding functions will replace all periods (‘.’) in the group names with slashes (‘/’)—which means that the functions will generate hierarchies of directories instead of having all the files in the top level directory (‘~/News/alt/andrea-dworkin’ instead of ‘~/News/alt.andrea-dworkin’.) This variable is `t` by default on most systems. However, for historical reasons, this is `nil` on Xenix and usg-unix-v machines by default.

This function also affects kill and score file names. If this variable is a list, and the list contains the element `not-score`, long file names will not be used for score files, if it contains the element `not-save`, long file names will not be used for saving, and if it contains the element `not-kill`, long file names will not be used for kill files.

If you’d like to save articles in a hierarchy that looks something like a spool, you could

```
(setq gnus-use-long-file-name '(not-save)) ; to get a hierarchy
(setq gnus-default-article-saver
  'gnus-summary-save-in-file) ; no encoding
```

Then just save with `o`. You’d then read this hierarchy with ephemeral `nneething` groups—`GD` in the group buffer, and the top level directory as the argument (‘~/News/’). Then just walk around to the groups/directories with `nneething`.

### 3.16 Decoding Articles

Sometime users post articles (or series of articles) that have been encoded in some way or other. Gnus can decode them for you.

All these functions use the process/prefix convention (see [\[Process/Prefix\]](#), [page \[undefined\]](#)) for finding out what articles to work on, with the extension that a “single article” means “a single series”. Gnus can find out by itself what articles belong to a series, decode all the articles and unpack/view/save the resulting file(s).

Gnus guesses what articles are in the series according to the following simplish rule: The subjects must be (nearly) identical, except for the last two numbers of the line. (Spaces are largely ignored, however.)

For example: If you choose a subject called ‘cat.gif (2/3)’, Gnus will find all the articles that match the regexp ‘^cat.gif ([0-9]+)/[0-9]+).\*\$’.

Subjects that are non-standard, like ‘cat.gif (2/3) Part 6 of a series’, will not be properly recognized by any of the automatic viewing commands, and you have to mark the articles manually with `#`.





### 3.16.5 Decoding Variables

Adjective, not verb.

#### 3.16.5.1 Rule Variables

Gnus uses *rule variables* to decide how to view a file. All these variables are of the form

```
(list '(regexp1 command2)
      '(regexp2 command2)
      ...)
```

##### gnus-uu-user-view-rules

This variable is consulted first when viewing files. If you wish to use, for instance, `sox` to convert an `.au` sound file, you could say something like:

```
(setq gnus-uu-user-view-rules
      (list '("\\\\.au$" "sox %s -t .aiff > /dev/audio")))
```

##### gnus-uu-user-view-rules-end

This variable is consulted if Gnus couldn't make any matches from the user and default view rules.

##### gnus-uu-user-archive-rules

This variable can be used to say what commands should be used to unpack archives.

#### 3.16.5.2 Other Decode Variables

##### gnus-uu-grabbed-file-functions

All functions in this list will be called right after each file has been successfully decoded—so that you can move or view files right away, and don't have to wait for all files to be decoded before you can do anything. Ready-made functions you can put in this list are:

##### gnus-uu-grab-view

View the file.

##### gnus-uu-grab-move

Move the file (if you're using a saving function.)

##### gnus-uu-be-dangerous

Specifies what to do if unusual situations arise during decoding. If `nil`, be as conservative as possible. If `t`, ignore things that didn't work, and overwrite existing files. Otherwise, ask each time.

##### gnus-uu-ignore-files-by-name

Files with name matching this regular expression won't be viewed.

##### gnus-uu-ignore-files-by-type

Files with a MIME type matching this variable won't be viewed. Note that Gnus tries to guess what type the file is based on the name. `gnus-uu` is not a MIME package (yet), so this is slightly kludgy.

##### gnus-uu-tmp-dir

Where `gnus-uu` does its work.

**gnus-uu-do-not-unpack-archives**

Non-`nil` means that `gnus-uu` won't peek inside archives looking for files to display.

**gnus-uu-view-and-save**

Non-`nil` means that the user will always be asked to save a file after viewing it.

**gnus-uu-ignore-default-view-rules**

Non-`nil` means that `gnus-uu` will ignore the default viewing rules.

**gnus-uu-ignore-default-archive-rules**

Non-`nil` means that `gnus-uu` will ignore the default archive unpacking commands.

**gnus-uu-kill-carriage-return**

Non-`nil` means that `gnus-uu` will strip all carriage returns from articles.

**gnus-uu-unmark-articles-not-decoded**

Non-`nil` means that `gnus-uu` will mark unsuccessfully decoded articles as unread.

**gnus-uu-correct-stripped-uucode**

Non-`nil` means that `gnus-uu` will *try* to fix uuencoded files that have had trailing spaces deleted.

**gnus-uu-pre-uudecode-hook**

Hook run before sending a message to `uudecode`.

**gnus-uu-view-with-metamail**

Non-`nil` means that `gnus-uu` will ignore the viewing commands defined by the rule variables and just fudge a MIME content type based on the file name. The result will be fed to `metamail` for viewing.

**gnus-uu-save-in-digest**

Non-`nil` means that `gnus-uu`, when asked to save without decoding, will save in digests. If this variable is `nil`, `gnus-uu` will just save everything in a file without any embellishments. The digesting almost conforms to RFC 1153—no easy way to specify any meaningful volume and issue numbers were found, so I simply dropped them.

### 3.16.5.3 Uuencoding and Posting

**gnus-uu-post-include-before-composing**

Non-`nil` means that `gnus-uu` will ask for a file to encode before you compose the article. If this variable is `t`, you can either include an encoded file with `C-c C-i` or have one included for you when you post the article.

**gnus-uu-post-length**

Maximum length of an article. The encoded file will be split into how many articles it takes to post the entire file.

**gnus-uu-post-threaded**

Non-`nil` means that `gnus-uu` will post the encoded file in a thread. This may not be smart, as no other decoder I have seen is able to follow threads when

collecting uuencoded articles. (Well, I have seen one package that does that—`gnus-uu`, but somehow, I don't think that counts...) Default is `nil`.

#### `gnus-uu-post-separate-description`

Non-`nil` means that the description will be posted in a separate article. The first article will typically be numbered (0/x). If this variable is `nil`, the description the user enters will be included at the beginning of the first article, which will be numbered (1/x). Default is `t`.

### 3.16.6 Viewing Files

After decoding, if the file is some sort of archive, Gnus will attempt to unpack the archive and see if any of the files in the archive can be viewed. For instance, if you have a gzipped tar file `'pics.tar.gz'` containing the files `'pic1.jpg'` and `'pic2.gif'`, Gnus will uncompress and de-tar the main file, and then view the two pictures. This unpacking process is recursive, so if the archive contains archives of archives, it'll all be unpacked.

Finally, Gnus will normally insert a *pseudo-article* for each extracted file into the summary buffer. If you go to these “articles”, you will be prompted for a command to run (usually Gnus will make a suggestion), and then the command will be run.

If `gnus-view-pseudo-asynchronously` is `nil`, Emacs will wait until the viewing is done before proceeding.

If `gnus-view-pseudos` is `automatic`, Gnus will not insert the pseudo-articles into the summary buffer, but view them immediately. If this variable is `not-confirm`, the user won't even be asked for a confirmation before viewing is done.

If `gnus-view-pseudos-separately` is non-`nil`, one pseudo-article will be created for each file to be viewed. If `nil`, all files that use the same viewing command will be given as a list of parameters to that command.

If `gnus-insert-pseudo-articles` is non-`nil`, insert pseudo-articles when decoding. It is `t` by default.

So; there you are, reading your *pseudo-articles* in your *virtual newsgroup* from the *virtual server*; and you think: Why isn't anything real anymore? How did we get here?

## 3.17 Article Treatment

Reading through this huge manual, you may have quite forgotten that the object of news-readers is to actually, like, read what people have written. Reading articles. Unfortunately, people are quite bad at writing, so there are tons of functions and variables to make reading these articles easier.

### 3.17.1 Article Highlighting

Not only do you want your article buffer to look like fruit salad, but you want it to look like technicolor fruit salad.

**W H a** Do much highlighting of the current article (`gnus-article-highlight`). This function highlights header, cited text, the signature, and adds buttons to the body and the head.

**W H h** Highlight the headers (`gnus-article-highlight-headers`). The highlighting will be done according to the `gnus-header-face-alist` variable, which is a list

where each element has the form (*regexp name content*). *regexp* is a regular expression for matching the header, *name* is the face used for highlighting the header name (see [\[Faces and Fonts\]](#), page [\[undefined\]](#)) and *content* is the face for highlighting the header value. The first match made will be used. Note that *regexp* shouldn't have '^' prepended—Gnus will add one.

*W H c* Highlight cited text (*gnus-article-highlight-citation*).

Some variables to customize the citation highlights:

*gnus-cite-parse-max-size*

If the article size is bigger than this variable (which is 25000 by default), no citation highlighting will be performed.

*gnus-cite-max-prefix*

Maximum possible length for a citation prefix (default 20).

*gnus-cite-face-list*

List of faces used for highlighting citations (see [\[Faces and Fonts\]](#), page [\[undefined\]](#)). When there are citations from multiple articles in the same message, Gnus will try to give each citation from each article its own face. This should make it easier to see who wrote what.

*gnus-supercite-regexp*

Regexp matching normal Supercite attribution lines.

*gnus-supercite-secondary-regexp*

Regexp matching mangled Supercite attribution lines.

*gnus-cite-minimum-match-count*

Minimum number of identical prefixes we have to see before we believe that it's a citation.

*gnus-cite-attribution-prefix*

Regexp matching the beginning of an attribution line.

*gnus-cite-attribution-suffix*

Regexp matching the end of an attribution line.

*gnus-cite-attribution-face*

Face used for attribution lines. It is merged with the face for the cited text belonging to the attribution.

*gnus-cite-ignore-quoted-from*

If non-*nil*, no citation highlighting will be performed on lines beginning with '>From '. Those lines may have been quoted by MTAs in order not to mix up with the envelope From line. The default value is *t*.

*W H s* Highlight the signature (*gnus-article-highlight-signature*). Everything after *gnus-signature-separator* (see [\[Article Signature\]](#), page [\[undefined\]](#)) in an article will be considered a signature and will be highlighted with *gnus-signature-face*, which is *italic* by default.

See [\[Customizing Articles\]](#), page [\[undefined\]](#), for how to highlight articles automatically.

### 3.17.2 Article Fontisizing

People commonly add emphasis to words in news articles by writing things like ‘`_this_`’ or ‘`*this*`’ or ‘`/this/`’. Gnus can make this look nicer by running the article through the `W e` (`gnus-article-emphasize`) command.

How the emphasis is computed is controlled by the `gnus-emphasis-alist` variable. This is an alist where the first element is a regular expression to be matched. The second is a number that says what regular expression grouping is used to find the entire emphasized word. The third is a number that says what regexp grouping should be displayed and highlighted. (The text between these two groupings will be hidden.) The fourth is the face used for highlighting.

```
(setq gnus-emphasis-alist
      '(("_\\((\\w+\\))_" 0 1 gnus-emphasis-underline)
        ("\\*\\((\\w+\\))\\*" 0 1 gnus-emphasis-bold)))
```

By default, there are seven rules, and they use the following faces: `gnus-emphasis-bold`, `gnus-emphasis-italic`, `gnus-emphasis-underline`, `gnus-emphasis-bold-italic`, `gnus-emphasis-underline-italic`, `gnus-emphasis-underline-bold`, and `gnus-emphasis-underline-bold-italic`.

If you want to change these faces, you can either use `M-x customize`, or you can use `copy-face`. For instance, if you want to make `gnus-emphasis-italic` use a red face instead, you could say something like:

```
(copy-face 'red 'gnus-emphasis-italic)
```

If you want to highlight arbitrary words, you can use the `gnus-group-highlight-words-alist` variable, which uses the same syntax as `gnus-emphasis-alist`. The `highlight-words` group parameter (see [\[Group Parameters\]](#), page [\[undefined\]](#)) can also be used.

See [\[Customizing Articles\]](#), page [\[undefined\]](#), for how to fontize articles automatically.

### 3.17.3 Article Hiding

Or rather, hiding certain things in each article. There usually is much too much cruft in most articles.

- |                    |  |
|--------------------|--|
| <code>W W a</code> | Do quite a lot of hiding on the article buffer ( <code>gnus-article-hide</code> ). In particular, this function will hide headers, PGP, cited text and the signature.          |
| <code>W W h</code> | Hide headers ( <code>gnus-article-hide-headers</code> ). See <a href="#">[Hiding Headers]</a> , page <a href="#">[undefined]</a> .   |
| <code>W W b</code> | Hide headers that aren't particularly interesting ( <code>gnus-article-hide-boring-headers</code> ). See <a href="#">[Hiding Headers]</a> , page <a href="#">[undefined]</a> . |
| <code>W W s</code> | Hide signature ( <code>gnus-article-hide-signature</code> ). See <a href="#">[Article Signature]</a> , page <a href="#">[undefined]</a> .                                      |

*W W l* Strip list identifiers specified in `gnus-list-identifiers`. These are strings some mailing list servers add to the beginning of all `Subject` headers—for example, `[zebra 4711]`. Any leading `Re:`  is skipped before stripping. `gnus-list-identifiers` may not contain `\\(\\.\\)`.

`gnus-list-identifiers`

A regular expression that matches list identifiers to be removed from subject. This can also be a list of regular expressions.

*W W P* Hide PEM (privacy enhanced messages) cruft (`gnus-article-hide-pem`).

*W W B* Strip the banner specified by the `banner` group parameter (`gnus-article-strip-banner`). This is mainly used to hide those annoying banners and/or signatures that some mailing lists and moderated groups adds to all the messages. The way to use this function is to add the `banner` group parameter (see [\(undefined\) \[Group Parameters\]](#), page [\(undefined\)](#)) to the group you want banners stripped from. The parameter either be a string, which will be interpreted as a regular expression matching text to be removed, or the symbol `signature`, meaning that the (last) signature should be removed, or other symbol, meaning that the corresponding regular expression in `gnus-article-banner-alist` is used.

Regardless of a group, you can hide things like advertisements only when the sender of an article has a certain mail address specified in `gnus-article-address-banner-alist`.

`gnus-article-address-banner-alist`

Alist of mail addresses and banners. Each element has the form `(address . banner)`, where `address` is a regexp matching a mail address in the `From` header, `banner` is one of a symbol `signature`, an item in `gnus-article-banner-alist`, a regexp and `nil`. If `address` matches author's mail address, it will remove things like advertisements. For example, if a sender has the mail address `'hail@yoo-hoo.co.jp'` and there is a banner something like `'Do You Yoo-hoo!?'` in all articles he sends, you can use the following element to remove them:

```
("@yoo-hoo\\.co\\.jp\\'" .
  "\\n_+\\nDo You Yoo-hoo!\\?\\n.*\\n.*\\n")
```

*W W c* Hide citation (`gnus-article-hide-citation`). Some variables for customizing the hiding:

`gnus-cited-opened-text-button-line-format`

`gnus-cited-closed-text-button-line-format`

Gnus adds buttons to show where the cited text has been hidden, and to allow toggle hiding the text. The format of the variable is specified by these format-like variable (see [\(undefined\) \[Formatting Variables\]](#), page [\(undefined\)](#)). These specs are valid:

`'b'` Starting point of the hidden text.  
`'e'` Ending point of the hidden text.

- ‘1’            Number of characters in the hidden region.
- ‘n’            Number of lines of hidden text.

#### `gnus-cited-lines-visible`

The number of lines at the beginning of the cited text to leave shown. This can also be a cons cell with the number of lines at the top and bottom of the text, respectively, to remain visible.

#### *W W C-c*

Hide citation (`gnus-article-hide-citation-maybe`) depending on the following two variables:

#### `gnus-cite-hide-percentage`

If the cited text is of a bigger percentage than this variable (default 50), hide the cited text.

#### `gnus-cite-hide-absolute`

The cited text must have at least this length (default 10) before it is hidden.

#### *W W C*

Hide cited text in articles that aren’t roots (`gnus-article-hide-citation-in-followups`). This isn’t very useful as an interactive command, but might be a handy function to stick have happen automatically (see [\(undefined\)](#) [Customizing Articles], page [\(undefined\)](#)).

All these “hiding” commands are toggles, but if you give a negative prefix to these commands, they will show what they have previously hidden. If you give a positive prefix, they will always hide.

Also see [\(undefined\)](#) [Article Highlighting], page [\(undefined\)](#) for further variables for citation customization.

See [\(undefined\)](#) [Customizing Articles], page [\(undefined\)](#), for how to hide article elements automatically.

### 3.17.4 Article Washing

We call this “article washing” for a really good reason. Namely, the *A* key was taken, so we had to use the *W* key instead.

*Washing* is defined by us as “changing something from something to something else”, but normally results in something looking better. Cleaner, perhaps.

See [\(undefined\)](#) [Customizing Articles], page [\(undefined\)](#), if you want to change how Gnus displays articles by default.

#### *C-u g*

This is not really washing, it’s sort of the opposite of washing. If you type this, you see the article exactly as it exists on disk or on the server.

#### *g*

Force redisplaying of the current article (`gnus-summary-show-article`). This is also not really washing. If you type this, you see the article without any previously applied interactive Washing functions but with all default treatments (see [\(undefined\)](#) [Customizing Articles], page [\(undefined\)](#)).



<i>W l</i>	Remove page breaks from the current article ( <code>gnus-summary-stop-page-breaking</code> ). See <code>&lt;undefined&gt;</code> [Misc Article], page <code>&lt;undefined&gt;</code> , for page delimiters.
<i>W r</i>	Do a Caesar rotate ( <code>rot13</code> ) on the article buffer ( <code>gnus-summary-caesar-message</code> ). Unreadable articles that tell you to read them with Caesar rotate or <code>rot13</code> . (Typically offensive jokes and such.)  It's commonly called "rot13" because each letter is rotated 13 positions in the alphabet, e. g. 'B' (letter #2) -> 'O' (letter #15). It is sometimes referred to as "Caesar rotate" because Caesar is rumored to have employed this form of, uh, somewhat weak encryption.
<i>W m</i>	Morse decode the article buffer ( <code>gnus-summary-morse-message</code> ).
<i>W t</i>	
<i>t</i>	Toggle whether to display all headers in the article buffer ( <code>gnus-summary-toggle-header</code> ).
<i>W v</i>	Toggle whether to display all headers in the article buffer permanently ( <code>gnus-summary-verbose-headers</code> ).
<i>W o</i>	Treat overstrike ( <code>gnus-article-treat-overstrike</code> ).
<i>W d</i>	Treat <code>M****s*** sm*rtq**t*s</code> according to <code>gnus-article-dumbquotes-map</code> ( <code>gnus-article-treat-dumbquotes</code> ). Note that this function guesses whether a character is a <code>sm*rtq**t*</code> or not, so it should only be used interactively.  <code>Sm*rtq**t*s</code> are <code>M****s****</code> 's unilateral extension to the character map in an attempt to provide more quoting characters. If you see something like <code>\222</code> or <code>\264</code> where you're expecting some kind of apostrophe or quotation mark, then try this wash.
<i>W Y f</i>	Full deuglify of broken Outlook (Express) articles: Treat dumbquotes, unwrap lines, repair attribution and rearrange citation. ( <code>gnus-article-outlook-deuglify-article</code> ).
<i>W Y u</i>	Unwrap lines that appear to be wrapped citation lines. You can control what lines will be unwrapped by frobbing <code>gnus-outlook-deuglify-unwrap-min</code> and <code>gnus-outlook-deuglify-unwrap-max</code> , indicating the minimum and maximum length of an unwrapped citation line. ( <code>gnus-article-outlook-unwrap-lines</code> ).
<i>W Y a</i>	Repair a broken attribution line. ( <code>gnus-article-outlook-repair-attribution</code> ).
<i>W Y c</i>	Repair broken citations by rearranging the text. ( <code>gnus-article-outlook-rearrange-citation</code> ).
<i>W w</i>	Do word wrap ( <code>gnus-article-fill-cited-article</code> ).  You can give the command a numerical prefix to specify the width to use when filling.
<i>W Q</i>	Fill long lines ( <code>gnus-article-fill-long-lines</code> ).

- W C* Capitalize the first word in each sentence (`gnus-article-capitalize-sentences`).
- W c* Translate CRLF pairs (i. e., ‘^M’s on the end of the lines) into LF (this takes care of DOS line endings), and then translate any remaining CRs into LF (this takes care of Mac line endings) (`gnus-article-remove-cr`).
- W q* Treat quoted-printable (`gnus-article-de-quoted-unreadable`). Quoted-Printable is one common MIME encoding employed when sending non-ASCII (i.e., 8-bit) articles. It typically makes strings like ‘dj vu’ look like ‘d=E9j=E0 vu’, which doesn’t look very readable to me. Note that this is usually done automatically by Gnus if the message in question has a `Content-Transfer-Encoding` header that says that this encoding has been done. If a prefix is given, a charset will be asked for.
- W 6* Treat base64 (`gnus-article-de-base64-unreadable`). Base64 is one common MIME encoding employed when sending non-ASCII (i.e., 8-bit) articles. Note that this is usually done automatically by Gnus if the message in question has a `Content-Transfer-Encoding` header that says that this encoding has been done. If a prefix is given, a charset will be asked for.
- W Z* Treat HZ or HZP (`gnus-article-decode-HZ`). HZ (or HZP) is one common encoding employed when sending Chinese articles. It typically makes strings look like ‘~{<:Ky2;S{#,NpJ)l6HK!#~}’.
- W u* Remove newlines from within URLs. Some mailers insert newlines into outgoing email messages to keep lines short. This reformatting can split long URLs onto multiple lines. Repair those URLs by removing the newlines (`gnus-article-unsplit-urls`).
- W h* Treat HTML (`gnus-article-wash-html`). Note that this is usually done automatically by Gnus if the message in question has a `Content-Type` header that says that the message is HTML.
- If a prefix is given, a charset will be asked for.
- The default is to use the function specified by `mm-text-html-renderer` (see section “Display Customization” in *The Emacs MIME Manual*) to convert the HTML, but this is controlled by the `gnus-article-wash-function` variable. Pre-defined functions you can use include:
- `w3`            Use Emacs/w3.
  - `w3m`          Use `emacs-w3m`.
  - `links`        Use `Links`.
  - `lynx`         Use `Lynx`.
  - `html2text`    Use `html2text`—a simple HTML converter included with Gnus.
- W b* Add clickable buttons to the article (`gnus-article-add-buttons`). See [\[Article Buttons\]](#), page [\[undefined\]](#).

<i>W B</i>	Add clickable buttons to the article headers ( <code>gnus-article-add-buttons-to-head</code> ).
<i>W p</i>	Verify a signed control message ( <code>gnus-article-verify-x-pgp-sig</code> ). Control messages such as <code>newgroup</code> and <code>checkgroups</code> are usually signed by the hierarchy maintainer. You need to add the PGP public key of the maintainer to your keyring to verify the message. <sup>1</sup>
<i>W s</i>	Verify a signed (PGP, PGP/MIME or S/MIME) message ( <code>gnus-summary-force-verify-and-decrypt</code> ). See <a href="#">[Security]</a> , page <a href="#">[undefined]</a> .
<i>W a</i>	Strip headers like the <code>X-No-Archive</code> header from the beginning of article bodies ( <code>gnus-article-strip-headers-in-body</code> ).
<i>W E l</i>	Remove all blank lines from the beginning of the article ( <code>gnus-article-strip-leading-blank-lines</code> ).
<i>W E m</i>	Replace all blank lines with empty lines and then all multiple empty lines with a single empty line. ( <code>gnus-article-strip-multiple-blank-lines</code> ).
<i>W E t</i>	Remove all blank lines at the end of the article ( <code>gnus-article-remove-trailing-blank-lines</code> ).
<i>W E a</i>	Do all the three commands above ( <code>gnus-article-strip-blank-lines</code> ).
<i>W E A</i>	Remove all blank lines ( <code>gnus-article-strip-all-blank-lines</code> ).
<i>W E s</i>	Remove all white space from the beginning of all lines of the article body ( <code>gnus-article-strip-leading-space</code> ).
<i>W E e</i>	Remove all white space from the end of all lines of the article body ( <code>gnus-article-strip-trailing-space</code> ).

See [\[Customizing Articles\]](#), page [\[undefined\]](#), for how to wash articles automatically.

### 3.17.5 Article Header

These commands perform various transformations of article header.

<i>W G u</i>	Unfold folded header lines ( <code>gnus-article-treat-unfold-headers</code> ).
<i>W G n</i>	Fold the <code>Newsgroups</code> and <code>Followup-To</code> headers ( <code>gnus-article-treat-fold-newsgroups</code> ).
<i>W G f</i>	Fold all the message headers ( <code>gnus-article-treat-fold-headers</code> ).
<i>W E w</i>	Remove excessive whitespace from all headers ( <code>gnus-article-remove-leading-whitespace</code> ).

### 3.17.6 Article Buttons

People often include references to other stuff in articles, and it would be nice if Gnus could just fetch whatever it is that people talk about with the minimum of fuzz when you hit `RET` or use the middle mouse button on these references.

---

<sup>1</sup> PGP keys for many hierarchies are available at <ftp://ftp.isc.org/pub/pgpcontrol/README.html>

Gnus adds *buttons* to certain standard references by default: Well-formed URLs, mail addresses, Message-IDs, Info links, man pages and Emacs or Gnus related references. This is controlled by two variables, one that handles article bodies and one that handles article heads:

#### `gnus-button-alist`

This is an alist where each entry has this form:

`(regexp button-par use-p function data-par)`

*regexp* All text that match this regular expression (case insensitive) will be considered an external reference. Here's a typical regexp that matches embedded URLs: '`<URL:\\[^\n\r\]*\]>`'. This can also be a variable containing a regexp, useful variables to use include `gnus-button-url-regexp` and `gnus-button-mid-or-mail-regexp`.

*button-par*

Gnus has to know which parts of the matches is to be highlighted. This is a number that says what sub-expression of the regexp is to be highlighted. If you want it all highlighted, you use 0 here.

*use-p*

This form will be `eval`ed, and if the result is non-`nil`, this is considered a match. This is useful if you want extra sifting to avoid false matches. Often variables named `gnus-button-*-level` are used here, See [\[Article Button Levels\]](#), page [\[undefined\]](#), but any other form may be used too.

*function*

This function will be called when you click on this button.

*data-par*

As with *button-par*, this is a sub-expression number, but this one says which part of the match is to be sent as data to *function*.

So the full entry for buttonizing URLs is then

`("<URL:\\[^\n\r\]*\]>" 0 t gnus-button-url 1)`

#### `gnus-header-button-alist`

This is just like the other alist, except that it is applied to the article head only, and that each entry has an additional element that is used to say what headers to apply the buttonize coding to:

`(header regexp button-par use-p function data-par)`

*header* is a regular expression.

### 3.17.6.1 Related variables and functions

#### `gnus-button-*-level`

See [\[Article Button Levels\]](#), page [\[undefined\]](#).

#### `gnus-button-url-regexp`

A regular expression that matches embedded URLs. It is used in the default values of the variables above.

#### `gnus-button-man-handler`

The function to use for displaying man pages. It must take at least one argument with a string naming the man page.

**gnus-button-mid-or-mail-regexp**

Regular expression that matches a message ID or a mail address.

**gnus-button-prefer-mid-or-mail**

This variable determines what to do when the button on a string as ‘foo123@bar.invalid’ is pushed. Strings like this can be either a message ID or a mail address. If it is one of the symbols `mid` or `mail`, Gnus will always assume that the string is a message ID or a mail address, respectively. If this variable is set to the symbol `ask`, always query the user what to do. If it is a function, this function will be called with the string as its only argument. The function must return `mid`, `mail`, `invalid` or `ask`. The default value is the function `gnus-button-mid-or-mail-heuristic`.

**gnus-button-mid-or-mail-heuristic**

Function that guesses whether its argument is a message ID or a mail address. Returns `mid` if it’s a message IDs, `mail` if it’s a mail address, `ask` if unsure and `invalid` if the string is invalid.

**gnus-button-mid-or-mail-heuristic-alist**

An alist of (RATE . REGEXP) pairs used by the function `gnus-button-mid-or-mail-heuristic`.

**gnus-button-ctan-handler**

The function to use for displaying CTAN links. It must take one argument, the string naming the URL.

**gnus-ctan-url**

Top directory of a CTAN (Comprehensive TeX Archive Network) archive used by `gnus-button-ctan-handler`.

**gnus-article-button-face**

Face used on buttons.

**gnus-article-mouse-face**

Face used when the mouse cursor is over a button.

See [\(undefined\)](#) [Customizing Articles], page [\(undefined\)](#), for how to buttonize articles automatically.

### 3.17.7 Article button levels

The higher the value of the variables `gnus-button-*-level`, the more buttons will appear. If the level is zero, no corresponding buttons are displayed. With the default value (which is 5) you should already see quite a lot of buttons. With higher levels, you will see more buttons, but you may also get more false positives. To avoid them, you can set the variables `gnus-button-*-level` local to specific groups (see [\(undefined\)](#) [Group Parameters], page [\(undefined\)](#)). Here’s an example for the variable `gnus-parameters`:

```
;; increase gnus-button-*-level in some groups:
(setq gnus-parameters
      '(("\\<\\(emacs\\|gnus\\)\\>" (gnus-button-emacs-level 10))
        ("\\<unix\\>" (gnus-button-man-level 10))
        ("\\<tex\\>" (gnus-button-tex-level 10))))
```

**gnus-button-browse-level**

Controls the display of references to message IDs, mail addresses and news URLs. Related variables and functions include `gnus-button-url-regexp`, `browse-url`, and `browse-url-browser-function`.

**gnus-button-emacs-level**

Controls the display of Emacs or Gnus references. Related functions are `gnus-button-handle-custom`, `gnus-button-handle-describe-function`, `gnus-button-handle-describe-variable`, `gnus-button-handle-symbol`, `gnus-button-handle-describe-key`, `gnus-button-handle-apropos`, `gnus-button-handle-apropos-command`, `gnus-button-handle-apropos-variable`, `gnus-button-handle-apropos-documentation`, and `gnus-button-handle-library`.

**gnus-button-man-level**

Controls the display of references to (Unix) man pages. See `gnus-button-man-handler`.

**gnus-button-message-level**

Controls the display of message IDs, mail addresses and news URLs. Related variables and functions include `gnus-button-mid-or-mail-regexp`, `gnus-button-prefer-mid-or-mail`, `gnus-button-mid-or-mail-heuristic`, and `gnus-button-mid-or-mail-heuristic-alist`.

**gnus-button-tex-level**

Controls the display of references to T<sub>E</sub>X or LaTeX stuff, e.g. for CTAN URLs. See the variables `gnus-ctan-url`, `gnus-button-ctan-handler`, `gnus-button-ctan-directory-regexp`, and `gnus-button-handle-ctan-bogus-regexp`.

### 3.17.8 Article Date

The date is most likely generated in some obscure timezone you've never heard of, so it's quite nice to be able to find out what the time was when the article was sent.

<i>W T u</i>	Display the date in UT (aka. GMT, aka ZULU) ( <code>gnus-article-date-ut</code> ).
<i>W T i</i>	Display the date in international format, aka. ISO 8601 ( <code>gnus-article-date-iso8601</code> ).
<i>W T l</i>	Display the date in the local timezone ( <code>gnus-article-date-local</code> ).
<i>W T p</i>	Display the date in a format that's easily pronounceable in English ( <code>gnus-article-date-english</code> ).
<i>W T s</i>	Display the date using a user-defined format ( <code>gnus-article-date-user</code> ). The format is specified by the <code>gnus-article-time-format</code> variable, and is a string that's passed to <code>format-time-string</code> . See the documentation of that variable for a list of possible format specs.
<i>W T e</i>	Say how much time has elapsed between the article was posted and now ( <code>gnus-article-date-lapsed</code> ). It looks something like:

X-Sent: 6 weeks, 4 days, 1 hour, 3 minutes, 8 seconds ago

The value of `gnus-article-date-lapsed-new-header` determines whether this header will just be added below the old Date one, or will replace it.

An advantage of using Gnus to read mail is that it converts simple bugs into wonderful absurdities.

If you want to have this line updated continually, you can put

```
(gnus-start-date-timer)
```

in your ‘`~/.gnus.el`’ file, or you can run it off of some hook. If you want to stop the timer, you can use the `gnus-stop-date-timer` command.

*W T o*      Display the original date (`gnus-article-date-original`). This can be useful if you normally use some other conversion function and are worried that it might be doing something totally wrong. Say, claiming that the article was posted in 1854. Although something like that is *totally* impossible. Don’t you trust me? *\*titter\**

See [\(undefined\) \[Customizing Articles\]](#), page [\(undefined\)](#), for how to display the date in your preferred format automatically.

### 3.17.9 Article Display

These commands add various frivolous display gimmicks to the article buffer in Emacs versions that support them.

**X-Face** headers are small black-and-white images supplied by the message headers (see [\(undefined\) \[X-Face\]](#), page [\(undefined\)](#)).

**Face** headers are small colored images supplied by the message headers (see [\(undefined\) \[Face\]](#), page [\(undefined\)](#)).

Smileys are those little ‘:-)’ symbols that people like to litter their messages with (see [\(undefined\) \[Smileys\]](#), page [\(undefined\)](#)).

Picons, on the other hand, reside on your own system, and Gnus will try to match the headers to what you have (see [\(undefined\) \[Picons\]](#), page [\(undefined\)](#)).

All these functions are toggles—if the elements already exist, they’ll be removed.

*W D x*      Display an X-Face in the From header. (`gnus-article-display-x-face`).

*W D d*      Display a Face in the From header. (`gnus-article-display-face`).

*W D s*      Display smileys (`gnus-treat-smiley`).

*W D f*      Piconify the From header (`gnus-treat-from-picon`).

*W D m*      Piconify all mail headers (i. e., Cc, To) (`gnus-treat-mail-picon`).

*W D n*      Piconify all news headers (i. e., Newsgroups and Followup-To) (`gnus-treat-newsgroups-picon`).

*W D D*      Remove all images from the article buffer (`gnus-article-remove-images`).



### 3.17.10 Article Signature

Each article is divided into two parts—the head and the body. The body can be divided into a signature part and a text part. The variable that says what is to be considered a signature is `gnus-signature-separator`. This is normally the standard ‘`^-- $`’ as mandated by son-of-RFC 1036. However, many people use non-standard signature separators, so this variable can also be a list of regular expressions to be tested, one by one. (Searches are done from the end of the body towards the beginning.) One likely value is:

```
(setq gnus-signature-separator
      '("^-- $"      ; The standard
        "^-- *$"     ; A common mangling
        "^-----*$"  ; Many people just use a looong
                      ; line of dashes. Shame!
        "^ *-----*$" ; Double-shame!
        "^ _-----*$" ; Underscores are also popular
        "^=====*$")) ; Pervert!
```

The more permissive you are, the more likely it is that you’ll get false positives.

`gnus-signature-limit` provides a limit to what is considered a signature when displaying articles.

1. If it is an integer, no signature may be longer (in characters) than that integer.
2. If it is a floating point number, no signature may be longer (in lines) than that number.
3. If it is a function, the function will be called without any parameters, and if it returns `nil`, there is no signature in the buffer.
4. If it is a string, it will be used as a regexp. If it matches, the text in question is not a signature.

This variable can also be a list where the elements may be of the types listed above. Here’s an example:

```
(setq gnus-signature-limit
      '(200.0 "^---*Forwarded article"))
```

This means that if there are more than 200 lines after the signature separator, or the text after the signature separator is matched by the regular expression ‘`^---*Forwarded article`’, then it isn’t a signature after all.

### 3.17.11 Article Miscellanea

**A t**            Translate the article from one language to another (`gnus-article-babel`).

## 3.18 MIME Commands

The following commands all understand the numerical prefix. For instance, **3 b** means “view the third MIME part”.

**b**  
**K v**            View the MIME part.  
**K o**            Save the MIME part.  
**K c**            Copy the MIME part.



- K e* View the MIME part externally.
- K i* View the MIME part internally.
- K |* Pipe the MIME part to an external command.

The rest of these MIME commands do not use the numerical prefix in the same manner:

- K b* Make all the MIME parts have buttons in front of them. This is mostly useful if you wish to save (or perform other actions) on inlined parts.
- K m* Some multipart messages are transmitted with missing or faulty headers. This command will attempt to “repair” these messages so that they can be viewed in a more pleasant manner (`gnus-summary-repair-multipart`).
- X m* Save all parts matching a MIME type to a directory (`gnus-summary-save-parts`). Understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\[undefined\]](#)).
- M-t* Toggle the buttonized display of the article buffer (`gnus-summary-toggle-display-buttonized`).
- W M w* Decode RFC 2047-encoded words in the article headers (`gnus-article-decode-mime-words`).
- W M c* Decode encoded article bodies as well as charsets (`gnus-article-decode-charset`).  
This command looks in the **Content-Type** header to determine the charset. If there is no such header in the article, you can give it a prefix, which will prompt for the charset to decode as. In regional groups where people post using some common encoding (but do not include MIME headers), you can set the `charset` group/topic parameter to the required charset (see [\[Group Parameters\]](#), page [\[undefined\]](#)).
- W M v* View all the MIME parts in the current article (`gnus-mime-view-all-parts`).

Relevant variables:

#### `gnus-ignored-mime-types`

This is a list of regexps. MIME types that match a regexp from this list will be completely ignored by Gnus. The default value is `nil`.

To have all Vcards be ignored, you’d say something like this:

```
(setq gnus-ignored-mime-types
      '("text/x-vcard"))
```

#### `gnus-article-loose-mime`

If non-`nil`, Gnus won’t require the ‘**MIME-Version**’ header before interpreting the message as a MIME message. This helps when reading messages from certain broken mail user agents. The default is `nil`.

#### `gnus-article-emulate-mime`

There are other, non-MIME encoding methods used. The most common is ‘`uuencode`’, but `yEncode` is also getting to be popular. If this variable is non-`nil`, Gnus will look in message bodies to see if it finds these encodings, and if so, it’ll run them through the Gnus MIME machinery. The default is `t`.

**gnus-unbuttonized-mime-types**

This is a list of regexps. MIME types that match a regexp from this list won't have MIME buttons inserted unless they aren't displayed or this variable is overridden by **gnus-buttonized-mime-types**. The default value is `(".*/*.*")`. This variable is only used when **gnus-inhibit-mime-unbuttonizing** is `nil`.

**gnus-buttonized-mime-types**

This is a list of regexps. MIME types that match a regexp from this list will have MIME buttons inserted unless they aren't displayed. This variable overrides **gnus-unbuttonized-mime-types**. The default value is `nil`. This variable is only used when **gnus-inhibit-mime-unbuttonizing** is `nil`.

To see e.g. security buttons but no other buttons, you could set this variable to `("multipart/signed")` and leave **gnus-unbuttonized-mime-types** at the default value.

**gnus-inhibit-mime-unbuttonizing**

If this is non-`nil`, then all MIME parts get buttons. The default value is `nil`.

**gnus-article-mime-part-function**

For each MIME part, this function will be called with the MIME handle as the parameter. The function is meant to be used to allow users to gather information from the article (e. g., add Vcard info to the bdb database) or to do actions based on parts (e. g., automatically save all jpegs into some directory).

Here's an example function the does the latter:

```
(defun my-save-all-jpeg-parts (handle)
  (when (equal (car (mm-handle-type handle)) "image/jpeg")
    (with-temp-buffer
      (insert (mm-get-part handle))
      (write-region (point-min) (point-max)
                    (read-file-name "Save jpeg to: "))))
  (setq gnus-article-mime-part-function
        'my-save-all-jpeg-parts))
```

**gnus-mime-multipart-functions**

Alist of MIME multipart types and functions to handle them.

**gnus-mime-display-multipart-alternative-as-mixed**

Display "multipart/alternative" parts as "multipart/mixed".

**gnus-mime-display-multipart-related-as-mixed**

Display "multipart/related" parts as "multipart/mixed".

If displaying "text/html" is discouraged, see **mm-discouraged-alternatives** in [section "Display Customization" in \*Emacs-Mime Manual\*](#). Images or other material inside a "multipart/related" part might be overlooked when this variable is `nil`.

**gnus-mime-display-multipart-as-mixed**

Display "multipart" parts as "multipart/mixed". If `t`, it overrides `nil` values of **gnus-mime-display-multipart-alternative-as-mixed** and **gnus-mime-display-multipart-related-as-mixed**.

**mm-file-name-rewrite-functions**

List of functions used for rewriting file names of MIME parts. Each function takes a file name as input and returns a file name.

Ready-made functions include

`mm-file-name-delete-whitespace`, `mm-file-name-trim-whitespace`, `mm-file-name-collapse-whitespace`, and `mm-file-name-replace-whitespace`. The later uses the value of the variable `mm-file-name-replace-whitespace` to replace each whitespace character in a file name with that string; default value is `"_"` (a single underscore).

The standard functions `capitalize`, `downcase`, `upcase`, and `upcase-initials` may be useful, too.

Everybody knows that whitespace characters in file names are evil, except those who don't know. If you receive lots of attachments from such unenlightened users, you can make live easier by adding

```
(setq mm-file-name-rewrite-functions
      '(mm-file-name-trim-whitespace
        mm-file-name-collapse-whitespace
        mm-file-name-replace-whitespace))
```

to your `'~/.gnus.el'` file.

### 3.19 Charsets

People use different charsets, and we have MIME to let us know what charsets they use. Or rather, we wish we had. Many people use newsreaders and mailers that do not understand or use MIME, and just send out messages without saying what character sets they use. To help a bit with this, some local news hierarchies have policies that say what character set is the default. For instance, the `'fj'` hierarchy uses `iso-2022-jp-2`.

This knowledge is encoded in the `gnus-group-charset-alist` variable, which is an alist of regexps (use the first item to match full group names) and default charsets to be used when reading these groups.

In addition, some people do use soi-disant MIME-aware agents that aren't. These blithely mark messages as being in `iso-8859-1` even if they really are in `koi-8`. To help here, the `gnus-newsgroup-ignored-charsets` variable can be used. The charsets that are listed here will be ignored. The variable can be set on a group-by-group basis using the group parameters (see [\[Group Parameters\]](#), page [\[undefined\]](#)). The default value is `(unknown-8bit x-unknown)`, which includes values some agents insist on having in there.

When posting, `gnus-group-posting-charset-alist` is used to determine which charsets should not be encoded using the MIME encodings. For instance, some hierarchies discourage using quoted-printable header encoding.

This variable is an alist of regexps and permitted unencoded charsets for posting. Each element of the alist has the form `(test header body-list)`, where:

- test* is either a regular expression matching the newsgroup header or a variable to query,
- header* is the charset which may be left unencoded in the header (`nil` means encode all charsets),

*body-list* is a list of charsets which may be encoded using 8bit content-transfer encoding in the body, or one of the special values `nil` (always encode using quoted-printable) or `t` (always use 8bit).

See section “Encoding Customization” in *The Emacs MIME Manual*, for additional variables that control which MIME charsets are used when sending messages.

Other charset tricks that may be useful, although not Gnus-specific:

If there are several MIME charsets that encode the same Emacs charset, you can choose what charset to use by saying the following:

```
(put-charset-property 'cyrillic-iso8859-5
                      'preferred-coding-system 'koi8-r)
```

This means that Russian will be encoded using `koi8-r` instead of the default `iso-8859-5` MIME charset.

If you want to read messages in `koi8-u`, you can cheat and say

```
(define-coding-system-alias 'koi8-u 'koi8-r)
```

This will almost do the right thing.

And finally, to read charsets like `windows-1251`, you can say something like

```
(codepage-setup 1251)
(define-coding-system-alias 'windows-1251 'cp1251)
```

## 3.20 Article Commands

**A P** Generate and print a PostScript image of the article buffer (`gnus-summary-print-article`). `gnus-ps-print-hook` will be run just before printing the buffer. An alternative way to print article is to use Muttprint (see [\(undefined\)](#) [Saving Articles], page [\(undefined\)](#)).

## 3.21 Summary Sorting

You can have the summary buffer sorted in various ways, even though I can’t really see why you’d want that.

**C-c C-s C-n**

Sort by article number (`gnus-summary-sort-by-number`).

**C-c C-s C-a**

Sort by author (`gnus-summary-sort-by-author`).

**C-c C-s C-s**

Sort by subject (`gnus-summary-sort-by-subject`).

**C-c C-s C-d**

Sort by date (`gnus-summary-sort-by-date`).

**C-c C-s C-l**

Sort by lines (`gnus-summary-sort-by-lines`).

**C-c C-s C-c**

Sort by article length (`gnus-summary-sort-by-chars`).

*C-c C-s C-i*

Sort by score (`gnus-summary-sort-by-score`).

*C-c C-s C-r*

Randomize (`gnus-summary-sort-by-random`).

*C-c C-s C-o*

Sort using the default sorting method (`gnus-summary-sort-by-original`).

These functions will work both when you use threading and when you don't use threading. In the latter case, all summary lines will be sorted, line by line. In the former case, sorting will be done on a root-by-root basis, which might not be what you were looking for. To toggle whether to use threading, type *T T* (see [\(undefined\)](#) [Thread Commands], page [\(undefined\)](#)).

## 3.22 Finding the Parent

*^* If you'd like to read the parent of the current article, and it is not displayed in the summary buffer, you might still be able to. That is, if the current group is fetched by NNTP, the parent hasn't expired and the **References** in the current article are not mangled, you can just press *^* or *A r* (`gnus-summary-refer-parent-article`). If everything goes well, you'll get the parent. If the parent is already displayed in the summary buffer, point will just move to this article.

If given a positive numerical prefix, fetch that many articles back into the ancestry. If given a negative numerical prefix, fetch just that ancestor. So if you say *3 ^*, Gnus will fetch the parent, the grandparent and the grandgrandparent of the current article. If you say *-3 ^*, Gnus will only fetch the grandgrandparent of the current article.

*A R (Summary)*

Fetch all articles mentioned in the **References** header of the article (`gnus-summary-refer-references`).

*A T (Summary)*

Display the full thread where the current article appears (`gnus-summary-refer-thread`). This command has to fetch all the headers in the current group to work, so it usually takes a while. If you do it often, you may consider setting `gnus-fetch-old-headers` to `invisible` (see [\(undefined\)](#) [Filling In Threads], page [\(undefined\)](#)). This won't have any visible effects normally, but it'll make this command work a whole lot faster. Of course, it'll make group entry somewhat slow.

The `gnus-refer-thread-limit` variable says how many old (i. e., articles before the first displayed in the current group) headers to fetch when doing this command. The default is 200. If *t*, all the available headers will be fetched. This variable can be overridden by giving the *A T* command a numerical prefix.

*M-^ (Summary)*

You can also ask the NNTP server for an arbitrary article, no matter what group it belongs to. *M-^* (`gnus-summary-refer-article`) will ask you for a **Message-ID**, which is one of those long, hard-to-read thingies that look something like

'<38o6up\$6f2@hymir.ifi.uio.no>'. You have to get it all exactly right. No fuzzy searches, I'm afraid.

The current select method will be used when fetching by **Message-ID** from non-news select method, but you can override this by giving this command a prefix.

If the group you are reading is located on a back end that does not support fetching by **Message-ID** very well (like **nnsPOOL**), you can set **gnus-refer-article-method** to an NNTP method. It would, perhaps, be best if the NNTP server you consult is the one updating the spool you are reading from, but that's not really necessary.

It can also be a list of select methods, as well as the special symbol **current**, which means to use the current select method. If it is a list, Gnus will try all the methods in the list until it finds a match.

Here's an example setting that will first try the current method, and then ask Google if that fails:

```
(setq gnus-refer-article-method
      '(current
        (nnweb "google" (nnweb-type google))))
```

Most of the mail back ends support fetching by **Message-ID**, but do not do a particularly excellent job at it. That is, **nnmbox**, **nnbabyl**, **nnmaildir**, **nnml**, are able to locate articles from any groups, while **nnfolder**, and **nnimap** are only able to locate articles that have been posted to the current group. (Anything else would be too time consuming.) **nnmh** does not support this at all.

## 3.23 Alternative Approaches

Different people like to read news using different methods. This being Gnus, we offer a small selection of minor modes for the summary buffers.

### 3.23.1 Pick and Read

Some newsreaders (like **nn** and, uhm, **Netnews** on VM/CMS) use a two-phased reading interface. The user first marks in a summary buffer the articles she wants to read. Then she starts reading the articles with just an article buffer displayed.

Gnus provides a summary buffer minor mode that allows this—**gnus-pick-mode**. This basically means that a few process mark commands become one-keystroke commands to allow easy marking, and it provides one additional command for switching to the summary buffer.

Here are the available keystrokes when using pick mode:

- . Pick the article or thread on the current line (**gnus-pick-article-or-thread**). If the variable **gnus-thread-hide-subtree** is true, then this key selects the entire thread when used at the first article of the thread. Otherwise, it selects just the article. If given a numerical prefix, go to that thread or article and pick it. (The line number is normally displayed at the beginning of the summary pick lines.)
- SPACE** Scroll the summary buffer up one page (**gnus-pick-next-page**). If at the end of the buffer, start reading the picked articles.

- u** Unpick the thread or article (`gnus-pick-unmark-article-or-thread`). If the variable `gnus-thread-hide-subtree` is true, then this key unpicks the thread if used at the first article of the thread. Otherwise it unpicks just the article. You can give this key a numerical prefix to unpick the thread or article at that line.
- RET** Start reading the picked articles (`gnus-pick-start-reading`). If given a prefix, mark all unpicked articles as read first. If `gnus-pick-display-summary` is non-`nil`, the summary buffer will still be visible when you are reading.

All the normal summary mode commands are still available in the pick-mode, with the exception of **u**. However **!** is available which is mapped to the same function `gnus-summary-tick-article-forward`.

If this sounds like a good idea to you, you could say:

```
(add-hook 'gnus-summary-mode-hook 'gnus-pick-mode)
```

`gnus-pick-mode-hook` is run in pick minor mode buffers.

If `gnus-mark-unpicked-articles-as-read` is non-`nil`, mark all unpicked articles as read. The default is `nil`.

The summary line format in pick mode is slightly different from the standard format. At the beginning of each line the line number is displayed. The pick mode line format is controlled by the `gnus-summary-pick-line-format` variable (see [\(undefined\) \[Formatting Variables\]](#), page [\(undefined\)](#)). It accepts the same format specs that `gnus-summary-line-format` does (see [\(undefined\) \[Summary Buffer Lines\]](#), page [\(undefined\)](#)).

### 3.23.2 Binary Groups

If you spend much time in binary groups, you may grow tired of hitting **X u, n, RET** all the time. *M-x* `gnus-binary-mode` is a minor mode for summary buffers that makes all ordinary Gnus article selection functions udecode series of articles and display the result instead of just displaying the articles the normal way.

The only way, in fact, to see the actual articles is the **g** command, when you have turned on this mode (`gnus-binary-show-article`).

`gnus-binary-mode-hook` is called in binary minor mode buffers.

## 3.24 Tree Display

If you don't like the normal Gnus summary display, you might try setting `gnus-use-trees` to `t`. This will create (by default) an additional *tree buffer*. You can execute all summary mode commands in the tree buffer.

There are a few variables to customize the tree display, of course:

**gnus-tree-mode-hook**

A hook called in all tree mode buffers.

**gnus-tree-mode-line-format**

A format string for the mode bar in the tree mode buffers (see [\(undefined\) \[Mode Line Formatting\]](#), page [\(undefined\)](#)). The default is `'Gnus: %b %S %Z'`. For a list of valid specs, see [\(undefined\) \[Summary Buffer Mode Line\]](#), page [\(undefined\)](#).



**gnus-selected-tree-face**

Face used for highlighting the selected article in the tree buffer. The default is `modeline`.

**gnus-tree-line-format**

A format string for the tree nodes. The name is a bit of a misnomer, though—it doesn’t define a line, but just the node. The default value is ‘`%( [%3,3n%] )`’, which displays the first three characters of the name of the poster. It is vital that all nodes are of the same length, so you *must* use ‘`%4,4n`’-like specifiers.

Valid specs are:

‘n’	The name of the poster.
‘f’	The <b>From</b> header.
‘N’	The number of the article.
‘[’	The opening bracket.
‘]’	The closing bracket.
‘s’	The subject.

See [\(undefined\) \[Formatting Variables\]](#), page [\(undefined\)](#).

Variables related to the display are:

**gnus-tree-brackets**

This is used for differentiating between “real” articles and “sparse” articles. The format is

```
((real-open . real-close)
 (sparse-open . sparse-close)
 (dummy-open . dummy-close))
```

and the default is `((?[ . ?]) (? ( . ?)) ({ . ?}) (< . ?>))`.

**gnus-tree-parent-child-edges**

This is a list that contains the characters used for connecting parent nodes to their children. The default is `(?- ?\ \ ?|)`.

**gnus-tree-minimize-window**

If this variable is non-`nil`, Gnus will try to keep the tree buffer as small as possible to allow more room for the other Gnus windows. If this variable is a number, the tree buffer will never be higher than that number. The default is `t`. Note that if you have several windows displayed side-by-side in a frame and the tree buffer is one of these, minimizing the tree window will also resize all other windows displayed next to it.

You may also wish to add the following hook to keep the window minimized at all times:

```
(add-hook 'gnus-configure-windows-hook
          'gnus-tree-perhaps-minimize)
```

**gnus-generate-tree-function**

The function that actually generates the thread tree. Two predefined functions are available: `gnus-generate-horizontal-tree` and `gnus-generate-vertical-tree` (which is the default).



Here's an example from a horizontal tree buffer:

```
{***}-(***)-[odd]-[Gun]
      |      \[Jan]
      |      \[odd]-[Eri]
      |      \(***)-[Eri]
      |      \[odd]-[Paa]
\ [Bjo]
\ [Gun]
\ [Gun]-[Jor]
```

Here's the same thread displayed in a vertical tree buffer:

```
{***}
|-----\-----\-----\
(***)                                [Bjo] [Gun] [Gun]
|--\-----\-----\
[odd] [Jan] [odd] (***)                                [Jor]
|      |      |--\
[Gun]      [Eri] [Eri] [odd]
|
[Paa]
```

If you're using horizontal trees, it might be nice to display the trees side-by-side with the summary buffer. You could add something like the following to your '~/.gnus.el' file:

```
(setq gnus-use-trees t
      gnus-generate-tree-function 'gnus-generate-horizontal-tree
      gnus-tree-minimize-window nil)
(gnus-add-configuration
 '(article
   (vertical 1.0
    (horizontal 0.25
     (summary 0.75 point)
     (tree 1.0))
    (article 1.0))))
```

See [\[Window Layout\]](#), page [\[undefined\]](#).

### 3.25 Mail Group Commands

Some commands only make sense in mail groups. If these commands are invalid in the current group, they will raise a hell and let you know.

All these commands (except the expiry and edit commands) use the process/prefix convention (see [\[Process/Prefix\]](#), page [\[undefined\]](#)).

**B e** Run all expirable articles in the current group through the expiry process (**gnus-summary-expire-articles**). That is, delete all expirable articles in the group that have been around for a while. (see [\[Expiring Mail\]](#), page [\[undefined\]](#)).

- B C-M-e* Delete all the expirable articles in the group (`gnus-summary-expire-articles-now`). This means that **all** articles eligible for expiry in the current group will disappear forever into that big ‘/dev/null’ in the sky.
- B DEL* Delete the mail article. This is “delete” as in “delete it from your disk forever and ever, never to return again.” Use with caution. (`gnus-summary-delete-article`).
- B m* Move the article from one mail group to another (`gnus-summary-move-article`). Marks will be preserved if `gnus-preserve-marks` is non-nil (which is the default).
- B c* Copy the article from one group (mail group or not) to a mail group (`gnus-summary-copy-article`). Marks will be preserved if `gnus-preserve-marks` is non-nil (which is the default).
- B B* Crosspost the current article to some other group (`gnus-summary-crosspost-article`). This will create a new copy of the article in the other group, and the Xref headers of the article will be properly updated.
- B i* Import an arbitrary file into the current mail newsgroup (`gnus-summary-import-article`). You will be prompted for a file name, a From header and a Subject header.
- B I* Create an empty article in the current mail newsgroups (`gnus-summary-create-article`). You will be prompted for a From header and a Subject header.
- B r* Respool the mail article (`gnus-summary-respool-article`). `gnus-summary-respool-default-method` will be used as the default select method when re-spooling. This variable is nil by default, which means that the current group select method will be used instead. Marks will be preserved if `gnus-preserve-marks` is non-nil (which is the default).
- B w*  
*e* Edit the current article (`gnus-summary-edit-article`). To finish editing and make the changes permanent, type `C-c C-c` (`gnus-summary-edit-article-done`). If you give a prefix to the `C-c C-c` command, Gnus won’t re-highlight the article.
- B q* If you want to re-spool an article, you might be curious as to what group the article will end up in before you do the re-spooling. This command will tell you (`gnus-summary-respool-query`).
- B t* Similarly, this command will display all fancy splitting patterns used when re-spooling, if any (`gnus-summary-respool-trace`).
- B p* Some people have a tendency to send you “courtesy” copies when they follow up to articles you have posted. These usually have a `Newsgroups` header in them, but not always. This command (`gnus-summary-article-posted-p`) will try to fetch the current article from your news server (or rather, from `gnus-refer-article-method` or `gnus-select-method`) and will report back whether it found the article or not. Even if it says that it didn’t find the article, it

may have been posted anyway—mail propagation is much faster than news propagation, and the news copy may just not have arrived yet.

***K E*** Encrypt the body of an article (`gnus-article-encrypt-body`). The body is encrypted with the encryption protocol specified by the variable `gnus-article-encrypt-protocol`.

If you move (or copy) articles regularly, you might wish to have Gnus suggest where to put the articles. `gnus-move-split-methods` is a variable that uses the same syntax as `gnus-split-methods` (see [\[Saving Articles\]](#), page [\[undefined\]](#)). You may customize that variable to create suggestions you find reasonable. (Note that `gnus-move-split-methods` uses group names where `gnus-split-methods` uses file names.)

```
(setq gnus-move-split-methods
      '(("^From:.*Lars Magne" "nnml:junk")
        ("^Subject:.*gnus" "nnfolder:important")
        (".*" "nnml:misc")))
```

## 3.26 Various Summary Stuff

`gnus-summary-display-while-building`

If non-`nil`, show and update the summary buffer as it's being built. If `t`, update the buffer after every line is inserted. If the value is an integer, *n*, update the display every *n* lines. The default is `nil`.

`gnus-summary-display-arrow`

If non-`nil`, display an arrow in the fringe to indicate the current article.

`gnus-summary-mode-hook`

This hook is called when creating a summary mode buffer.

`gnus-summary-generate-hook`

This is called as the last thing before doing the threading and the generation of the summary buffer. It's quite convenient for customizing the threading variables based on what data the newsgroup has. This hook is called from the summary buffer after most summary buffer variables have been set.

`gnus-summary-prepare-hook`

It is called after the summary buffer has been generated. You might use it to, for instance, highlight lines or modify the look of the buffer in some other ungodly manner. I don't care.

`gnus-summary-prepared-hook`

A hook called as the very last thing after the summary buffer has been generated.

`gnus-summary-ignore-duplicates`

When Gnus discovers two articles that have the same `Message-ID`, it has to do something drastic. No articles are allowed to have the same `Message-ID`, but this may happen when reading mail from some sources. Gnus allows you to customize what happens with this variable. If it is `nil` (which is the default), Gnus will rename the `Message-ID` (for display purposes only) and display the

article as any other article. If this variable is `t`, it won't display the article—it'll be as if it never existed.

#### `gnus-alter-articles-to-read-function`

This function, which takes two parameters (the group name and the list of articles to be selected), is called to allow the user to alter the list of articles to be selected.

For instance, the following function adds the list of cached articles to the list in one particular group:

```
(defun my-add-cached-articles (group articles)
  (if (string= group "some.group")
      (append gnus-newsgroup-cached articles)
      articles))
```

#### `gnus-newsgroup-variables`

A list of newsgroup (summary buffer) local variables, or cons of variables and their default expressions to be evalled (when the default values are not `nil`), that should be made global while the summary buffer is active.

Note: The default expressions will be evaluated (using function `eval`) before assignment to the local variable rather than just assigned to it. If the default expression is the symbol `global`, that symbol will not be evaluated but the global value of the local variable will be used instead.

These variables can be used to set variables in the group parameters while still allowing them to affect operations done in other buffers. For example:

```
(setq gnus-newsgroup-variables
      '(message-use-followup-to
        (gnus-visible-headers .
          "^From:\\|\\^Newsgroups:\\|\\^Subject:\\|\\^Date:\\|\\^To:")))
```

Also see [\[Group Parameters\]](#), page [\[undefined\]](#).

### 3.26.1 Summary Group Information

- H f* Try to fetch the FAQ (list of frequently asked questions) for the current group (`gnus-summary-fetch-faq`). Gnus will try to get the FAQ from `gnus-group-faq-directory`, which is usually a directory on a remote machine. This variable can also be a list of directories. In that case, giving a prefix to this command will allow you to choose between the various sites. `ange-ftp` or `efs` will probably be used for fetching the file.
- H d* Give a brief description of the current group (`gnus-summary-describe-group`). If given a prefix, force rereading the description from the server.
- H h* Give an extremely brief description of the most important summary keystrokes (`gnus-summary-describe-briefly`).
- H i* Go to the Gnus info node (`gnus-info-find-node`).

### 3.26.2 Searching for Articles

- M-s* Search through all subsequent (raw) articles for a regexp (`gnus-summary-search-article-forward`).

<i>M-r</i>	Search through all previous (raw) articles for a regexp ( <code>gnus-summary-search-article-backward</code> ).
<i>&amp;</i>	This command will prompt you for a header, a regular expression to match on this field, and a command to be executed if the match is made ( <code>gnus-summary-execute-command</code> ). If the header is an empty string, the match is done on the entire article. If given a prefix, search backward instead.  For instance, <i>&amp; RET some.*string RET #</i> will put the process mark on all articles that have heads or bodies that match ‘some.*string’.
<i>M-&amp;</i>	Perform any operation on all articles that have been marked with the process mark ( <code>gnus-summary-universal-argument</code> ).

### 3.26.3 Summary Generation Commands

<i>Y g</i>	Regenerate the current summary buffer ( <code>gnus-summary-prepare</code> ).
<i>Y c</i>	Pull all cached articles (for the current group) into the summary buffer ( <code>gnus-summary-insert-cached-articles</code> ).
<i>Y d</i>	Pull all dormant articles (for the current group) into the summary buffer ( <code>gnus-summary-insert-dormant-articles</code> ).

### 3.26.4 Really Various Summary Commands

<i>A D</i>	
<i>C-d</i>	If the current article is a collection of other articles (for instance, a digest), you might use this command to enter a group based on the that article ( <code>gnus-summary-enter-digest-group</code> ). Gnus will try to guess what article type is currently displayed unless you give a prefix to this command, which forces a “digest” interpretation. Basically, whenever you see a message that is a collection of other messages of some format, you <i>C-d</i> and read these messages in a more convenient fashion.
<i>C-M-d</i>	This command is very similar to the one above, but lets you gather several documents into one biiig group ( <code>gnus-summary-read-document</code> ). It does this by opening several <code>nndoc</code> groups for each document, and then opening an <code>nvirtual</code> group on top of these <code>nndoc</code> groups. This command understands the process/prefix convention (see <a href="#">[Process/Prefix]</a> , page <a href="#">[undefined]</a> ).
<i>C-t</i>	Toggle truncation of summary lines ( <code>gnus-summary-toggle-truncation</code> ). This will probably confuse the line centering function in the summary buffer, so it’s not a good idea to have truncation switched off while reading articles.
<i>=</i>	Expand the summary buffer window ( <code>gnus-summary-expand-window</code> ). If given a prefix, force an <code>article</code> window configuration.
<i>C-M-e</i>	Edit the group parameters (see <a href="#">[Group Parameters]</a> , page <a href="#">[undefined]</a> ) of the current group ( <code>gnus-summary-edit-parameters</code> ).
<i>C-M-a</i>	Customize the group parameters (see <a href="#">[Group Parameters]</a> , page <a href="#">[undefined]</a> ) of the current group ( <code>gnus-summary-customize-parameters</code> ).

### 3.27 Exiting the Summary Buffer

Exiting from the summary buffer will normally update all info on the group and return you to the group buffer.

**Z Z**

**Z Q**

**q** Exit the current group and update all information on the group (`gnus-summary-exit`). `gnus-summary-prepare-exit-hook` is called before doing much of the exiting, which calls `gnus-summary-expire-articles` by default. `gnus-summary-exit-hook` is called after finishing the exit process. `gnus-group-no-more-groups-hook` is run when returning to group mode having no more (unread) groups.

**Z E**

**Q** Exit the current group without updating any information on the group (`gnus-summary-exit-no-update`).

**Z c**

**c** Mark all unticked articles in the group as read and then exit (`gnus-summary-catchup-and-exit`).

**Z C**

Mark all articles, even the ticked ones, as read and then exit (`gnus-summary-catchup-all-and-exit`).

**Z n**

Mark all articles as read and go to the next group (`gnus-summary-catchup-and-goto-next-group`).

**Z R**

**C-x C-s** Exit this group, and then enter it again (`gnus-summary-reselect-current-group`). If given a prefix, select all articles, both read and unread.

**Z G**

**M-g** Exit the group, check for new articles in the group, and select the group (`gnus-summary-rescan-group`). If given a prefix, select all articles, both read and unread.

**Z N**

Exit the group and go to the next group (`gnus-summary-next-group`).

**Z P**

Exit the group and go to the previous group (`gnus-summary-prev-group`).

**Z s**

Save the current number of read/marked articles in the dribble buffer and then save the dribble buffer (`gnus-summary-save-newssrc`). If given a prefix, also save the `‘.newsrc’` file(s). Using this command will make exit without updating (the `Q` command) worthless.

`gnus-exit-group-hook` is called when you exit the current group with an “updating” exit. For instance `Q` (`gnus-summary-exit-no-update`) does not call this hook.

If you’re in the habit of exiting groups, and then changing your mind about it, you might set `gnus-kill-summary-on-exit` to `nil`. If you do that, Gnus won’t kill the summary buffer when you exit it. (Quelle surprise!) Instead it will change the name of the buffer to something like `‘*Dead Summary ... *’` and install a minor mode called `gnus-dead-summary-mode`. Now, if you switch back to this buffer, you’ll find that all keys are mapped to a

function called `gnus-summary-wake-up-the-dead`. So tapping any keys in a dead summary buffer will result in a live, normal summary buffer.

There will never be more than one dead summary buffer at any one time.

The data on the current group will be updated (which articles you have read, which articles you have replied to, etc.) when you exit the summary buffer. If the `gnus-use-cross-reference` variable is `t` (which is the default), articles that are cross-referenced to this group and are marked as read, will also be marked as read in the other subscribed groups they were cross-posted to. If this variable is neither `nil` nor `t`, the article will be marked as read in both subscribed and unsubscribed groups (see [\[Crosspost Handling\]](#), page [\[undefined\]](#)).

### 3.28 Crosspost Handling

Marking cross-posted articles as read ensures that you'll never have to read the same article more than once. Unless, of course, somebody has posted it to several groups separately. Posting the same article to several groups (not cross-posting) is called *spamming*, and you are by law required to send nasty-grams to anyone who perpetrates such a heinous crime. You may want to try NoCeM handling to filter out spam (see [\[NoCeM\]](#), page [\[undefined\]](#)).

Remember: Cross-posting is kinda ok, but posting the same article separately to several groups is not. Massive cross-posting (aka. *velveeta*) is to be avoided at all costs, and you can even use the `gnus-summary-mail-crosspost-complaint` command to complain about excessive crossposting (see [\[Summary Mail Commands\]](#), page [\[undefined\]](#)).

One thing that may cause Gnus to not do the cross-posting thing correctly is if you use an NNTP server that supports XOVER (which is very nice, because it speeds things up considerably) which does not include the `Xref` header in its NOV lines. This is Evil, but all too common, alas, alack. Gnus tries to Do The Right Thing even with XOVER by registering the `Xref` lines of all articles you actually read, but if you kill the articles, or just mark them as read without reading them, Gnus will not get a chance to snoop the `Xref` lines out of these articles, and will be unable to use the cross reference mechanism.

To check whether your NNTP server includes the `Xref` header in its overview files, try `'telnet your.nntp.server nntp'`, `'MODE READER'` on inn servers, and then say `'LIST overview.fmt'`. This may not work, but if it does, and the last line you get does not read `'Xref:full'`, then you should shout and whine at your news admin until she includes the `Xref` header in the overview files.

If you want Gnus to get the `Xrefs` right all the time, you have to set `gnus-nov-is-evil` to `t`, which slows things down considerably.

C'est la vie.

For an alternative approach, see [\[Duplicate Suppression\]](#), page [\[undefined\]](#).

### 3.29 Duplicate Suppression

By default, Gnus tries to make sure that you don't have to read the same article more than once by utilizing the crossposting mechanism (see [\[Crosspost Handling\]](#), page [\[undefined\]](#)). However, that simple and efficient approach may not work satisfactory for some users for various reasons.



1. The NNTP server may fail to generate the **Xref** header. This is evil and not very common.
2. The NNTP server may fail to include the **Xref** header in the `‘.overview’` data bases. This is evil and all too common, alas.
3. You may be reading the same group (or several related groups) from different NNTP servers.
4. You may be getting mail that duplicates articles posted to groups.

I’m sure there are other situations where **Xref** handling fails as well, but these four are the most common situations.

If, and only if, **Xref** handling fails for you, then you may consider switching on *duplicate suppression*. If you do so, Gnus will remember the **Message-IDs** of all articles you have read or otherwise marked as read, and then, as if by magic, mark them as read all subsequent times you see them—in *all* groups. Using this mechanism is quite likely to be somewhat inefficient, but not overly so. It’s certainly preferable to reading the same articles more than once.

Duplicate suppression is not a very subtle instrument. It’s more like a sledge hammer than anything else. It works in a very simple fashion—if you have marked an article as read, it adds this **Message-ID** to a cache. The next time it sees this **Message-ID**, it will mark the article as read with the ‘M’ mark. It doesn’t care what group it saw the article in.

#### **gnus-suppress-duplicates**

If non-**nil**, suppress duplicates.

#### **gnus-save-duplicate-list**

If non-**nil**, save the list of duplicates to a file. This will make startup and shutdown take longer, so the default is **nil**. However, this means that only duplicate articles read in a single Gnus session are suppressed.

#### **gnus-duplicate-list-length**

This variable says how many **Message-IDs** to keep in the duplicate suppression list. The default is 10000.

#### **gnus-duplicate-file**

The name of the file to store the duplicate suppression list in. The default is `‘~/News/suppression’`.

If you have a tendency to stop and start Gnus often, setting **gnus-save-duplicate-list** to **t** is probably a good idea. If you leave Gnus running for weeks on end, you may have it **nil**. On the other hand, saving the list makes startup and shutdown much slower, so that means that if you stop and start Gnus often, you should set **gnus-save-duplicate-list** to **nil**. Uhm. I’ll leave this up to you to figure out, I think.

## **3.30 Security**

Gnus is able to verify signed messages or decrypt encrypted messages. The formats that are supported are PGP, PGP/MIME and S/MIME, however you need some external programs to get things to work:



1. To handle PGP and PGP/MIME messages, you have to install an OpenPGP implementation such as GnuPG. The Lisp interface to GnuPG included with Gnus is called PGG (see [section “PGG” in \*PGG Manual\*](#)), but Mailcrypt and gpg.el are also supported.
2. To handle S/MIME message, you need to install OpenSSL. OpenSSL 0.9.6 or newer is recommended.

More information on how to set things up can be found in the message manual (see [section “Security” in \*Message Manual\*](#)).

#### mm-verify-option

Option of verifying signed parts. **never**, not verify; **always**, always verify; **known**, only verify known protocols. Otherwise, ask user.

#### mm-decrypt-option

Option of decrypting encrypted parts. **never**, no decryption; **always**, always decrypt; **known**, only decrypt known protocols. Otherwise, ask user.

#### mml1991-use

Symbol indicating elisp interface to OpenPGP implementation for PGP messages. The default is **pgg**, but **mailcrypt** and **gpg** are also supported although deprecated.

#### mml2015-use

Symbol indicating elisp interface to OpenPGP implementation for PGP/MIME messages. The default is **pgg**, but **mailcrypt** and **gpg** are also supported although deprecated.

Snarfing OpenPGP keys (i.e., importing keys from articles into your key ring) is not supported explicitly through a menu item or command, rather Gnus do detect and label keys as ‘**application/pgp-keys**’, allowing you to specify whatever action you think is appropriate through the usual MIME infrastructure. You can use a ‘**~/mailcap**’ entry (see [section “mailcap” in \*The Emacs MIME Manual\*](#)) such as the following to import keys using GNU Privacy Guard when you click on the MIME button (see [\[Using MIME\]](#), [page \[undefined\]](#)).

```
application/pgp-keys; gpg --import --interactive --verbose; needsterminal
```

This happens to also be the default action defined in **mailcap-mime-data**.

## 3.31 Mailing List

Gnus understands some mailing list fields of RFC 2369. To enable it, add a **to-list** group parameter (see [\[Group Parameters\]](#), [page \[undefined\]](#)), possibly using **A M (gnus-mailing-list-insinuate)** in the summary buffer.

That enables the following commands to the summary buffer:

- C-c C-n h** Send a message to fetch mailing list help, if List-Help field exists.
- C-c C-n s** Send a message to subscribe the mailing list, if List-Subscribe field exists.
- C-c C-n u** Send a message to unsubscribe the mailing list, if List-Unsubscribe field exists.
- C-c C-n p** Post to the mailing list, if List-Post field exists.
- C-c C-n o** Send a message to the mailing list owner, if List-Owner field exists.
- C-c C-n a** Browse the mailing list archive, if List-Archive field exists.



## 4 Article Buffer

The articles are displayed in the article buffer, of which there is only one. All the summary buffers share the same article buffer unless you tell Gnus otherwise.

### 4.1 Hiding Headers

The top section of each article is the *head*. (The rest is the *body*, but you may have guessed that already.)

There is a lot of useful information in the head: the name of the person who wrote the article, the date it was written and the subject of the article. That's well and nice, but there's also lots of information most people do not want to see—what systems the article has passed through before reaching you, the **Message-ID**, the **References**, etc. ad nauseam—and you'll probably want to get rid of some of those lines. If you want to keep all those lines in the article buffer, you can set `gnus-show-all-headers` to `t`.

Gnus provides you with two variables for sifting headers:

#### `gnus-visible-headers`

If this variable is non-`nil`, it should be a regular expression that says what headers you wish to keep in the article buffer. All headers that do not match this variable will be hidden.

For instance, if you only want to see the name of the person who wrote the article and the subject, you'd say:

```
(setq gnus-visible-headers "^From:\\|^Subject:")
```

This variable can also be a list of regexps to match headers to remain visible.

#### `gnus-ignored-headers`

This variable is the reverse of `gnus-visible-headers`. If this variable is set (and `gnus-visible-headers` is `nil`), it should be a regular expression that matches all lines that you want to hide. All lines that do not match this variable will remain visible.

For instance, if you just want to get rid of the **References** line and the **Xref** line, you might say:

```
(setq gnus-ignored-headers "^References:\\|^Xref:")
```

This variable can also be a list of regexps to match headers to be removed.

Note that if `gnus-visible-headers` is non-`nil`, this variable will have no effect.

Gnus can also sort the headers for you. (It does this by default.) You can control the sorting by setting the `gnus-sorted-header-list` variable. It is a list of regular expressions that says in what order the headers are to be displayed.

For instance, if you want the name of the author of the article first, and then the subject, you might say something like:

```
(setq gnus-sorted-header-list '("^From:" "^Subject:"))
```

Any headers that are to remain visible, but are not listed in this variable, will be displayed in random order after all the headers listed in this variable.

You can hide further boring headers by setting `gnus-treat-hide-boring-headers` to `head`. What this function does depends on the `gnus-boring-article-headers` variable. It's a list, but this list doesn't actually contain header names. Instead it lists various *boring conditions* that Gnus can check and remove from sight.

These conditions are:

<code>empty</code>	Remove all empty headers.
<code>followup-to</code>	Remove the <code>Followup-To</code> header if it is identical to the <code>Newsgroups</code> header.
<code>reply-to</code>	Remove the <code>Reply-To</code> header if it lists the same addresses as the <code>From</code> header, or if the <code>broken-reply-to</code> group parameter is set.
<code>newsgroups</code>	Remove the <code>Newsgroups</code> header if it only contains the current group name.
<code>to-address</code>	Remove the <code>To</code> header if it only contains the address identical to the current group's <code>to-address</code> parameter.
<code>to-list</code>	Remove the <code>To</code> header if it only contains the address identical to the current group's <code>to-list</code> parameter.
<code>cc-list</code>	Remove the <code>CC</code> header if it only contains the address identical to the current group's <code>to-list</code> parameter.
<code>date</code>	Remove the <code>Date</code> header if the article is less than three days old.
<code>long-to</code>	Remove the <code>To</code> header if it is very long.
<code>many-to</code>	Remove all <code>To</code> headers if there are more than one.

To include these three elements, you could say something like:

```
(setq gnus-boring-article-headers
      '(empty followup-to reply-to))
```

This is also the default value for this variable.

## 4.2 Using MIME

Mime is a standard for waving your hands through the air, aimlessly, while people stand around yawning.

MIME, however, is a standard for encoding your articles, aimlessly, while all newsreaders die of fear.

MIME may specify what character set the article uses, the encoding of the characters, and it also makes it possible to embed pictures and other naughty stuff in innocent-looking articles.

Gnus pushes MIME articles through `gnus-display-mime-function` to display the MIME parts. This is `gnus-display-mime` by default, which creates a bundle of clickable buttons that can be used to display, save and manipulate the MIME objects.

The following commands are available when you have placed point over a MIME button:

*RET (Article)*

*BUTTON-2 (Article)*

Toggle displaying of the MIME object (`gnus-article-press-button`). If built-in viewers can not display the object, Gnus resorts to external viewers in the ‘mailcap’ files. If a viewer has the ‘copiousoutput’ specification, the object is displayed inline.

*M-RET (Article)*

*v (Article)*

Prompt for a method, and then view the MIME object using this method (`gnus-mime-view-part`).

*t (Article)*

View the MIME object as if it were a different MIME media type (`gnus-mime-view-part-as-type`).

*C (Article)*

Prompt for a charset, and then view the MIME object using this charset (`gnus-mime-view-part-as-charset`).

*o (Article)*

Prompt for a file name, and then save the MIME object (`gnus-mime-save-part`).

*C-o (Article)*

Prompt for a file name, then save the MIME object and strip it from the article. Then proceed to article editing, where a reasonable suggestion is being made on how the altered article should look like. The stripped MIME object will be referred via the message/external-body MIME type. (`gnus-mime-save-part-and-strip`).

*d (Article)*

Delete the MIME object from the article and replace it with some information about the removed MIME object (`gnus-mime-delete-part`).

*c (Article)*

Copy the MIME object to a fresh buffer and display this buffer (`gnus-mime-copy-part`). Compressed files like ‘.gz’ and ‘.bz2’ are automatically decompressed if `auto-compression-mode` is enabled (see [section “Accessing Compressed Files”](#) in *The Emacs Editor*).

*p (Article)*

Print the MIME object (`gnus-mime-print-part`). This command respects the ‘print=’ specifications in the ‘.mailcap’ file.

*i (Article)*

Insert the contents of the MIME object into the buffer (`gnus-mime-inline-part`) as text/plain. If given a prefix, insert the raw contents without decoding. If given a numerical prefix, you can do semi-manual charset stuff (see `gnus-summary-show-article-charset-alist` in [\(undefined\)](#) [Paging the Article], page [\(undefined\)](#)).

***E (Article)***

View the MIME object with an internal viewer. If no internal viewer is available, use an external viewer (`gnus-mime-view-part-internally`).

***e (Article)***

View the MIME object with an external viewer. (`gnus-mime-view-part-externally`).

***| (Article)***

Output the MIME object to a process (`gnus-mime-pipe-part`).

***. (Article)***

Interactively run an action on the MIME object (`gnus-mime-action-on-part`).

Gnus will display some MIME objects automatically. The way Gnus determines which parts to do this with is described in the Emacs MIME manual.

It might be best to just use the toggling functions from the article buffer to avoid getting nasty surprises. (For instance, you enter the group `alt.sing-a-long` and, before you know it, MIME has decoded the sound file in the article and some horrible sing-a-long song comes screaming out your speakers, and you can't find the volume button, because there isn't one, and people are starting to look at you, and you try to stop the program, but you can't, and you can't find the program to control the volume, and everybody else in the room suddenly decides to look at you disdainfully, and you'll feel rather stupid.)

Any similarity to real events and people is purely coincidental. Ahem.

Also see [\[undefined\]](#) [MIME Commands], page [\[undefined\]](#).

### 4.3 Customizing Articles

A slew of functions for customizing how the articles are to look like exist. You can call these functions interactively (see [\[undefined\]](#) [Article Washing], page [\[undefined\]](#)), or you can have them called automatically when you select the articles.

To have them called automatically, you should set the corresponding “treatment” variable. For instance, to have headers hidden, you'd set `gnus-treat-hide-headers`. Below is a list of variables that can be set, but first we discuss the values these variables can have.

Note: Some values, while valid, make little sense. Check the list below for sensible values.

1. `nil`: Don't do this treatment.
2. `t`: Do this treatment on all body parts.
3. `head`: Do the treatment on the headers.
4. `last`: Do this treatment on the last part.
5. An integer: Do this treatment on all body parts that have a length less than this number.
6. A list of strings: Do this treatment on all body parts that are in articles that are read in groups that have names that match one of the regexps in the list.
7. A list where the first element is not a string:

The list is evaluated recursively. The first element of the list is a predicate. The following predicates are recognized: `or`, `and`, `not` and `typep`. Here's an example:

```
(or last
  (typep "text/x-vcard"))
```

You may have noticed that the word *part* is used here. This refers to the fact that some messages are MIME multipart articles that may be divided into several parts. Articles that are not multipart are considered to contain just a single part.

Are the treatments applied to all sorts of multipart parts? Yes, if you want to, but by default, only ‘text/plain’ parts are given the treatment. This is controlled by the `gnus-article-treat-types` variable, which is a list of regular expressions that are matched to the type of the part. This variable is ignored if the value of the controlling variable is a predicate list, as described above.

The following treatment options are available. The easiest way to customize this is to examine the `gnus-article-treat` customization group. Values in parenthesis are suggested sensible values. Others are possible but those listed are probably sufficient for most people.

```
gnus-treat-buttonize (t, integer)
gnus-treat-buttonize-head (head)
```

See [\(undefined\)](#) [Article Buttons], page [\(undefined\)](#).

```
gnus-treat-capitalize-sentences (t, integer)
gnus-treat-overstrike (t, integer)
gnus-treat-strip-cr (t, integer)
gnus-treat-strip-headers-in-body (t, integer)
gnus-treat-strip-leading-blank-lines (t, integer)
gnus-treat-strip-multiple-blank-lines (t, integer)
gnus-treat-strip-pem (t, last, integer)
gnus-treat-strip-trailing-blank-lines (t, last, integer)
gnus-treat-unsplit-urls (t, integer)
gnus-treat-wash-html (t, integer)
```

See [\(undefined\)](#) [Article Washing], page [\(undefined\)](#).

```
gnus-treat-date-english (head)
gnus-treat-date-iso8601 (head)
gnus-treat-date-lapsed (head)
gnus-treat-date-local (head)
gnus-treat-date-original (head)
gnus-treat-date-user-defined (head)
gnus-treat-date-ut (head)
```

See [\(undefined\)](#) [Article Date], page [\(undefined\)](#).

```
gnus-treat-from-picon (head)
gnus-treat-mail-picon (head)
gnus-treat-newsgroups-picon (head)
```

See [\(undefined\)](#) [Picons], page [\(undefined\)](#).

```
gnus-treat-display-smileys (t, integer)
gnus-treat-body-boundary (head)
```

Adds a delimiter between header and body, the string used as delimiter is controlled by `gnus-body-boundary-delimiter`.

See [\(undefined\)](#) [Smileys], page [\(undefined\)](#).

`gnus-treat-display-x-face` (head)  
 See [\(undefined\)](#) [X-Face], page [\(undefined\)](#).

`gnus-treat-display-face` (head)  
 See [\(undefined\)](#) [Face], page [\(undefined\)](#).

`gnus-treat-emphasize` (t, head, integer)  
`gnus-treat-fill-article` (t, integer)  
`gnus-treat-fill-long-lines` (t, integer)  
`gnus-treat-hide-boring-headers` (head)  
`gnus-treat-hide-citation` (t, integer)  
`gnus-treat-hide-citation-maybe` (t, integer)  
`gnus-treat-hide-headers` (head)  
`gnus-treat-hide-signature` (t, last)  
`gnus-treat-strip-banner` (t, last)  
`gnus-treat-strip-list-identifiers` (head)  
 See [\(undefined\)](#) [Article Hiding], page [\(undefined\)](#).

`gnus-treat-highlight-citation` (t, integer)  
`gnus-treat-highlight-headers` (head)  
`gnus-treat-highlight-signature` (t, last, integer)  
 See [\(undefined\)](#) [Article Highlighting], page [\(undefined\)](#).

`gnus-treat-play-sounds`  
`gnus-treat-translate`  
`gnus-treat-x-pgp-sig` (head)  
`gnus-treat-unfold-headers` (head)  
`gnus-treat-fold-headers` (head)  
`gnus-treat-fold-newsgroups` (head)  
`gnus-treat-leading-whitespace` (head)  
 See [\(undefined\)](#) [Article Header], page [\(undefined\)](#).

You can, of course, write your own functions to be called from `gnus-part-display-hook`. The functions are called narrowed to the part, and you can do anything you like, pretty much. There is no information that you have to keep in the buffer—you can change everything.

## 4.4 Article Keymap

Most of the keystrokes in the summary buffer can also be used in the article buffer. They should behave as if you typed them in the summary buffer, which means that you don't actually have to have a summary buffer displayed while reading. You can do it all from the article buffer.

A few additional keystrokes are available:

<b>SPACE</b>	Scroll forwards one page ( <code>gnus-article-next-page</code> ). This is exactly the same as <code>h SPACE h</code> .
<b>DEL</b>	Scroll backwards one page ( <code>gnus-article-prev-page</code> ). This is exactly the same as <code>h DEL h</code> .



<i>C-c ^</i>	If point is in the neighborhood of a <code>Message-ID</code> and you press <i>C-c ^</i> , Gnus will try to get that article from the server ( <code>gnus-article-refer-article</code> ).
<i>C-c C-m</i>	Send a reply to the address near point ( <code>gnus-article-mail</code> ). If given a prefix, include the mail.
<i>s</i>	Reconfigure the buffers so that the summary buffer becomes visible ( <code>gnus-article-show-summary</code> ).
<i>?</i>	Give a very brief description of the available keystrokes ( <code>gnus-article-describe-briefly</code> ).
<i>TAB</i>	Go to the next button, if any ( <code>gnus-article-next-button</code> ). This only makes sense if you have buttonizing turned on.
<i>M-TAB</i>	Go to the previous button, if any ( <code>gnus-article-prev-button</code> ).
<i>R</i>	Send a reply to the current article and yank the current article ( <code>gnus-article-reply-with-original</code> ). If given a prefix, make a wide reply. If the region is active, only yank the text in the region.
<i>F</i>	Send a followup to the current article and yank the current article ( <code>gnus-article-followup-with-original</code> ). If given a prefix, make a wide reply. If the region is active, only yank the text in the region.

## 4.5 Misc Article

### `gnus-single-article-buffer`

If non-`nil`, use the same article buffer for all the groups. (This is the default.)  
If `nil`, each group will have its own article buffer.

### `gnus-article-decode-hook`

Hook used to decode MIME articles. The default value is (`article-decode-charset article-decode-encoded-words`)

### `gnus-article-prepare-hook`

This hook is called right after the article has been inserted into the article buffer. It is mainly intended for functions that do something depending on the contents; it should probably not be used for changing the contents of the article buffer.

### `gnus-article-mode-hook`

Hook called in article mode buffers.

### `gnus-article-mode-syntax-table`

Syntax table used in article buffers. It is initialized from `text-mode-syntax-table`.

### `gnus-article-over-scroll`

If non-`nil`, allow scrolling the article buffer even when there no more new text to scroll in. The default is `nil`.

### `gnus-article-mode-line-format`

This variable is a format string along the same lines as `gnus-summary-mode-line-format` (see [\[Summary Buffer Mode Line\]](#), page [\[undefined\]](#)). It accepts the same format specifications as that variable, with two extensions:

‘w’	The <i>wash status</i> of the article. This is a short string with one character for each possible article wash operation that may have been performed. The characters and their meaning:
‘c’	Displayed when cited text may be hidden in the article buffer.
‘h’	Displayed when headers are hidden in the article buffer.
‘p’	Displayed when article is digitally signed or encrypted, and Gnus has hidden the security headers. (N.B. does not tell anything about security status, i.e. good or bad signature.)
‘s’	Displayed when the signature has been hidden in the Article buffer.
‘o’	Displayed when Gnus has treated overstrike characters in the article buffer.
‘e’	Displayed when Gnus has treated emphasised strings in the article buffer.
‘m’	The number of MIME parts in the article.

**gnus-break-pages**

Controls whether *page breaking* is to take place. If this variable is non-**nil**, the articles will be divided into pages whenever a page delimiter appears in the article. If this variable is **nil**, paging will not be done.

**gnus-page-delimiter**

This is the delimiter mentioned above. By default, it is ‘^L’ (formfeed).

**gnus-use-idna**

This variable controls whether Gnus performs IDNA decoding of internationalized domain names inside ‘From’, ‘To’ and ‘Cc’ headers. This requires **GNU Libidn**, and this variable is only enabled if you have installed it.

## 5 Composing Messages

All commands for posting and mailing will put you in a message buffer where you can edit the article all you like, before you send the article by pressing `C-c C-c`. See [section “Overview” in \*Message Manual\*](#). Where the message will be posted/mailed to depends on your setup (see [\[Posting Server\]](#), page [\[undefined\]](#)).

Also see [\[Canceling and Superseding\]](#), page [\[undefined\]](#) for information on how to remove articles you shouldn’t have posted.

### 5.1 Mail

Variables for customizing outgoing mail:

#### `gnus-uu-digest-headers`

List of regexps to match headers included in digested messages. The headers will be included in the sequence they are matched. If `nil` include all headers.

#### `gnus-add-to-list`

If non-`nil`, add a `to-list` group parameter to mail groups that have none when you do a `a`.

#### `gnus-confirm-mail-reply-to-news`

This can also be a function receiving the group name as the only parameter which should return non-`nil` if a confirmation is needed, or a regular expression matching group names, where confirmation is should be asked for.

If you find yourself never wanting to reply to mail, but occasionally press `R` anyway, this variable might be for you.

#### `gnus-confirm-treat-mail-like-news`

If non-`nil`, Gnus also requests confirmation according to `gnus-confirm-mail-reply-to-news` when replying to mail. This is useful for treating mailing lists like newsgroups.

### 5.2 Posting Server

When you press those magical `C-c C-c` keys to ship off your latest (extremely intelligent, of course) article, where does it go?

Thank you for asking. I hate you.

It can be quite complicated.

When posting news, Message usually invokes `message-send-news` (see [section “News Variables” in \*Message Manual\*](#)). Normally, Gnus will post using the same select method as you’re reading from (which might be convenient if you’re reading lots of groups from different private servers). However. If the server you’re reading from doesn’t allow posting, just reading, you probably want to use some other server to post your (extremely intelligent and fabulously interesting) articles. You can then set the `gnus-post-method` to some other method:

```
(setq gnus-post-method '(nnspool ""))
```

Now, if you've done this, and then this server rejects your article, or this server is down, what do you do then? To override this variable you can use a non-zero prefix to the `C-c C-c` command to force using the "current" server, to get back the default behavior, for posting.

If you give a zero prefix (i.e., `C-u 0 C-c C-c`) to that command, Gnus will prompt you for what method to use for posting.

You can also set `gnus-post-method` to a list of select methods. If that's the case, Gnus will always prompt you for what method to use for posting.

Finally, if you want to always post using the native select method, you can set this variable to `native`.

When sending mail, Message invokes `message-send-mail-function`. The default function, `message-send-mail-with-sendmail`, pipes your article to the `sendmail` binary for further queuing and sending. When your local system is not configured for sending mail using `sendmail`, and you have access to a remote SMTP server, you can set `message-send-mail-function` to `smtpmail-send-it` and make sure to setup the `smtpmail` package correctly. An example:

```
(setq message-send-mail-function 'smtpmail-send-it
      smtpmail-default-smtp-server "YOUR SMTP HOST")
```

To the thing similar to this, there is `message-smtpmail-send-it`. It is useful if your ISP requires the POP-before-SMTP authentication. See [\[POP before SMTP\]](#), page [\[undefined\]](#).

Other possible choices for `message-send-mail-function` includes `message-send-mail-with-mh`, `message-send-mail-with-qmail`, and `feedmail-send-it`.

### 5.3 POP before SMTP

Does your ISP require the POP-before-SMTP authentication? It is whether you need to connect to the POP mail server within a certain time before sending mails. If so, there is a convenient way. To do that, put the following lines in your `~/gnus.el` file:

```
(setq message-send-mail-function 'message-smtpmail-send-it)
(add-hook 'message-send-mail-hook 'mail-source-touch-pop)
```

It means to let Gnus connect to the POP mail server in advance whenever you send a mail. The `mail-source-touch-pop` function does only a POP authentication according to the value of `mail-sources` without fetching mails, just before sending a mail. Note that you have to use `message-smtpmail-send-it` which runs `message-send-mail-hook` rather than `smtpmail-send-it` and set the value of `mail-sources` for a POP connection correctly. See [\[Mail Sources\]](#), page [\[undefined\]](#).

If you have two or more POP mail servers set in `mail-sources`, you may want to specify one of them to `mail-source-primary-source` as the POP mail server to be used for the POP-before-SMTP authentication. If it is your primary POP mail server (i.e., you are fetching mails mainly from that server), you can set it permanently as follows:

```
(setq mail-source-primary-source
      '(pop :server "pop3.mail.server"
            :password "secret"))
```

Otherwise, bind it dynamically only when performing the POP-before-SMTP authentication as follows:

```
(add-hook 'message-send-mail-hook
  (lambda ()
    (let ((mail-source-primary-source
          '(pop :server "pop3.mail.server"
                :password "secret"))))
      (mail-source-touch-pop))))
```

## 5.4 Mail and Post

Here's a list of variables relevant to both mailing and posting:

### `gnus-mailing-list-groups`

If your news server offers groups that are really mailing lists gatewayed to the NNTP server, you can read those groups without problems, but you can't post/followup to them without some difficulty. One solution is to add a `to-address` to the group parameters (see [\(undefined\) \[Group Parameters\]](#), [page \(undefined\)](#)). An easier thing to do is set the `gnus-mailing-list-groups` to a regexp that matches the groups that really are mailing lists. Then, at least, followups to the mailing lists will work most of the time. Posting to these groups (a) is still a pain, though.

### `gnus-user-agent`

This variable controls which information should be exposed in the User-Agent header. It can be one of the symbols `gnus` (show only Gnus version), `emacs-gnus` (show only Emacs and Gnus versions), `emacs-gnus-config` (same as `emacs-gnus` plus system configuration), `emacs-gnus-type` (same as `emacs-gnus` plus system type) or a custom string. If you set it to a string, be sure to use a valid format, see RFC 2616.

You may want to do spell-checking on messages that you send out. Or, if you don't want to spell-check by hand, you could add automatic spell-checking via the `ispell` package:

```
(add-hook 'message-send-hook 'ispell-message)
```

If you want to change the `ispell` dictionary based on what group you're in, you could say something like the following:

```
(add-hook 'gnus-select-group-hook
  (lambda ()
    (cond
      ((string-match
        "^de\\\\" (gnus-group-real-name gnus-newsgroup-name))
       (ispell-change-dictionary "deutsch"))
      (t
       (ispell-change-dictionary "english")))))
```

Modify to suit your needs.

## 5.5 Archived Messages

Gnus provides a few different methods for storing the mail and news you send. The default method is to use the *archive virtual server* to store the messages. If you want to disable

this completely, the `gnus-message-archive-group` variable should be `nil`, which is the default.

For archiving interesting messages in a group you read, see the `B c` (`gnus-summary-copy-article`) command (see [\[Mail Group Commands\]](#), page [\[undefined\]](#)).

`gnus-message-archive-method` says what virtual server Gnus is to use to store sent messages. The default is:

```
(nnfolder "archive"
  (nnfolder-directory "~/Mail/archive")
  (nnfolder-active-file "~/Mail/archive/active")
  (nnfolder-get-new-mail nil)
  (nnfolder-inhibit-expiry t))
```

You can, however, use any mail select method (`nnml`, `nnmbox`, etc.). `nnfolder` is a quite likable select method for doing this sort of thing, though. If you don't like the default directory chosen, you could say something like:

```
(setq gnus-message-archive-method
  '(nnfolder "archive"
    (nnfolder-inhibit-expiry t)
    (nnfolder-active-file "~/News/sent-mail/active")
    (nnfolder-directory "~/News/sent-mail/")))
```

Gnus will insert `Gcc` headers in all outgoing messages that point to one or more group(s) on that server. Which group to use is determined by the `gnus-message-archive-group` variable.

This variable can be used to do the following:

a string      Messages will be saved in that group.

Note that you can include a select method in the group name, then the message will not be stored in the select method given by `gnus-message-archive-method`, but in the select method specified by the group name, instead. Suppose `gnus-message-archive-method` has the default value shown above. Then setting `gnus-message-archive-group` to `"foo"` means that outgoing messages are stored in `'nnfolder+archive:foo'`, but if you use the value `"nnml:foo"`, then outgoing messages will be stored in `'nnml:foo'`.

a list of strings

Messages will be saved in all those groups.

an alist of regexps, functions and forms

When a key “matches”, the result is used.

`nil`          No message archiving will take place. This is the default.

Let's illustrate:

Just saving to a single group called `'MisK'`:

```
(setq gnus-message-archive-group "MisK")
```

Saving to two groups, `'MisK'` and `'safe'`:

```
(setq gnus-message-archive-group '("MisK" "safe"))
```

Save to different groups based on what group you are in:

```
(setq gnus-message-archive-group
      '(("^alt" "sent-to-alt")
        ("mail" "sent-to-mail")
        (".*" "sent-to-misc")))
```

More complex stuff:

```
(setq gnus-message-archive-group
      '((if (message-news-p)
            "misc-news"
            "misc-mail")))
```

How about storing all news messages in one file, but storing all mail messages in one file per month:

```
(setq gnus-message-archive-group
      '((if (message-news-p)
            "misc-news"
            (concat "mail." (format-time-string "%Y-%m"))))))
```

Now, when you send a message off, it will be stored in the appropriate group. (If you want to disable storing for just one particular message, you can just remove the `Gcc` header that has been inserted.) The archive group will appear in the group buffer the next time you start Gnus, or the next time you press `F` in the group buffer. You can enter it and read the articles in it just like you'd read any other group. If the group gets really big and annoying, you can simply rename it (using `G r` in the group buffer) to something nice—`'misc-mail-september-1995'`, or whatever. New messages will continue to be stored in the old (now empty) group.

That's the default method of archiving sent messages. Gnus offers a different way for the people who don't like the default method. In that case you should set `gnus-message-archive-group` to `nil`; this will disable archiving.

#### `gnus-outgoing-message-group`

All outgoing messages will be put in this group. If you want to store all your outgoing mail and articles in the group `'nnml:archive'`, you set this variable to that value. This variable can also be a list of group names.

If you want to have greater control over what group to put each message in, you can set this variable to a function that checks the current newsgroup name and then returns a suitable group name (or list of names).

This variable can be used instead of `gnus-message-archive-group`, but the latter is the preferred method.

#### `gnus-gcc-mark-as-read`

If non-`nil`, automatically mark `Gcc` articles as read.

#### `gnus-gcc-externalize-attachments`

If `nil`, attach files as normal parts in `Gcc` copies; if a regexp and matches the `Gcc` group name, attach files as external parts; if it is `all`, attach local files as external parts; if it is other non-`nil`, the behavior is the same as `all`, but it may be changed in the future.

## 5.6 Posting Styles

All them variables, they make my head swim.

So what if you want a different **Organization** and signature based on what groups you post to? And you post both from your home machine and your work machine, and you want different **From** lines, and so on?

One way to do stuff like that is to write clever hooks that change the variables you need to have changed. That's a bit boring, so somebody came up with the bright idea of letting the user specify these things in a handy alist. Here's an example of a `gnus-posting-styles` variable:

```
((".*"
  (signature "Peace and happiness")
  (organization "What me?"))
 ("^comp"
  (signature "Death to everybody"))
 ("comp.emacs.i-love-it"
  (organization "Emacs is it")))
```

As you might surmise from this example, this alist consists of several *styles*. Each style will be applicable if the first element “matches”, in some form or other. The entire alist will be iterated over, from the beginning towards the end, and each match will be applied, which means that attributes in later styles that match override the same attributes in earlier matching styles. So ‘`comp.programming.literate`’ will have the ‘`Death to everybody`’ signature and the ‘`What me?`’ **Organization** header.

The first element in each style is called the *match*. If it's a string, then Gnus will try to regexp match it against the group name. If it is the form `(header match regexp)`, then Gnus will look in the original article for a header whose name is *match* and compare that *regexp*. *match* and *regexp* are strings. (The original article is the one you are replying or following up to. If you are not composing a reply or a followup, then there is nothing to match against.) If the *match* is a function symbol, that function will be called with no arguments. If it's a variable symbol, then the variable will be referenced. If it's a list, then that list will be `eval`ed. In any case, if this returns a non-`nil` value, then the style is said to *match*.

Each style may contain an arbitrary amount of *attributes*. Each attribute consists of a `(name value)` pair. In addition, you can also use the `(name :file value)` form or the `(name :value value)` form. Where `:file` signifies *value* represents a file name and its contents should be used as the attribute value, `:value` signifies *value* does not represent a file name explicitly. The attribute name can be one of:

- `signature`
- `signature-file`
- `x-face-file`
- `address`, overriding `user-mail-address`
- `name`, overriding `(user-full-name)`
- `body`

The attribute name can also be a string or a symbol. In that case, this will be used as a header name, and the value will be inserted in the headers of the article; if the value is `nil`,



the header name will be removed. If the attribute name is `eval`, the form is evaluated, and the result is thrown away.

The attribute value can be a string (used verbatim), a function with zero arguments (the return value will be used), a variable (its value will be used) or a list (it will be `eval`ed and the return value will be used). The functions and sexps are called/`eval`ed in the message buffer that is being set up. The headers of the current article are available through the `message-reply-headers` variable, which is a vector of the following headers: number subject from date id references chars lines xref extra.

If you wish to check whether the message you are about to compose is meant to be a news article or a mail message, you can check the values of the `message-news-p` and `message-mail-p` functions.

So here's a new example:

```
(setq gnus-posting-styles
  '((".*"
    (signature-file "~/signature")
    (name "User Name")
    ("X-Home-Page" (getenv "WWW_HOME"))
    (organization "People's Front Against MWM"))
    ("^rec.humor"
    (signature my-funny-signature-randomizer))
    ((equal (system-name) "gnarly") ;; A form
    (signature my-quote-randomizer))
    (message-news-p ;; A function symbol
    (signature my-news-signature))
    (window-system ;; A value symbol
    ("X-Window-System" (format "%s" window-system)))
    ;; If I'm replying to Larsi, set the Organization header.
    ((header "from" "larsi.*org")
    (Organization "Somewhere, Inc.))
    ((posting-from-work-p) ;; A user defined function
    (signature-file "~/work-signature")
    (address "user@bar.foo")
    (body "You are fired.\n\nSincerely, your boss.")
    (organization "Important Work, Inc.))
    ("nnml:.*"
    (From (save-excursion
            (set-buffer gnus-article-buffer)
            (message-fetch-field "to"))))
    ("^nn.+:"
    (signature-file "~/mail-signature"))))
```

The `'nnml:.*'` rule means that you use the `To` address as the `From` address in all your outgoing replies, which might be handy if you fill many roles.

## 5.7 Drafts

If you are writing a message (mail or news) and suddenly remember that you have a steak in the oven (or some pesto in the food processor, you craaaazy vegetarians), you'll probably wish there was a method to save the message you are writing so that you can continue editing it some other day, and send it when you feel its finished.

Well, don't worry about it. Whenever you start composing a message of some sort using the Gnus mail and post commands, the buffer you get will automatically associate to an article in a special *draft* group. If you save the buffer the normal way (`C-x C-s`, for instance), the article will be saved there. (Auto-save files also go to the draft group.)

The draft group is a special group (which is implemented as an `nndraft` group, if you absolutely have to know) called '`nndraft:drafts`'. The variable `nndraft-directory` says where `nndraft` is to store its files. What makes this group special is that you can't tick any articles in it or mark any articles as read—all articles in the group are permanently unread.

If the group doesn't exist, it will be created and you'll be subscribed to it. The only way to make it disappear from the Group buffer is to unsubscribe it. The special properties of the draft group comes from a group property (see [\[Group Parameters\]](#), page [\(undefined\)](#)), and if lost the group behaves like any other group. This means the commands below will not be available. To restore the special properties of the group, the simplest way is to kill the group, using `C-k`, and restart Gnus. The group is automatically created again with the correct parameters. The content of the group is not lost.

When you want to continue editing the article, you simply enter the draft group and push `D e` (`gnus-draft-edit-message`) to do that. You will be placed in a buffer where you left off.

Rejected articles will also be put in this draft group (see [\[Rejected Articles\]](#), page [\(undefined\)](#)).

If you have lots of rejected messages you want to post (or mail) without doing further editing, you can use the `D s` command (`gnus-draft-send-message`). This command understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\(undefined\)](#)). The `D S` command (`gnus-draft-send-all-messages`) will ship off all messages in the buffer.

If you have some messages that you wish not to send, you can use the `D t` (`gnus-draft-toggle-sending`) command to mark the message as unsendable. This is a toggling command.

## 5.8 Rejected Articles

Sometimes a news server will reject an article. Perhaps the server doesn't like your face. Perhaps it just feels miserable. Perhaps *there be demons*. Perhaps you have included too much cited text. Perhaps the disk is full. Perhaps the server is down.

These situations are, of course, totally beyond the control of Gnus. (Gnus, of course, loves the way you look, always feels great, has angels fluttering around inside of it, doesn't care about how much cited text you include, never runs full and never goes down.) So Gnus saves these articles until some later time when the server feels better.

The rejected articles will automatically be put in a special draft group (see [\[Drafts\]](#), page [\[Security\]](#)). When the server comes back up again, you'd then typically enter that group and send all the articles off.

## 5.9 Signing and encrypting

Gnus can digitally sign and encrypt your messages, using vanilla PGP format or PGP/MIME or S/MIME. For decoding such messages, see the `mm-verify-option` and `mm-decrypt-option` options (see [\[Security\]](#), page [\[Security\]](#)).

Often, you would like to sign replies to people who send you signed messages. Even more often, you might want to encrypt messages which are in reply to encrypted messages. Gnus offers `gnus-message-replysign` to enable the former, and `gnus-message-replyencrypt` for the latter. In addition, setting `gnus-message-replysignencrypted` (on by default) will sign automatically encrypted messages.

Instructing MML to perform security operations on a MIME part is done using the `C-c C-m s` key map for signing and the `C-c C-m c` key map for encryption, as follows.

- `C-c C-m s`  
Digitally sign current message using S/MIME.
- `C-c C-m s o`  
Digitally sign current message using PGP.
- `C-c C-m s p`  
Digitally sign current message using PGP/MIME.
- `C-c C-m c s`  
Digitally encrypt current message using S/MIME.
- `C-c C-m c o`  
Digitally encrypt current message using PGP.
- `C-c C-m c p`  
Digitally encrypt current message using PGP/MIME.
- `C-c C-m C-n`  
Remove security related MML tags from message.

See [section “Security” in Message Manual](#), for more information.



## 6 Select Methods

A *foreign group* is a group not read by the usual (or default) means. It could be, for instance, a group from a different NNTP server, it could be a virtual group, or it could be your own personal mail group.

A foreign group (or any group, really) is specified by a *name* and a *select method*. To take the latter first, a select method is a list where the first element says what back end to use (e.g. `nntp`, `nnspool`, `nnml`) and the second element is the *server name*. There may be additional elements in the select method, where the value may have special meaning for the back end in question.

One could say that a select method defines a *virtual server*—so we do just that (see [\[Server Buffer\]](#), page [\(undefined\)](#)).

The *name* of the group is the name the back end will recognize the group as.

For instance, the group `'soc.motss'` on the NNTP server `'some.where.edu'` will have the name `'soc.motss'` and select method `(nntp "some.where.edu")`. Gnus will call this group `'nntp+some.where.edu:soc.motss'`, even though the `nntp` back end just knows this group as `'soc.motss'`.

The different methods all have their peculiarities, of course.

### 6.1 Server Buffer

Traditionally, a *server* is a machine or a piece of software that one connects to, and then requests information from. Gnus does not connect directly to any real servers, but does all transactions through one back end or other. But that's just putting one layer more between the actual media and Gnus, so we might just as well say that each back end represents a virtual server.

For instance, the `nntp` back end may be used to connect to several different actual NNTP servers, or, perhaps, to many different ports on the same actual NNTP server. You tell Gnus which back end to use, and what parameters to set by specifying a *select method*.

These select method specifications can sometimes become quite complicated—say, for instance, that you want to read from the NNTP server `'news.funet.fi'` on port number 13, which hangs if queried for NOV headers and has a buggy select. Ahem. Anyway, if you had to specify that for each group that used this server, that would be too much work, so Gnus offers a way of naming select methods, which is what you do in the server buffer.

To enter the server buffer, use the `^ (gnus-group-enter-server-mode)` command in the group buffer.

`gnus-server-mode-hook` is run when creating the server buffer.

#### 6.1.1 Server Buffer Format

You can change the look of the server buffer lines by changing the `gnus-server-line-format` variable. This is a *format*-like variable, with some simple extensions:

- 'h'        How the news is fetched—the back end name.
- 'n'        The name of this server.
- 'w'        Where the news is to be fetched from—the address.

‘s’            The opened/closed/denied status of the server.

The mode line can also be customized by using the `gnus-server-mode-line-format` variable (see [\[Mode Line Formatting\]](#), page [\[undefined\]](#)). The following specs are understood:

‘S’            Server name.

‘M’            Server method.

Also see [\[Formatting Variables\]](#), page [\[undefined\]](#).

### 6.1.2 Server Commands

*a*            Add a new server (`gnus-server-add-server`).

*e*            Edit a server (`gnus-server-edit-server`).

*SPACE*       Browse the current server (`gnus-server-read-server`).

*q*            Return to the group buffer (`gnus-server-exit`).

*k*            Kill the current server (`gnus-server-kill-server`).

*y*            Yank the previously killed server (`gnus-server-yank-server`).

*c*            Copy the current server (`gnus-server-copy-server`).

*l*            List all servers (`gnus-server-list-servers`).

*s*            Request that the server scan its sources for new articles (`gnus-server-scan-server`). This is mainly sensible with mail servers.

*g*            Request that the server regenerate all its data structures (`gnus-server-regenerate-server`). This can be useful if you have a mail back end that has gotten out of sync.

### 6.1.3 Example Methods

Most select methods are pretty simple and self-explanatory:

```
(nntp "news.funet.fi")
```

Reading directly from the spool is even simpler:

```
(nnspool "")
```

As you can see, the first element in a select method is the name of the back end, and the second is the *address*, or *name*, if you will.

After these two elements, there may be an arbitrary number of (*variable form*) pairs.

To go back to the first example—imagine that you want to read from port 15 on that machine. This is what the select method should look like then:

```
(nntp "news.funet.fi" (nntp-port-number 15))
```

You should read the documentation to each back end to find out what variables are relevant, but here’s an `nnmh` example:

`nnmh` is a mail back end that reads a spool-like structure. Say you have two structures that you wish to access: One is your private mail spool, and the other is a public one. Here’s the possible spec for your private mail:

```
(nnmh "private" (nnmh-directory "~/private/mail/"))
```

(This server is then called ‘private’, but you may have guessed that.)

Here’s the method for a public spool:

```
(nnmh "public"
  (nnmh-directory "/usr/information/spool/")
  (nnmh-get-new-mail nil))
```

If you are behind a firewall and only have access to the NNTP server from the firewall machine, you can instruct Gnus to `rlogin` on the firewall machine and telnet from there to the NNTP server. Doing this can be rather fiddly, but your virtual server definition should probably look something like this:

```
(nntp "firewall"
  (nntp-open-connection-function nntp-open-via-rlogin-and-telnet)
  (nntp-via-address "the.firewall.machine")
  (nntp-address "the.real.nntp.host")
  (nntp-end-of-line "\n"))
```

If you want to use the wonderful `ssh` program to provide a compressed connection over the modem line, you could add the following configuration to the example above:

```
(nntp-via-rlogin-command "ssh")
```

See also `nntp-via-rlogin-command-switches`.

If you’re behind a firewall, but have direct access to the outside world through a wrapper command like `runsocks`, you could open a socksified telnet connection to the news server as follows:

```
(nntp "outside"
  (nntp-pre-command "runsocks")
  (nntp-open-connection-function nntp-open-via-telnet)
  (nntp-address "the.news.server")
  (nntp-end-of-line "\n"))
```

This means that you have to have set up `ssh-agent` correctly to provide automatic authorization, of course. And to get a compressed connection, you have to have the ‘`Compression`’ option in the `ssh` ‘`config`’ file.

### 6.1.4 Creating a Virtual Server

If you’re saving lots of articles in the cache by using persistent articles, you may want to create a virtual server to read the cache.

First you need to add a new server. The `a` command does that. It would probably be best to use `nnml` to read the cache. You could also use `nnspool` or `nnmh`, though.

Type `a nnml RET cache RET`.

You should now have a brand new `nnml` virtual server called ‘`cache`’. You now need to edit it to have the right definitions. Type `e` to edit the server. You’ll be entered into a buffer that will contain the following:

```
(nnml "cache")
```

Change that to:

```
(nnml "cache"
  (nnml-directory "~/News/cache/")
  (nnml-active-file "~/News/cache/active"))
```

Type `C-c C-c` to return to the server buffer. If you now press `RET` over this virtual server, you should be entered into a browse buffer, and you should be able to enter any of the groups displayed.

### 6.1.5 Server Variables

One sticky point when defining variables (both on back ends and in Emacs in general) is that some variables are typically initialized from other variables when the definition of the variables is being loaded. If you change the “base” variable after the variables have been loaded, you won’t change the “derived” variables.

This typically affects directory and file variables. For instance, `nnml-directory` is ‘~/Mail/’ by default, and all `nnml` directory variables are initialized from that variable, so `nnml-active-file` will be ‘~/Mail/active’. If you define a new virtual `nnml` server, it will *not* suffice to set just `nnml-directory`—you have to explicitly set all the file variables to be what you want them to be. For a complete list of variables for each back end, see each back end’s section later in this manual, but here’s an example `nnml` definition:

```
(nnml "public"
  (nnml-directory "~/my-mail/")
  (nnml-active-file "~/my-mail/active")
  (nnml-newsgroups-file "~/my-mail/newsgroups"))
```

Server variables are often called *server parameters*.

### 6.1.6 Servers and Methods

Wherever you would normally use a select method (e.g. `gnus-secondary-select-method`, in the group select method, when browsing a foreign server) you can use a virtual server name instead. This could potentially save lots of typing. And it’s nice all over.

### 6.1.7 Unavailable Servers

If a server seems to be unreachable, Gnus will mark that server as **denied**. That means that any subsequent attempt to make contact with that server will just be ignored. “It can’t be opened,” Gnus will tell you, without making the least effort to see whether that is actually the case or not.

That might seem quite naughty, but it does make sense most of the time. Let’s say you have 10 groups subscribed to on server ‘`nephelococcygia.com`’. This server is located somewhere quite far away from you and the machine is quite slow, so it takes 1 minute just to find out that it refuses connection to you today. If Gnus were to attempt to do that 10 times, you’d be quite annoyed, so Gnus won’t attempt to do that. Once it has gotten a single “connection refused”, it will regard that server as “down”.

So, what happens if the machine was only feeling unwell temporarily? How do you test to see whether the machine has come up again?

You jump to the server buffer (see [\[Server Buffer\]](#), page [\[undefined\]](#)) and poke it with the following commands:



<i>O</i>	Try to establish connection to the server on the current line ( <code>gnus-server-open-server</code> ).
<i>C</i>	Close the connection (if any) to the server ( <code>gnus-server-close-server</code> ).
<i>D</i>	Mark the current server as unreachable ( <code>gnus-server-deny-server</code> ).
<i>M-o</i>	Open the connections to all servers in the buffer ( <code>gnus-server-open-all-servers</code> ).
<i>M-c</i>	Close the connections to all servers in the buffer ( <code>gnus-server-close-all-servers</code> ).
<i>R</i>	Remove all marks to whether Gnus was denied connection from any servers ( <code>gnus-server-remove-denials</code> ).
<i>L</i>	Set server status to offline ( <code>gnus-server-offline-server</code> ).

## 6.2 Getting News

A newsreader is normally used for reading news. Gnus currently provides only two methods of getting news—it can read from an NNTP server, or it can read from a local spool.

### 6.2.1 NNTP

Subscribing to a foreign group from an NNTP server is rather easy. You just specify `nntp` as method and the address of the NNTP server as the, uhm, address.

If the NNTP server is located at a non-standard port, setting the third element of the select method to this port number should allow you to connect to the right port. You'll have to edit the group info for that (see [Foreign Groups](#), page [134](#)).

The name of the foreign group can be the same as a native group. In fact, you can subscribe to the same group from as many different servers you feel like. There will be no name collisions.

The following variables can be used to create a virtual `nntp` server:

#### `nntp-server-opened-hook`

is run after a connection has been made. It can be used to send commands to the NNTP server after it has been contacted. By default it sends the command `MODE READER` to the server with the `nntp-send-mode-reader` function. This function should always be present in this hook.

#### `nntp-authinfo-function`

This function will be used to send `'AUTHINFO'` to the NNTP server. The default function is `nntp-send-authinfo`, which looks through your `'~/authinfo'` (or whatever you've set the `nntp-authinfo-file` variable to) for applicable entries. If none are found, it will prompt you for a login name and a password. The format of the `'~/authinfo'` file is (almost) the same as the `ftp '~/netrc'` file, which is defined in the `ftp` manual page, but here are the salient facts:

1. The file contains one or more line, each of which define one server.
2. Each line may contain an arbitrary number of token/value pairs.

The valid tokens include `'machine'`, `'login'`, `'password'`, `'default'`. In addition Gnus introduces two new tokens, not present in the original

`‘.netrc’/ftp` syntax, namely `‘port’` and `‘force’`. (This is the only way the `‘.authinfo’` file format deviates from the `‘.netrc’` file format.) `‘port’` is used to indicate what port on the server the credentials apply to and `‘force’` is explained below.

Here’s an example file:

```
machine news.uio.no login larsi password geheimnis
machine nntp.ifi.uio.no login larsi force yes
```

The token/value pairs may appear in any order; `‘machine’` doesn’t have to be first, for instance.

In this example, both login name and password have been supplied for the former server, while the latter has only the login name listed, and the user will be prompted for the password. The latter also has the `‘force’` tag, which means that the authinfo will be sent to the `nntp` server upon connection; the default (i.e., when there is not `‘force’` tag) is to not send authinfo to the `nntp` server until the `nntp` server asks for it.

You can also add `‘default’` lines that will apply to all servers that don’t have matching `‘machine’` lines.

```
default force yes
```

This will force sending `‘AUTHINFO’` commands to all servers not previously mentioned.

Remember to not leave the `‘~/.authinfo’` file world-readable.

#### `nntp-server-action-alist`

This is a list of regexps to match on server types and actions to be taken when matches are made. For instance, if you want Gnus to beep every time you connect to innd, you could say something like:

```
(setq nntp-server-action-alist
      '(("innd" (ding))))
```

You probably don’t want to do that, though.

The default value is

```
'(("nntpd 1\\.5\\.11t"
  (remove-hook 'nntp-server-opened-hook
               'nntp-send-mode-reader)))
```

This ensures that Gnus doesn’t send the `MODE READER` command to `nntpd 1.5.11t`, since that command chokes that server, I’ve been told.

#### `nntp-maximum-request`

If the NNTP server doesn’t support NOV headers, this back end will collect headers by sending a series of `head` commands. To speed things up, the back end sends lots of these commands without waiting for reply, and then reads all the replies. This is controlled by the `nntp-maximum-request` variable, and is 400 by default. If your network is buggy, you should set this to 1.

#### `nntp-connection-timeout`

If you have lots of foreign `nntp` groups that you connect to regularly, you’re sure to have problems with NNTP servers not responding properly, or being too

loaded to reply within reasonable time. This can lead to awkward problems, which can be helped somewhat by setting `nntp-connection-timeout`. This is an integer that says how many seconds the `nntp` back end should wait for a connection before giving up. If it is `nil`, which is the default, no timeouts are done.

#### `nntp-server-hook`

This hook is run as the last step when connecting to an NNTP server.

#### `nntp-buggy-select`

Set this to non-`nil` if your select routine is buggy.

#### `nntp-nov-is-evil`

If the NNTP server does not support NOV, you could set this variable to `t`, but `nntp` usually checks automatically whether NOV can be used.

#### `nntp-xover-commands`

List of strings used as commands to fetch NOV lines from a server. The default value of this variable is ("`XOVER`" "`XOVERVIEW`").

#### `nntp-nov-gap`

`nntp` normally sends just one big request for NOV lines to the server. The server responds with one huge list of lines. However, if you have read articles 2-5000 in the group, and only want to read article 1 and 5001, that means that `nntp` will fetch 4999 NOV lines that you will not need. This variable says how big a gap between two consecutive articles is allowed to be before the `XOVER` request is split into several requests. Note that if your network is fast, setting this variable to a really small number means that fetching will probably be slower. If this variable is `nil`, `nntp` will never split requests. The default is 5.

#### `nntp-prepare-server-hook`

A hook run before attempting to connect to an NNTP server.

#### `nntp-warn-about-losing-connection`

If this variable is non-`nil`, some noise will be made when a server closes connection.

#### `nntp-record-commands`

If non-`nil`, `nntp` will log all commands it sends to the NNTP server (along with a timestamp) in the '`*nntp-log*`' buffer. This is useful if you are debugging a Gnus/NNTP connection that doesn't seem to work.

#### `nntp-open-connection-function`

It is possible to customize how the connection to the `nntp` server will be opened. If you specify an `nntp-open-connection-function` parameter, Gnus will use that function to establish the connection. Five pre-made functions are supplied. These functions can be grouped in two categories: direct connection functions (three pre-made), and indirect ones (two pre-made).

#### `nntp-prepare-post-hook`

A hook run just before posting an article. If there is no `Message-ID` header in the article and the news server provides the recommended ID, it will be added to

the article before running this hook. It is useful to make **Cancel-Lock** headers even if you inhibit Gnus to add a **Message-ID** header, you could say:

```
(add-hook 'nntp-prepare-post-hook 'canlock-insert-header)
```

Note that not all servers support the recommended ID. This works for INN versions 2.3.0 and later, for instance.

#### **nntp-read-timeout**

How long nntp should wait between checking for the end of output. Shorter values mean quicker response, but is more CPU intensive. The default is 0.1 seconds. If you have a slow line to the server (and don't like to see Emacs eat your available CPU power), you might set this to, say, 1.

### **6.2.1.1 Direct Functions**

These functions are called direct because they open a direct connection between your machine and the NNTP server. The behavior of these functions is also affected by commonly understood variables (see [Common Variables](#), page [Common Variables](#)).

#### **nntp-open-network-stream**

This is the default, and simply connects to some port or other on the remote system.

#### **nntp-open-tls-stream**

Opens a connection to a server over a *secure* channel. To use this you must have **GNUTLS** installed. You then define a server as follows:

```
;; "nntps" is port 563 and is predefined in our '/etc/services'
;; however, 'gnutls-cli -p' doesn't like named ports.
;;
(nntp "snews.bar.com"
      (nntp-open-connection-function nntp-open-tls-stream)
      (nntp-port-number )
      (nntp-address "snews.bar.com"))
```

#### **nntp-open-ssl-stream**

Opens a connection to a server over a *secure* channel. To use this you must have **OpenSSL** or **SSLeay** installed. You then define a server as follows:

```
;; "snews" is port 563 and is predefined in our '/etc/services'
;; however, 'openssl s_client -port' doesn't like named ports.
;;
(nntp "snews.bar.com"
      (nntp-open-connection-function nntp-open-ssl-stream)
      (nntp-port-number 563)
      (nntp-address "snews.bar.com"))
```

#### **nntp-open-telnet-stream**

Opens a connection to an NNTP server by simply *'telnet'*ing it. You might wonder why this function exists, since we have the default **nntp-open-network-stream** which would do the job. (One of) the reason(s) is that if you are behind a firewall but have direct connections to the outside world thanks to a command wrapper like **runsocks**, you can use it like this:

```
(nntp "socksified"
      (nntp-pre-command "runsocks")
      (nntp-open-connection-function nntp-open-telnet-stream)
      (nntp-address "the.news.server"))
```

With the default method, you would need to wrap your whole Emacs session, which is not a good idea.

### 6.2.1.2 Indirect Functions

These functions are called indirect because they connect to an intermediate host before actually connecting to the NNTP server. All of these functions and related variables are also said to belong to the “via” family of connection: they’re all prefixed with “via” to make things cleaner. The behavior of these functions is also affected by commonly understood variables (see [\(undefined\)](#) [Common Variables], page [\(undefined\)](#)).

#### **nntp-open-via-rlogin-and-telnet**

Does an ‘rlogin’ on a remote system, and then does a ‘telnet’ to the real NNTP server from there. This is useful for instance if you need to connect to a firewall machine first.

**nntp-open-via-rlogin-and-telnet-specific** variables:

##### **nntp-via-rlogin-command**

Command used to log in on the intermediate host. The default is ‘rsh’, but ‘ssh’ is a popular alternative.

##### **nntp-via-rlogin-command-switches**

List of strings to be used as the switches to **nntp-via-rlogin-command**. The default is `nil`. If you use ‘ssh’ for **nntp-via-rlogin-command**, you may set this to ‘(“-C”)’ in order to compress all data connections, otherwise set this to ‘(“-t” “-e” “none”)’ or ‘(“-C” “-t” “-e” “none”)’ if the telnet command requires a pseudo-tty allocation on an intermediate host.

#### **nntp-open-via-telnet-and-telnet**

Does essentially the same, but uses ‘telnet’ instead of ‘rlogin’ to connect to the intermediate host.

**nntp-open-via-telnet-and-telnet-specific** variables:

##### **nntp-via-telnet-command**

Command used to telnet the intermediate host. The default is ‘telnet’.

##### **nntp-via-telnet-switches**

List of strings to be used as the switches to the **nntp-via-telnet-command** command. The default is ‘(“-8”)’.

##### **nntp-via-user-password**

Password to use when logging in on the intermediate host.

##### **nntp-via-envuser**

If non-`nil`, the intermediate `telnet` session (client and server both) will support the `ENVIRON` option and not prompt for login name. This works for Solaris `telnet`, for instance.

**nntp-via-shell-prompt**

Regexp matching the shell prompt on the intermediate host. The default is `'bash\\|\\$ *\\r?$\\|> *\\r?'`.

Here are some additional variables that are understood by all the above functions:

**nntp-via-user-name**

User name to use when connecting to the intermediate host.

**nntp-via-address**

Address of the intermediate host to connect to.

**6.2.1.3 Common Variables**

The following variables affect the behavior of all, or several of the pre-made connection functions. When not specified, all functions are affected.

**nntp-pre-command**

A command wrapper to use when connecting through a non native connection function (all except `nntp-open-network-stream`, `nntp-open-tls-stream`, and `nntp-open-ssl-stream`. This is where you would put a `'SOCKS'` wrapper for instance.

**nntp-address**

The address of the NNTP server.

**nntp-port-number**

Port number to connect to the NNTP server. The default is `'nntp'`. If you use NNTP over `tls/ssl`, you may want to use integer ports rather than named ports (i.e, use `'563'` instead of `'snews'` or `'nntps'`), because external TLS/SSL tools may not work with named ports.

**nntp-end-of-line**

String to use as end-of-line marker when talking to the NNTP server. This is `'\\r\\n'` by default, but should be `'\\n'` when using a non native connection function.

**nntp-telnet-command**

Command to use when connecting to the NNTP server through `'telnet'`. This is *not* for an intermediate host. This is just for the real NNTP server. The default is `'telnet'`.

**nntp-telnet-switches**

A list of switches to pass to `nntp-telnet-command`. The default is `'("-8")'`.

**6.2.2 News Spool**

Subscribing to a foreign group from the local spool is extremely easy, and might be useful, for instance, to speed up reading groups that contain very big articles—`'alt.binaries.pictures.furniture'`, for instance.

Anyway, you just specify `nnsPOOL` as the method and `"` (or anything else) as the address.

If you have access to a local spool, you should probably use that as the native select method (see [\[Finding the News\]](#), page [\[undefined\]](#)). It is normally faster than

using an `nntp` select method, but might not be. It depends. You just have to try to find out what's best at your site.

**nnsPOOL-inews-program**

Program used to post an article.

**nnsPOOL-inews-switches**

Parameters given to the inews program when posting an article.

**nnsPOOL-sPOOL-directory**

Where `nnsPOOL` looks for the articles. This is normally `‘/usr/spool/news/’`.

**nnsPOOL-nov-directory**

Where `nnsPOOL` will look for NOV files. This is normally `‘/usr/spool/news/over.view/’`.

**nnsPOOL-lib-dir**

Where the news lib dir is (`‘/usr/lib/news/’` by default).

**nnsPOOL-active-file**

The name of the active file.

**nnsPOOL-newsgroups-file**

The name of the group descriptions file.

**nnsPOOL-history-file**

The name of the news history file.

**nnsPOOL-active-times-file**

The name of the active date file.

**nnsPOOL-nov-is-evil**

If non-`nil`, `nnsPOOL` won't try to use any NOV files that it finds.

**nnsPOOL-sift-nov-with-sed**

If non-`nil`, which is the default, use `sed` to get the relevant portion from the overview file. If `nil`, `nnsPOOL` will load the entire file into a buffer and process it there.

## 6.3 Getting Mail

Reading mail with a newsreader— isn't that just plain WeIrD? But of course.

### 6.3.1 Mail in a Newsreader

If you are used to traditional mail readers, but have decided to switch to reading mail with Gnus, you may find yourself experiencing something of a culture shock.

Gnus does not behave like traditional mail readers. If you want to make it behave that way, you can, but it's an uphill battle.

Gnus, by default, handles all its groups using the same approach. This approach is very newsreaderly—you enter a group, see the new/unread messages, and when you read the messages, they get marked as read, and you don't see them any more. (Unless you explicitly ask for them.)

In particular, you do not do anything explicitly to delete messages.

Does this mean that all the messages that have been marked as read are deleted? How awful!

But, no, it means that old messages are *expired* according to some scheme or other. For news messages, the expire process is controlled by the news administrator; for mail, the expire process is controlled by you. The expire process for mail is covered in depth in [\[Expiring Mail\]](#), page [\[undefined\]](#).

What many Gnus users find, after using it a while for both news and mail, is that the transport mechanism has very little to do with how they want to treat a message.

Many people subscribe to several mailing lists. These are transported via SMTP, and are therefore mail. But we might go for weeks without answering, or even reading these messages very carefully. We may not need to save them because if we should need to read one again, they are archived somewhere else.

Some people have local news groups which have only a handful of readers. These are transported via NNTP, and are therefore news. But we may need to read and answer a large fraction of the messages very carefully in order to do our work. And there may not be an archive, so we may need to save the interesting messages the same way we would personal mail.

The important distinction turns out to be not the transport mechanism, but other factors such as how interested we are in the subject matter, or how easy it is to retrieve the message if we need to read it again.

Gnus provides many options for sorting mail into “groups” which behave like newsgroups, and for treating each group (whether mail or news) differently.

Some users never get comfortable using the Gnus (ahem) paradigm and wish that Gnus should grow up and be a male, er, mail reader. It is possible to whip Gnus into a more mailreaderly being, but, as said before, it’s not easy. People who prefer proper mail readers should try VM instead, which is an excellent, and proper, mail reader.

I don’t mean to scare anybody off, but I want to make it clear that you may be required to learn a new way of thinking about messages. After you’ve been subjected to The Gnus Way, you will come to love it. I can guarantee it. (At least the guy who sold me the Emacs Subliminal Brain-Washing Functions that I’ve put into Gnus did guarantee it. You Will Be Assimilated. You Love Gnus. You Love The Gnus Mail Way. You Do.)

### 6.3.2 Getting Started Reading Mail

It’s quite easy to use Gnus to read your new mail. You just plonk the mail back end of your choice into `gnus-secondary-select-methods`, and things will happen automatically.

For instance, if you want to use `nnml` (which is a “one file per mail” back end), you could put the following in your ‘`~/gnus.el`’ file:

```
(setq gnus-secondary-select-methods '((nnml "")))
```

Now, the next time you start Gnus, this back end will be queried for new articles, and it will move all the messages in your spool file to its directory, which is ‘`~/Mail/`’ by default. The new group that will be created (`mail.misc`) will be subscribed, and you can read it like any other group.

You will probably want to split the mail into several groups, though:



```
(setq nnmail-split-methods
  '(("junk" "^From:.*Lars Ingebrigtsen")
    ("crazy" "^Subject:.*die\\|^Organization:.*flabby")
    ("other" "")))
```

This will result in three new `nnml` mail groups being created: `'nnml:junk'`, `'nnml:crazy'`, and `'nnml:other'`. All the mail that doesn't fit into the first two groups will be placed in the last group.

This should be sufficient for reading mail with Gnus. You might want to give the other sections in this part of the manual a perusal, though. Especially see [\[Choosing a Mail Back End\]](#), page [\[undefined\]](#) and see [\[Expiring Mail\]](#), page [\[undefined\]](#).

### 6.3.3 Splitting Mail

The `nnmail-split-methods` variable says how the incoming mail is to be split into groups.

```
(setq nnmail-split-methods
  '(("mail.junk" "^From:.*Lars Ingebrigtsen")
    ("mail.crazy" "^Subject:.*die\\|^Organization:.*flabby")
    ("mail.other" "")))
```

This variable is a list of lists, where the first element of each of these lists is the name of the mail group (they do not have to be called something beginning with `'mail'`, by the way), and the second element is a regular expression used on the header of each mail to determine if it belongs in this mail group. The first string may contain `'\\1'` forms, like the ones used by `replace-match` to insert sub-expressions from the matched text. For instance:

```
("list.\\1" "From:.* \\(.*\\)-list@majordomo.com")
```

In that case, `nnmail-split-lowercase-expanded` controls whether the inserted text should be made lowercase. See [\[Fancy Mail Splitting\]](#), page [\[undefined\]](#).

The second element can also be a function. In that case, it will be called narrowed to the headers with the first element of the rule as the argument. It should return a non-`nil` value if it thinks that the mail belongs in that group.

The last of these groups should always be a general one, and the regular expression should *always* be `""` so that it matches any mails that haven't been matched by any of the other regexps. (These rules are processed from the beginning of the alist toward the end. The first rule to make a match will “win”, unless you have crossposting enabled. In that case, all matching rules will “win”.) If no rule matched, the mail will end up in the `'bogus'` group. When new groups are created by splitting mail, you may want to run `gnus-group-find-new-groups` to see the new groups. This also applies to the `'bogus'` group.

If you like to tinker with this yourself, you can set this variable to a function of your choice. This function will be called without any arguments in a buffer narrowed to the headers of an incoming mail message. The function should return a list of group names that it thinks should carry this mail message.

Note that the mail back ends are free to maul the poor, innocent, incoming headers all they want to. They all add `Lines` headers; some add `X-Gnus-Group` headers; most rename the Unix mbox `From<SPACE>` line to something else.

The mail back ends all support cross-posting. If several regexps match, the mail will be “cross-posted” to all those groups. `nnmail-crosspost` says whether to use this mechanism or not. Note that no articles are crossposted to the general (“”) group.

`nnmh` and `nnml` makes crossposts by creating hard links to the crossposted articles. However, not all file systems support hard links. If that’s the case for you, set `nnmail-crosspost-link-function` to `copy-file`. (This variable is `add-name-to-file` by default.)

If you wish to see where the previous mail split put the messages, you can use the `M-x nnmail-split-history` command. If you wish to see where re-spooling messages would put the messages, you can use `gnus-summary-respool-trace` and related commands (see [\[Mail Group Commands\]](#), page [\(undefined\)](#)).

Header lines longer than the value of `nnmail-split-header-length-limit` are excluded from the split function.

By default the splitting codes MIME decodes headers so you can match on non-ASCII strings. The `nnmail-mail-splitting-charset` variable specifies the default charset for decoding. The behaviour can be turned off completely by binding `nnmail-mail-splitting-decodes` to `nil`, which is useful if you want to match articles based on the raw header data.

By default, splitting is performed on all incoming messages. If you specify a `directory` entry for the variable `mail-sources` (see [\(undefined\) \[Mail Source Specifiers\]](#), page [\(undefined\)](#)), however, then splitting does *not* happen by default. You can set the variable `nnmail-resplit-incoming` to a non-`nil` value to make splitting happen even in this case. (This variable has no effect on other kinds of entries.)

Gnus gives you all the opportunity you could possibly want for shooting yourself in the foot. Let’s say you create a group that will contain all the mail you get from your boss. And then you accidentally unsubscribe from the group. Gnus will still put all the mail from your boss in the unsubscribed group, and so, when your boss mails you “Have that report ready by Monday or you’re fired!”, you’ll never see it and, come Tuesday, you’ll still believe that you’re gainfully employed while you really should be out collecting empty bottles to save up for next month’s rent money.

### 6.3.4 Mail Sources

Mail can be gotten from many different sources—the mail spool, from a POP mail server, from a procmail directory, or from a maildir, for instance.

#### 6.3.4.1 Mail Source Specifiers

You tell Gnus how to fetch mail by setting `mail-sources` (see [\(undefined\) \[Fetching Mail\]](#), page [\(undefined\)](#)) to a *mail source specifier*.

Here’s an example:

```
(pop :server "pop3.mailserver.com" :user "myname")
```

As can be observed, a mail source specifier is a list where the first element is a *mail source type*, followed by an arbitrary number of *keywords*. Keywords that are not explicitly specified are given default values.

The following mail source types are available:

**file** Get mail from a single file; typically from the mail spool.

Keywords:

**:path** The file name. Defaults to the value of the **MAIL** environment variable or the value of **rmail-spool-directory** (usually something like `‘/usr/mail/spool/user-name’`).

**:prescript**

**:postscript**

Script run before/after fetching mail.

An example file mail source:

```
(file :path "/usr/spool/mail/user-name")
```

Or using the default file name:

```
(file)
```

If the mail spool file is not located on the local machine, it's best to use POP or IMAP or the like to fetch the mail. You can not use `ange-ftp` file names here—it has no way to lock the mail spool while moving the mail.

If it's impossible to set up a proper server, you can use `ssh` instead.

```
(setq mail-sources
  '((file :prescript "ssh host bin/getmail >%t")))
```

The `‘getmail’` script would look something like the following:

```
#!/bin/sh
# getmail - move mail from spool to stdout
# flu@iki.fi

MOVEMAIL=/usr/lib/emacs/20.3/i386-redhat-linux/movemail
TMP=$HOME/Mail/tmp
rm -f $TMP; $MOVEMAIL $MAIL $TMP >/dev/null && cat $TMP
```

Alter this script to fit find the `‘movemail’` you want to use.

**directory**

Get mail from several files in a directory. This is typically used when you have `procmail` split the incoming mail into several files. That is, there is a one-to-one correspondence between files in that directory and groups, so that mail from the file `‘foo.bar.spool’` will be put in the group `foo.bar`. (You can change the suffix to be used instead of `.spool`.) Setting **nnmail-scan-directory-mail-source-once** to `non-nil` forces Gnus to scan the mail source only once. This is particularly useful if you want to scan mail groups at a specified level.

There is also the variable **nnmail-resplit-incoming**, if you set that to a `non-nil` value, then the normal splitting process is applied to all the files from the directory, [\[Splitting Mail\]](#), page [\[undefined\]](#).

Keywords:

**:path** The name of the directory where the files are. There is no default value.

**:suffix** Only files ending with this suffix are used. The default is `‘.spool’`.

**:predicate**

Only files that have this predicate return non-**nil** are returned. The default is **identity**. This is used as an additional filter—only files that have the right suffix *and* satisfy this predicate are considered.

**:prescript****:postscript**

Script run before/after fetching mail.

An example directory mail source:

```
(directory :path "/home/user-name/procmail-dir/"
           :suffix ".prcml")
```

**pop**

Get mail from a POP server.

Keywords:

**:server** The name of the POP server. The default is taken from the **MAILHOST** environment variable.

**:port** The port number of the POP server. This can be a number (eg, **:port 1234**) or a string (eg, **:port "pop3"**). If it is a string, it should be a service name as listed in **/etc/services** on Unix systems. The default is **"pop3"**. On some systems you might need to specify it as **"pop-3"** instead.

**:user** The user name to give to the POP server. The default is the login name.

**:password**

The password to give to the POP server. If not specified, the user is prompted.

**:program** The program to use to fetch mail from the POP server. This should be a **format**-like string. Here's an example:

```
fetchmail %u@%s -P %p %t
```

The valid format specifier characters are:

**'t'** The name of the file the mail is to be moved to. This must always be included in this string.

**'s'** The name of the server.

**'P'** The port number of the server.

**'u'** The user name to use.

**'p'** The password to use.

The values used for these specs are taken from the values you give the corresponding keywords.

**:prescript**

A script to be run before fetching the mail. The syntax is the same as the **:program** keyword. This can also be a function to be run.

**:postscript**

A script to be run after fetching the mail. The syntax is the same as the **:program** keyword. This can also be a function to be run.

**:function**

The function to use to fetch mail from the POP server. The function is called with one parameter—the name of the file where the mail should be moved to.

**:authentication**

This can be either the symbol **password** or the symbol **apop** and says what authentication scheme to use. The default is **password**.

If the **:program** and **:function** keywords aren't specified, **pop3-movemail** will be used. If the **pop3-leave-mail-on-server** is non-nil the mail is to be left on the POP server after fetching when using **pop3-movemail**. Note that POP servers maintain no state information between sessions, so what the client believes is there and what is actually there may not match up. If they do not, then the whole thing can fall apart and leave you with a corrupt mailbox.

Here are some examples. Fetch from the default POP server, using the default user name, and default fetcher:

```
(pop)
```

Fetch from a named server with a named user and password:

```
(pop :server "my.pop.server"
     :user "user-name" :password "secret")
```

Use 'movemail' to move the mail:

```
(pop :program "movemail po:%u %t %p")
```

**maildir** Get mail from a maildir. This is a type of mailbox that is supported by at least gmail and postfix, where each file in a special directory contains exactly one mail.

Keywords:

**:path** The name of the directory where the mails are stored. The default is taken from the **MAILDIR** environment variable or **~/Maildir/**.

**:subdirs** The subdirectories of the Maildir. The default is **("new" "cur")**. You can also get mails from remote hosts (because maildirs don't suffer from locking problems).

Two example maildir mail sources:

```
(maildir :path "/home/user-name/Maildir/"
         :subdirs ("cur" "new"))

(maildir :path "/user@remotehost.org:~/Maildir/"
         :subdirs ("new"))
```

**imap** Get mail from a IMAP server. If you don't want to use IMAP as intended, as a network mail reading protocol (ie with **nimap**), for some reason or other, Gnus let you treat it similar to a POP server and fetches articles from a given IMAP mailbox. See [\[IMAP\]](#), page [\[IMAP\]](#), for more information.

Note that for the Kerberos, GSSAPI, TLS/SSL and STARTTLS support you may need external programs and libraries, See [\[IMAP\]](#), page [\[undefined\]](#).

Keywords:

- :server**     The name of the IMAP server. The default is taken from the MAILHOST environment variable.
- :port**       The port number of the IMAP server. The default is '143', or '993' for TLS/SSL connections.
- :user**        The user name to give to the IMAP server. The default is the login name.
- :password**    The password to give to the IMAP server. If not specified, the user is prompted.
- :stream**      What stream to use for connecting to the server, this is one of the symbols in `imap-stream-alist`. Right now, this means 'gssapi', 'kerberos4', 'starttls', 'tls', 'ssl', 'shell' or the default 'network'.
- :authentication**  
Which authenticator to use for authenticating to the server, this is one of the symbols in `imap-authenticator-alist`. Right now, this means 'gssapi', 'kerberos4', 'digest-md5', 'cram-md5', 'anonymous' or the default 'login'.
- :program**    When using the 'shell' :stream, the contents of this variable is mapped into the `imap-shell-program` variable. This should be a format-like string (or list of strings). Here's an example:  

`ssh %s imapd`

The valid format specifier characters are:
  - 's'            The name of the server.
  - 'l'            User name from `imap-default-user`.
  - 'p'            The port number of the server.
The values used for these specs are taken from the values you give the corresponding keywords.
- :mailbox**    The name of the mailbox to get mail from. The default is 'INBOX' which normally is the mailbox which receive incoming mail.
- :predicate**  
The predicate used to find articles to fetch. The default, 'UNSEEN UNDELETED', is probably the best choice for most people, but if you sometimes peek in your mailbox with a IMAP client and mark some articles as read (or; SEEN) you might want to set this to '1:\*'. Then all articles in the mailbox is fetched, no matter what. For a complete list of predicates, see RFC 2060 section 6.4.4.

**:fetchflag**

How to flag fetched articles on the server, the default ‘\Deleted’ will mark them as deleted, an alternative would be ‘\Seen’ which would simply mark them as read. These are the two most likely choices, but more flags are defined in RFC 2060 section 2.3.2.

**:dontexpunge**

If non-`nil`, don’t remove all articles marked as deleted in the mailbox after finishing the fetch.

An example IMAP mail source:

```
(imap :server "mail.mycorp.com"
      :stream kerberos4
      :fetchflag "\\Seen")
```

**webmail** Get mail from a webmail server, such as <http://www.hotmail.com/>, <http://webmail.netscape.com/>, <http://www.netaddress.com/>, <http://mail.yahoo.com/>.

NOTE: Webmail largely depends on cookies. A "one-line-cookie" patch is required for url "4.0pre.46".

WARNING: Mails may be lost. NO WARRANTY.

Keywords:

**:subtype** The type of the webmail server. The default is `hotmail`. The alternatives are `netscape`, `netaddress`, `my-deja`.

**:user** The user name to give to the webmail server. The default is the login name.

**:password**

The password to give to the webmail server. If not specified, the user is prompted.

**:dontexpunge**

If non-`nil`, only fetch unread articles and don’t move them to trash folder after finishing the fetch.

An example webmail source:

```
(webmail :subtype 'hotmail
          :user "user-name"
          :password "secret")
```

*Common Keywords*

Common keywords can be used in any type of mail source.

Keywords:

**:plugged** If non-`nil`, fetch the mail even when Gnus is unplugged. If you use directory source to get mail, you can specify it as in this example:

```
(setq mail-sources
      '((directory :path "/home/pavel/.Spool/"
                  :suffix ""
```

```
:plugged t)))
```

Gnus will then fetch your mail even when you are unplugged. This is useful when you use local mail and news.

### 6.3.4.2 Function Interface

Some of the above keywords specify a Lisp function to be executed. For each keyword `:foo`, the Lisp variable `foo` is bound to the value of the keyword while the function is executing. For example, consider the following mail-source setting:

```
(setq mail-sources '((pop :user "jrl"
                          :server "pophost" :function fetchfunc)))
```

While the function `fetchfunc` is executing, the symbol `user` is bound to `"jrl"`, and the symbol `server` is bound to `"pophost"`. The symbols `port`, `password`, `program`, `prescript`, `postscript`, `function`, and `authentication` are also bound (to their default values).

See above for a list of keywords for each type of mail source.

### 6.3.4.3 Mail Source Customization

The following is a list of variables that influence how the mail is fetched. You would normally not need to set or change any of these variables.

#### `mail-source-crash-box`

File where mail will be stored while processing it. The default is `'~/ .emacs-mail-crash-box'`.

#### `mail-source-delete-incoming`

If non-`nil`, delete incoming files after handling them. If `t`, delete the files immediately, if `nil`, never delete any files. If a positive number, delete files older than number of days (This will only happen, when receiving new mail). You may also set `mail-source-delete-incoming` to `nil` and call `mail-source-delete-old-incoming` from a hook or interactively.

#### `mail-source-delete-old-incoming-confirm`

If non-`nil`, ask for confirmation before deleting old incoming files. This variable only applies when `mail-source-delete-incoming` is a positive number.

#### `mail-source-ignore-errors`

If non-`nil`, ignore errors when reading mail from a mail source.

#### `mail-source-directory`

Directory where incoming mail source files (if any) will be stored. The default is `'~/Mail/'`. At present, the only thing this is used for is to say where the incoming files will be stored if the variable `mail-source-delete-incoming` is `nil` or a number.

#### `mail-source-incoming-file-prefix`

Prefix for file name for storing incoming mail. The default is `'Incoming'`, in which case files will end up with names like `'Incoming30630D_'` or `'Incoming298602ZD'`. This is really only relevant if `mail-source-delete-incoming` is `nil`.



**mail-source-default-file-modes**

All new mail files will get this file mode. The default is 384.

**mail-source-movemail-program**

If non-`nil`, name of program for fetching new mail. If `nil`, `movemail` in *exec-directory*.

**6.3.4.4 Fetching Mail**

The way to actually tell Gnus where to get new mail from is to set `mail-sources` to a list of mail source specifiers (see [\[Mail Source Specifiers\]](#), page [\[undefined\]](#)).

If this variable (and the obsolescent `nnmail-spool-file`) is `nil`, the mail back ends will never attempt to fetch mail by themselves.

If you want to fetch mail both from your local spool as well as a POP mail server, you'd say something like:

```
(setq mail-sources
      '((file)
        (pop :server "pop3.mail.server"
              :password "secret")))
```

Or, if you don't want to use any of the keyword defaults:

```
(setq mail-sources
      '((file :path "/var/spool/mail/user-name")
        (pop :server "pop3.mail.server"
              :user "user-name"
              :port "pop3"
              :password "secret")))
```

When you use a mail back end, Gnus will slurp all your mail from your inbox and plonk it down in your home directory. Gnus doesn't move any mail if you're not using a mail back end—you have to do a lot of magic invocations first. At the time when you have finished drawing the pentagram, lightened the candles, and sacrificed the goat, you really shouldn't be too surprised when Gnus moves your mail.

**6.3.5 Mail Back End Variables**

These variables are (for the most part) pertinent to all the various mail back ends.

**nnmail-read-incoming-hook**

The mail back ends all call this hook after reading new mail. You can use this hook to notify any mail watch programs, if you want to.

**nnmail-split-hook**

Hook run in the buffer where the mail headers of each message is kept just before the splitting based on these headers is done. The hook is free to modify the buffer contents in any way it sees fit—the buffer is discarded after the splitting has been done, and no changes performed in the buffer will show up in any files. `gnus-article-decode-encoded-words` is one likely function to add to this hook.

**nnmail-pre-get-new-mail-hook**

**nnmail-post-get-new-mail-hook**

These are two useful hooks executed when treating new incoming mail—**nnmail-pre-get-new-mail-hook** (is called just before starting to handle the new mail) and **nnmail-post-get-new-mail-hook** (is called when the mail handling is done). Here's an example of using these two hooks to change the default file modes the new mail files get:

```
(add-hook 'nnmail-pre-get-new-mail-hook
          (lambda () (set-default-file-modes 511)))
```

```
(add-hook 'nnmail-post-get-new-mail-hook
          (lambda () (set-default-file-modes 551)))
```

**nnmail-use-long-file-names**

If non-**nil**, the mail back ends will use long file and directory names. Groups like 'mail.misc' will end up in directories (assuming use of **nnml** back end) or files (assuming use of **nnfolder** back end) like 'mail.misc'. If it is **nil**, the same group will end up in 'mail/misc'.

**nnmail-delete-file-function**

Function called to delete files. It is **delete-file** by default.

**nnmail-cache-accepted-message-ids**

If non-**nil**, put the **Message-IDs** of articles imported into the back end (via **Gcc**, for instance) into the mail duplication discovery cache. The default is **nil**.

**nnmail-cache-ignore-groups**

This can be a regular expression or a list of regular expressions. Group names that match any of the regular expressions will never be recorded in the **Message-ID** cache.

This can be useful, for example, when using Fancy Splitting (see [\[Fancy Mail Splitting\]](#), page [\[undefined\]](#)) together with the function **nnmail-split-fancy-with-parent**.

### 6.3.6 Fancy Mail Splitting

If the rather simple, standard method for specifying how to split mail doesn't allow you to do what you want, you can set **nnmail-split-methods** to **nnmail-split-fancy**. Then you can play with the **nnmail-split-fancy** variable.

Let's look at an example value of this variable first:

```
;; Messages from the mailer daemon are not crossposted to any of
;; the ordinary groups. Warnings are put in a separate group
;; from real errors.
(| ("from" mail (| ("subject" "warn.*" "mail.warning")
                  "mail.misc"))
  ;; Non-error messages are crossposted to all relevant
  ;; groups, but we don't crosspost between the group for the
  ;; (ding) list and the group for other (ding) related mail.
  (& (| (any "ding@ifi\\.uio\\.no" "ding.list"))
```

```

      ("subject" "ding" "ding.misc"))
;; Other mailing lists...
(any "procmail@informatik\\.rwth-aachen\\.de" "procmail.list")
(any "SmartList@informatik\\.rwth-aachen\\.de" "SmartList.list")
;; Both lists below have the same suffix, so prevent
;; cross-posting to mkpkg.list of messages posted only to
;; the bugs- list, but allow cross-posting when the
;; message was really cross-posted.
(any "bugs-mypackage@somewhere" "mypkg.bugs")
(any "mypackage@somewhere" - "bugs-mypackage" "mypkg.list")
;; People...
(any "larsi@ifi\\.uio\\.no" "people.Lars_Magne_Ingebrigtsen"))
;; Unmatched mail goes to the catch all group.
"misc.misc")

```

This variable has the format of a *split*. A *split* is a (possibly) recursive structure where each *split* may contain other *splits*. Here are the possible *split* syntaxes:

**group**      If the *split* is a string, that will be taken as a group name. Normal regexp match expansion will be done. See below for examples.

**(field value [- restrict [...] ] split)**

If the *split* is a list, the first element of which is a string, then store the message as specified by *split*, if header *field* (a regexp) contains *value* (also a regexp). If *restrict* (yet another regexp) matches some string after *field* and before the end of the matched *value*, the *split* is ignored. If none of the *restrict* clauses match, *split* is processed.

**(| split ...)**

If the *split* is a list, and the first element is | (vertical bar), then process each *split* until one of them matches. A *split* is said to match if it will cause the mail message to be stored in one or more groups.

**(& split ...)**

If the *split* is a list, and the first element is &, then process all *splits* in the list.

**junk**      If the *split* is the symbol **junk**, then don't save (i.e., delete) this message. Use with extreme caution.

**(: function arg1 arg2 ...)**

If the *split* is a list, and the first element is ':', then the second element will be called as a function with *args* given as arguments. The function should return a *split*.

For instance, the following function could be used to split based on the body of the messages:

```

(defun split-on-body ()
  (save-excursion
    (save-restriction
      (widen)
      (goto-char (point-min))
      (when (re-search-forward "Some.*string" nil t)

```

```
"string.group"))))
```

The buffer is narrowed to the message in question when *function* is run. That's why (*widen*) needs to be called after *save-excursion* and *save-restriction* in the example above. Also note that with the *nnimap* backend, message bodies will not be downloaded by default. You need to set *nnimap-split-download-body* to *t* to do that (see [\(undefined\)](#) [Splitting in IMAP], page [\(undefined\)](#)).

```
(! func split)
```

If the *split* is a list, and the first element is *!*, then *split* will be processed, and *func* will be called as a function with the result of *split* as argument. *func* should return a split.

*nil* If the *split* is *nil*, it is ignored.

In these splits, *field* must match a complete field name. *value* must match a complete word according to the fundamental mode syntax table. You can use *.\** in the regexps to match partial field names or words. In other words, all *value*'s are wrapped in '*\<*' and '*\>*' pairs.

*field* and *value* can also be Lisp symbols, in that case they are expanded as specified by the variable *nnmail-split-abbrev-alist*. This is an alist of cons cells, where the CAR of a cell contains the key, and the CDR contains the associated value. Predefined entries in *nnmail-split-abbrev-alist* include:

*from* Matches the 'From', 'Sender' and 'Resent-From' fields.

*to* Matches the 'To', 'Cc', 'Apparently-To', 'Resent-To' and 'Resent-Cc' fields.

*any* Is the union of the *from* and *to* entries.

*nnmail-split-fancy-syntax-table* is the syntax table in effect when all this splitting is performed.

If you want to have Gnus create groups dynamically based on some information in the headers (i.e., do *replace-match*-like substitutions in the group names), you can say things like:

```
(any "debian-\\b\\(\\w\\|@lists.debian.org" "mail.debian.\\1")
```

In this example, messages sent to 'debian-foo@lists.debian.org' will be filed in 'mail.debian.foo'.

If the string contains the element '*\&*', then the previously matched string will be substituted. Similarly, the elements '*\\1*' up to '*\\9*' will be substituted with the text matched by the groupings 1 through 9.

Where *nnmail-split-lowercase-expanded* controls whether the lowercase of the matched string should be used for the substitution. Setting it as non-*nil* is useful to avoid the creation of multiple groups when users send to an address using different case (i.e. mailing-list@domain vs Mailing-List@Domain). The default value is *t*.

*nnmail-split-fancy-match-partial-words* controls whether partial words are matched during fancy splitting.

Normally, regular expressions given in *nnmail-split-fancy* are implicitly surrounded by '*\<... \>*' markers, which are word delimiters. If this variable is true, they are not implicitly surrounded by anything.

```
(any "joe" "joemail")
```

In this example, messages sent from ‘joedavis@foo.org’ will normally not be filed in ‘joemail’. With `nnmail-split-fancy-match-partial-words` set to `t`, however, the match will happen. In effect, the requirement of a word boundary is removed and instead the match becomes more like a `grep`.

`nnmail-split-fancy-with-parent` is a function which allows you to split followups into the same groups their parents are in. Sometimes you can’t make splitting rules for all your mail. For example, your boss might send you personal mail regarding different projects you are working on, and as you can’t tell your boss to put a distinguishing string into the subject line, you have to resort to manually moving the messages into the right group. With this function, you only have to do it once per thread.

To use this feature, you have to set `nnmail-treat-duplicates` and `nnmail-cache-accepted-message-ids` to a non-`nil` value. And then you can include `nnmail-split-fancy-with-parent` using the colon feature, like so:

```
(setq nnmail-treat-duplicates 'warn      ; or delete
      nnmail-cache-accepted-message-ids t
      nnmail-split-fancy
      '(| (: nnmail-split-fancy-with-parent)
          ;; other splits go here
      ))
```

This feature works as follows: when `nnmail-treat-duplicates` is non-`nil`, Gnus records the message id of every message it sees in the file specified by the variable `nnmail-message-id-cache-file`, together with the group it is in (the group is omitted for non-mail messages). When mail splitting is invoked, the function `nnmail-split-fancy-with-parent` then looks at the References (and In-Reply-To) header of each message to split and searches the file specified by `nnmail-message-id-cache-file` for the message ids. When it has found a parent, it returns the corresponding group name unless the group name matches the regexp `nnmail-split-fancy-with-parent-ignore-groups`. It is recommended that you set `nnmail-message-id-cache-length` to a somewhat higher number than the default so that the message ids are still in the cache. (A value of 5000 appears to create a file some 300 kBytes in size.) When `nnmail-cache-accepted-message-ids` is non-`nil`, Gnus also records the message ids of moved articles, so that the followup messages goes into the new group.

Also see the variable `nnmail-cache-ignore-groups` if you don’t want certain groups to be recorded in the cache. For example, if all outgoing messages are written to an “outgoing” group, you could set `nnmail-cache-ignore-groups` to match that group name. Otherwise, answers to all your messages would end up in the “outgoing” group.

### 6.3.7 Group Mail Splitting

If you subscribe to dozens of mailing lists but you don’t want to maintain mail splitting rules manually, group mail splitting is for you. You just have to set `to-list` and/or `to-address` in group parameters or group customization and set `nnmail-split-methods` to `gnus-group-split`. This splitting function will scan all groups for those parameters and split mail accordingly, i.e., messages posted from or to the addresses specified in the parameters `to-list` or `to-address` of a mail group will be stored in that group.

Sometimes, mailing lists have multiple addresses, and you may want mail splitting to recognize them all: just set the `extra-aliases` group parameter to the list of additional addresses and it's done. If you'd rather use a regular expression, set `split-regexp`.

All these parameters in a group will be used to create an `nnmail-split-fancy` split, in which the *field* is 'any', the *value* is a single regular expression that matches `to-list`, `to-address`, all of `extra-aliases` and all matches of `split-regexp`, and the *split* is the name of the group. *restricts* are also supported: just set the `split-exclude` parameter to a list of regular expressions.

If you can't get the right split to be generated using all these parameters, or you just need something fancier, you can set the parameter `split-spec` to an `nnmail-split-fancy` split. In this case, all other aforementioned parameters will be ignored by `gnus-group-split`. In particular, `split-spec` may be set to `nil`, in which case the group will be ignored by `gnus-group-split`.

`gnus-group-split` will do cross-posting on all groups that match, by defining a single & fancy split containing one split for each group. If a message doesn't match any split, it will be stored in the group named in `gnus-group-split-default-catch-all-group`, unless some group has `split-spec` set to `catch-all`, in which case that group is used as the catch-all group. Even though this variable is often used just to name a group, it may also be set to an arbitrarily complex fancy split (after all, a group name is a fancy split), and this may be useful to split mail that doesn't go to any mailing list to personal mail folders. Note that this fancy split is added as the last element of a | split list that also contains a & split with the rules extracted from group parameters.

It's time for an example. Assume the following group parameters have been defined:

```
nnml:mail.bar:
((to-address . "bar@femail.com")
 (split-regexp . ".*@femail\\.com"))
nnml:mail.foo:
((to-list . "foo@nowhere.gov")
 (extra-aliases "foo@localhost" "foo-redirect@home")
 (split-exclude "bugs-foo" "rambling-foo")
 (admin-address . "foo-request@nowhere.gov"))
nnml:mail.others:
((split-spec . catch-all))
```

Setting `nnmail-split-methods` to `gnus-group-split` will behave as if `nnmail-split-fancy` had been selected and variable `nnmail-split-fancy` had been set as follows:

```
(| (& (any "\\(bar@femail\\.com\\|.*@femail\\.com\\)" "mail.bar")
      (any "\\(foo@nowhere\\.gov\\|foo@localhost\\|foo-redirect@home\\)"
            - "bugs-foo" - "rambling-foo" "mail.foo"))
  "mail.others")
```

If you'd rather not use group splitting for all your mail groups, you may use it for only some of them, by using `nnmail-split-fancy` splits like this:

```
(: gnus-group-split-fancy groups no-crosspost catch-all)
```

*groups* may be a regular expression or a list of group names whose parameters will be scanned to generate the output split. *no-crosspost* can be used to disable cross-posting; in

this case, a single `|` split will be output. *catch-all* is the fall back fancy split, used like `gnus-group-split-default-catch-all-group`. If *catch-all* is `nil`, or if `split-regexp` matches the empty string in any selected group, no catch-all split will be issued. Otherwise, if some group has `split-spec` set to *catch-all*, this group will override the value of the *catch-all* argument.

Unfortunately, scanning all groups and their parameters can be quite slow, especially considering that it has to be done for every message. But don't despair! The function `gnus-group-split-setup` can be used to enable `gnus-group-split` in a much more efficient way. It sets `nnmail-split-methods` to `nnmail-split-fancy` and sets `nnmail-split-fancy` to the split produced by `gnus-group-split-fancy`. Thus, the group parameters are only scanned once, no matter how many messages are split.

However, if you change group parameters, you'd have to update `nnmail-split-fancy` manually. You can do it by running `gnus-group-split-update`. If you'd rather have it updated automatically, just tell `gnus-group-split-setup` to do it for you. For example, add to your `~/.gnus.el`:

```
(gnus-group-split-setup auto-update catch-all)
```

If *auto-update* is non-`nil`, `gnus-group-split-update` will be added to `nnmail-pre-get-new-mail-hook`, so you won't ever have to worry about updating `nnmail-split-fancy` again. If you don't omit *catch-all* (it's optional, equivalent to `nil`), `gnus-group-split-default-catch-all-group` will be set to its value.

Because you may want to change `nnmail-split-fancy` after it is set by `gnus-group-split-update`, this function will run `gnus-group-split-updated-hook` just before finishing.

### 6.3.8 Incorporating Old Mail

Most people have lots of old mail stored in various file formats. If you have set up Gnus to read mail using one of the spiffy Gnus mail back ends, you'll probably wish to have that old mail incorporated into your mail groups.

Doing so can be quite easy.

To take an example: You're reading mail using `nnml` (see [\[Mail Spool\]](#), page [\[undefined\]](#)), and have set `nnmail-split-methods` to a satisfactory value (see [\[Splitting Mail\]](#), page [\[undefined\]](#)). You have an old Unix mbox file filled with important, but old, mail. You want to move it into your `nnml` groups.

Here's how:

1. Go to the group buffer.
2. Type `G f` and give the file name to the mbox file when prompted to create an `nn doc` group from the mbox file (see [\[Foreign Groups\]](#), page [\[undefined\]](#)).
3. Type `SPACE` to enter the newly created group.
4. Type `M P b` to process-mark all articles in this group's buffer (see [\[Setting Process Marks\]](#), page [\[undefined\]](#)).
5. Type `B r` to respool all the process-marked articles, and answer '`nnml`' when prompted (see [\[Mail Group Commands\]](#), page [\[undefined\]](#)).

All the mail messages in the mbox file will now also be spread out over all your `nnml` groups. Try entering them and check whether things have gone without a glitch. If things



look ok, you may consider deleting the mbox file, but I wouldn't do that unless I was absolutely sure that all the mail has ended up where it should be.

Respooling is also a handy thing to do if you're switching from one mail back end to another. Just respool all the mail in the old mail groups using the new mail back end.

### 6.3.9 Expiring Mail

Traditional mail readers have a tendency to remove mail articles when you mark them as read, in some way. Gnus takes a fundamentally different approach to mail reading.

Gnus basically considers mail just to be news that has been received in a rather peculiar manner. It does not think that it has the power to actually change the mail, or delete any mail messages. If you enter a mail group, and mark articles as “read”, or kill them in some other fashion, the mail articles will still exist on the system. I repeat: Gnus will not delete your old, read mail. Unless you ask it to, of course.

To make Gnus get rid of your unwanted mail, you have to mark the articles as *expirable*. (With the default key bindings, this means that you have to type *E*.) This does not mean that the articles will disappear right away, however. In general, a mail article will be deleted from your system if, 1) it is marked as expirable, AND 2) it is more than one week old. If you do not mark an article as expirable, it will remain on your system until hell freezes over. This bears repeating one more time, with some spurious capitalizations: IF you do NOT mark articles as EXPIRABLE, Gnus will NEVER delete those ARTICLES.

You do not have to mark articles as expirable by hand. Gnus provides two features, called “auto-expire” and “total-expire”, that can help you with this. In a nutshell, “auto-expire” means that Gnus hits *E* for you when you select an article. And “total-expire” means that Gnus considers all articles as expirable that are read. So, in addition to the articles marked ‘E’, also the articles marked ‘r’, ‘R’, ‘O’, ‘K’, ‘Y’ and so on are considered expirable.

When should either auto-expire or total-expire be used? Most people who are subscribed to mailing lists split each list into its own group and then turn on auto-expire or total-expire for those groups. (See [\[Splitting Mail\]](#), page [\[undefined\]](#), for more information on splitting each list into its own group.)

Which one is better, auto-expire or total-expire? It's not easy to answer. Generally speaking, auto-expire is probably faster. Another advantage of auto-expire is that you get more marks to work with: for the articles that are supposed to stick around, you can still choose between tick and dormant and read marks. But with total-expire, you only have dormant and ticked to choose from. The advantage of total-expire is that it works well with adaptive scoring (see [\[Adaptive Scoring\]](#), page [\[undefined\]](#)). Auto-expire works with normal scoring but not with adaptive scoring.

Groups that match the regular expression `gnus-auto-expirable-newsgroups` will have all articles that you read marked as expirable automatically. All articles marked as expirable have an ‘E’ in the first column in the summary buffer.

By default, if you have auto expiry switched on, Gnus will mark all the articles you read as expirable, no matter if they were read or unread before. To avoid having articles marked as read marked as expirable automatically, you can put something like the following in your ‘`~/gnus.el`’ file:



```
(remove-hook 'gnus-mark-article-hook
             'gnus-summary-mark-read-and-unread-as-read)
(add-hook 'gnus-mark-article-hook 'gnus-summary-mark-unread-as-read)
```

Note that making a group auto-expirable doesn't mean that all read articles are expired—only the articles marked as expirable will be expired. Also note that using the `d` command won't make articles expirable—only semi-automatic marking of articles as read will mark the articles as expirable in auto-expirable groups.

Let's say you subscribe to a couple of mailing lists, and you want the articles you have read to disappear after a while:

```
(setq gnus-auto-expirable-newsgroups
      "mail.nonsense-list\\|mail.nice-list")
```

Another way to have auto-expiry happen is to have the element `auto-expire` in the group parameters of the group.

If you use adaptive scoring (see [\[Adaptive Scoring\]](#), page [\[undefined\]](#)) and auto-expiring, you'll have problems. Auto-expiring and adaptive scoring don't really mix very well.

The `nnmail-expiry-wait` variable supplies the default time an expirable article has to live. Gnus starts counting days from when the message *arrived*, not from when it was sent. The default is seven days.

Gnus also supplies a function that lets you fine-tune how long articles are to live, based on what group they are in. Let's say you want to have one month expiry period in the 'mail.private' group, a one day expiry period in the 'mail.junk' group, and a six day expiry period everywhere else:

```
(setq nnmail-expiry-wait-function
      (lambda (group)
        (cond ((string= group "mail.private")
                31)
              ((string= group "mail.junk")
                1)
              ((string= group "important")
                'never)
              (t
                6))))
```

The group names this function is fed are “unadorned” group names—no 'nnml:' prefixes and the like.

The `nnmail-expiry-wait` variable and `nnmail-expiry-wait-function` function can either be a number (not necessarily an integer) or one of the symbols `immediate` or `never`.

You can also use the `expiry-wait` group parameter to selectively change the expiry period (see [\[Group Parameters\]](#), page [\[undefined\]](#)).

The normal action taken when expiring articles is to delete them. However, in some circumstances it might make more sense to move them to other groups instead of deleting them. The variable `nnmail-expiry-target` (and the `expiry-target` group parameter) controls this. The variable supplies a default value for all groups, which can be overridden for specific groups by the group parameter. default value is `delete`, but this can also be

a string (which should be the name of the group the message should be moved to), or a function (which will be called in a buffer narrowed to the message in question, and with the name of the group being moved from as its parameter) which should return a target—either a group name or `delete`.

Here’s an example for specifying a group name:

```
(setq nnmail-expiry-target "nnml:expired")
```

Gnus provides a function `nnmail-fancy-expiry-target` which will expire mail to groups according to the variable `nnmail-fancy-expiry-targets`. Here’s an example:

```
(setq nnmail-expiry-target 'nnmail-fancy-expiry-target
  nnmail-fancy-expiry-targets
  '((to-from "boss" "nnfolder:Work")
    ("subject" "IMPORTANT" "nnfolder:IMPORTANT.%Y.%b")
    ("from" ".*" "nnfolder:Archive-%Y"))))
```

With this setup, any mail that has `IMPORTANT` in its Subject header and was sent in the year `YYYY` and month `MMM`, will get expired to the group `nnfolder:IMPORTANT.YYYY.MMM`. If its From or To header contains the string `boss`, it will get expired to `nnfolder:Work`. All other mail will get expired to `nnfolder:Archive-YYYY`.

If `nnmail-keep-last-article` is non-`nil`, Gnus will never expire the final article in a mail newsgroup. This is to make life easier for procmail users.

By the way: That line up there, about Gnus never expiring non-expirable articles, is a lie. If you put `total-expire` in the group parameters, articles will not be marked as expirable, but all read articles will be put through the expiry process. Use with extreme caution. Even more dangerous is the `gnus-total-expirable-newsgroups` variable. All groups that match this regexp will have all read articles put through the expiry process, which means that *all* old mail articles in the groups in question will be deleted after a while. Use with extreme caution, and don’t come crying to me when you discover that the regexp you used matched the wrong group and all your important mail has disappeared. Be a *man*! Or a *woman*! Whatever you feel more comfortable with! So there!

Most people make most of their mail groups total-expirable, though.

If `gnus-inhibit-user-auto-expire` is non-`nil`, user marking commands will not mark an article as expirable, even if the group has auto-expire turned on.

### 6.3.10 Washing Mail

Mailers and list servers are notorious for doing all sorts of really, really stupid things with mail. “Hey, RFC 822 doesn’t explicitly prohibit us from adding the string `wE aRe ElItE!!!!1!!` to the end of all lines passing through our server, so let’s do that!!!!1!” Yes, but RFC 822 wasn’t designed to be read by morons. Things that were considered to be self-evident were not discussed. So. Here we are.

Case in point: The German version of Microsoft Exchange adds ‘`AW:` ’ to the subjects of replies instead of ‘`Re:` ’. I could pretend to be shocked and dismayed by this, but I haven’t got the energy. It is to laugh.

Gnus provides a plethora of functions for washing articles while displaying them, but it might be nicer to do the filtering before storing the mail to disk. For that purpose, we have three hooks and various functions that can be put in these hooks.

**nnmail-prepare-incoming-hook**

This hook is called before doing anything with the mail and is meant for grand, sweeping gestures. It is called in a buffer that contains all the new, incoming mail. Functions to be used include:

**nnheader-ms-strip-cr**

Remove trailing carriage returns from each line. This is default on Emacs running on MS machines.

**nnmail-prepare-incoming-header-hook**

This hook is called narrowed to each header. It can be used when cleaning up the headers. Functions that can be used include:

**nnmail-remove-leading-whitespace**

Clear leading white space that “helpful” listservs have added to the headers to make them look nice. Aaah.

(Note that this function works on both the header on the body of all messages, so it is a potentially dangerous function to use (if a body of a message contains something that looks like a header line). So rather than fix the bug, it is of course the right solution to make it into a feature by documenting it.)

**nnmail-remove-list-identifiers**

Some list servers add an identifier—for example, ‘(idm)’—to the beginning of all **Subject** headers. I’m sure that’s nice for people who use stone age mail readers. This function will remove strings that match the **nnmail-list-identifiers** regexp, which can also be a list of regexp. **nnmail-list-identifiers** may not contain `\\(\\.\\)`.

For instance, if you want to remove the ‘(idm)’ and the ‘nagnagnag’ identifiers:

```
(setq nnmail-list-identifiers
      '("(idm)" "nagnagnag"))
```

This can also be done non-destructively with **gnus-list-identifiers**, See [\[Article Hiding\]](#), page [\[undefined\]](#).

**nnmail-remove-tabs**

Translate all ‘TAB’ characters into ‘SPACE’ characters.

**nnmail-fix-eudora-headers**

Eudora produces broken **References** headers, but OK **In-Reply-To** headers. This function will get rid of the **References** headers.

**nnmail-prepare-incoming-message-hook**

This hook is called narrowed to each message. Functions to be used include:

**article-de-quoted-unreadable**

Decode Quoted Readable encoding.

### 6.3.11 Duplicates

If you are a member of a couple of mailing lists, you will sometimes receive two copies of the same mail. This can be quite annoying, so `nnmail` checks for and treats any duplicates it might find. To do this, it keeps a cache of old `Message-IDs`— `nnmail-message-id-cache-file`, which is `~/ .nnmail-cache` by default. The approximate maximum number of `Message-IDs` stored there is controlled by the `nnmail-message-id-cache-length` variable, which is 1000 by default. (So 1000 `Message-IDs` will be stored.) If all this sounds scary to you, you can set `nnmail-treat-duplicates` to `warn` (which is what it is by default), and `nnmail` won't delete duplicate mails. Instead it will insert a warning into the head of the mail saying that it thinks that this is a duplicate of a different message.

This variable can also be a function. If that's the case, the function will be called from a buffer narrowed to the message in question with the `Message-ID` as a parameter. The function must return either `nil`, `warn`, or `delete`.

You can turn this feature off completely by setting the variable to `nil`.

If you want all the duplicate mails to be put into a special *duplicates* group, you could do that using the normal mail split methods:

```
(setq nnmail-split-fancy
  '(| ;; Messages duplicates go to a separate group.
    ("gnus-warning" "duplicat\\(e\\|ion\\) of message" "duplicate")
    ;; Message from daemons, postmaster, and the like to another.
    (any mail "mail.misc")
    ;; Other rules.
    [...] ))
```

Or something like:

```
(setq nnmail-split-methods
  '(("duplicates" "^Gnus-Warning:.*duplicate")
    ;; Other rules.
    [...]))
```

Here's a neat feature: If you know that the recipient reads her mail with Gnus, and that she has `nnmail-treat-duplicates` set to `delete`, you can send her as many insults as you like, just by using a `Message-ID` of a mail that you know that she's already received. Think of all the fun! She'll never see any of it! Whee!

### 6.3.12 Not Reading Mail

If you start using any of the mail back ends, they have the annoying habit of assuming that you want to read mail with them. This might not be unreasonable, but it might not be what you want.

If you set `mail-sources` and `nnmail-spool-file` to `nil`, none of the back ends will ever attempt to read incoming mail, which should help.

This might be too much, if, for instance, you are reading mail quite happily with `nnml` and just want to peek at some old Rmail file you have stashed away with `nnbaby1`. All back ends have variables called `back-end-get-new-mail`. If you want to disable the `nnbaby1` mail reading, you edit the virtual server for the group to have a setting where `nnbaby1-get-new-mail` to `nil`.

All the mail back ends will call `nn*-prepare-save-mail-hook` narrowed to the article to be saved before saving it when reading incoming mail.

### 6.3.13 Choosing a Mail Back End

Gnus will read the mail spool when you activate a mail group. The mail file is first copied to your home directory. What happens after that depends on what format you want to store your mail in.

There are six different mail back ends in the standard Gnus, and more back ends are available separately. The mail back end most people use (because it is possibly the fastest) is `nnml` (see [\[Mail Spool\]](#), page [\[undefined\]](#)).

#### 6.3.13.1 Unix Mail Box

The `nnmbox` back end will use the standard `Un*x` mbox file to store mail. `nnmbox` will add extra headers to each mail article to say which group it belongs in.

Virtual server settings:

`nnmbox-mbox-file`

The name of the mail box in the user's home directory. Default is `'~/mbox'`.

`nnmbox-active-file`

The name of the active file for the mail box. Default is `'~/mbox-active'`.

`nnmbox-get-new-mail`

If non-nil, `nnmbox` will read incoming mail and split it into groups. Default is `t`.

#### 6.3.13.2 Rmail Baby1

The `nnbabyl` back end will use a Babyl mail box (aka. *Rmail mbox*) to store mail. `nnbabyl` will add extra headers to each mail article to say which group it belongs in.

Virtual server settings:

`nnbabyl-mbox-file`

The name of the Rmail mbox file. The default is `'~/RMAIL'`

`nnbabyl-active-file`

The name of the active file for the rmail box. The default is `'~/rmail-active'`

`nnbabyl-get-new-mail`

If non-nil, `nnbabyl` will read incoming mail. Default is `t`

#### 6.3.13.3 Mail Spool

The `nnml` spool mail format isn't compatible with any other known format. It should be used with some caution.

If you use this back end, Gnus will split all incoming mail into files, one file for each mail, and put the articles into the corresponding directories under the directory specified by the `nnml-directory` variable. The default value is `'~/Mail/'`.

You do not have to create any directories beforehand; Gnus will take care of all that.

If you have a strict limit as to how many files you are allowed to store in your account, you should not use this back end. As each mail gets its own file, you might very well occupy

thousands of inodes within a few weeks. If this is no problem for you, and it isn't a problem for you having your friendly systems administrator walking around, madly, shouting "Who is eating all my inodes?! Who? Who!?!", then you should know that this is probably the fastest format to use. You do not have to trudge through a big mbox file just to read your new mail.

`nnml` is probably the slowest back end when it comes to article splitting. It has to create lots of files, and it also generates NOV databases for the incoming mails. This makes it possibly the fastest back end when it comes to reading mail.

When the marks file is used (which it is by default), `nnml` servers have the property that you may backup them using `tar` or similar, and later be able to restore them into Gnus (by adding the proper `nnml` server) and have all your marks be preserved. Marks for a group is usually stored in the `.marks` file (but see `nnml-marks-file-name`) within each `nnml` group's directory. Individual `nnml` groups are also possible to backup, use `G m` to restore the group (after restoring the backup into the `nnml` directory).

If for some reason you believe your `'marks'` files are screwed up, you can just delete them all. Gnus will then correctly regenerate them next time it starts.

Virtual server settings:

#### `nnml-directory`

All `nnml` directories will be placed under this directory. The default is the value of `message-directory` (whose default value is `'~/Mail'`).

#### `nnml-active-file`

The active file for the `nnml` server. The default is `'~/Mail/active'`.

#### `nnml-newsgroups-file`

The `nnml` group descriptions file. See [\[Newsgroups File Format\]](#), page [\[undefined\]](#). The default is `'~/Mail/newsgroups'`.

#### `nnml-get-new-mail`

If non-`nil`, `nnml` will read incoming mail. The default is `t`.

#### `nnml-nov-is-evil`

If non-`nil`, this back end will ignore any NOV files. The default is `nil`.

#### `nnml-nov-file-name`

The name of the NOV files. The default is `'overview'`.

#### `nnml-prepare-save-mail-hook`

Hook run narrowed to an article before saving.

#### `nnml-marks-is-evil`

If non-`nil`, this back end will ignore any MARKS files. The default is `nil`.

#### `nnml-marks-file-name`

The name of the *marks* files. The default is `'marks'`.

#### `nnml-use-compressed-files`

If non-`nil`, `nnml` will allow using compressed message files.

If your `nnml` groups and NOV files get totally out of whack, you can do a complete update by typing `M-x nnml-generate-nov-databases`. This command will trawl through

the entire `nnml` hierarchy, looking at each and every article, so it might take a while to complete. A better interface to this functionality can be found in the server buffer (see [\[Server Commands\]](#), page [\[undefined\]](#)).

#### 6.3.13.4 MH Spool

`nnmh` is just like `nnml`, except that it doesn't generate NOV databases and it doesn't keep an active file or marks file. This makes `nnmh` a *much* slower back end than `nnml`, but it also makes it easier to write procmail scripts for.

Virtual server settings:

##### `nnmh-directory`

All `nnmh` directories will be located under this directory. The default is the value of `message-directory` (whose default is `~/Mail`)

##### `nnmh-get-new-mail`

If non-`nil`, `nnmh` will read incoming mail. The default is `t`.

##### `nnmh-be-safe`

If non-`nil`, `nnmh` will go to ridiculous lengths to make sure that the articles in the folder are actually what Gnus thinks they are. It will check date stamps and stat everything in sight, so setting this to `t` will mean a serious slow-down. If you never use anything but Gnus to read the `nnmh` articles, you do not have to set this variable to `t`. The default is `nil`.

#### 6.3.13.5 Maildir

`nnmaildir` stores mail in the maildir format, with each maildir corresponding to a group in Gnus. This format is documented here: <http://cr.yp.to/proto/maildir.html> and here: <http://www.qmail.org/man/man5/maildir.html>. `nnmaildir` also stores extra information in the `'.nnmaildir/'` directory within a maildir.

Maildir format was designed to allow concurrent deliveries and reading, without needing locks. With other back ends, you would have your mail delivered to a spool of some kind, and then you would configure Gnus to split mail from that spool into your groups. You can still do that with `nnmaildir`, but the more common configuration is to have your mail delivered directly to the maildirs that appear as group in Gnus.

`nnmaildir` is designed to be perfectly reliable: `C-g` will never corrupt its data in memory, and `SIGKILL` will never corrupt its data in the filesystem.

`nnmaildir` stores article marks and NOV data in each maildir. So you can copy a whole maildir from one Gnus setup to another, and you will keep your marks.

Virtual server settings:

##### `directory`

For each of your `nnmaildir` servers (it's very unlikely that you'd need more than one), you need to create a directory and populate it with maildirs or symlinks to maildirs (and nothing else; do not choose a directory already used for other purposes). Each maildir will be represented in Gnus as a newsgroup on that server; the filename of the symlink will be the name of the group. Any filenames in the directory starting with `'.'` are ignored. The directory is scanned



when you first start Gnus, and each time you type `g` in the group buffer; if any maildirs have been removed or added, `nnmaildir` notices at these times.

The value of the `directory` parameter should be a Lisp form which is processed by `eval` and `expand-file-name` to get the path of the directory for this server. The form is `eval`ed only when the server is opened; the resulting string is used until the server is closed. (If you don't know about forms and `eval`, don't worry—a simple string will work.) This parameter is not optional; you must specify it. I don't recommend using `"~/Mail"` or a subdirectory of it; several other parts of Gnus use that directory by default for various things, and may get confused if `nnmaildir` uses it too. `"~/nnmaildir"` is a typical value.

#### `target-prefix`

This should be a Lisp form which is processed by `eval` and `expand-file-name`. The form is `eval`ed only when the server is opened; the resulting string is used until the server is closed.

When you create a group on an `nnmaildir` server, the maildir is created with `target-prefix` prepended to its name, and a symlink pointing to that maildir is created, named with the plain group name. So if `directory` is `"~/nnmaildir"` and `target-prefix` is `"../maildirs/"`, then when you create the group `foo`, `nnmaildir` will create `'~/nnmaildir/../maildirs/foo'` as a maildir, and will create `'~/nnmaildir/foo'` as a symlink pointing to `'../maildirs/foo'`.

You can set `target-prefix` to a string without any slashes to create both maildirs and symlinks in the same `directory`; in this case, any maildirs found in `directory` whose names start with `target-prefix` will not be listed as groups (but the symlinks pointing to them will be).

As a special case, if `target-prefix` is `"` (the default), then when you create a group, the maildir will be created in `directory` without a corresponding symlink. Beware that you cannot use `gnus-group-delete-group` on such groups without the `force` argument.

#### `directory-files`

This should be a function with the same interface as `directory-files` (such as `directory-files` itself). It is used to scan the server's `directory` for maildirs. This parameter is optional; the default is `nnheader-directory-files-safe` if `nnheader-directory-files-is-safe` is `nil`, and `directory-files` otherwise. (`nnheader-directory-files-is-safe` is checked only once when the server is opened; if you want to check it each time the directory is scanned, you'll have to provide your own function that does that.)

#### `get-new-mail`

If non-`nil`, then after scanning for new mail in the group maildirs themselves as usual, this server will also incorporate mail the conventional Gnus way, from `mail-sources` according to `nnmail-split-methods` or `nnmail-split-fancy`. The default value is `nil`.

Do *not* use the same maildir both in `mail-sources` and as an `nnmaildir` group. The results might happen to be useful, but that would be by chance, not by design, and the results might be different in the future. If your split rules create new groups, remember to supply a `create-directory` server parameter.



### 6.3.13.6 Group parameters

`nnmaildir` uses several group parameters. It's safe to ignore all this; the default behavior for `nnmaildir` is the same as the default behavior for other mail back ends: articles are deleted after one week, etc. Except for the expiry parameters, all this functionality is unique to `nnmaildir`, so you can ignore it if you're just trying to duplicate the behavior you already have with another back end.

If the value of any of these parameters is a vector, the first element is evaluated as a Lisp form and the result is used, rather than the original value. If the value is not a vector, the value itself is evaluated as a Lisp form. (This is why these parameters use names different from those of other, similar parameters supported by other back ends: they have different, though similar, meanings.) (For numbers, strings, `nil`, and `t`, you can ignore the `eval` business again; for other values, remember to use an extra quote and wrap the value in a vector when appropriate.)

#### `expire-age`

An integer specifying the minimum age, in seconds, of an article before it will be expired, or the symbol `never` to specify that articles should never be expired. If this parameter is not set, `nnmaildir` falls back to the usual `nnmail-expiry-wait(-function)` variables (overrideable by the `expiry-wait(-function)` group parameters. If you wanted a value of 3 days, you could use something like `[(* 3 24 60 60)]`; `nnmaildir` will evaluate the form and use the result. An article's age is measured starting from the article file's modification time. Normally, this is the same as the article's delivery time, but editing an article makes it younger. Moving an article (other than via expiry) may also make an article younger.

#### `expire-group`

If this is set to a string such as a full Gnus group name, like

```
"backend+server.address.string:group.name"
```

and if it is not the name of the same group that the parameter belongs to, then articles will be moved to the specified group during expiry before being deleted. *If this is set to an `nnmaildir` group, the article will be just as old in the destination group as it was in the source group.* So be careful with `expire-age` in the destination group. If this is set to the name of the same group that the parameter belongs to, then the article is not expired at all. If you use the vector form, the first element is evaluated once for each article. So that form can refer to `nnmaildir-article-file-name`, etc., to decide where to put the article. *If this parameter is not set, `nnmaildir` does not fall back to the `expiry-target` group parameter or the `nnmail-expiry-target` variable.*

#### `read-only`

If this is set to `t`, `nnmaildir` will treat the articles in this maildir as read-only. This means: articles are not renamed from `'new/` into `'cur/`; articles are only found in `'new/`, not `'cur/`; articles are never deleted; articles cannot be edited. `'new/` is expected to be a symlink to the `'new/` directory of another maildir—e.g., a system-wide mailbox containing a mailing list of common interest. Everything in the maildir outside `'new/` is *not* treated as read-only, so for a shared mailbox, you do still need to set up your own maildir (or have write

permission to the shared mailbox); your maildir just won't contain extra copies of the articles.

#### **directory-files**

A function with the same interface as **directory-files**. It is used to scan the directories in the maildir corresponding to this group to find articles. The default is the function specified by the server's **directory-files** parameter.

#### **distrust-Lines:**

If non-**nil**, **nnmaildir** will always count the lines of an article, rather than use the **Lines:** header field. If **nil**, the header field will be used if present.

#### **always-marks**

A list of mark symbols, such as **['(read expire)]**. Whenever Gnus asks **nnmaildir** for article marks, **nnmaildir** will say that all articles have these marks, regardless of whether the marks stored in the filesystem say so. This is a proof-of-concept feature that will probably be removed eventually; it ought to be done in Gnus proper, or abandoned if it's not worthwhile.

#### **never-marks**

A list of mark symbols, such as **['(tick expire)]**. Whenever Gnus asks **nnmaildir** for article marks, **nnmaildir** will say that no articles have these marks, regardless of whether the marks stored in the filesystem say so. **never-marks** overrides **always-marks**. This is a proof-of-concept feature that will probably be removed eventually; it ought to be done in Gnus proper, or abandoned if it's not worthwhile.

#### **nov-cache-size**

An integer specifying the size of the NOV memory cache. To speed things up, **nnmaildir** keeps NOV data in memory for a limited number of articles in each group. (This is probably not worthwhile, and will probably be removed in the future.) This parameter's value is noticed only the first time a group is seen after the server is opened—i.e., when you first start Gnus, typically. The NOV cache is never resized until the server is closed and reopened. The default is an estimate of the number of articles that would be displayed in the summary buffer: a count of articles that are either marked with **tick** or not marked with **read**, plus a little extra.

### **6.3.13.7 Article identification**

Articles are stored in the **'cur/'** subdirectory of each maildir. Each article file is named like **uniq:info**, where **uniq** contains no colons. **nnmaildir** ignores, but preserves, the **:info** part. (Other maildir readers typically use this part of the filename to store marks.) The **uniq** part uniquely identifies the article, and is used in various places in the **'.nnmaildir/'** subdirectory of the maildir to store information about the corresponding article. The full pathname of an article is available in the variable **nnmaildir-article-file-name** after you request the article in the summary buffer.

### **6.3.13.8 NOV data**

An article identified by **uniq** has its NOV data (used to generate lines in the summary buffer) stored in **.nnmaildir/nov/uniq**. There is no **nnmaildir-generate-nov-databases**

function. (There isn't much need for it—an article's NOV data is updated automatically when the article or `nnmail-extra-headers` has changed.) You can force `nnmaildir` to regenerate the NOV data for a single article simply by deleting the corresponding NOV file, but *beware*: this will also cause `nnmaildir` to assign a new article number for this article, which may cause trouble with `seen` marks, the Agent, and the cache.

### 6.3.13.9 Article marks

An article identified by `uniq` is considered to have the mark `flag` when the file `‘.nnmaildir/marks/flag/uniq’` exists. When Gnus asks `nnmaildir` for a group's marks, `nnmaildir` looks for such files and reports the set of marks it finds. When Gnus asks `nnmaildir` to store a new set of marks, `nnmaildir` creates and deletes the corresponding files as needed. (Actually, rather than create a new file for each mark, it just creates hard links to `‘.nnmaildir/markfile’`, to save inodes.)

You can invent new marks by creating a new directory in `‘.nnmaildir/marks/’`. You can tar up a maildir and remove it from your server, untar it later, and keep your marks. You can add and remove marks yourself by creating and deleting mark files. If you do this while Gnus is running and your `nnmaildir` server is open, it's best to exit all summary buffers for `nnmaildir` groups and type `s` in the group buffer first, and to type `g` or `M-g` in the group buffer afterwards. Otherwise, Gnus might not pick up the changes, and might undo them.

### 6.3.13.10 Mail Folders

`nnfolder` is a back end for storing each mail group in a separate file. Each file is in the standard `Un*x` mbox format. `nnfolder` will add extra headers to keep track of article numbers and arrival dates.

When the marks file is used (which it is by default), `nnfolder` servers have the property that you may backup them using `tar` or similar, and later be able to restore them into Gnus (by adding the proper `nnfolder` server) and have all your marks be preserved. Marks for a group is usually stored in a file named as the mbox file with `.mrk` concatenated to it (but see `nnfolder-marks-file-suffix`) within the `nnfolder` directory. Individual `nnfolder` groups are also possible to backup, use `G m` to restore the group (after restoring the backup into the `nnfolder` directory).

Virtual server settings:

#### `nnfolder-directory`

All the `nnfolder` mail boxes will be stored under this directory. The default is the value of `message-directory` (whose default is `‘~/Mail’`)

#### `nnfolder-active-file`

The name of the active file. The default is `‘~/Mail/active’`.

#### `nnfolder-newsgroups-file`

The name of the group descriptions file. See [\[Newsgroups File Format\]](#), page [\[undefined\]](#). The default is `‘~/Mail/newsgroups’`

#### `nnfolder-get-new-mail`

If non-nil, `nnfolder` will read incoming mail. The default is `t`

**nnfolder-save-buffer-hook**

Hook run before saving the folders. Note that Emacs does the normal backup renaming of files even with the **nnfolder** buffers. If you wish to switch this off, you could say something like the following in your `‘.emacs’` file:

```
(defun turn-off-backup ()
  (set (make-local-variable 'backup-inhibited) t))

(add-hook 'nnfolder-save-buffer-hook 'turn-off-backup)
```

**nnfolder-delete-mail-hook**

Hook run in a buffer narrowed to the message that is to be deleted. This function can be used to copy the message to somewhere else, or to extract some information from it before removing it.

**nnfolder-nov-is-evil**

If non-`nil`, this back end will ignore any NOV files. The default is `nil`.

**nnfolder-nov-file-suffix**

The extension for NOV files. The default is `‘.nov’`.

**nnfolder-nov-directory**

The directory where the NOV files should be stored. If `nil`, **nnfolder-directory** is used.

**nnfolder-marks-is-evil**

If non-`nil`, this back end will ignore any MARKS files. The default is `nil`.

**nnfolder-marks-file-suffix**

The extension for MARKS files. The default is `‘.mrk’`.

**nnfolder-marks-directory**

The directory where the MARKS files should be stored. If `nil`, **nnfolder-directory** is used.

If you have lots of **nnfolder**-like files you’d like to read with **nnfolder**, you can use the `M-x nnfolder-generate-active-file` command to make **nnfolder** aware of all likely files in **nnfolder-directory**. This only works if you use long file names, though.

### 6.3.13.11 Comparing Mail Back Ends

First, just for terminology, the *back end* is the common word for a low-level access method—a transport, if you will, by which something is acquired. The sense is that one’s mail has to come from somewhere, and so selection of a suitable back end is required in order to get that mail within spitting distance of Gnus.

The same concept exists for Usenet itself: Though access to articles is typically done by NNTP these days, once upon a midnight dreary, everyone in the world got at Usenet by running a reader on the machine where the articles lay (the machine which today we call an NNTP server), and access was by the reader stepping into the articles’ directory spool area directly. One can still select between either the **nnntp** or **nnspool** back ends, to select between these methods, if one happens actually to live on the server (or can see its spool directly, anyway, via NFS).

The goal in selecting a mail back end is to pick one which simultaneously represents a suitable way of dealing with the original format plus leaving mail in a form that is convenient to use in the future. Here are some high and low points on each:

#### **nnmbox**

UNIX systems have historically had a single, very common, and well-defined format. All messages arrive in a single *spool file*, and they are delineated by a line whose regular expression matches `^From_`. (My notational use of `_` is to indicate a space, to make it clear in this instance that this is not the RFC-specified `From:` header.) Because Emacs and therefore Gnus emanate historically from the Unix environment, it is simplest if one does not mess a great deal with the original mailbox format, so if one chooses this back end, Gnus' primary activity in getting mail from the real spool area to Gnus' preferred directory is simply to copy it, with no (appreciable) format change in the process. It is the "dumbest" way to move mail into availability in the Gnus environment. This makes it fast to move into place, but slow to parse, when Gnus has to look at what's where.

#### **nnbabyl**

Once upon a time, there was the DEC-10 and DEC-20, running operating systems called TOPS and related things, and the usual (only?) mail reading environment was a thing called Babyl. I don't know what format was used for mail landing on the system, but Babyl had its own internal format to which mail was converted, primarily involving creating a spool-file-like entity with a scheme for inserting Babyl-specific headers and status bits above the top of each message in the file. Rmail was Emacs' first mail reader, it was written by Richard Stallman, and Stallman came out of that TOPS/Babyl environment, so he wrote Rmail to understand the mail files folks already had in existence. Gnus (and VM, for that matter) continue to support this format because it's perceived as having some good qualities in those mailer-specific headers/status bits stuff. Rmail itself still exists as well, of course, and is still maintained by Stallman.

Both of the above forms leave your mail in a single file on your file system, and they must parse that entire file each time you take a look at your mail.

#### **nnml**

**nnml** is the back end which smells the most as though you were actually operating with an **nnspool**-accessed Usenet system. (In fact, I believe **nnml** actually derived from **nnspool** code, lo these years ago.) One's mail is taken from the original spool file, and is then cut up into individual message files, 1:1. It maintains a Usenet-style active file (analogous to what one finds in an INN- or CNews-based news system in (for instance) `/var/lib/news/active`, or what is returned via the `NNTP LIST` verb) and also creates overview files for efficient group entry, as has been defined for NNTP servers for some years now. It is slower in mail-splitting, due to the creation of lots of files, updates to the **nnml** active file, and additions to overview files on a per-message basis, but it is extremely fast on access because of what amounts to the indexing support provided by the active file and overviews.

**nnml** costs *inodes* in a big way; that is, it soaks up the resource which defines available places in the file system to put new files. Sysadmins take a dim view of heavy inode occupation within tight, shared file systems. But if you live on a personal machine where the file system is your own and space is not at a premium, **nnml** wins big.

It is also problematic using this back end if you are living in a FAT16-based Windows world, since much space will be wasted on all these tiny files.

#### **nnmh**

The Rand MH mail-reading system has been around UNIX systems for a very long time; it operates by splitting one's spool file of messages into individual files, but with little or no indexing support—**nnmh** is considered to be semantically equivalent to “**nnml** without active file or overviews”. This is arguably the worst choice, because one gets the slowness of individual file creation married to the slowness of access parsing when learning what's new in one's groups.

#### **nnfolder**

Basically the effect of **nnfolder** is **nnmbox** (the first method described above) on a per-group basis. That is, **nnmbox** itself puts *all* one's mail in one file; **nnfolder** provides a little bit of optimization to this so that each of one's mail groups has a Unix mail box file. It's faster than **nnmbox** because each group can be parsed separately, and still provides the simple Unix mail box format requiring minimal effort in moving the mail around. In addition, it maintains an “active” file making it much faster for Gnus to figure out how many messages there are in each separate group.

If you have groups that are expected to have a massive amount of messages, **nnfolder** is not the best choice, but if you receive only a moderate amount of mail, **nnfolder** is probably the most friendly mail back end all over.

#### **nnmaildir**

For configuring expiry and other things, **nnmaildir** uses incompatible group parameters, slightly different from those of other mail back ends.

**nnmaildir** is largely similar to **nnml**, with some notable differences. Each message is stored in a separate file, but the filename is unrelated to the article number in Gnus. **nnmaildir** also stores the equivalent of **nnml**'s overview files in one file per article, so it uses about twice as many inodes as **nnml**. (Use `df -i` to see how plentiful your inode supply is.) If this slows you down or takes up very much space, consider switching to **ReiserFS** or another non-block-structured file system.

Since maildirs don't require locking for delivery, the maildirs you use as groups can also be the maildirs your mail is directly delivered to. This means you can skip Gnus' mail splitting if your mail is already organized into different mailboxes during delivery. A **directory** entry in **mail-sources** would have a similar effect, but would require one set of mailboxes for spooling deliveries (in mbox format, thus damaging message bodies), and another set to be used as groups (in whatever format you like). A maildir has a built-in spool, in the **new/** subdirectory. Beware that currently, mail moved from **new/** to **cur/** instead of via mail splitting will not undergo treatment such as duplicate checking.



`nnmaildir` stores article marks for a given group in the corresponding maildir, in a way designed so that it's easy to manipulate them from outside Gnus. You can tar up a maildir, unpack it somewhere else, and still have your marks. `nnml` also stores marks, but it's not as easy to work with them from outside Gnus as with `nnmaildir`.

`nnmaildir` uses a significant amount of memory to speed things up. (It keeps in memory some of the things that `nnml` stores in files and that `nnmh` repeatedly parses out of message files.) If this is a problem for you, you can set the `nov-cache-size` group parameter to something small (0 would probably not work, but 1 probably would) to make it use less memory. This caching will probably be removed in the future.

Startup is likely to be slower with `nnmaildir` than with other back ends. Everything else is likely to be faster, depending in part on your file system.

`nnmaildir` does not use `nnoo`, so you cannot use `nnoo` to write an `nnmaildir`-derived back end.

## 6.4 Browsing the Web

Web-based discussion forums are getting more and more popular. On many subjects, the web-based forums have become the most important forums, eclipsing the importance of mailing lists and news groups. The reason is easy to understand—they are friendly to new users; you just point and click, and there's the discussion. With mailing lists, you have to go through a cumbersome subscription procedure, and most people don't even know what a news group is.

The problem with this scenario is that web browsers are not very good at being news-readers. They do not keep track of what articles you've read; they do not allow you to score on subjects you're interested in; they do not allow off-line browsing; they require you to click around and drive you mad in the end.

So—if web browsers suck at reading discussion forums, why not use Gnus to do it instead?

Gnus has been getting a bit of a collection of back ends for providing interfaces to these sources.

All the web sources require Emacs/w3 and the url library to work.

The main caveat with all these web sources is that they probably won't work for a very long time. Gleaning information from the HTML data is guesswork at best, and when the layout is altered, the Gnus back end will fail. If you have reasonably new versions of these back ends, though, you should be ok.

One thing all these Web methods have in common is that the Web sources are often down, unavailable or just plain too slow to be fun. In those cases, it makes a lot of sense to let the Gnus Agent (see [\[Gnus Unplugged\]](#), page [\[undefined\]](#)) handle downloading articles, and then you can read them at leisure from your local disk. No more World Wide Wait for you.

### 6.4.1 Archiving Mail

Some of the back ends, notably `nnml`, `nnfolder`, and `nnmaildir`, now actually store the article marks with each group. For these servers, archiving and restoring a group while preserving marks is fairly simple.

(Preserving the group level and group parameters as well still requires ritual dancing and sacrifices to the ‘.newsrc.eld’ deity though.)

To archive an entire **nnml**, **nnfolder**, or **nnmaildir** server, take a recursive copy of the server directory. There is no need to shut down Gnus, so archiving may be invoked by **cron** or similar. You restore the data by restoring the directory tree, and adding a server definition pointing to that directory in Gnus. The [\[Article Backlog\]](#), page [\[undefined\]](#), [\[Asynchronous Fetching\]](#), page [\[undefined\]](#) and other things might interfere with overwriting data, so you may want to shut down Gnus before you restore the data.

It is also possible to archive individual **nnml**, **nnfolder**, or **nnmaildir** groups, while preserving marks. For **nnml** or **nnmaildir**, you copy all files in the group’s directory. For **nnfolder** you need to copy both the base folder file itself (‘F00’, say), and the marks file (‘F00.mrk’ in this example). Restoring the group is done with **G m** from the Group buffer. The last step makes Gnus notice the new directory. **nnmaildir** notices the new directory automatically, so **G m** is unnecessary in that case.

### 6.4.2 Web Searches

It’s, like, too neat to search the Usenet for articles that match a string, but it, like, totally *sucks*, like, totally, to use one of those, like, Web browsers, and you, like, have to, rilly, like, look at the commercials, so, like, with Gnus you can do *rad*, rilly, searches without having to use a browser.

The **nnweb** back end allows an easy interface to the mighty search engine. You create an **nnweb** group, enter a search pattern, and then enter the group and read the articles like you would any normal group. The **G w** command in the group buffer (see [\[Foreign Groups\]](#), page [\[undefined\]](#)) will do this in an easy-to-use fashion.

**nnweb** groups don’t really lend themselves to being solid groups—they have a very fleeting idea of article numbers. In fact, each time you enter an **nnweb** group (not even changing the search pattern), you are likely to get the articles ordered in a different manner. Not even using duplicate suppression (see [\[Duplicate Suppression\]](#), page [\[undefined\]](#)) will help, since **nnweb** doesn’t even know the **Message-ID** of the articles before reading them using some search engines (Google, for instance). The only possible way to keep track of which articles you’ve read is by scoring on the **Date** header—mark all articles posted before the last date you read the group as read.

If the search engine changes its output substantially, **nnweb** won’t be able to parse it and will fail. One could hardly fault the Web providers if they were to do this—their *raison d’être* is to make money off of advertisements, not to provide services to the community. Since **nnweb** washes the ads off all the articles, one might think that the providers might be somewhat miffed. We’ll see.

You must have the **url** and **w3** package installed to be able to use **nnweb**.

Virtual server variables:

#### **nnweb-type**

What search engine type is being used. The currently supported types are **google**, **dejanews**, and **gmane**. Note that **dejanews** is an alias to **google**.

#### **nnweb-search**

The search string to feed to the search engine.



**nnweb-max-hits**

Advisory maximum number of hits per search to display. The default is 999.

**nnweb-type-definition**

Type-to-definition alist. This alist says what **nnweb** should do with the various search engine types. The following elements must be present:

<b>article</b>	Function to decode the article and provide something that Gnus understands.
<b>map</b>	Function to create an article number to message header and URL alist.
<b>search</b>	Function to send the search string to the search engine.
<b>address</b>	The address the aforementioned function should send the search string to.
<b>id</b>	Format string URL to fetch an article by <b>Message-ID</b> .

**6.4.3 Slashdot**

**Slashdot** is a popular news site, with lively discussion following the news articles. **nnslashdot** will let you read this forum in a convenient manner.

The easiest way to read this source is to put something like the following in your ‘~/gnus.el’ file:

```
(setq gnus-secondary-select-methods
      '((nnslashdot "")))
```

This will make Gnus query the **nnslashdot** back end for new comments and groups. The **F** command will subscribe each new news article as a new Gnus group, and you can read the comments by entering these groups. (Note that the default subscription method is to subscribe new groups as zombies. Other methods are available (see [\[Subscription Methods\]](#), page [\[Subscription Methods\]](#))).

If you want to remove an old **nnslashdot** group, the **G DEL** command is the most handy tool (see [\[Foreign Groups\]](#), page [\[Foreign Groups\]](#)).

When following up to **nnslashdot** comments (or posting new comments), some light HTMLizations will be performed. In particular, text quoted with ‘>’ will be quoted with ‘**blockquote**’ instead, and signatures will have ‘**br**’ added to the end of each line. Other than that, you can just write HTML directly into the message buffer. Note that **Slashdot** filters out some HTML forms.

The following variables can be altered to change its behavior:

**nnslashdot-threaded**

Whether **nnslashdot** should display threaded groups or not. The default is **t**. To be able to display threads, **nnslashdot** has to retrieve absolutely all comments in a group upon entry. If a threaded display is not required, **nnslashdot** will only retrieve the comments that are actually wanted by the user. Threading is nicer, but much, much slower than unthreaded.

**nnslashdot-login-name**

The login name to use when posting.

**nnslashdot-password**

The password to use when posting.

**nnslashdot-directory**

Where **nnslashdot** will store its files. The default is ‘~/News/slashdot/’.

**nnslashdot-active-url**

The URL format string that will be used to fetch the information on news articles and comments. The default is

‘<http://slashdot.org/search.pl?section=&min=%d>’.

**nnslashdot-comments-url**

The URL format string that will be used to fetch comments.

**nnslashdot-article-url**

The URL format string that will be used to fetch the news article. The default is ‘<http://slashdot.org/article.pl?sid=%s&mode=nocomment>’.

**nnslashdot-threshold**

The score threshold. The default is -1.

**nnslashdot-group-number**

The number of old groups, in addition to the ten latest, to keep updated. The default is 0.

### 6.4.4 Ultimate

The **Ultimate Bulletin Board** is probably the most popular Web bulletin board system used. It has a quite regular and nice interface, and it’s possible to get the information Gnus needs to keep groups updated.

The easiest way to get started with **nnultimate** is to say something like the following in the group buffer: *B nnultimate RET <http://www.tcj.com/messboard/ubbcgi/> RET*. (Substitute the URL (not including ‘Ultimate.cgi’ or the like at the end) for a forum you’re interested in; there’s quite a list of them on the Ultimate web site.) Then subscribe to the groups you’re interested in from the server buffer, and read them from the group buffer.

The following **nnultimate** variables can be altered:

**nnultimate-directory**

The directory where **nnultimate** stores its files. The default is ‘~/News/ultimate/’.

### 6.4.5 Web Archive

Some mailing lists only have archives on Web servers, such as <http://www.egroups.com/> and <http://www.mail-archive.com/>. It has a quite regular and nice interface, and it’s possible to get the information Gnus needs to keep groups updated.

The easiest way to get started with **nnwarchive** is to say something like the following in the group buffer: *M-x gnus-group-make-warchive-group RET an\_egroup RET egroups RET [www.egroups.com](http://www.egroups.com) RET your@email.address RET*. (Substitute the *an\_egroup* with the mailing list you subscribed, the *your@email.address* with your email address.), or to browse the back end by *B nnwarchive RET mail-archive RET*.

The following **nnwarchive** variables can be altered:

**nnwarchive-directory**

The directory where **nnwarchive** stores its files. The default is  
 ‘~/News/warchive/’.

**nnwarchive-login**

The account name on the web server.

**nnwarchive-passwd**

The password for your account on the web server.

**6.4.6 RSS**

Some web sites have an RDF Site Summary (RSS). RSS is a format for summarizing headlines from news related sites (such as BBC or CNN). But basically anything list-like can be presented as an RSS feed: weblogs, changelogs or recent changes to a wiki (e.g. <http://cliki.net/recent-changes.rdf>).

RSS has a quite regular and nice interface, and it’s possible to get the information Gnus needs to keep groups updated.

Use **G R** from the summary buffer to subscribe to a feed—you will be prompted for the location of the feed.

An easy way to get started with **nnrss** is to say something like the following in the group buffer: **B nnrss RET RET y**, then subscribe to groups.

The following **nnrss** variables can be altered:

**nnrss-directory**

The directory where **nnrss** stores its files. The default is ‘~/News/rss/’.

**nnrss-use-local**

If you set **nnrss-use-local** to **t**, **nnrss** will read the feeds from local files in **nnrss-directory**. You can use the command **nnrss-generate-download-script** to generate a download script using **wget**.

The following code may be helpful, if you want to show the description in the summary buffer.

```
(add-to-list 'nnmail-extra-headers nnrss-description-field)
(setq gnus-summary-line-format "%U%R%z%I%([%4L: %-15,15f%]) %s%uX\n")
```

```
(defun gnus-user-format-function-X (header)
  (let ((descr
        (assq nnrss-description-field (mail-header-extra header))))
    (if descr (concat "\n\t" (cdr descr)) "")))
```

The following code may be useful to open an **nnrss** url directly from the summary buffer.

```
(require 'browse-url)

(defun browse-nnrss-url( arg )
  (interactive "p")
  (let ((url (assq nnrss-url-field
                  (mail-header-extra
                   (gnus-data-header
```

```

                                (assq (gnus-summary-article-number)
                                      gnus-newsgroup-data))))))
(if url
    (progn
      (browse-url (cdr url))
      (gnus-summary-mark-as-read-forward 1))
    (gnus-summary-scroll-up arg))))

(eval-after-load "gnus"
  #'(define-key gnus-summary-mode-map
    (kbd "<RET>") 'browse-nnrss-url))
(add-to-list 'nnmail-extra-headers nnrss-url-field)

```

### 6.4.7 Customizing w3

Gnus uses the url library to fetch web pages and Emacs/w3 to display web pages. Emacs/w3 is documented in its own manual, but there are some things that may be more relevant for Gnus users.

For instance, a common question is how to make Emacs/w3 follow links using the `browse-url` functions (which will call some external web browser like Netscape). Here's one way:

```

(eval-after-load "w3"
  '(progn
    (fset 'w3-fetch-orig (symbol-function 'w3-fetch))
    (defun w3-fetch (&optional url target)
      (interactive (list (w3-read-url-with-default)))
      (if (eq major-mode 'gnus-article-mode)
          (browse-url url)
          (w3-fetch-orig url target)))))

```

Put that in your `.emacs` file, and hitting links in w3-rendered HTML in the Gnus article buffers will use `browse-url` to follow the link.

## 6.5 IMAP

IMAP is a network protocol for reading mail (or news, or . . .), think of it as a modernized NNTP. Connecting to a IMAP server is much similar to connecting to a news server, you just specify the network address of the server.

IMAP has two properties. First, IMAP can do everything that POP can, it can hence be viewed as a POP++. Secondly, IMAP is a mail storage protocol, similar to NNTP being a news storage protocol—however, IMAP offers more features than NNTP because news is more or less read-only whereas mail is read-write.

If you want to use IMAP as a POP++, use an `imap` entry in `mail-sources`. With this, Gnus will fetch mails from the IMAP server and store them on the local disk. This is not the usage described in this section—See [\[Mail Sources\]](#), page [\[undefined\]](#).

If you want to use IMAP as a mail storage protocol, use an `nnimap` entry in `gnus-secondary-select-methods`. With this, Gnus will manipulate mails stored on the IMAP server. This is the kind of usage explained in this section.

A server configuration in ‘`~/.gnus.el`’ with a few IMAP servers might look something like the following. (Note that for TLS/SSL, you need external programs and libraries, see below.)

```
(setq gnus-secondary-select-methods
      '((nnimap "simpleserver") ; no special configuration
        ; perhaps a ssh port forwarded server:
        (nnimap "dolk"
                  (nnimap-address "localhost")
                  (nnimap-server-port 1430))
        ; a UW server running on localhost
        (nnimap "barbar"
                  (nnimap-server-port 143)
                  (nnimap-address "localhost")
                  (nnimap-list-pattern ("INBOX" "mail/*")))
        ; anonymous public cyrus server:
        (nnimap "cyrus.andrew.cmu.edu"
                  (nnimap-authenticator anonymous)
                  (nnimap-list-pattern "archive.*")
                  (nnimap-stream network))
        ; a ssl server on a non-standard port:
        (nnimap "vic20"
                  (nnimap-address "vic20.somewhere.com")
                  (nnimap-server-port 9930)
                  (nnimap-stream ssl))))
```

After defining the new server, you can subscribe to groups on the server using normal Gnus commands such as *U* in the Group Buffer (see [\[Subscription Commands\]](#), page [\[undefined\]](#)) or via the Server Buffer (see [\[Server Buffer\]](#), page [\[undefined\]](#)).

The following variables can be used to create a virtual `nnimap` server:

#### `nnimap-address`

The address of the remote IMAP server. Defaults to the virtual server name if not specified.

#### `nnimap-server-port`

Port on server to contact. Defaults to port 143, or 993 for TLS/SSL.

Note that this should be an integer, example server specification:

```
(nnimap "mail.server.com"
      (nnimap-server-port 4711))
```

#### `nnimap-list-pattern`

String or list of strings of mailboxes to limit available groups to. This is used when the server has very many mailboxes and you’re only interested in a few—some servers export your home directory via IMAP, you’ll probably want to limit the mailboxes to those in ‘`~/Mail/*`’ then.

The string can also be a cons of REFERENCE and the string as above, what REFERENCE is used for is server specific, but on the University of Washington server it’s a directory that will be concatenated with the mailbox.

Example server specification:

```
(nnimap "mail.server.com"
  (nnimap-list-pattern ("INBOX" "Mail/*" "alt.sex.*"
    ("~friend/Mail/" . "list/*"))))
```

#### **nnimap-stream**

The type of stream used to connect to your server. By default, nnimap will detect and automatically use all of the below, with the exception of TLS/SSL. (IMAP over TLS/SSL is being replaced by STARTTLS, which can be automatically detected, but it's not widely deployed yet.)

Example server specification:

```
(nnimap "mail.server.com"
  (nnimap-stream ssl))
```

Please note that the value of `nnimap-stream` is a symbol!

- *gssapi*: Connect with GSSAPI (usually Kerberos 5). Requires the `'gsasl'` or `'imtest'` program.
- *kerberos4*: Connect with Kerberos 4. Requires the `'imtest'` program.
- *starttls*: Connect via the STARTTLS extension (similar to TLS/SSL). Requires the external library `'starttls.el'` and program `'starttls'`.
- *tls*: Connect through TLS. Requires GNUTLS (the program `'gnutls-cli'`).
- *ssl*: Connect through SSL. Requires OpenSSL (the program `'openssl'`) or SSLeay (`'s_client'`).
- *shell*: Use a shell command to start IMAP connection.
- *network*: Plain, TCP/IP network connection.

The `'imtest'` program is shipped with Cyrus IMAPD. If you're using `'imtest'` from Cyrus IMAPD < 2.0.14 (which includes version 1.5.x and 1.6.x) you need to frob `imap-process-connection-type` to make `imap.el` use a pty instead of a pipe when communicating with `'imtest'`. You will then suffer from a line length restrictions on IMAP commands, which might make Gnus seem to hang indefinitely if you have many articles in a mailbox. The variable `imap-kerberos4-program` contain parameters to pass to the `imtest` program.

For TLS connection, the `gnutls-cli` program from GNUTLS is needed. It is available from <http://www.gnu.org/software/gnutls/>.

This parameter specifies a list of command lines that invoke a GSSAPI authenticated IMAP stream in a subshell. They are tried sequentially until a connection is made, or the list has been exhausted. By default, `'gsasl'` from GNU SASL, available from <http://www.gnu.org/software/gsas1/>, and the `'imtest'` program from Cyrus IMAPD (see `imap-kerberos4-program`), are tried.

For SSL connections, the OpenSSL program is available from <http://www.openssl.org/>. OpenSSL was formerly known as SSLeay, and nnimap support it too—although the most recent versions of SSLeay, 0.9.x, are known to have serious bugs making it useless. Earlier versions, especially 0.8.x, of SSLeay are known to work. The variable `imap-ssl-program` contain parameters to pass to OpenSSL/SSLeay.

For IMAP connections using the `shell` stream, the variable `imap-shell-program` specify what program to call.

#### `nnimap-authenticator`

The authenticator used to connect to the server. By default, `nnimap` will use the most secure authenticator your server is capable of.

Example server specification:

```
(nnimap "mail.server.com"
      (nnimap-authenticator anonymous))
```

Please note that the value of `nnimap-authenticator` is a symbol!

- *gssapi*: GSSAPI (usually kerberos 5) authentication. Requires external program `gsasl` or `imtest`.
- *kerberos4*: Kerberos 4 authentication. Requires external program `imtest`.
- *digest-md5*: Encrypted username/password via DIGEST-MD5. Requires external library `digest-md5.el`.
- *cram-md5*: Encrypted username/password via CRAM-MD5.
- *login*: Plain-text username/password via LOGIN.
- *anonymous*: Login as “anonymous”, supplying your email address as password.

#### `nnimap-expunge-on-close`

Unlike *Parmenides* the IMAP designers have decided things that don’t exist actually do exist. More specifically, IMAP has this concept of marking articles **Deleted** which doesn’t actually delete them, and this (marking them **Deleted**, that is) is what `nnimap` does when you delete an article in Gnus (with *B DEL* or similar).

Since the articles aren’t really removed when we mark them with the **Deleted** flag we’ll need a way to actually delete them. Feel like running in circles yet?

Traditionally, `nnimap` has removed all articles marked as **Deleted** when closing a mailbox but this is now configurable by this server variable.

The possible options are:

- |               |  |
|---------------|--|
| <b>always</b> | The default behavior, delete all articles marked as “Deleted” when closing a mailbox.  |
| <b>never</b>  | Never actually delete articles. Currently there is no way of showing the articles marked for deletion in <code>nnimap</code> , but other IMAP clients may allow you to do this. If you ever want to run the EXPUNGE command manually, See <a href="#">[Expunging mailboxes]</a> , page <a href="#">[Expunging mailboxes]</a> . |
| <b>ask</b>    | When closing mailboxes, <code>nnimap</code> will ask if you wish to expunge deleted articles or not.   |

#### `nnimap-importantize-dormant`

If non-`nil` (the default), marks dormant articles as ticked (as well), for other IMAP clients. Within Gnus, dormant articles will naturally still (only) be marked as dormant. This is to make dormant articles stand out, just like



ticked articles, in other IMAP clients. (In other words, Gnus has two “Tick” marks and IMAP has only one.)

Probably the only reason for frobing this would be if you’re trying enable per-user persistent dormant flags, using something like:

```
(setcdr (assq 'dormant nnimap-mark-to-flag-alist)
        (format "gnus-dormant-%s" (user-login-name)))
(setcdr (assq 'dormant nnimap-mark-to-predicate-alist)
        (format "KEYWORD gnus-dormant-%s" (user-login-name)))
```

In this case, you would not want the per-user dormant flag showing up as ticked for other users.

#### **nnimap-expunge-search-string**

This variable contain the IMAP search command sent to server when searching for articles eligible for expiring. The default is "UID %s NOT SINCE %s", where the first %s is replaced by UID set and the second %s is replaced by a date.

Probably the only useful value to change this to is "UID %s NOT SENT SINCE %s", which makes nnimap use the Date: in messages instead of the internal article date. See section 6.4.4 of RFC 2060 for more information on valid strings.

#### **nnimap-authinfo-file**

A file containing credentials used to log in on servers. The format is (almost) the same as the ftp ‘~/.netrc’ file. See the variable **nnntp-authinfo-file** for exact syntax; also see [\[NNTP\]](#), page [\[undefined\]](#). An example of an .authinfo line for an IMAP server, is:

```
machine students.uio.no login larsi password geheimnis port imap
```

Note that it should be **port imap**, or **port 143**, if you use a **nnimap-stream** of **tls** or **ssl**, even if the actual port number used is port 993 for secured IMAP. For convenience, Gnus will accept **port imaps** as a synonym of **port imap**.

#### **nnimap-need-unselect-to-notice-new-mail**

Unselect mailboxes before looking for new mail in them. Some servers seem to need this under some circumstances; it was reported that Courier 1.7.1 did.

### **6.5.1 Splitting in IMAP**

Splitting is something Gnus users have loved and used for years, and now the rest of the world is catching up. Yeah, dream on, not many IMAP servers have server side splitting and those that have splitting seem to use some non-standard protocol. This means that IMAP support for Gnus has to do its own splitting.

And it does.

(Incidentally, people seem to have been dreaming on, and Sieve has gaining a market share and is supported by several IMAP servers. Fortunately, Gnus support it too, See [\[Sieve Commands\]](#), page [\[undefined\]](#).)

Here are the variables of interest:

#### **nnimap-split-crosspost**

If non-**nil**, do crossposting if several split methods match the mail. If **nil**, the first match in **nnimap-split-rule** found will be used.



Nnmail equivalent: `nnmail-crosspost`.

#### `nnimap-split-inbox`

A string or a list of strings that gives the name(s) of IMAP mailboxes to split from. Defaults to `nil`, which means that splitting is disabled!

```
(setq nnimap-split-inbox
      '("INBOX" ("~/friend/Mail" . "lists/*") "lists.imap"))
```

No `nnmail` equivalent.

#### `nnimap-split-rule`

New mail found in `nnimap-split-inbox` will be split according to this variable. This variable contains a list of lists, where the first element in the sublist gives the name of the IMAP mailbox to move articles matching the regexp in the second element in the sublist. Got that? Neither did I, we need examples.

```
(setq nnimap-split-rule
      '(("INBOX.nnimap"
         "^Sender: owner-nnimap@vic20.globalcom.se")
        ("INBOX.junk"    "^Subject:.*MAKE MONEY")
        ("INBOX.private" "")))
```

This will put all articles from the `nnimap` mailing list into mailbox `INBOX.nnimap`, all articles containing `MAKE MONEY` in the `Subject:` line into `INBOX.junk` and everything else in `INBOX.private`.

The first string may contain ‘`\1`’ forms, like the ones used by `replace-match` to insert sub-expressions from the matched text. For instance:

```
("INBOX.lists.\1"      "^Sender: owner-\\([a-z-]+\\)@")
```

The first element can also be the symbol `junk` to indicate that matching messages should simply be deleted. Use with care.

The second element can also be a function. In that case, it will be called with the first element of the rule as the argument, in a buffer containing the headers of the article. It should return a non-`nil` value if it thinks that the mail belongs in that group.

Nnmail users might recollect that the last regexp had to be empty to match all articles (like in the example above). This is not required in `nnimap`. Articles not matching any of the regexps will not be moved out of your inbox. (This might affect performance if you keep lots of unread articles in your inbox, since the splitting code would go over them every time you fetch new mail.)

These rules are processed from the beginning of the alist toward the end. The first rule to make a match will “win”, unless you have crossposting enabled. In that case, all matching rules will “win”.

This variable can also have a function as its value, the function will be called with the headers narrowed and should return a group where it thinks the article should be split to. See `nnimap-split-fancy`.

The splitting code tries to create mailboxes if it needs to.

To allow for different split rules on different virtual servers, and even different split rules in different inboxes on the same server, the syntax of this variable have been extended along the lines of:

```
(setq nnimap-split-rule
      '(("my1server"      (".*" (("ding"      "ding@gnus.org")
                                   ("junk"      "From:.*Simon"))))
        ("my2server"      ("INBOX" nnimap-split-fancy))
        ("my[34]server"   (".*" (("private"   "To:.*Simon")
                                   ("junk"      my-junk-func))))))
```

The virtual server name is in fact a regexp, so that the same rules may apply to several servers. In the example, the servers `my3server` and `my4server` both use the same rules. Similarly, the inbox string is also a regexp. The actual splitting rules are as before, either a function, or a list with group/regexp or group/function elements.

Nnmail equivalent: `nnmail-split-methods`.

#### `nnimap-split-predicate`

Mail matching this predicate in `nnimap-split-inbox` will be split, it is a string and the default is `'UNSEEN UNDELETED'`.

This might be useful if you use another IMAP client to read mail in your inbox but would like Gnus to split all articles in the inbox regardless of readedness. Then you might change this to `'UNDELETED'`.

#### `nnimap-split-fancy`

It's possible to set `nnimap-split-rule` to `nnmail-split-fancy` if you want to use fancy splitting. See [\[Fancy Mail Splitting\]](#), page [\[Fancy Mail Splitting\]](#).

However, to be able to have different fancy split rules for nnmail and nnimap back ends you can set `nnimap-split-rule` to `nnimap-split-fancy` and define the nnimap specific fancy split rule in `nnimap-split-fancy`.

Example:

```
(setq nnimap-split-rule 'nnimap-split-fancy
      nnimap-split-fancy ...)
```

Nnmail equivalent: `nnmail-split-fancy`.

#### `nnimap-split-download-body`

Set to non-`nil` to download entire articles during splitting. This is generally not required, and will slow things down considerably. You may need it if you want to use an advanced splitting function that analyses the body to split the article.

## 6.5.2 Expiring in IMAP

Even though `nnimap` is not a proper `nnmail` derived back end, it supports most features in regular expiring (see [\[Expiring Mail\]](#), page [\[Expiring Mail\]](#)). Unlike splitting in IMAP (see [\[Splitting in IMAP\]](#), page [\[Splitting in IMAP\]](#)) it does not clone the `nnmail` variables (i.e., creating `nnimap-expiry-wait`) but reuse the `nnmail` variables. What follows below are the variables used by the `nnimap` expiry process.

A note on how the expire mark is stored on the IMAP server is appropriate here as well. The expire mark is translated into a `imap` client specific mark, `gnus-expire`, and stored on the message. This means that likely only Gnus will understand and treat the `gnus-expire` mark properly, although other clients may allow you to view client specific flags on the

message. It also means that your server must support permanent storage of client specific flags on messages. Most do, fortunately.

`nnmail-expiry-wait`

`nnmail-expiry-wait-function`

These variables are fully supported. The expire value can be a number, the symbol `immediate` or `never`.

`nnmail-expiry-target`

This variable is supported, and internally implemented by calling the `nnmail` functions that handle this. It contains an optimization that if the destination is a IMAP group on the same server, the article is copied instead of appended (that is, uploaded again).

### 6.5.3 Editing IMAP ACLs

ACL stands for Access Control List. ACLs are used in IMAP for limiting (or enabling) other users access to your mail boxes. Not all IMAP servers support this, this function will give an error if it doesn't.

To edit an ACL for a mailbox, type `G 1` (`gnus-group-edit-nnimap-acl`) and you'll be presented with an ACL editing window with detailed instructions.

Some possible uses:

- Giving "anyone" the "lrs" rights (lookup, read, keep seen/unseen flags) on your mailing list mailboxes enables other users on the same server to follow the list without subscribing to it.
- At least with the Cyrus server, you are required to give the user "anyone" posting ("p") capabilities to have "plussing" work (that is, mail sent to `user+mailbox@domain` ending up in the IMAP mailbox `INBOX.mailbox`).

### 6.5.4 Expunging mailboxes

If you're using the `never` setting of `nnimap-expunge-on-close`, you may want the option of expunging all deleted articles in a mailbox manually. This is exactly what `G x` does.

Currently there is no way of showing deleted articles, you can just delete them.

### 6.5.5 A note on namespaces

The IMAP protocol has a concept called namespaces, described by the following text in the RFC:

#### 5.1.2. Mailbox Namespace Naming Convention

By convention, the first hierarchical element of any mailbox name which begins with `"#"` identifies the "namespace" of the remainder of the name. This makes it possible to disambiguate between different types of mailbox stores, each of which have their own namespaces.

For example, implementations which offer access to USENET newsgroups MAY use the `"#news"` namespace to partition the USENET newsgroup namespace from that of other mailboxes. Thus, the

comp.mail.misc newsgroup would have an mailbox name of "#news.comp.mail.misc", and the name "comp.mail.misc" could refer to a different object (e.g. a user's private mailbox).

While there is nothing in this text that warrants concern for the IMAP implementation in Gnus, some servers use namespace prefixes in a way that does not work with how Gnus uses mailbox names.

Specifically, University of Washington's IMAP server uses mailbox names like `#driver.mbx/read-mail` which are valid only in the CREATE and APPEND commands. After the mailbox is created (or a messages is appended to a mailbox), it must be accessed without the namespace prefix, i.e. `read-mail`. Since Gnus do not make it possible for the user to guarantee that user entered mailbox names will only be used with the CREATE and APPEND commands, you should simply not use the namespace prefixed mailbox names in Gnus.

See the UoW IMAPD documentation for the `#driver.*/` prefix for more information on how to use the prefixes. They are a power tool and should be used only if you are sure what the effects are.

### 6.5.6 Debugging IMAP

IMAP is a complex protocol, more so than NNTP or POP3. Implementation bugs are not unlikely, and we do our best to fix them right away. If you encounter odd behaviour, chances are that either the server or Gnus is buggy.

If you are familiar with network protocols in general, you will probably be able to extract some clues from the protocol dump of the exchanges between Gnus and the server. Even if you are not familiar with network protocols, when you include the protocol dump in IMAP-related bug reports you are helping us with data critical to solving the problem. Therefore, we strongly encourage you to include the protocol dump when reporting IMAP bugs in Gnus.

Because the protocol dump, when enabled, generates lots of data, it is disabled by default. You can enable it by setting `imap-log` as follows:

```
(setq imap-log t)
```

This instructs the `imap.el` package to log any exchanges with the server. The log is stored in the buffer `*imap-log*`. Look for error messages, which sometimes are tagged with the keyword BAD—but when submitting a bug, make sure to include all the data.

## 6.6 Other Sources

Gnus can do more than just read news or mail. The methods described below allow Gnus to view directories and files as if they were newsgroups.

### 6.6.1 Directory Groups

If you have a directory that has lots of articles in separate files in it, you might treat it as a newsgroup. The files have to have numerical names, of course.

This might be an opportune moment to mention `ange-ftp` (and its successor `efs`), that most wonderful of all wonderful Emacs packages. When I wrote `nndir`, I didn't think much about it—a back end to read directories. Big deal.

**ange-ftp** changes that picture dramatically. For instance, if you enter the **ange-ftp** file name `‘/ftp.hpc.uh.edu:/pub/emacs/ding-list/’` as the directory name, **ange-ftp** or **efs** will actually allow you to read this directory over at `‘sina’` as a newsgroup. Distributed news ahoy!

**nndir** will use NOV files if they are present.

**nndir** is a “read-only” back end—you can’t delete or expire articles with this method. You can use **nnmh** or **nnml** for whatever you use **nndir** for, so you could switch to any of those methods if you feel the need to have a non-read-only **nndir**.

## 6.6.2 Anything Groups

From the **nndir** back end (which reads a single spool-like directory), it’s just a hop and a skip to **nneething**, which pretends that any arbitrary directory is a newsgroup. Strange, but true.

When **nneething** is presented with a directory, it will scan this directory and assign article numbers to each file. When you enter such a group, **nneething** must create “headers” that Gnus can use. After all, Gnus is a newsreader, in case you’re forgetting. **nneething** does this in a two-step process. First, it snoops each file in question. If the file looks like an article (i.e., the first few lines look like headers), it will use this as the head. If this is just some arbitrary file without a head (e.g. a C source file), **nneething** will cobble up a header out of thin air. It will use file ownership, name and date and do whatever it can with these elements.

All this should happen automatically for you, and you will be presented with something that looks very much like a newsgroup. Totally like a newsgroup, to be precise. If you select an article, it will be displayed in the article buffer, just as usual.

If you select a line that represents a directory, Gnus will pop you into a new summary buffer for this **nneething** group. And so on. You can traverse the entire disk this way, if you feel like, but remember that Gnus is not dired, really, and does not intend to be, either.

There are two overall modes to this action—ephemeral or solid. When doing the ephemeral thing (i.e., **G D** from the group buffer), Gnus will not store information on what files you have read, and what files are new, and so on. If you create a solid **nneething** group the normal way with **G m**, Gnus will store a mapping table between article numbers and file names, and you can treat this group like any other groups. When you activate a solid **nneething** group, you will be told how many unread articles it contains, etc., etc.

Some variables:

### **nneething-map-file-directory**

All the mapping files for solid **nneething** groups will be stored in this directory, which defaults to `‘~/nneething/’`.

### **nneething-exclude-files**

All files that match this regexp will be ignored. Nice to use to exclude auto-save files and the like, which is what it does by default.

### **nneething-include-files**

Regexp saying what files to include in the group. If this variable is non-**nil**, only files matching this regexp will be included.

**nneething-map-file**  
Name of the map files.

### 6.6.3 Document Groups

**ndoc** is a cute little thing that will let you read a single file as a newsgroup. Several files types are supported:

**babyl**        The Babyl (Rmail) mail box.  
**mbox**        The standard Unix mbox file.  
**mmdf**        The MMDF mail box format.  
**news**        Several news articles appended into a file.  
**rnews**        The rnews batch transport format.  
**forward**     Forwarded articles.  
**nsmail**      Netscape mail boxes.  
**mime-parts**  
              MIME multipart messages.  
**standard-digest**  
              The standard (RFC 1153) digest format.  
**mime-digest**  
              A MIME digest of messages.  
**lanl-gov-announce**  
              Announcement messages from LANL Gov Announce.  
**rfc822-forward**  
              A message forwarded according to RFC822.  
**outlook**     The Outlook mail box.  
**oe-dbx**      The Outlook Express dbx mail box.  
**exim-bounce**  
              A bounce message from the Exim MTA.  
**forward**     A message forwarded according to informal rules.  
**rfc934**      An RFC934-forwarded message.  
**mailman**     A mailman digest.  
**clari-briefs**  
              A digest of Clarinet brief news items.  
**slack-digest**  
              Non-standard digest format—matches most things, but does it badly.  
**mail-in-mail**  
              The last resort.

You can also use the special “file type” `guess`, which means that `nndoc` will try to guess what file type it is looking at. `digest` means that `nndoc` should guess what digest type the file is.

`nndoc` will not try to change the file or insert any extra headers into it—it will simply, like, let you use the file as the basis for a group. And that’s it.

If you have some old archived articles that you want to insert into your new & spiffy Gnus mail back end, `nndoc` can probably help you with that. Say you have an old ‘RMAIL’ file with mail that you now want to split into your new `nnml` groups. You look at that file using `nndoc` (using the `G f` command in the group buffer (see [\[Foreign Groups\]](#), page [\[undefined\]](#))), set the process mark on all the articles in the buffer (`M P b`, for instance), and then re-spool (`B r`) using `nnml`. If all goes well, all the mail in the ‘RMAIL’ file is now also stored in lots of `nnml` directories, and you can delete that pesky ‘RMAIL’ file. If you have the guts!

Virtual server variables:

#### `nndoc-article-type`

This should be one of `mbox`, `babyl`, `digest`, `news`, `rnews`, `mmdf`, `forward`, `rfc934`, `rfc822-forward`, `mime-parts`, `standard-digest`, `slack-digest`, `clari-briefs`, `nsmail`, `outlook`, `oe-dbx`, `mailman`, and `mail-in-mail` or `guess`.

#### `nndoc-post-type`

This variable says whether Gnus is to consider the group a news group or a mail group. There are two valid values: `mail` (the default) and `news`.

### 6.6.3.1 Document Server Internals

Adding new document types to be recognized by `nndoc` isn’t difficult. You just have to whip up a definition of what the document looks like, write a predicate function to recognize that document type, and then hook into `nndoc`.

First, here’s an example document type definition:

```
(mmdf
 (article-begin . "^\\^A\\^A\\^A\\^A\\n")
 (body-end . "^\\^A\\^A\\^A\\^A\\n"))
```

The definition is simply a unique *name* followed by a series of regexp pseudo-variable settings. Below are the possible variables—don’t be daunted by the number of variables; most document types can be defined with very few settings:

#### `first-article`

If present, `nndoc` will skip past all text until it finds something that match this regexp. All text before this will be totally ignored.

#### `article-begin`

This setting has to be present in all document type definitions. It says what the beginning of each article looks like.

#### `head-begin-function`

If present, this should be a function that moves point to the head of the article.

**nndoc-head-begin**

If present, this should be a regexp that matches the head of the article.

**nndoc-head-end**

This should match the end of the head of the article. It defaults to ‘`^$`’—the empty line.

**body-begin-function**

If present, this function should move point to the beginning of the body of the article.

**body-begin**

This should match the beginning of the body of the article. It defaults to ‘`^\n`’.

**body-end-function**

If present, this function should move point to the end of the body of the article.

**body-end** If present, this should match the end of the body of the article.

**file-end** If present, this should match the end of the file. All text after this regexp will be totally ignored.

So, using these variables **nndoc** is able to dissect a document file into a series of articles, each with a head and a body. However, a few more variables are needed since not all document types are all that news-like—variables needed to transform the head or the body into something that’s palatable for Gnus:

**prepare-body-function**

If present, this function will be called when requesting an article. It will be called with point at the start of the body, and is useful if the document has encoded some parts of its contents.

**article-transform-function**

If present, this function is called when requesting an article. It’s meant to be used for more wide-ranging transformation of both head and body of the article.

**generate-head-function**

If present, this function is called to generate a head that Gnus can understand. It is called with the article number as a parameter, and is expected to generate a nice head for the article in question. It is called when requesting the headers of all articles.

Let’s look at the most complicated example I can come up with—standard digests:

```
(standard-digest
 (first-article . ,(concat "^" (make-string 70 ?-) "\n\n+"))
 (article-begin . ,(concat "\n\n" (make-string 30 ?-) "\n\n+"))
 (prepare-body-function . nndoc-unquote-dashes)
 (body-end-function . nndoc-digest-body-end)
 (head-end . "^ ?$")
 (body-begin . "^ ?\n")
 (file-end . "^End of .*digest.*[0-9].*\n\\*\\*\\*\\|^End of.*Digest *$")
 (subtype digest guess))
```



We see that all text before a 70-width line of dashes is ignored; all text after a line that starts with that ‘`^End of`’ is also ignored; each article begins with a 30-width line of dashes; the line separating the head from the body may contain a single space; and that the body is run through `nndoc-unquote-dashes` before being delivered.

To hook your own document definition into `nndoc`, use the `nndoc-add-type` function. It takes two parameters—the first is the definition itself and the second (optional) parameter says where in the document type definition alist to put this definition. The alist is traversed sequentially, and `nndoc-type-type-p` is called for a given type *type*. So `nndoc-mmdf-type-p` is called to see whether a document is of `mmdf` type, and so on. These type predicates should return `nil` if the document is not of the correct type; `t` if it is of the correct type; and a number if the document might be of the correct type. A high number means high probability; a low number means low probability with ‘0’ being the lowest valid number.

### 6.6.4 SOUP

In the PC world people often talk about “offline” newsreaders. These are thingies that are combined reader/news transport monstrosities. With built-in modem programs. Yecchh!

Of course, us Unix Weenie types of human beans use things like `uucp` and, like, `nntpd` and set up proper news and mail transport things like Ghod intended. And then we just use normal newsreaders.

However, it can sometimes be convenient to do something that’s a bit easier on the brain if you have a very slow modem, and you’re not really that interested in doing things properly.

A file format called SOUP has been developed for transporting news and mail from servers to home machines and back again. It can be a bit fiddly.

First some terminology:

*server*            This is the machine that is connected to the outside world and where you get news and/or mail from.

*home machine*    This is the machine that you want to do the actual reading and responding on. It is typically not connected to the rest of the world in any way.

*packet*            Something that contains messages and/or commands. There are two kinds of packets:

*message packets*

These are packets made at the server, and typically contain lots of messages for you to read. These are called ‘`SoupoutX.tgz`’ by default, where *x* is a number.

*response packets*

These are packets made at the home machine, and typically contains replies that you’ve written. These are called ‘`SoupinX.tgz`’ by default, where *x* is a number.

1. You log in on the server and create a SOUP packet. You can either use a dedicated SOUP thingie (like the `awk` program), or you can use Gnus to create the packet with its SOUP commands (`O s` and/or `G s b`; and then `G s p`) (see [\[SOUP Commands\]](#), [page `<undefined>`](#)).

2. You transfer the packet home. Rail, boat, car or modem will do fine.
3. You put the packet in your home directory.
4. You fire up Gnus on your home machine using the `nnsoup` back end as the native or secondary server.
5. You read articles and mail and answer and followup to the things you want (see [\[SOUP Replies\]](#), page [\[undefined\]](#)).
6. You do the `G s r` command to pack these replies into a SOUP packet.
7. You transfer this packet to the server.
8. You use Gnus to mail this packet out with the `G s s` command.
9. You then repeat until you die.

So you basically have a bipartite system—you use `nnsoup` for reading and Gnus for packing/sending these SOUP packets.

#### 6.6.4.1 SOUP Commands

These are commands for creating and manipulating SOUP packets.

<code>G s b</code>	Pack all unread articles in the current group ( <code>gnus-group-brew-soup</code> ). This command understands the process/prefix convention.
<code>G s w</code>	Save all SOUP data files ( <code>gnus-soup-save-areas</code> ).
<code>G s s</code>	Send all replies from the replies packet ( <code>gnus-soup-send-replies</code> ).
<code>G s p</code>	Pack all files into a SOUP packet ( <code>gnus-soup-pack-packet</code> ).
<code>G s r</code>	Pack all replies into a replies packet ( <code>nnsoup-pack-replies</code> ).
<code>O s</code>	This summary-mode command adds the current article to a SOUP packet ( <code>gnus-soup-add-article</code> ). It understands the process/prefix convention (see <a href="#">[Process/Prefix]</a> , page <a href="#">[undefined]</a> ).

There are a few variables to customize where Gnus will put all these thingies:

<code>gnus-soup-directory</code>	Directory where Gnus will save intermediate files while composing SOUP packets. The default is <code>~/SoupBrew/</code> .
<code>gnus-soup-replies-directory</code>	This is what Gnus will use as a temporary directory while sending our reply packets. <code>~/SoupBrew/SoupReplies/</code> is the default.
<code>gnus-soup-prefix-file</code>	Name of the file where Gnus stores the last used prefix. The default is <code>'gnus-prefix'</code> .
<code>gnus-soup-packer</code>	A format string command for packing a SOUP packet. The default is <code>'tar cf - %s   gzip &gt; \$HOME/Soupout%d.tgz'</code> .
<code>gnus-soup-unpacker</code>	Format string command for unpacking a SOUP packet. The default is <code>'gunzip -c %s   tar xvf -'</code> .

**gnus-soup-packet-directory**

Where Gnus will look for reply packets. The default is ‘~/’.

**gnus-soup-packet-regexp**

Regular expression matching SOUP reply packets in **gnus-soup-packet-directory**.

**6.6.4.2 SOUP Groups**

**nnsoup** is the back end for reading SOUP packets. It will read incoming packets, unpack them, and put them in a directory where you can read them at leisure.

These are the variables you can use to customize its behavior:

**nnsoup-tmp-directory**

When **nnsoup** unpacks a SOUP packet, it does it in this directory. (‘/tmp/’ by default.)

**nnsoup-directory**

**nnsoup** then moves each message and index file to this directory. The default is ‘~/SOUP/’.

**nnsoup-replies-directory**

All replies will be stored in this directory before being packed into a reply packet. The default is ‘~/SOUP/replies/’.

**nnsoup-replies-format-type**

The SOUP format of the replies packets. The default is ‘?n’ (rnews), and I don’t think you should touch that variable. I probably shouldn’t even have documented it. Drats! Too late!

**nnsoup-replies-index-type**

The index type of the replies packet. The default is ‘?n’, which means “none”. Don’t fiddle with this one either!

**nnsoup-active-file**

Where **nnsoup** stores lots of information. This is not an “active file” in the **nntp** sense; it’s an Emacs Lisp file. If you lose this file or mess it up in any way, you’re dead. The default is ‘~/SOUP/active’.

**nnsoup-packer**

Format string command for packing a reply SOUP packet. The default is ‘tar cf - %s | gzip > \$HOME/Soupin%d.tgz’.

**nnsoup-unpacker**

Format string command for unpacking incoming SOUP packets. The default is ‘gunzip -c %s | tar xvf -’.

**nnsoup-packet-directory**

Where **nnsoup** will look for incoming packets. The default is ‘~/’.

**nnsoup-packet-regexp**

Regular expression matching incoming SOUP packets. The default is ‘Soupout’.

**nnsoup-always-save**

If non-**nil**, save the replies buffer after each posted message.

### 6.6.4.3 SOUP Replies

Just using `nnsoup` won't mean that your postings and mailings end up in SOUP reply packets automatically. You have to work a bit more for that to happen.

The `nnsoup-set-variables` command will set the appropriate variables to ensure that all your followups and replies end up in the SOUP system.

In specific, this is what it does:

```
(setq message-send-news-function 'nnsoup-request-post)
(setq message-send-mail-function 'nnsoup-request-mail)
```

And that's it, really. If you only want news to go into the SOUP system you just use the first line. If you only want mail to be SOUPed you use the second.

### 6.6.5 Mail-To-News Gateways

If your local `nntp` server doesn't allow posting, for some reason or other, you can post using one of the numerous mail-to-news gateways. The `nngateway` back end provides the interface.

Note that you can't read anything from this back end—it can only be used to post with.

Server variables:

`nngateway-address`

This is the address of the mail-to-news gateway.

`nngateway-header-transformation`

News headers often have to be transformed in some odd way or other for the mail-to-news gateway to accept it. This variable says what transformation should be called, and defaults to `nngateway-simple-header-transformation`. The function is called narrowed to the headers to be transformed and with one parameter—the gateway address.

This default function just inserts a new `To` header based on the `Newsgroups` header and the gateway address. For instance, an article with this `Newsgroups` header:

```
Newsgroups: alt.religion.emacs
```

will get this `To` header inserted:

```
To: alt-religion-emacs@GATEWAY
```

The following pre-defined functions exist:

`nngateway-simple-header-transformation`

Creates a `To` header that looks like `newsgroup@nngateway-address`.

`nngateway-mail2news-header-transformation`

Creates a `To` header that looks like `nngateway-address`.

Here's an example:

```
(setq gnus-post-method
      '(nngateway
        "mail2news@replay.com")
```

```
(nngateway-header-transformation
 nngateway-mail2news-header-transformation)))
```

So, to use this, simply say something like:

```
(setq gnus-post-method '(nngateway "GATEWAY.ADDRESS"))
```

## 6.7 Combined Groups

Gnus allows combining a mixture of all the other group types into bigger groups.

### 6.7.1 Virtual Groups

An *nnvirtual* group is really nothing more than a collection of other groups.

For instance, if you are tired of reading many small groups, you can put them all in one big group, and then grow tired of reading one big, unwieldy group. The joys of computing!

You specify **nnvirtual** as the method. The address should be a regexp to match component groups.

All marks in the virtual group will stick to the articles in the component groups. So if you tick an article in a virtual group, the article will also be ticked in the component group from whence it came. (And vice versa—marks from the component groups will also be shown in the virtual group.). To create an empty virtual group, run *G V* (**gnus-group-make-empty-virtual**) in the group buffer and edit the method regexp with *M-e* (**gnus-group-edit-group-method**)

Here's an example **nnvirtual** method that collects all Andrea Dworkin newsgroups into one, big, happy newsgroup:

```
(nnvirtual "^alt\\.fan\\.andrea-dworkin$\\|^rec\\.dworkin.*")
```

The component groups can be native or foreign; everything should work smoothly, but if your computer explodes, it was probably my fault.

Collecting the same group from several servers might actually be a good idea if users have set the Distribution header to limit distribution. If you would like to read `'soc.motss'` both from a server in Japan and a server in Norway, you could use the following as the group regexp:

```
"^nntp\\+server\\.jp:soc\\.motss$\\|^nntp\\+server\\.no:soc\\.motss$"
```

(Remember, though, that if you're creating the group with *G m*, you shouldn't double the backslashes, and you should leave off the quote characters at the beginning and the end of the string.)

This should work kinda smoothly—all articles from both groups should end up in this one, and there should be no duplicates. Threading (and the rest) will still work as usual, but there might be problems with the sequence of articles. Sorting on date might be an option here (see [\[Selecting a Group\]](#), page [\[undefined\]](#)).

One limitation, however—all groups included in a virtual group have to be alive (i.e., subscribed or unsubscribed). Killed or zombie groups can't be component groups for **nnvirtual** groups.

If the **nnvirtual-always-rescan** is non-nil, **nnvirtual** will always scan groups for unread articles when entering a virtual group. If this variable is nil (which is the default) and you read articles in a component group after the virtual group has been activated, the

read articles from the component group will show up when you enter the virtual group. You'll also see this effect if you have two virtual groups that have a component group in common. If that's the case, you should set this variable to `t`. Or you can just tap `M-g` on the virtual group every time before you enter it—it'll have much the same effect.

`nnvirtual` can have both mail and news groups as component groups. When responding to articles in `nnvirtual` groups, `nnvirtual` has to ask the back end of the component group the article comes from whether it is a news or mail back end. However, when you do a `^`, there is typically no sure way for the component back end to know this, and in that case `nnvirtual` tells Gnus that the article came from a not-news back end. (Just to be on the safe side.)

`C-c C-n` in the message buffer will insert the `Newsgroups` line from the article you respond to in these cases.

`nnvirtual` groups do not inherit anything but articles and marks from component groups—group parameters, for instance, are not inherited.

### 6.7.2 Kibozed Groups

*Kibozing* is defined by the OED as “grepping through (parts of) the news feed”. `nnkiboze` is a back end that will do this for you. Oh joy! Now you can grind any NNTP server down to a halt with useless requests! Oh happiness!

To create a kibozed group, use the `G k` command in the group buffer.

The address field of the `nnkiboze` method is, as with `nnvirtual`, a regexp to match groups to be “included” in the `nnkiboze` group. That's where most similarities between `nnkiboze` and `nnvirtual` end.

In addition to this regexp detailing component groups, an `nnkiboze` group must have a score file to say what articles are to be included in the group (see [\[Scoring\]](#), page [\[undefined\]](#)).

You must run `M-x nnkiboze-generate-groups` after creating the `nnkiboze` groups you want to have. This command will take time. Lots of time. Oodles and oodles of time. Gnus has to fetch the headers from all the articles in all the component groups and run them through the scoring process to determine if there are any articles in the groups that are to be part of the `nnkiboze` groups.

Please limit the number of component groups by using restrictive regexps. Otherwise your sysadmin may become annoyed with you, and the NNTP site may throw you off and never let you back in again. Stranger things have happened.

`nnkiboze` component groups do not have to be alive—they can be dead, and they can be foreign. No restrictions.

The generation of an `nnkiboze` group means writing two files in `nnkiboze-directory`, which is `~/News/kiboze/` by default. One contains the NOV header lines for all the articles in the group, and the other is an additional `.newsrsc` file to store information on what groups have been searched through to find component articles.

Articles marked as read in the `nnkiboze` group will have their NOV lines removed from the NOV file.

## 6.8 Gnus Unplugged

In olden times (ca. February '88), people used to run their newsreaders on big machines with permanent connections to the net. News transport was dealt with by news servers, and all the newsreaders had to do was to read news. Believe it or not.

Nowadays most people read news and mail at home, and use some sort of modem to connect to the net. To avoid running up huge phone bills, it would be nice to have a way to slurp down all the news and mail, hang up the phone, read for several hours, and then upload any responses you have to make. And then you repeat the procedure.

Of course, you can use news servers for doing this as well. I've used `inn` together with `slurp`, `pop` and `sendmail` for some years, but doing that's a bore. Moving the news server functionality up to the newsreader makes sense if you're the only person reading news on a machine.

Setting up Gnus as an “offline” newsreader is quite simple. In fact, you don't even have to configure anything.

Of course, to use it as such, you have to learn a few new commands.

### 6.8.1 Agent Basics

First, let's get some terminology out of the way.

The Gnus Agent is said to be *unplugged* when you have severed the connection to the net (and notified the Agent that this is the case). When the connection to the net is up again (and Gnus knows this), the Agent is *plugged*.

The *local* machine is the one you're running on, and which isn't connected to the net continuously.

*Downloading* means fetching things from the net to your local machine. *Uploading* is doing the opposite.

You know that Gnus gives you all the opportunity you'd ever want for shooting yourself in the foot. Some people call it flexibility. Gnus is also customizable to a great extent, which means that the user has a say on how Gnus behaves. Other newsreaders might unconditionally shoot you in your foot, but with Gnus, you have a choice!

Gnus is never really in plugged or unplugged state. Rather, it applies that state to each server individually. This means that some servers can be plugged while others can be unplugged. Additionally, some servers can be ignored by the Agent altogether (which means that they're kinda like plugged always).

So when you unplug the Agent and then wonder why is Gnus opening a connection to the Net, the next step to do is to look whether all servers are agentized. If there is an unagentized server, you found the culprit.

Another thing is the *offline* state. Sometimes, servers aren't reachable. When Gnus notices this, it asks you whether you want the server to be switched to offline state. If you say yes, then the server will behave somewhat as if it was unplugged, except that Gnus will ask you whether you want to switch it back online again.

Let's take a typical Gnus session using the Agent.

- You start Gnus with `gnus-unplugged`. This brings up the Gnus Agent in a disconnected state. You can read all the news that you have already fetched while in this mode.



- You then decide to see whether any new news has arrived. You connect your machine to the net (using PPP or whatever), and then hit *J j* to make Gnus become *plugged* and use *g* to check for new mail as usual. To check for new mail in unplugged mode (see [\[Mail Source Specifiers\]](#), page [\[undefined\]](#)).
- You can then read the new news immediately, or you can download the news onto your local machine. If you want to do the latter, you press *g* to check if there are any new news and then *J s* to fetch all the eligible articles in all the groups. (To let Gnus know which articles you want to download, see [\[Agent Categories\]](#), page [\[undefined\]](#)).
- After fetching the articles, you press *J j* to make Gnus become unplugged again, and you shut down the PPP thing (or whatever). And then you read the news offline.
- And then you go to step 2.

Here are some things you should do the first time (or so) that you use the Agent.

- Decide which servers should be covered by the Agent. If you have a mail back end, it would probably be nonsensical to have it covered by the Agent. Go to the server buffer (*^* in the group buffer) and press *J a* on the server (or servers) that you wish to have covered by the Agent (see [\[Server Agent Commands\]](#), page [\[undefined\]](#)), or *J r* on automatically added servers you do not wish to have covered by the Agent. By default, all `nntp` and `nnimap` servers in `gnus-select-method` and `gnus-secondary-select-methods` are agentized.
- Decide on download policy. It's fairly simple once you decide whether you are going to use agent categories, topic parameters, and/or group parameters to implement your policy. If you're new to gnus, it is probably best to start with a category, See [\[Agent Categories\]](#), page [\[undefined\]](#).  
Both topic parameters (see [\[Topic Parameters\]](#), page [\[undefined\]](#)) and agent categories (see [\[Agent Categories\]](#), page [\[undefined\]](#)) provide for setting a policy that applies to multiple groups. Which you use is entirely up to you. Topic parameters do override categories so, if you mix the two, you'll have to take that into account. If you have a few groups that deviate from your policy, you can use group parameters (see [\[Group Parameters\]](#), page [\[undefined\]](#)) to configure them.
- Uhm. . . that's it.

### 6.8.2 Agent Categories

One of the main reasons to integrate the news transport layer into the newsreader is to allow greater control over what articles to download. There's not much point in downloading huge amounts of articles, just to find out that you're not interested in reading any of them. It's better to be somewhat more conservative in choosing what to download, and then mark the articles for downloading manually if it should turn out that you're interested in the articles anyway.

One of the more effective methods for controlling what is to be downloaded is to create a *category* and then assign some (or all) groups to this category. Groups that do not belong in any other category belong to the `default` category. Gnus has its own buffer for creating and managing categories.

If you prefer, you can also use group parameters (see [\[Group Parameters\]](#), page [\[undefined\]](#)) and topic parameters (see [\[Topic Parameters\]](#), page [\[undefined\]](#)).



fined)) for an alternative approach to controlling the agent. The only real difference is that categories are specific to the agent (so there is less to learn) while group and topic parameters include the kitchen sink.

Since you can set agent parameters in several different places we have a rule to decide which source to believe. This rule specifies that the parameter sources are checked in the following order: group parameters, topic parameters, agent category, and finally customizable variables. So you can mix all of these sources to produce a wide range of behavior, just don't blame me if you don't remember where you put your settings.

### 6.8.2.1 Category Syntax

A category consists of a name, the list of groups belonging to the category, and a number of optional parameters that override the customizable variables. The complete list of agent parameters are listed below.

**gnus-agent-cat-name**

The name of the category.

**gnus-agent-cat-groups**

The list of groups that are in this category.

**gnus-agent-cat-predicate**

A predicate which (generally) gives a rough outline of which articles are eligible for downloading; and

**gnus-agent-cat-score-file**

a score rule which (generally) gives you a finer granularity when deciding what articles to download. (Note that this *download score* is not necessarily related to normal scores.)

**gnus-agent-cat-enable-expiration**

a boolean indicating whether the agent should expire old articles in this group. Most groups should be expired to conserve disk space. In fact, its probably safe to say that the gnus.\* hierarchy contains the only groups that should not be expired.

**gnus-agent-cat-days-until-old**

an integer indicating the number of days that the agent should wait before deciding that a read article is safe to expire.

**gnus-agent-cat-low-score**

an integer that overrides the value of **gnus-agent-low-score**.

**gnus-agent-cat-high-score**

an integer that overrides the value of **gnus-agent-high-score**.

**gnus-agent-cat-length-when-short**

an integer that overrides the value of **gnus-agent-short-article**.

**gnus-agent-cat-length-when-long**

an integer that overrides the value of **gnus-agent-long-article**.

**gnus-agent-cat-enable-undownloaded-faces**

a symbol indicating whether the summary buffer should display undownloaded articles using the `gnus-summary-*-undownloaded-face` faces. The symbol `nil` will disable the use of undownloaded faces while all other symbols enable them.

The name of a category can not be changed once the category has been created.

Each category maintains a list of groups that are exclusive members of that category. The exclusivity rule is automatically enforced, add a group to a new category and it is automatically removed from its old category.

A predicate in its simplest form can be a single predicate such as `true` or `false`. These two will download every available article or nothing respectively. In the case of these two special predicates an additional score rule is superfluous.

Predicates of `high` or `low` download articles in respect of their scores in relationship to `gnus-agent-high-score` and `gnus-agent-low-score` as described below.

To gain even finer control of what is to be regarded eligible for download a predicate can consist of a number of predicates with logical operators sprinkled in between.

Perhaps some examples are in order.

Here's a simple predicate. (It's the default predicate, in fact, used for all groups that don't belong to any other category.)

**short**

Quite simple, eh? This predicate is true if and only if the article is short (for some value of "short").

Here's a more complex predicate:

```
(or high
  (and
    (not low)
    (not long)))
```

This means that an article should be downloaded if it has a high score, or if the score is not low and the article is not long. You get the drift.

The available logical operators are `or`, `and` and `not`. (If you prefer, you can use the more "C"-ish operators `'|'`, `&` and `!` instead.)

The following predicates are pre-defined, but if none of these fit what you want to do, you can write your own.

When evaluating each of these predicates, the named constant will be bound to the value determined by calling `gnus-agent-find-parameter` on the appropriate parameter. For example, `gnus-agent-short-article` will be bound to `(gnus-agent-find-parameter group 'agent-short-article)`. This means that you can specify a predicate in your category then tune that predicate to individual groups.

<b>short</b>	True iff the article is shorter than <code>gnus-agent-short-article</code> lines; default 100.
<b>long</b>	True iff the article is longer than <code>gnus-agent-long-article</code> lines; default 200.
<b>low</b>	True iff the article has a download score less than <code>gnus-agent-low-score</code> ; default 0.

<b>high</b>	True iff the article has a download score greater than <b>gnus-agent-high-score</b> ; default 0.
<b>spam</b>	True iff the Gnus Agent guesses that the article is spam. The heuristics may change over time, but at present it just computes a checksum and sees whether articles match.
<b>true</b>	Always true.
<b>false</b>	Always false.

If you want to create your own predicate function, here's what you have to know: The functions are called with no parameters, but the **gnus-headers** and **gnus-score** dynamic variables are bound to useful values.

For example, you could decide that you don't want to download articles that were posted more than a certain number of days ago (e.g. posted more than **gnus-agent-expire-days** ago) you might write a function something along the lines of the following:

```
(defun my-article-old-p ()
  "Say whether an article is old."
  (< (time-to-days (date-to-time (mail-header-date gnus-headers)))
    (- (time-to-days (current-time)) gnus-agent-expire-days)))
```

with the predicate then defined as:

```
(not my-article-old-p)
```

or you could append your predicate to the predefined **gnus-category-predicate-alist** in your '~/.gnus.el' or wherever.

```
(require 'gnus-agent)
(setq gnus-category-predicate-alist
      (append gnus-category-predicate-alist
               '((old . my-article-old-p))))
```

and simply specify your predicate as:

```
(not old)
```

If/when using something like the above, be aware that there are many misconfigured systems/mailers out there and so an article's date is not always a reliable indication of when it was posted. Hell, some people just don't give a damn.

The above predicates apply to *all* the groups which belong to the category. However, if you wish to have a specific predicate for an individual group within a category, or you're just too lazy to set up a new category, you can enter a group's individual predicate in its group parameters like so:

```
(agent-predicate . short)
```

This is the group/topic parameter equivalent of the agent category default. Note that when specifying a single word predicate like this, the **agent-predicate** specification must be in dotted pair notation.

The equivalent of the longer example from above would be:

```
(agent-predicate or high (and (not low) (not long)))
```

The outer parenthesis required in the category specification are not entered here as, not being in dotted pair notation, the value of the predicate is assumed to be a list.

Now, the syntax of the download score is the same as the syntax of normal score files, except that all elements that require actually seeing the article itself are verboten. This means that only the following headers can be scored on: **Subject**, **From**, **Date**, **Message-ID**, **References**, **Chars**, **Lines**, and **Xref**.

As with predicates, the specification of the **download score rule** to use in respect of a group can be in either the category definition if it's to be applicable to all groups in therein, or a group's parameters if it's to be specific to that group.

In both of these places the **download score rule** can take one of three forms:

#### 1. Score rule

This has the same syntax as a normal Gnus score file except only a subset of scoring keywords are available as mentioned above.

example:

- Category specification
 

```
((("from"
      ("Lars Ingebrigtsen" 1000000 nil s))
      ("lines"
      (500 -100 nil <))))
```
- Group/Topic Parameter specification
 

```
(agent-score ("from"
      ("Lars Ingebrigtsen" 1000000 nil s))
      ("lines"
      (500 -100 nil <)))
```

Again, note the omission of the outermost parenthesis here.

#### 2. Agent score file

These score files must *only* contain the permitted scoring keywords stated above.

example:

- Category specification
 

```
("~/News/agent.SCORE")
```

 or perhaps
 

```
("~/News/agent.SCORE" "~/News/agent.group.SCORE")
```
- Group Parameter specification
 

```
(agent-score "~/News/agent.SCORE")
```

Additional score files can be specified as above. Need I say anything about parenthesis?

#### 3. Use normal score files

If you don't want to maintain two sets of scoring rules for a group, and your desired **downloading** criteria for a group are the same as your **reading** criteria then you can tell the agent to refer to your **normal** score files when deciding what to download.

These directives in either the category definition or a group's parameters will cause the agent to read in all the applicable score files for a group, *filtering out* those sections that do not relate to one of the permitted subset of scoring keywords.

- Category Specification

- file**  
 • Group Parameter specification  
 (agent-score . file)

### 6.8.2.2 Category Buffer

You'd normally do all category maintenance from the category buffer. When you enter it for the first time (with the `J c` command from the group buffer), you'll only see the **default** category.

The following commands are available in this buffer:

- q** Return to the group buffer (`gnus-category-exit`).
- e** Use a customization buffer to set all of the selected category's parameters at one time (`gnus-category-customize-category`).
- k** Kill the current category (`gnus-category-kill`).
- c** Copy the current category (`gnus-category-copy`).
- a** Add a new category (`gnus-category-add`).
- p** Edit the predicate of the current category (`gnus-category-edit-predicate`).
- g** Edit the list of groups belonging to the current category (`gnus-category-edit-groups`).
- s** Edit the download score rule of the current category (`gnus-category-edit-score`).
- l** List all the categories (`gnus-category-list`).

### 6.8.2.3 Category Variables

**gnus-category-mode-hook**

Hook run in category buffers.

**gnus-category-line-format**

Format of the lines in the category buffer (see [\[Formatting Variables\]](#), page [\[undefined\]](#)). Valid elements are:

- 'c' The name of the category.
- 'g' The number of groups in the category.

**gnus-category-mode-line-format**

Format of the category mode line (see [\[Mode Line Formatting\]](#), page [\[undefined\]](#)).

**gnus-agent-short-article**

Articles that have fewer lines than this are short. Default 100.

**gnus-agent-long-article**

Articles that have more lines than this are long. Default 200.

**gnus-agent-low-score**

Articles that have a score lower than this have a low score. Default 0.

**gnus-agent-high-score**

Articles that have a score higher than this have a high score. Default 0.

**gnus-agent-expire-days**

The number of days that a ‘read’ article must stay in the agent’s local disk before becoming eligible for expiration (While the name is the same, this doesn’t mean expiring the article on the server. It just means deleting the local copy of the article). What is also important to understand is that the counter starts with the time the article was written to the local disk and not the time the article was read. Default 7.

**gnus-agent-enable-expiration**

Determines whether articles in a group are, by default, expired or retained indefinitely. The default is **ENABLE** which means that you’ll have to disable expiration when desired. On the other hand, you could set this to **DISABLE**. In that case, you would then have to enable expiration in selected groups.

**6.8.3 Agent Commands**

All the Gnus Agent commands are on the *J* submap. The *J j* (**gnus-agent-toggle-plugged**) command works in all modes, and toggles the plugged/unplugged state of the Gnus Agent.

**6.8.3.1 Group Agent Commands**

- J u*      Fetch all eligible articles in the current group (**gnus-agent-fetch-groups**).
- J c*      Enter the Agent category buffer (**gnus-enter-category-buffer**).
- J s*      Fetch all eligible articles in all groups (**gnus-agent-fetch-session**).
- J S*      Send all sendable messages in the queue group (**gnus-group-send-queue**). See [\[Drafts\]](#), page [\[undefined\]](#).
- J a*      Add the current group to an Agent category (**gnus-agent-add-group**). This command understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\[undefined\]](#)).
- J r*      Remove the current group from its category, if any (**gnus-agent-remove-group**). This command understands the process/prefix convention (see [\[Process/Prefix\]](#), page [\[undefined\]](#)).
- J Y*      Synchronize flags changed while unplugged with remote server, if any.

**6.8.3.2 Summary Agent Commands**

- J #*      Mark the article for downloading (**gnus-agent-mark-article**).
- J M-#*    Remove the downloading mark from the article (**gnus-agent-unmark-article**).
- @*        Toggle whether to download the article (**gnus-agent-toggle-mark**). The download mark is ‘%’ by default.
- J c*      Mark all articles as read (**gnus-agent-catchup**) that are neither cached, downloaded, nor downloadable.

- J S*        Download all eligible (see [\[Agent Categories\]](#), page [\[Agent Categories\]](#)) articles in this group. (`gnus-agent-fetch-group`).
- J s*        Download all processable articles in this group. (`gnus-agent-fetch-series`).
- J u*        Download all downloadable articles in the current group (`gnus-agent-summary-fetch-group`).

### 6.8.3.3 Server Agent Commands

- J a*        Add the current server to the list of servers covered by the Gnus Agent (`gnus-agent-add-server`).
- J r*        Remove the current server from the list of servers covered by the Gnus Agent (`gnus-agent-remove-server`).

### 6.8.4 Agent Visuals

If you open a summary while unplugged and, Gnus knows from the group's active range that there are more articles than the headers currently stored in the Agent, you may see some articles whose subject looks something like `[Undownloaded article #####]`. These are placeholders for the missing headers. Aside from setting a mark, there is not much that can be done with one of these placeholders. When Gnus finally gets a chance to fetch the group's headers, the placeholders will automatically be replaced by the actual headers. You can configure the summary buffer's maneuvering to skip over the placeholders if you care (See `gnus-auto-goto-ignores`).

While it may be obvious to all, the only headers and articles available while unplugged are those headers and articles that were fetched into the Agent while previously plugged. To put it another way, "If you forget to fetch something while plugged, you might have a less than satisfying unplugged session". For this reason, the Agent adds two visual effects to your summary buffer. These effects display the download status of each article so that you always know which articles will be available when unplugged.

The first visual effect is the `%0` spec. If you customize `gnus-summary-line-format` to include this specifier, you will add a single character field that indicates an article's download status. Articles that have been fetched into either the Agent or the Cache, will display `gnus-downloaded-mark` (defaults to `+`). All other articles will display `gnus-undownloaded-mark` (defaults to `-`). If you open a group that has not been agentized, a space (`' '`) will be displayed.

The second visual effect are the undownloaded faces. The faces, there are three indicating the article's score (low, normal, high), seem to result in a love/hate response from many Gnus users. The problem is that the face selection is controlled by a list of condition tests and face names (See `gnus-summary-highlight`). Each condition is tested in the order in which it appears in the list so early conditions have precedence over later conditions. All of this means that, if you tick an undownloaded article, the article will continue to be displayed in the undownloaded face rather than the ticked face.

If you use the Agent as a cache (to avoid downloading the same article each time you visit it or to minimize your connection time), the undownloaded face will probably seem like a good idea. The reason being that you do all of our work (marking, reading, deleting) with downloaded articles so the normal faces always appear.



For occasional Agent users, the undownloaded faces may appear to be an absolutely horrible idea. The issue being that, since most of their articles have not been fetched into the Agent, most of the normal faces will be obscured by the undownloaded faces. If this is your situation, you have two choices available. First, you can completely disable the undownload faces by customizing `gnus-summary-highlight` to delete the three cons-cells that refer to the `gnus-summary-*-undownloaded-face` faces. Second, if you prefer to take a more fine-grained approach, you may set the `agent-disable-undownloaded-faces` group parameter to `t`. This parameter, like all other agent parameters, may be set on an Agent Category (see [\[Agent Categories\]](#), page [\(undefined\)](#)), a Group Topic (see [\[Topic Parameters\]](#), page [\(undefined\)](#)), or an individual group (see [\[Group Parameters\]](#), page [\(undefined\)](#)).

### 6.8.5 Agent as Cache

When Gnus is plugged, it is not efficient to download headers or articles from the server again, if they are already stored in the Agent. So, Gnus normally only downloads headers once, and stores them in the Agent. These headers are later used when generating the summary buffer, regardless of whether you are plugged or unplugged. Articles are not cached in the Agent by default though (that would potentially consume lots of disk space), but if you have already downloaded an article into the Agent, Gnus will not download the article from the server again but use the locally stored copy instead.

If you so desire, you can configure the agent (see `gnus-agent-cache` see [\(undefined\)](#) [\[Agent Variables\]](#), page [\(undefined\)](#)) to always download headers and articles while plugged. Gnus will almost certainly be slower, but it will be kept synchronized with the server. That last point probably won't make any sense if you are using a `nntp` or `nimap` back end.

### 6.8.6 Agent Expiry

The Agent back end, `nagent`, doesn't handle expiry. Well, at least it doesn't handle it like other back ends. Instead, there are special `gnus-agent-expire` and `gnus-agent-expire-group` commands that will expire all read articles that are older than `gnus-agent-expire-days` days. They can be run whenever you feel that you're running out of space. Neither are particularly fast or efficient, and it's not a particularly good idea to interrupt them (with `C-g` or anything else) once you've started one of them.

Note that other functions, e.g. `gnus-request-expire-articles`, might run `gnus-agent-expire` for you to keep the agent synchronized with the group.

The agent parameter `agent-enable-expiration` may be used to prevent expiration in selected groups.

If `gnus-agent-expire-all` is non-`nil`, the agent expiration commands will expire all articles—unread, read, ticked and dormant. If `nil` (which is the default), only read articles are eligible for expiry, and unread, ticked and dormant articles will be kept indefinitely.

If you find that some articles eligible for expiry are never expired, perhaps some Gnus Agent files are corrupted. There's are special commands, `gnus-agent-regenerate` and `gnus-agent-regenerate-group`, to fix possible problems.

### 6.8.7 Agent Regeneration

The local data structures used by `nagent` may become corrupted due to certain exceptional conditions. When this happens, `nagent` functionality may degrade or even fail. The



solution to this problem is to repair the local data structures by removing all internal inconsistencies.

For example, if your connection to your server is lost while downloading articles into the agent, the local data structures will not know about articles successfully downloaded prior to the connection failure. Running `gnus-agent-regenerate` or `gnus-agent-regenerate-group` will update the data structures such that you don't need to download these articles a second time.

The command `gnus-agent-regenerate` will perform `gnus-agent-regenerate-group` on every agentized group. While you can run `gnus-agent-regenerate` in any buffer, it is strongly recommended that you first close all summary buffers.

The command `gnus-agent-regenerate-group` uses the local copies of individual articles to repair the local NOV(header) database. It then updates the internal data structures that document which articles are stored locally. An optional argument will mark articles in the agent as unread.

### 6.8.8 Agent and IMAP

The Agent works with any Gnus back end, including `nnimap`. However, since there are some conceptual differences between NNTP and IMAP, this section (should) provide you with some information to make Gnus Agent work smoother as a IMAP Disconnected Mode client.

The first thing to keep in mind is that all flags (read, ticked, etc) are kept on the IMAP server, rather than in `newsrsrc` as is the case for `nnntp`. Thus Gnus needs to remember flag changes when disconnected, and synchronize these flags when you plug back in.

Gnus keeps track of flag changes when reading `nnimap` groups under the Agent. When you plug back in, Gnus will check if you have any changed any flags and ask if you wish to synchronize these with the server. The behavior is customizable by `gnus-agent-synchronize-flags`.

If `gnus-agent-synchronize-flags` is `nil`, the Agent will never automatically synchronize flags. If it is `ask`, which is the default, the Agent will check if you made any changes and if so ask if you wish to synchronize these when you re-connect. If it has any other value, all flags will be synchronized automatically.

If you do not wish to synchronize flags automatically when you re-connect, you can do it manually with the `gnus-agent-synchronize-flags` command that is bound to `J Y` in the group buffer.

Some things are currently not implemented in the Agent that you'd might expect from a disconnected IMAP client, including:

- Copying/moving articles into `nnimap` groups when unplugged.
- Creating/deleting `nnimap` groups when unplugged.

Technical note: the synchronization algorithm does not work by “pushing” all local flags to the server, but rather incrementally update the server view of flags by changing only those flags that were changed by the user. Thus, if you set one flag on an article, quit the group and re-select the group and remove the flag; the flag will be set and removed from the server when you “synchronize”. The queued flag operations can be found in the per-server `flags` file in the Agent directory. It's emptied when you synchronize flags.

### 6.8.9 Outgoing Messages

When Gnus is unplugged, all outgoing messages (both mail and news) are stored in the draft group “queue” (see [\[Drafts\]](#), page [\[undefined\]](#)). You can view them there after posting, and edit them at will.

When Gnus is plugged again, you can send the messages either from the draft group with the special commands available there, or you can use the *J S* command in the group buffer to send all the sendable messages in the draft group.

### 6.8.10 Agent Variables

#### `gnus-agent-directory`

Where the Gnus Agent will store its files. The default is ‘~/News/agent/’.

#### `gnus-agent-handle-level`

Groups on levels (see [\[Group Levels\]](#), page [\[undefined\]](#)) higher than this variable will be ignored by the Agent. The default is `gnus-level-subscribed`, which means that only subscribed group will be considered by the Agent by default.

#### `gnus-agent-plugged-hook`

Hook run when connecting to the network.

#### `gnus-agent-unplugged-hook`

Hook run when disconnecting from the network.

#### `gnus-agent-fetched-hook`

Hook run when finished fetching articles.

#### `gnus-agent-cache`

Variable to control whether use the locally stored NOV and articles when plugged, e.g. essentially using the Agent as a cache. The default is non-`nil`, which means to use the Agent as a cache.

#### `gnus-agent-go-online`

If `gnus-agent-go-online` is `nil`, the Agent will never automatically switch offline servers into online status. If it is `ask`, the default, the Agent will ask if you wish to switch offline servers into online status when you re-connect. If it has any other value, all offline servers will be automatically switched into online status.

#### `gnus-agent-mark-unread-after-downloaded`

If `gnus-agent-mark-unread-after-downloaded` is non-`nil`, mark articles as unread after downloading. This is usually a safe thing to do as the newly downloaded article has obviously not been read. The default is `t`.

#### `gnus-agent-consider-all-articles`

If `gnus-agent-consider-all-articles` is non-`nil`, the agent will let the agent predicate decide whether articles need to be downloaded or not, for all articles. When `nil`, the default, the agent will only let the predicate decide whether unread articles are downloaded or not. If you enable this, you may also want to look into the agent expiry settings (see [\[Category Variables\]](#),

page [\(undefined\)](#)), so that the agent doesn't download articles which the agent will later expire, over and over again.

#### **gnus-agent-max-fetch-size**

The agent fetches articles into a temporary buffer prior to parsing them into individual files. To avoid exceeding the max. buffer size, the agent alternates between fetching and parsing until all articles have been fetched. **gnus-agent-max-fetch-size** provides a size limit to control how often the cycling occurs. A large value improves performance. A small value minimizes the time lost should the connection be lost while fetching (You may need to run **gnus-agent-regenerate-group** to update the group's state. However, all articles parsed prior to losing the connection will be available while unplugged). The default is 10M so it is unusual to see any cycling.

#### **gnus-server-unopen-status**

Perhaps not an Agent variable, but closely related to the Agent, this variable says what will happen if Gnus cannot open a server. If the Agent is enabled, the default, **nil**, makes Gnus ask the user whether to deny the server or whether to unplug the agent. If the Agent is disabled, Gnus always simply deny the server. Other choices for this variable include **denied** and **offline** the latter is only valid if the Agent is used.

#### **gnus-auto-goto-ignores**

Another variable that isn't an Agent variable, yet so closely related that most will look for it here, this variable tells the summary buffer how to maneuver around undownloaded (only headers stored in the agent) and unfetched (neither article nor headers stored) articles.

The legal values are **nil** (maneuver to any article), **undownloaded** (maneuvering while unplugged ignores articles that have not been fetched), **always-undownloaded** (maneuvering always ignores articles that have not been fetched), **unfetched** (maneuvering ignores articles whose headers have not been fetched).

#### **gnus-agent-auto-agentize-methods**

If you have never used the Agent before (or more technically, if `~/News/agent/lib/servers` does not exist), Gnus will automatically agentize a few servers for you. This variable control which backends should be auto-agentized. It is typically only useful to agentize remote backends. The auto-agentizing has the same effect as running **J a** on the servers (see [\(undefined\)](#) [Server Agent Commands], [page \(undefined\)](#)). If the file exist, you must manage the servers manually by adding or removing them, this variable is only applicable the first time you start Gnus. The default is `'(nnntp nnimap)'`.

### **6.8.11 Example Setup**

If you don't want to read this manual, and you have a fairly standard setup, you may be able to use something like the following as your `~/gnus.el` file to get started.

```
;;; Define how Gnus is to fetch news. We do this over NNTP
;;; from your ISP's server.
```

```
(setq gnus-select-method '(nntp "news.your-isp.com"))

;;; Define how Gnus is to read your mail. We read mail from
;;; your ISP's POP server.
(setq mail-sources '((pop :server "pop.your-isp.com")))

;;; Say how Gnus is to store the mail. We use nnml groups.
(setq gnus-secondary-select-methods '((nnml "")))

;;; Make Gnus into an offline newsreader.
;;; (gnus-agentize) ; The obsolete setting.
;;; (setq gnus-agent t) ; Now the default.
```

That should be it, basically. Put that in your `~/gnus.el` file, edit to suit your needs, start up PPP (or whatever), and type `M-x gnus`.

If this is the first time you've run Gnus, you will be subscribed automatically to a few default newsgroups. You'll probably want to subscribe to more groups, and to do that, you have to query the NNTP server for a complete list of groups with the `A A` command. This usually takes quite a while, but you only have to do it once.

After reading and parsing a while, you'll be presented with a list of groups. Subscribe to the ones you want to read with the `u` command. `1` to make all the killed groups disappear after you've subscribe to all the groups you want to read. (`A k` will bring back all the killed groups.)

You can now read the groups at once, or you can download the articles with the `J s` command. And then read the rest of this manual to find out which of the other gazillion things you want to customize.

### 6.8.12 Batching Agents

Having the Gnus Agent fetch articles (and post whatever messages you've written) is quite easy once you've gotten things set up properly. The following shell script will do everything that is necessary:

You can run a complete batch command from the command line with the following incantation:

```
#!/bin/sh
emacs -batch -l ~/.emacs -l ~/.gnus.el gnus-agent-batch >/dev/null 2>&1
```

### 6.8.13 Agent Caveats

The Gnus Agent doesn't seem to work like most other offline newsreaders. Here are some common questions that some imaginary people may ask:

*If I read an article while plugged, do they get entered into the Agent?*

**No.** If you want this behaviour, add `gnus-agent-fetch-selected-article` to `gnus-select-article-hook`.

*If I read an article while plugged, and the article already exists in the Agent, will it get downloaded once more?*

**No,** unless `gnus-agent-cache` is `nil`.

In short, when Gnus is unplugged, it only looks into the locally stored articles; when it's plugged, it talks to your ISP and may also use the locally stored articles.



## 7 Scoring

Other people use *kill files*, but we here at Gnus Towers like scoring better than killing, so we'd rather switch than fight. They do something completely different as well, so sit up straight and pay attention!

All articles have a default score (`gnus-summary-default-score`), which is 0 by default. This score may be raised or lowered either interactively or by score files. Articles that have a score lower than `gnus-summary-mark-below` are marked as read.

Gnus will read any *score files* that apply to the current group before generating the summary buffer.

There are several commands in the summary buffer that insert score entries based on the current article. You can, for instance, ask Gnus to lower or increase the score of all articles with a certain subject.

There are two sorts of scoring entries: Permanent and temporary. Temporary score entries are self-expiring entries. Any entries that are temporary and have not been used for, say, a week, will be removed silently to help keep the sizes of the score files down.

### 7.1 Summary Score Commands

The score commands that alter score entries do not actually modify real score files. That would be too inefficient. Gnus maintains a cache of previously loaded score files, one of which is considered the *current score file alist*. The score commands simply insert entries into this list, and upon group exit, this list is saved.

The current score file is by default the group's local score file, even if no such score file actually exists. To insert score commands into some other score file (e.g. 'all.SCORE'), you must first make this score file the current one.

General score commands that don't actually change the score file:

<code>V s</code>	Set the score of the current article ( <code>gnus-summary-set-score</code> ).
<code>V S</code>	Display the score of the current article ( <code>gnus-summary-current-score</code> ).
<code>V t</code>	Display all score rules that have been used on the current article ( <code>gnus-score-find-trace</code> ). In the <code>*Score Trace*</code> buffer, you may type <code>e</code> to edit score file corresponding to the score rule on current line and <code>f</code> to format ( <code>gnus-score-pretty-print</code> ) the score file and edit it.
<code>V w</code>	List words used in scoring ( <code>gnus-score-find-favourite-words</code> ).
<code>V R</code>	Run the current summary through the scoring process ( <code>gnus-summary-rescore</code> ). This might be useful if you're playing around with your score files behind Gnus' back and want to see the effect you're having.
<code>V c</code>	Make a different score file the current ( <code>gnus-score-change-score-file</code> ).
<code>V e</code>	Edit the current score file ( <code>gnus-score-edit-current-scores</code> ). You will be popped into a <code>gnus-score-mode</code> buffer (see <a href="#">[Score File Editing]</a> , <a href="#">page <code>undefined</code></a> ).
<code>V f</code>	Edit a score file and make this score file the current one ( <code>gnus-score-edit-file</code> ).

*V F* Flush the score cache (`gnus-score-flush-cache`). This is useful after editing score files.

*V C* Customize a score file in a visually pleasing manner (`gnus-score-customize`).

The rest of these commands modify the local score file.

*V m* Prompt for a score, and mark all articles with a score below this as read (`gnus-score-set-mark-below`).

*V x* Prompt for a score, and add a score rule to the current score file to expunge all articles below this score (`gnus-score-set-expunge-below`).

The keystrokes for actually making score entries follow a very regular pattern, so there's no need to list all the commands. (Hundreds of them.)

1. The first key is either *I* (upper case i) for increasing the score or *L* for lowering the score.
2. The second key says what header you want to score on. The following keys are available:

<i>a</i>	Score on the author name.
<i>s</i>	Score on the subject line.
<i>x</i>	Score on the <b>Xref</b> line—i.e., the cross-posting line.
<i>r</i>	Score on the <b>References</b> line.
<i>d</i>	Score on the date.
<i>l</i>	Score on the number of lines.
<i>i</i>	Score on the <b>Message-ID</b> header.
<i>e</i>	Score on an “extra” header, that is, one of those in <code>gnus-extra-headers</code> , if your NNTP server tracks additional header data in overviews.
<i>f</i>	Score on followups—this matches the author name, and adds scores to the followups to this author. (Using this key leads to the creation of ‘ADAPT’ files.)
<i>b</i>	Score on the body.
<i>h</i>	Score on the head.
<i>t</i>	Score on thread. (Using this key leads to the creation of ‘ADAPT’ files.)

3. The third key is the match type. Which match types are valid depends on what headers you are scoring on.

#### strings

<i>e</i>	Exact matching.
<i>s</i>	Substring matching.
<i>f</i>	Fuzzy matching (see <a href="#">[Fuzzy Matching]</a> , page <a href="#">[undefined]</a> ).
<i>r</i>	Regexp matching



**date**

<i>b</i>	Before date.
<i>a</i>	After date.
<i>n</i>	This date.

**number**

<	Less than number.
=	Equal to number.
>	Greater than number.

4. The fourth and usually final key says whether this is a temporary (i.e., expiring) score entry, or a permanent (i.e., non-expiring) score entry, or whether it is to be done immediately, without adding to the score file.

<i>t</i>	Temporary score entry.
<i>p</i>	Permanent score entry.
<i>i</i>	Immediately scoring.

5. If you are scoring on ‘e’ (extra) headers, you will then be prompted for the header name on which you wish to score. This must be a header named in gnus-extra-headers, and ‘TAB’ completion is available.

So, let’s say you want to increase the score on the current author with exact matching permanently: *I a e p*. If you want to lower the score based on the subject line, using substring matching, and make a temporary score entry: *L s s t*. Pretty easy.

To make things a bit more complicated, there are shortcuts. If you use a capital letter on either the second or third keys, Gnus will use defaults for the remaining one or two keystrokes. The defaults are “substring” and “temporary”. So *I A* is the same as *I a s t*, and *I a R* is the same as *I a r t*.

These functions take both the numerical prefix and the symbolic prefix (see [\[Symbolic Prefixes\]](#), page [\[Symbolic Prefixes\]](#)). A numerical prefix says how much to lower (or increase) the score of the article. A symbolic prefix of *a* says to use the ‘all.SCORE’ file for the command instead of the current score file.

The `gnus-score-mimic-keymap` says whether these commands will pretend they are keymaps or not.

## 7.2 Group Score Commands

There aren’t many of these as yet, I’m afraid.

***W f*** Gnus maintains a cache of score alists to avoid having to reload them all the time. This command will flush the cache (`gnus-score-flush-cache`).

You can do scoring from the command line by saying something like:

```
$ emacs -batch -l ~/.emacs -l ~/.gnus.el -f gnus-batch-score
```

## 7.3 Score Variables

### `gnus-use-scoring`

If `nil`, Gnus will not check for score files, and will not, in general, do any score-related work. This is `t` by default.

### `gnus-kill-killed`

If this variable is `nil`, Gnus will never apply score files to articles that have already been through the kill process. While this may save you lots of time, it also means that if you apply a kill file to a group, and then change the kill file and want to run it over you group again to kill more articles, it won't work. You have to set this variable to `t` to do that. (It is `t` by default.)

### `gnus-kill-files-directory`

All kill and score files will be stored in this directory, which is initialized from the `SAVEDIR` environment variable by default. This is `'~/News/'` by default.

### `gnus-score-file-suffix`

Suffix to add to the group name to arrive at the score file name (`'SCORE'` by default.)

### `gnus-score-uncacheable-files`

All score files are normally cached to avoid excessive re-loading of score files. However, if this might make your Emacs grow big and bloated, so this regexp can be used to weed out score files unlikely to be needed again. It would be a bad idea to deny caching of `'all.SCORE'`, while it might be a good idea to not cache `'comp.infosystems.www.authoring.misc.ADAPT'`. In fact, this variable is `'ADAPT$'` by default, so no adaptive score files will be cached.

### `gnus-save-score`

If you have really complicated score files, and do lots of batch scoring, then you might set this variable to `t`. This will make Gnus save the scores into the `'newsrsrc.eld'` file.

If you do not set this to `t`, then manual scores (like those set with `V s` (`gnus-summary-set-score`)) will not be preserved across group visits.

### `gnus-score-interactive-default-score`

Score used by all the interactive raise/lower commands to raise/lower score with. Default is 1000, which may seem excessive, but this is to ensure that the adaptive scoring scheme gets enough room to play with. We don't want the small changes from the adaptive scoring to overwrite manually entered data.

### `gnus-summary-default-score`

Default score of an article, which is 0 by default.

### `gnus-summary-expunge-below`

Don't display the summary lines of articles that have scores lower than this variable. This is `nil` by default, which means that no articles will be hidden. This variable is local to the summary buffers, and has to be set from `gnus-summary-mode-hook`.

**gnus-score-over-mark**

Mark (in the third column) used for articles with a score over the default. Default is '+’.

**gnus-score-below-mark**

Mark (in the third column) used for articles with a score below the default. Default is '-’.

**gnus-score-find-score-files-function**

Function used to find score files for the current group. This function is called with the name of the group as the argument.

Predefined functions available are:

**gnus-score-find-single**

Only apply the group’s own score file.

**gnus-score-find-bnews**

Apply all score files that match, using bnews syntax. This is the default. If the current group is ‘gnu.emacs.gnus’, for instance, ‘all.emacs.all.SCORE’, ‘not.alt.all.SCORE’ and ‘gnu.all.SCORE’ would all apply. In short, the instances of ‘all’ in the score file names are translated into ‘.\*’, and then a regexp match is done.

This means that if you have some score entries that you want to apply to all groups, then you put those entries in the ‘all.SCORE’ file.

The score files are applied in a semi-random order, although Gnus will try to apply the more general score files before the more specific score files. It does this by looking at the number of elements in the score file names—discarding the ‘all’ elements.

**gnus-score-find-hierarchical**

Apply all score files from all the parent groups. This means that you can’t have score files like ‘all.SCORE’, but you can have ‘SCORE’, ‘comp.SCORE’ and ‘comp.emacs.SCORE’ for each server.

This variable can also be a list of functions. In that case, all these functions will be called with the group name as argument, and all the returned lists of score files will be applied. These functions can also return lists of lists of score alists directly. In that case, the functions that return these non-file score alists should probably be placed before the “real” score file functions, to ensure that the last score file returned is the local score file. Phu.

For example, to do hierarchical scoring but use a non-server-specific overall score file, you could use the value

```
(list (lambda (group) ("all.SCORE"))
      'gnus-score-find-hierarchical)
```

**gnus-score-expiry-days**

This variable says how many days should pass before an unused score file entry is expired. If this variable is nil, no score file entries are expired. It’s 7 by default.

**gnus-update-score-entry-dates**

If this variable is non-`nil`, temporary score entries that have been triggered (matched) will have their dates updated. (This is how Gnus controls expiry—all non-matched-entries will become too old while matched entries will stay fresh and young.) However, if you set this variable to `nil`, even matched entries will grow old and will have to face that oh-so grim reaper.

**gnus-score-after-write-file-function**

Function called with the name of the score file just written.

**gnus-score-thread-simplify**

If this variable is non-`nil`, article subjects will be simplified for subject scoring purposes in the same manner as with threading—according to the current value of `gnus-simplify-subject-functions`. If the scoring entry uses `substring` or `exact` matching, the match will also be simplified in this manner.

## 7.4 Score File Format

A score file is an `emacs-lisp` file that normally contains just a single form. Casual users are not expected to edit these files; everything can be changed from the summary buffer.

Anyway, if you'd like to dig into it yourself, here's an example:

```
((("from"
  ("Lars Ingebrigtsen" -10000)
  ("Per Abrahamsen")
  ("larsi\\|lmi" -50000 nil R))
  ("subject"
   ("Ding is Badd" nil 728373))
  ("xref"
   ("alt.politics" -1000 728372 s))
  ("lines"
   (2 -100 nil <))
  (mark 0)
  (expunge -1000)
  (mark-and-expunge -10)
  (read-only nil)
  (orphan -10)
  (adapt t)
  (files "/hom/larsi/News/gnu.SCORE")
  (exclude-files "all.SCORE")
  (local (gnus-newsgroup-auto-expire t)
    (gnus-summary-make-false-root empty))
  (eval (ding)))
```

This example demonstrates most score file elements. See [\(undefined\)](#) [Advanced Scoring], page [\(undefined\)](#), for a different approach.

Even though this looks much like Lisp code, nothing here is actually `eval`ed. The Lisp reader is used to read this form, though, so it has to be valid syntactically, if not semantically.

Six keys are supported by this alist:

**STRING** If the key is a string, it is the name of the header to perform the match on. Scoring can only be performed on these eight headers: **From**, **Subject**, **References**, **Message-ID**, **Xref**, **Lines**, **Chars** and **Date**. In addition to these headers, there are three strings to tell Gnus to fetch the entire article and do the match on larger parts of the article: **Body** will perform the match on the body of the article, **Head** will perform the match on the head of the article, and **All** will perform the match on the entire article. Note that using any of these last three keys will slow down group entry *considerably*. The final “header” you can score on is **Followup**. These score entries will result in new score entries being added for all follow-ups to articles that matches these score entries.

Following this key is an arbitrary number of score entries, where each score entry has one to four elements.

1. The first element is the *match element*. On most headers this will be a string, but on the **Lines** and **Chars** headers, this must be an integer.
2. If the second element is present, it should be a number—the *score element*. This number should be an integer in the `neginf` to `posinf` interval. This number is added to the score of the article if the match is successful. If this element is not present, the `gnus-score-interactive-default-score` number will be used instead. This is 1000 by default.
3. If the third element is present, it should be a number—the *date element*. This date says when the last time this score entry matched, which provides a mechanism for expiring the score entries. If this element is not present, the score entry is permanent. The date is represented by the number of days since December 31, 1 BCE.
4. If the fourth element is present, it should be a symbol—the *type element*. This element specifies what function should be used to see whether this score entry matches the article. What match types that can be used depends on what header you wish to perform the match on.

*From, Subject, References, Xref, Message-ID*

For most header types, there are the **r** and **R** (regexp), as well as **s** and **S** (substring) types, and **e** and **E** (exact match), and **w** (word match) types. If this element is not present, Gnus will assume that substring matching should be used. **R**, **S**, and **E** differ from the others in that the matches will be done in a case-sensitive manner. All these one-letter types are really just abbreviations for the `regexp`, `string`, `exact`, and `word` types, which you can use instead, if you feel like.

*Extra* Just as for the standard string overview headers, if you are using `gnus-extra-headers`, you can score on these headers’ values. In this case, there is a 5th element in the score entry, being the name of the header to be scored. The following entry is useful in your ‘`all.SCORE`’ file in case of spam attacks from a single origin host, if your NNTP server tracks ‘`NNTP-Posting-Host`’ in overviews:

```
("111.222.333.444" -1000 nil s
```

"NNTP-Posting-Host")

### *Lines, Chars*

These two headers use different match types: `<`, `>`, `=`, `>=` and `<=`.

These predicates are true if

(PREDICATE HEADER MATCH)

evaluates to non-`nil`. For instance, the advanced match `("lines" 4 <)` (see [\[Advanced Scoring\]](#), page [\[undefined\]](#)) will result in the following form:

(`< header-value 4`)

Or to put it another way: When using `<` on `Lines` with 4 as the match, we get the score added if the article has less than 4 lines. (It's easy to get confused and think it's the other way around. But it's not. I think.)

When matching on `Lines`, be careful because some back ends (like `nndir`) do not generate `Lines` header, so every article ends up being marked as having 0 lines. This can lead to strange results if you happen to lower score of the articles with few lines.

### *Date*

For the `Date` header we have three kinda silly match types: `before`, `at` and `after`. I can't really imagine this ever being useful, but, like, it would feel kinda silly not to provide this function. Just in case. You never know. Better safe than sorry. Once burnt, twice shy. Don't judge a book by its cover. Never not have sex on a first date. (I have been told that at least one person, and I quote, "found this function indispensable", however.)

A more useful match type is `regexp`. With it, you can match the date string using a regular expression. The date is normalized to ISO8601 compact format first—`YYYYMMDDTHHMMSS`. If you want to match all articles that have been posted on April 1st in every year, you could use `'....0401.....'` as a match string, for instance. (Note that the date is kept in its original time zone, so this will match articles that were posted when it was April 1st where the article was posted from. Time zones are such wholesome fun for the whole family, eh?)

### *Head, Body, All*

These three match keys use the same match types as the `From` (etc) header uses.

### *Followup*

This match key is somewhat special, in that it will match the `From` header, and affect the score of not only the matching articles, but also all followups to the matching articles. This allows you e.g. increase the score of followups to your own

articles, or decrease the score of followups to the articles of some known trouble-maker. Uses the same match types as the **From** header uses. (Using this match key will lead to creation of ‘ADAPT’ files.)

*Thread* This match key works along the same lines as the **Followup** match key. If you say that you want to score on a (sub-)thread started by an article with a **Message-ID** *x*, then you add a ‘**thread**’ match. This will add a new ‘**thread**’ match for each article that has *x* in its **References** header. (These new ‘**thread**’ matches will use the **Message-IDs** of these matching articles.) This will ensure that you can raise/lower the score of an entire thread, even though some articles in the thread may not have complete **References** headers. Note that using this may lead to undeterministic scores of the articles in the thread. (Using this match key will lead to creation of ‘ADAPT’ files.)

**mark** The value of this entry should be a number. Any articles with a score lower than this number will be marked as read.

**expunge** The value of this entry should be a number. Any articles with a score lower than this number will be removed from the summary buffer.

**mark-and-expunge** The value of this entry should be a number. Any articles with a score lower than this number will be marked as read and removed from the summary buffer.

**thread-mark-and-expunge** The value of this entry should be a number. All articles that belong to a thread that has a total score below this number will be marked as read and removed from the summary buffer. **gnus-thread-score-function** says how to compute the total score for a thread.

**files** The value of this entry should be any number of file names. These files are assumed to be score files as well, and will be loaded the same way this one was.

**exclude-files** The value of this entry should be any number of files. These files will not be loaded, even though they would normally be so, for some reason or other.

**eval** The value of this entry will be **eval**ed. This element will be ignored when handling global score files.

**read-only** Read-only score files will not be updated or saved. Global score files should feature this atom (see [\[Global Score Files\]](#), page [\[undefined\]](#)). (Note: *Global* here really means *global*; not your personal apply-to-all-groups score files.)

**orphan** The value of this entry should be a number. Articles that do not have parents will get this number added to their scores. Imagine you follow some high-volume

newsgroup, like ‘`comp.lang.c`’. Most likely you will only follow a few of the threads, also want to see any new threads.

You can do this with the following two score file entries:

```
(orphan -500)
(mark-and-expunge -100)
```

When you enter the group the first time, you will only see the new threads. You then raise the score of the threads that you find interesting (with *I T* or *I S*), and ignore (*C y*) the rest. Next time you enter the group, you will see new articles in the interesting threads, plus any new threads.

I.e.—the orphan score atom is for high-volume groups where a few interesting threads which can’t be found automatically by ordinary scoring rules exist.

**adapt** This entry controls the adaptive scoring. If it is `t`, the default adaptive scoring rules will be used. If it is `ignore`, no adaptive scoring will be performed on this group. If it is a list, this list will be used as the adaptive scoring rules. If it isn’t present, or is something other than `t` or `ignore`, the default adaptive scoring rules will be used. If you want to use adaptive scoring on most groups, you’d set `gnus-use-adaptive-scoring` to `t`, and insert an `(adapt ignore)` in the groups where you do not want adaptive scoring. If you only want adaptive scoring in a few groups, you’d set `gnus-use-adaptive-scoring` to `nil`, and insert `(adapt t)` in the score files of the groups where you want it.

**adapt-file**

All adaptive score entries will go to the file named by this entry. It will also be applied when entering the group. This atom might be handy if you want to adapt on several groups at once, using the same adaptive file for a number of groups.

**local** The value of this entry should be a list of `(var value)` pairs. Each `var` will be made buffer-local to the current summary buffer, and set to the value specified. This is a convenient, if somewhat strange, way of setting variables in some groups if you don’t like hooks much. Note that the *value* won’t be evaluated.

## 7.5 Score File Editing

You normally enter all scoring commands from the summary buffer, but you might feel the urge to edit them by hand as well, so we’ve supplied you with a mode for that.

It’s simply a slightly customized `emacs-lisp` mode, with these additional commands:

**C-c C-c** Save the changes you have made and return to the summary buffer (`gnus-score-edit-done`).

**C-c C-d** Insert the current date in numerical format (`gnus-score-edit-insert-date`). This is really the day number, if you were wondering.

**C-c C-p** The adaptive score files are saved in an unformatted fashion. If you intend to read one of these files, you want to *pretty print* it first. This command (`gnus-score-pretty-print`) does that for you.

Type *M-x gnus-score-mode* to use this mode.



`gnus-score-menu-hook` is run in score mode buffers.

In the summary buffer you can use commands like `V f`, `V e` and `V t` to begin editing score files.

## 7.6 Adaptive Scoring

If all this scoring is getting you down, Gnus has a way of making it all happen automatically—as if by magic. Or rather, as if by artificial stupidity, to be precise.

When you read an article, or mark an article as read, or kill an article, you leave marks behind. On exit from the group, Gnus can sniff these marks and add score elements depending on what marks it finds. You turn on this ability by setting `gnus-use-adaptive-scoring` to `t` or `(line)`. If you want score adaptively on separate words appearing in the subjects, you should set this variable to `(word)`. If you want to use both adaptive methods, set this variable to `(word line)`.

To give you complete control over the scoring process, you can customize the `gnus-default-adaptive-score-alist` variable. For instance, it might look something like this:

```
(setq gnus-default-adaptive-score-alist
      '((gnus-unread-mark)
        (gnus-ticked-mark (from 4))
        (gnus-dormant-mark (from 5))
        (gnus-del-mark (from -4) (subject -1))
        (gnus-read-mark (from 4) (subject 2))
        (gnus-expirable-mark (from -1) (subject -1))
        (gnus-killed-mark (from -1) (subject -3))
        (gnus-kill-file-mark)
        (gnus-ancient-mark)
        (gnus-low-score-mark)
        (gnus-catchup-mark (from -1) (subject -1)))))
```

As you see, each element in this alist has a mark as a key (either a variable name or a “real” mark—a character). Following this key is a arbitrary number of header/score pairs. If there are no header/score pairs following the key, no adaptive scoring will be done on articles that have that key as the article mark. For instance, articles with `gnus-unread-mark` in the example above will not get adaptive score entries.

Each article can have only one mark, so just a single of these rules will be applied to each article.

To take `gnus-del-mark` as an example—this alist says that all articles that have that mark (i.e., are marked with ‘e’) will have a score entry added to lower based on the `From` header by -4, and lowered by `Subject` by -1. Change this to fit your prejudices.

If you have marked 10 articles with the same subject with `gnus-del-mark`, the rule for that mark will be applied ten times. That means that that subject will get a score of ten times -1, which should be, unless I’m much mistaken, -10.

If you have auto-expirable (mail) groups (see [\[Expiring Mail\]](#), page [\[undefined\]](#)), all the read articles will be marked with the ‘E’ mark. This’ll probably make adaptive scoring slightly impossible, so auto-expiring and adaptive scoring doesn’t really mix very well.

The headers you can score on are `from`, `subject`, `message-id`, `references`, `xref`, `lines`, `chars` and `date`. In addition, you can score on `followup`, which will create an adaptive score entry that matches on the `References` header using the `Message-ID` of the current article, thereby matching the following thread.

If you use this scheme, you should set the score file atom `mark` to something small—like `-300`, perhaps, to avoid having small random changes result in articles getting marked as read.

After using adaptive scoring for a week or so, Gnus should start to become properly trained and enhance the authors you like best, and kill the authors you like least, without you having to say so explicitly.

You can control what groups the adaptive scoring is to be performed on by using the score files (see [\[Score File Format\]](#), page [\[Score File Format\]](#)). This will also let you use different rules in different groups.

The adaptive score entries will be put into a file where the name is the group name with `gnus-adaptive-file-suffix` appended. The default is `'ADAPT'`.

When doing adaptive scoring, substring or fuzzy matching would probably give you the best results in most cases. However, if the header one matches is short, the possibility for false positives is great, so if the length of the match is less than `gnus-score-exact-adapt-limit`, exact matching will be used. If this variable is `nil`, exact matching will always be used to avoid this problem.

As mentioned above, you can adapt either on individual words or entire headers. If you adapt on words, the `gnus-default-adaptive-word-score-alist` variable says what score each instance of a word should add given a mark.

```
(setq gnus-default-adaptive-word-score-alist
      '((,gnus-read-mark . 30)
        (,gnus-catchup-mark . -10)
        (,gnus-killed-mark . -20)
        (,gnus-del-mark . -15)))
```

This is the default value. If you have adaption on words enabled, every word that appears in subjects of articles marked with `gnus-read-mark` will result in a score rule that increase the score with 30 points.

Words that appear in the `gnus-default-ignored-adaptive-words` list will be ignored. If you wish to add more words to be ignored, use the `gnus-ignored-adaptive-words` list instead.

Some may feel that short words shouldn't count when doing adaptive scoring. If so, you may set `gnus-adaptive-word-length-limit` to an integer. Words shorter than this number will be ignored. This variable defaults to `nil`.

When the scoring is done, `gnus-adaptive-word-syntax-table` is the syntax table in effect. It is similar to the standard syntax table, but it considers numbers to be non-word-constituent characters.

If `gnus-adaptive-word-minimum` is set to a number, the adaptive word scoring process will never bring down the score of an article to below this number. The default is `nil`.

If `gnus-adaptive-word-no-group-words` is set to `t`, gnus won't adaptively word score any of the words in the group name. Useful for groups like `'comp.editors.emacs'`, where most of the subject lines contain the word `'emacs'`.

After using this scheme for a while, it might be nice to write a `gnus-psychoanalyze-user` command to go through the rules and see what words you like and what words you don't like. Or perhaps not.

Note that the adaptive word scoring thing is highly experimental and is likely to change in the future. Initial impressions seem to indicate that it's totally useless as it stands. Some more work (involving more rigorous statistical methods) will have to be done to make this useful.

## 7.7 Home Score File

The score file where new score file entries will go is called the *home score file*. This is normally (and by default) the score file for the group itself. For instance, the home score file for `'gnu.emacs.gnus'` is `'gnu.emacs.gnus.SCORE'`.

However, this may not be what you want. It is often convenient to share a common home score file among many groups—all `'emacs'` groups could perhaps use the same home score file.

The variable that controls this is `gnus-home-score-file`. It can be:

1. A string. Then this file will be used as the home score file for all groups.
2. A function. The result of this function will be used as the home score file. The function will be called with the name of the group as the parameter.
3. A list. The elements in this list can be:
  1. `(regexp file-name)`. If the *regexp* matches the group name, the *file-name* will be used as the home score file.
  2. A function. If the function returns non-`nil`, the result will be used as the home score file.
  3. A string. Use the string as the home score file.

The list will be traversed from the beginning towards the end looking for matches.

So, if you want to use just a single score file, you could say:

```
(setq gnus-home-score-file
      "my-total-score-file.SCORE")
```

If you want to use `'gnu.SCORE'` for all `'gnu'` groups and `'rec.SCORE'` for all `'rec'` groups (and so on), you can say:

```
(setq gnus-home-score-file
      'gnus-hierarchical-home-score-file)
```

This is a ready-made function provided for your convenience. Other functions include `gnus-current-home-score-file`

Return the “current” regular score file. This will make scoring commands add entry to the “innermost” matching score file.

If you want to have one score file for the `'emacs'` groups and another for the `'comp'` groups, while letting all other groups use their own home score files:

```
(setq gnus-home-score-file
      ;; All groups that match the regexp "\\..emacs"
```

```
'(("\\.emacs" "emacs.SCORE")
;; All the comp groups in one score file
("^comp" "comp.SCORE")))
```

`gnus-home-adapt-file` works exactly the same way as `gnus-home-score-file`, but says what the home adaptive score file is instead. All new adaptive file entries will go into the file specified by this variable, and the same syntax is allowed.

In addition to using `gnus-home-score-file` and `gnus-home-adapt-file`, you can also use group parameters (see [\[Group Parameters\]](#), page [\[undefined\]](#)) and topic parameters (see [\[Topic Parameters\]](#), page [\[undefined\]](#)) to achieve much the same. Group and topic parameters take precedence over this variable.

## 7.8 Followups To Yourself

Gnus offers two commands for picking out the `Message-ID` header in the current buffer. Gnus will then add a score rule that scores using this `Message-ID` on the `References` header of other articles. This will, in effect, increase the score of all articles that respond to the article in the current buffer. Quite useful if you want to easily note when people answer what you’ve said.

`gnus-score-followup-article`

This will add a score to articles that directly follow up your own article.

`gnus-score-followup-thread`

This will add a score to all articles that appear in a thread “below” your own article.

These two functions are both primarily meant to be used in hooks like `message-sent-hook`, like this:

```
(add-hook 'message-sent-hook 'gnus-score-followup-thread)
```

If you look closely at your own `Message-ID`, you’ll notice that the first two or three characters are always the same. Here’s two of mine:

```
<x6u3u47icf.fsf@eyesore.no>
<x6sp9o7ibw.fsf@eyesore.no>
```

So “my” ident on this machine is ‘x6’. This can be exploited—the following rule will raise the score on all followups to myself:

```
("references"
("<x6[0-9a-z]+\\.fsf\\(_-\\)?@.*eyesore\\.no>"
1000 nil r))
```

Whether it’s the first two or first three characters that are “yours” is system-dependent.

## 7.9 Scoring On Other Headers

Gnus is quite fast when scoring the “traditional” headers—‘`From`’, ‘`Subject`’ and so on. However, scoring other headers requires writing a `head` scoring rule, which means that Gnus has to request every single article from the back end to find matches. This takes a long time in big groups.

Now, there’s not much you can do about this for news groups, but for mail groups, you have greater control. In [\[To From Newsgroups\]](#), page [\[undefined\]](#), it’s explained

in greater detail what this mechanism does, but here's a cookbook example for `nnml` on how to allow scoring on the 'To' and 'Cc' headers.

Put the following in your '~/.gnus.el' file.

```
(setq gnus-extra-headers '(To Cc Newsgroups Keywords)
      nnmail-extra-headers gnus-extra-headers)
```

Restart Gnus and rebuild your `nnml` overview files with the `M-x nnml-generate-nov-databases` command. This will take a long time if you have much mail.

Now you can score on 'To' and 'Cc' as "extra headers" like so: `I e s p To RET <your name> RET`.

See? Simple.

## 7.10 Scoring Tips

### *Crossposts*

If you want to lower the score of crossposts, the line to match on is the `Xref` header.

```
("xref" (" talk.politics.misc:" -1000))
```

### *Multiple crossposts*

If you want to lower the score of articles that have been crossposted to more than, say, 3 groups:

```
("xref"
 (" [^:\n]+:[0-9]+ +[^:\n]+:[0-9]+ +[^:\n]+:[0-9]+"
  -1000 nil r))
```

### *Matching on the body*

This is generally not a very good idea—it takes a very long time. Gnus actually has to fetch each individual article from the server. But you might want to anyway, I guess. Even though there are three match keys (`Head`, `Body` and `All`), you should choose one and stick with it in each score file. If you use any two, each article will be fetched *twice*. If you want to match a bit on the `Head` and a bit on the `Body`, just use `All` for all the matches.

### *Marking as read*

You will probably want to mark articles that have scores below a certain number as read. This is most easily achieved by putting the following in your 'all.SCORE' file:

```
((mark -100))
```

You may also consider doing something similar with `expunge`.

### *Negated character classes*

If you say stuff like `[^abcd]*`, you may get unexpected results. That will match newlines, which might lead to, well, The Unknown. Say `[^abcd\n]*` instead.

## 7.11 Reverse Scoring

If you want to keep just articles that have 'Sex with Emacs' in the subject header, and expunge all other articles, you could put something like this in your score file:

```
((("subject"
  ("Sex with Emacs" 2))
 (mark 1)
 (expunge 1))
```

So, you raise all articles that match ‘Sex with Emacs’ and mark the rest as read, and expunge them to boot.

## 7.12 Global Score Files

Sure, other newsreaders have “global kill files”. These are usually nothing more than a single kill file that applies to all groups, stored in the user’s home directory. Bah! Puny, weak newsreaders!

What I’m talking about here are Global Score Files. Score files from all over the world, from users everywhere, uniting all nations in one big, happy score file union! Ange-score! New and untested!

All you have to do to use other people’s score files is to set the `gnus-global-score-files` variable. One entry for each score file, or each score file directory. Gnus will decide by itself what score files are applicable to which group.

To use the score file ‘/ftp@ftp.gnus.org:/pub/larsi/ding/score/soc.motss.SCORE’ and all score files in the ‘/ftp@ftp.some-where:/pub/score’ directory, say this:

```
(setq gnus-global-score-files
      '("/ftp@ftp.gnus.org:/pub/larsi/ding/score/soc.motss.SCORE"
        "/ftp@ftp.some-where:/pub/score/"))
```

Simple, eh? Directory names must end with a ‘/’. These directories are typically scanned only once during each Gnus session. If you feel the need to manually re-scan the remote directories, you can use the `gnus-score-search-global-directories` command.

Note that, at present, using this option will slow down group entry somewhat. (That is—a lot.)

If you want to start maintaining score files for other people to use, just put your score file up for anonymous ftp and announce it to the world. Become a retro-moderator! Participate in the retro-moderator wars sure to ensue, where retro-moderators battle it out for the sympathy of the people, luring them to use their score files on false premises! Yay! The net is saved!

Here are some tips for the would-be retro-moderator, off the top of my head:

- Articles heavily crossposted are probably junk.
- To lower a single inappropriate article, lower by `Message-ID`.
- Particularly brilliant authors can be raised on a permanent basis.
- Authors that repeatedly post off-charter for the group can safely be lowered out of existence.
- Set the `mark` and `expunge` atoms to obliterate the nastiest articles completely.
- Use expiring score entries to keep the size of the file down. You should probably have a long expiry period, though, as some sites keep old articles for a long time.

... I wonder whether other newsreaders will support global score files in the future. *Snicker*. Yup, any day now, newsreaders like Blue Wave, xrn and 1stReader are bound to implement scoring. Should we start holding our breath yet?

## 7.13 Kill Files

Gnus still supports those pesky old kill files. In fact, the kill file entries can now be expiring, which is something I wrote before Daniel Quinlan thought of doing score files, so I've left the code in there.

In short, kill processing is a lot slower (and I do mean *a lot*) than score processing, so it might be a good idea to rewrite your kill files into score files.

Anyway, a kill file is a normal `emacs-lisp` file. You can put any forms into this file, which means that you can use kill files as some sort of primitive hook function to be run on group entry, even though that isn't a very good idea.

Normal kill files look like this:

```
(gnus-kill "From" "Lars Ingebrigtsen")
(gnus-kill "Subject" "ding")
(gnus-expunge "X")
```

This will mark every article written by me as read, and remove the marked articles from the summary buffer. Very useful, you'll agree.

Other programs use a totally different kill file syntax. If Gnus encounters what looks like a `rn` kill file, it will take a stab at interpreting it.

Two summary functions for editing a GNUS kill file:

```
M-k      Edit this group's kill file (gnus-summary-edit-local-kill).
M-K      Edit the general kill file (gnus-summary-edit-global-kill).
```

Two group mode functions for editing the kill files:

```
M-k      Edit this group's kill file (gnus-group-edit-local-kill).
M-K      Edit the general kill file (gnus-group-edit-global-kill).
```

Kill file variables:

**gnus-kill-file-name**

A kill file for the group 'soc.motss' is normally called 'soc.motss.KILL'. The suffix appended to the group name to get this file name is detailed by the `gnus-kill-file-name` variable. The "global" kill file (not in the score file sense of "global", of course) is just called 'KILL'.

**gnus-kill-save-kill-file**

If this variable is non-`nil`, Gnus will save the kill file after processing, which is necessary if you use expiring kills.

**gnus-apply-kill-hook**

A hook called to apply kill files to a group. It is `(gnus-apply-kill-file)` by default. If you want to ignore the kill file if you have a score file for the same group, you can set this hook to `(gnus-apply-kill-file-unless-scored)`. If you don't want kill files to be processed, you should set this variable to `nil`.



**gnus-kill-file-mode-hook**

A hook called in kill-file mode buffers.

## 7.14 Converting Kill Files

If you have loads of old kill files, you may want to convert them into score files. If they are “regular”, you can use the ‘`gnus-kill-to-score.el`’ package; if not, you’ll have to do it by hand.

The kill to score conversion package isn’t included in Gnus by default. You can fetch it from <http://www.stud.ifi.uio.no/~larsi/ding-various/gnus-kill-to-score.el>.

If your old kill files are very complex—if they contain more non-`gnus-kill` forms than not, you’ll have to convert them by hand. Or just let them be as they are. Gnus will still use them as before.

## 7.15 GroupLens

NOTE: Unfortunately the GroupLens system seems to have shut down, so this section is mostly of historical interest.

**GroupLens** is a collaborative filtering system that helps you work together with other people to find the quality news articles out of the huge volume of news articles generated every day.

To accomplish this the GroupLens system combines your opinions about articles you have already read with the opinions of others who have done likewise and gives you a personalized prediction for each unread news article. Think of GroupLens as a matchmaker. GroupLens watches how you rate articles, and finds other people that rate articles the same way. Once it has found some people you agree with it tells you, in the form of a prediction, what they thought of the article. You can use this prediction to help you decide whether or not you want to read the article.

### 7.15.1 Using GroupLens

To use GroupLens you must register a pseudonym with your local **Better Bit Bureau (BBB)** is the only better bit in town at the moment.

Once you have registered you’ll need to set a couple of variables.

**gnus-use-grouplens**

Setting this variable to a non-`nil` value will make Gnus hook into all the relevant GroupLens functions.

**grouplens-pseudonym**

This variable should be set to the pseudonym you got when registering with the Better Bit Bureau.

**grouplens-newsgroups**

A list of groups that you want to get GroupLens predictions for.

That’s the minimum of what you need to get up and running with GroupLens. Once you’ve registered, GroupLens will start giving you scores for articles based on the average of what other people think. But, to get the real benefit of GroupLens you need to start rating articles yourself. Then the scores GroupLens gives you will be personalized for you, based on how the people you usually agree with have already rated.



### 7.15.2 Rating Articles

In GroupLens, an article is rated on a scale from 1 to 5, inclusive. Where 1 means something like this article is a waste of bandwidth and 5 means that the article was really good. The basic question to ask yourself is, “on a scale from 1 to 5 would I like to see more articles like this one?”

There are four ways to enter a rating for an article in GroupLens.

- `r`            This function will prompt you for a rating on a scale of one to five.
- `k`            This function will prompt you for a rating, and rate all the articles in the thread. This is really useful for some of those long running giant threads in rec.humor.

The next two commands, `n` and `,`, take a numerical prefix to be the score of the article you’re reading.

- `1-5 n`        Rate the article and go to the next unread article.
- `1-5 ,`        Rate the article and go to the next unread article with the highest score.

If you want to give the current article a score of 4 and then go to the next article, just type `4 n`.

### 7.15.3 Displaying Predictions

GroupLens makes a prediction for you about how much you will like a news article. The predictions from GroupLens are on a scale from 1 to 5, where 1 is the worst and 5 is the best. You can use the predictions from GroupLens in one of three ways controlled by the variable `gnus-grouplens-override-scoring`.

There are three ways to display predictions in grouplens. You may choose to have the GroupLens scores contribute to, or override the regular Gnus scoring mechanism. `override` is the default; however, some people prefer to see the Gnus scores plus the grouplens scores. To get the separate scoring behavior you need to set `gnus-grouplens-override-scoring` to `'separate'`. To have the GroupLens predictions combined with the grouplens scores set it to `'override'` and to combine the scores set `gnus-grouplens-override-scoring` to `'combine'`. When you use the combine option you will also want to set the values for `grouplens-prediction-offset` and `grouplens-score-scale-factor`.

In either case, GroupLens gives you a few choices for how you would like to see your predictions displayed. The display of predictions is controlled by the `grouplens-prediction-display` variable.

The following are valid values for that variable.

- `prediction-spot`  
The higher the prediction, the further to the right an ‘\*’ is displayed.
- `confidence-interval`  
A numeric confidence interval.
- `prediction-bar`  
The higher the prediction, the longer the bar.
- `confidence-bar`  
Numerical confidence.

**confidence-spot**  
The spot gets bigger with more confidence.

**prediction-num**  
Plain-old numeric value.

**confidence-plus-minus**  
Prediction +/- confidence.

### 7.15.4 GroupLens Variables

**gnus-summary-grouplens-line-format**  
The summary line format used in GroupLens-enhanced summary buffers. It accepts the same specs as the normal summary line format (see [\[Summary Buffer Lines\]](#), page [\[undefined\]](#)). The default is ‘%U%R%Z%1%I%(%[%4L:%-23,23n%]) %s\n’.

**grouplens-bbb-host**  
Host running the bbbd server. ‘grouplens.cs.umn.edu’ is the default.

**grouplens-bbb-port**  
Port of the host running the bbbd server. The default is 9000.

**grouplens-score-offset**  
Offset the prediction by this value. In other words, subtract the prediction value by this number to arrive at the effective score. The default is 0.

**grouplens-score-scale-factor**  
This variable allows the user to magnify the effect of GroupLens scores. The scale factor is applied after the offset. The default is 1.

## 7.16 Advanced Scoring

Scoring on Subjects and From headers is nice enough, but what if you’re really interested in what a person has to say only when she’s talking about a particular subject? Or what if you really don’t want to read what person A has to say when she’s following up to person B, but want to read what she says when she’s following up to person C?

By using advanced scoring rules you may create arbitrarily complex scoring patterns.

### 7.16.1 Advanced Scoring Syntax

Ordinary scoring rules have a string as the first element in the rule. Advanced scoring rules have a list as the first element. The second element is the score to be applied if the first element evaluated to a non-`nil` value.

These lists may consist of three logical operators, one redirection operator, and various match operators.

Logical operators:

**&**

**and** This logical operator will evaluate each of its arguments until it finds one that evaluates to `false`, and then it’ll stop. If all arguments evaluate to `true` values, then this operator will return `true`.

|  
**or**            This logical operator will evaluate each of its arguments until it finds one that evaluates to **true**. If no arguments are **true**, then this operator will return **false**.

!  
**not**            This logical operator only takes a single argument. It returns the logical negation of the value of its argument.

There is an *indirection operator* that will make its arguments apply to the ancestors of the current article being scored. For instance, **1-** will make score rules apply to the parent of the current article. **2-** will make score rules apply to the grandparent of the current article. Alternatively, you can write **^^**, where the number of **^**s (carets) says how far back into the ancestry you want to go.

Finally, we have the match operators. These are the ones that do the real work. Match operators are header name strings followed by a match and a match type. A typical match operator looks like **'("from" "Lars Ingebrigtsen" s)'**. The header names are the same as when using simple scoring, and the match types are also the same.

### 7.16.2 Advanced Scoring Examples

Please note that the following examples are score file rules. To make a complete score file from them, surround them with another pair of parentheses.

Let's say you want to increase the score of articles written by Lars when he's talking about Gnus:

```
((&
  ("from" "Lars Ingebrigtsen")
  ("subject" "Gnus"))
  1000)
```

Quite simple, huh?

When he writes long articles, he sometimes has something nice to say:

```
((&
  ("from" "Lars Ingebrigtsen")
  (|
    ("subject" "Gnus")
    ("lines" 100 >)))
  1000)
```

However, when he responds to things written by Reig Eigil Logge, you really don't want to read what he's written:

```
((&
  ("from" "Lars Ingebrigtsen")
  (1- ("from" "Reig Eigir Logge")))
  -100000)
```

Everybody that follows up Redmondo when he writes about disappearing socks should have their scores raised, but only when they talk about white socks. However, when Lars talks about socks, it's usually not very interesting:

```
((&
  (1-
    (&
      ("from" "redmondo@.*no" r)
      ("body" "disappearing.*socks" t)))
  (! ("from" "Lars Ingebrigtsen"))
  ("body" "white.*socks"))
1000)
```

The possibilities are endless.

### 7.16.3 Advanced Scoring Tips

The `&` and `|` logical operators do short-circuit logic. That is, they stop processing their arguments when it's clear what the result of the operation will be. For instance, if one of the arguments of an `&` evaluates to `false`, there's no point in evaluating the rest of the arguments. This means that you should put slow matches (`'body'`, `'header'`) last and quick matches (`'from'`, `'subject'`) first.

The indirection arguments (`1-` and so on) will make their arguments work on previous generations of the thread. If you say something like:

```
...
(1-
  (1-
    ("from" "lars")))
...
```

Then that means “score on the from header of the grandparent of the current article”. An indirection is quite fast, but it's better to say:

```
(1-
  (&
    ("from" "Lars")
    ("subject" "Gnus")))
```

than it is to say:

```
(&
  (1- ("from" "Lars"))
  (1- ("subject" "Gnus")))
```

## 7.17 Score Decays

You may find that your scores have a tendency to grow without bounds, especially if you're using adaptive scoring. If scores get too big, they lose all meaning—they simply max out and it's difficult to use them in any sensible way.

Gnus provides a mechanism for decaying scores to help with this problem. When score files are loaded and `gnus-decay-scores` is non-`nil`, Gnus will run the score files through the decaying mechanism thereby lowering the scores of all non-permanent score rules. The decay itself is performed by the `gnus-decay-score-function` function, which is `gnus-decay-score` by default. Here's the definition of that function:

```
(defun gnus-decay-score (score)
```

```

"Decay SCORE according to 'gnus-score-decay-constant'
and 'gnus-score-decay-scale'."
(let ((n (- score
             (* (if (< score 0) -1 1)
                (min (abs score)
                     (max gnus-score-decay-constant
                          (* (abs score)
                             gnus-score-decay-scale)))))))
      (if (and (featurep 'xemacs)
                ;; XEmacs' floor can handle only the floating point
                ;; number below the half of the maximum integer.
                (> (abs n) (lsh -1 -2)))
          (string-to-number
            (car (split-string (number-to-string n) "\\.")))
          (floor n))))

```

`gnus-score-decay-constant` is 3 by default and `gnus-score-decay-scale` is 0.05. This should cause the following:

1. Scores between -3 and 3 will be set to 0 when this function is called.
2. Scores with magnitudes between 3 and 60 will be shrunk by 3.
3. Scores with magnitudes greater than 60 will be shrunk by 5% of the score.

If you don't like this decay function, write your own. It is called with the score to be decayed as its only parameter, and it should return the new score, which should be an integer.

Gnus will try to decay scores once a day. If you haven't run Gnus for four days, Gnus will decay the scores four times, for instance.



## 8 Message

\input texinfo

# Message Manual

---

by Lars Magne Ingebrigtsen

---

This file documents Message, the Emacs message composition mode.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being “A GNU Manual”, and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License” in the Emacs manual.

(a) The FSF’s Back-Cover Text is: “You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.”

This document is part of a collection distributed under the GNU Free Documentation License. If you want to distribute this document separately from the collection, you can do so by adding a copy of the license to the document, as described in section 6 of the license.



## Message

All message composition from Gnus (both mail and news) takes place in Message mode buffers.

This manual corresponds to Message v5.10.6. Message is distributed with the Gnus distribution bearing the same version number as this manual.



## 9 Interface

When a program (or a person) wants to respond to a message – reply, follow up, forward, cancel – the program (or person) should just put point in the buffer where the message is and call the required command. **Message** will then pop up a new **message** mode buffer with appropriate headers filled out, and the user can edit the message before sending it.

### 9.1 New Mail Message

The **message-mail** command pops up a new message buffer.

Two optional parameters are accepted: The first will be used as the **To** header and the second as the **Subject** header. If these are **nil**, those two headers will be empty.

### 9.2 New News Message

The **message-news** command pops up a new message buffer.

This function accepts two optional parameters. The first will be used as the **Newsgroups** header and the second as the **Subject** header. If these are **nil**, those two headers will be empty.

### 9.3 Reply

The **message-reply** function pops up a message buffer that's a reply to the message in the current buffer.

Message uses the normal methods to determine where replies are to go (see [\[Responses\]](#), page [\(undefined\)](#)), but you can change the behavior to suit your needs by fiddling with the **message-reply-to-function** variable.

If you want the replies to go to the **Sender** instead of the **From**, you could do something like this:

```
(setq message-reply-to-function
  (lambda ()
    (cond ((equal (mail-fetch-field "from") "somebody")
           (list (cons 'To (mail-fetch-field "sender"))))
          (t
           nil))))
```

This function will be called narrowed to the head of the article that is being replied to.

As you can see, this function should return a list. In this case, it returns `((To . "Whom"))` if it has an opinion as to what the **To** header should be. If it does not, it should just return **nil**, and the normal methods for determining the **To** header will be used.

Each list element should be a cons, where the **CAR** should be the name of a header (e.g. **Cc**) and the **CDR** should be the header value (e.g. `'larsi@ifi.uio.no'`). All these headers will be inserted into the head of the outgoing mail.

## 9.4 Wide Reply

The `message-wide-reply` pops up a message buffer that's a wide reply to the message in the current buffer. A *wide reply* is a reply that goes out to all people listed in the **To**, **From** (or **Reply-to**) and **Cc** headers.

Message uses the normal methods to determine where wide replies are to go, but you can change the behavior to suit your needs by fiddling with the `message-wide-reply-to-function`. It is used in the same way as `message-reply-to-function` (see [\[Reply\]](#), page [\(undefined\)](#)).

Addresses that match the `message-dont-reply-to-names` regular expression will be removed from the **Cc** header.

If `message-wide-reply-confirm-recipients` is non-`nil` you will be asked to confirm that you want to reply to multiple recipients. The default is `nil`.

## 9.5 Followup

The `message-followup` command pops up a message buffer that's a followup to the message in the current buffer.

Message uses the normal methods to determine where followups are to go, but you can change the behavior to suit your needs by fiddling with the `message-followup-to-function`. It is used in the same way as `message-reply-to-function` (see [\[Reply\]](#), page [\(undefined\)](#)).

The `message-use-followup-to` variable says what to do about **Followup-To** headers. If it is `use`, always use the value. If it is `ask` (which is the default), ask whether to use the value. If it is `t`, use the value unless it is `'poster'`. If it is `nil`, don't use the value.

## 9.6 Canceling News

The `message-cancel-news` command cancels the article in the current buffer.

The value of `message-cancel-message` is inserted in the body of the cancel message. The default is `'I am canceling my own article.'`

When Message posts news messages, it inserts **Cancel-Lock** headers by default. This is a cryptographic header that ensures that only you can cancel your own messages, which is nice. The downside is that if you lose your `‘.emacs’` file (which is where Gnus stores the secret cancel lock password (which is generated automatically the first time you use this feature)), you won't be able to cancel your message. If you want to manage a password yourself, you can put something like the following in your `‘~/gnus.el’` file:

```
(setq canlock-password "geheimnis"
      canlock-password-for-verify canlock-password)
```

Whether to insert the header or not is controlled by the `message-insert-canlock` variable.

Not many news servers respect the **Cancel-Lock** header yet, but this is expected to change in the future.

## 9.7 Superseding

The `message-supersede` command pops up a message buffer that will supersede the message in the current buffer.

Headers matching the `message-ignored-supersedes-headers` are removed before popping up the new message buffer. The default is

```
'^Path:\\|^Date:\\|^NNTP-Posting-Host:\\|^Xref:\\|^Lines:\\|^Received:\\|^X-From-Line:\\|^Return-Path:\\|^Supersedes:'.
```

## 9.8 Forwarding

The `message-forward` command pops up a message buffer to forward the message in the current buffer. If given a prefix, forward using news.

`message-forward-ignored-headers`

All headers that match this regexp will be deleted when forwarding a message.

`message-make-forward-subject-function`

A list of functions that are called to generate a subject header for forwarded messages. The subject generated by the previous function is passed into each successive function.

The provided functions are:

`message-forward-subject-author-subject`

Source of article (author or newsgroup), in brackets followed by the subject.

`message-forward-subject-fwd`

Subject of article with 'Fwd:' prepended to it.

`message-wash-forwarded-subjects`

If this variable is `t`, the subjects of forwarded messages have the evidence of previous forwards (such as 'Fwd:', 'Re:', '(fwd)') removed before the new subject is constructed. The default value is `nil`.

`message-forward-as-mime`

If this variable is `t` (the default), forwarded messages are included as inline MIME RFC822 parts. If it's `nil`, forwarded messages will just be copied inline to the new message, like previous, non MIME-savvy versions of Gnus would do.

`message-forward-before-signature`

If non-`nil`, put forwarded message before signature, else after.

## 9.9 Resending

The `message-resend` command will prompt the user for an address and resend the message in the current buffer to that address.

Headers that match the `message-ignored-resent-headers` regexp will be removed before sending the message.

## 9.10 Bouncing

The `message-bounce` command will, if the current buffer contains a bounced mail message, pop up a message buffer stripped of the bounce information. A *bounced message* is typically a mail you've sent out that has been returned by some `mailer-daemon` as undeliverable.

Headers that match the `message-ignored-bounced-headers` regexp will be removed before popping up the buffer. The default is `^\\(Received\\|Return-Path\\):'`.

## 9.11 Mailing Lists

Sometimes while posting to mailing lists, the poster needs to direct followups to the post to specific places. The Mail-Followup-To (MFT) was created to enable just this. Two example scenarios where this is useful:

- A mailing list poster can use MFT to express that responses should be sent to just the list, and not the poster as well. This will happen if the poster is already subscribed to the list.
- A mailing list poster can use MFT to express that responses should be sent to the list and the poster as well. This will happen if the poster is not subscribed to the list.
- If a message is posted to several mailing lists, MFT may also be used to direct the following discussion to one list only, because discussions that are spread over several lists tend to be fragmented and very difficult to follow.

Gnus honors the MFT header in other's messages (i.e. while following up to someone else's post) and also provides support for generating sensible MFT headers for outgoing messages as well.

### 9.11.1 Composing a correct MFT header automatically

The first step in getting Gnus to automatically generate a MFT header in posts you make is to give Gnus a list of the mailing lists addresses you are subscribed to. You can do this in more than one way. The following variables would come in handy.

`message-subscribed-addresses`

This should be a list of addresses the user is subscribed to. Its default value is `nil`. Example:

```
(setq message-subscribed-addresses
      '("ding@gnus.org" "bing@noose.org"))
```

`message-subscribed-regexps`

This should be a list of regexps denoting the addresses of mailing lists subscribed to. Default value is `nil`. Example: If you want to achieve the same result as above:

```
(setq message-subscribed-regexps
      '("\\(ding@gnus\\|bing@noose\\)\\.org"))
```

`message-subscribed-address-functions`

This can be a list of functions to be called (one at a time!!) to determine the value of MFT headers. It is advisable that these functions not take any arguments. Default value is `nil`.

There is a pre-defined function in Gnus that is a good candidate for this variable. `gnus-find-subscribed-addresses` is a function that returns a list of addresses corresponding to the groups that have the `subscribed` (see [section “Group Parameters” in \*The Gnus Manual\*](#)) group parameter set to a non-`nil` value. This is how you would do it.

```
(setq message-subscribed-address-functions
      '(gnus-find-subscribed-addresses))
```

#### `message-subscribed-address-file`

You might be one organised human freak and have a list of addresses of all subscribed mailing lists in a separate file! Then you can just set this variable to the name of the file and life would be good.

You can use one or more of the above variables. All their values are “added” in some way that works :-)

Now you are all set. Just start composing a message as you normally do. And just send it; as always. Just before the message is sent out, Gnus’ MFT generation thingy kicks in and checks if the message already has a MFT field. If there is one, it is left alone. (Except if it’s empty - in that case, the field is removed and is not replaced with an automatically generated one. This lets you disable MFT generation on a per-message basis.) If there is none, then the list of recipient addresses (in the To: and Cc: headers) is checked to see if one of them is a list address you are subscribed to. If none of them is a list address, then no MFT is generated; otherwise, a MFT is added to the other headers and set to the value of all addresses in To: and Cc:

Hm. “So”, you ask, “what if I send an email to a list I am not subscribed to? I want my MFT to say that I want an extra copy.” (This is supposed to be interpreted by others the same way as if there were no MFT, but you can use an explicit MFT to override someone else’s to-address group parameter.) The function `message-generate-unsubscribed-mail-followup-to` might come in handy. It is bound to `C-c C-f C-a` by default. In any case, you can insert a MFT of your own choice; `C-c C-f C-m` (`message-goto-mail-followup-to`) will help you get started.

### 9.11.2 Honoring an MFT post

When you followup to a post on a mailing list, and the post has a MFT header, Gnus’ action will depend on the value of the variable `message-use-mail-followup-to`. This variable can be one of:

<code>use</code>	Always honor MFTs. The To: and Cc: headers in your followup will be derived from the MFT header of the original post. This is the default.
<code>nil</code>	Always dishonor MFTs (just ignore the darned thing)
<code>ask</code>	Gnus will prompt you for an action.

It is considered good netiquette to honor MFT, as it is assumed the fellow who posted a message knows where the followups need to go better than you do.





## 10 Commands

### 10.1 Buffer Entry

You most often end up in a Message buffer when responding to some other message of some sort. Message does lots of handling of quoted text, and may remove signatures, reformat the text, or the like—depending on which used settings you’re using. Message usually gets things right, but sometimes it stumbles. To help the user unwind these stumblings, Message sets the undo boundary before each major automatic action it takes. If you press the undo key (usually located at `C-␣`) a few times, you will get back the un-edited message you’re responding to.

### 10.2 Header Commands

#### 10.2.1 Commands for moving to headers

These following commands move to the header in question. If it doesn’t exist, it will be inserted.

`C-c ?` Describe the message mode.

`C-c C-f C-t`  
Go to the To header (`message-goto-to`).

`C-c C-f C-o`  
Go to the From header (`message-goto-from`). (The “o” in the key binding is for Originator.)

`C-c C-f C-b`  
Go to the Bcc header (`message-goto-bcc`).

`C-c C-f C-f`  
Go to the Fcc header (`message-goto-fcc`).

`C-c C-f C-c`  
Go to the Cc header (`message-goto-cc`).

`C-c C-f C-s`  
Go to the Subject header (`message-goto-subject`).

`C-c C-f C-r`  
Go to the Reply-To header (`message-goto-reply-to`).

`C-c C-f C-n`  
Go to the Newsgroups header (`message-goto-newsgroups`).

`C-c C-f C-d`  
Go to the Distribution header (`message-goto-distribution`).

`C-c C-f C-o`  
Go to the Followup-To header (`message-goto-followup-to`).

`C-c C-f C-k`  
Go to the Keywords header (`message-goto-keywords`).

*C-c C-f C-u*

Go to the Summary header (`message-goto-summary`).

*C-c C-f C-i*

This inserts the ‘Importance:’ header with a value of ‘high’. This header is used to signal the importance of the message to the receiver. If the header is already present in the buffer, it cycles between the three valid values according to RFC 1376: ‘low’, ‘normal’ and ‘high’.

*C-c C-f C-a*

Insert a reasonable ‘Mail-Followup-To:’ header (see [\[Mailing Lists\]](#), page [\[undefined\]](#)) in a post to an unsubscribed list. When making original posts to a mailing list you are not subscribed to, you have to type in a ‘Mail-Followup-To:’ header by hand. The contents, usually, are the addresses of the list and your own address. This function inserts such a header automatically. It fetches the contents of the ‘To:’ header in the current mail buffer, and appends the current `user-mail-address`.

If the optional argument `include-cc` is non-`nil`, the addresses in the ‘Cc:’ header are also put into the ‘Mail-Followup-To:’ header.

## 10.2.2 Commands to change headers

*C-c C-o* Sort headers according to `message-header-format-alist` (`message-sort-headers`).

*C-c C-t* Insert a To header that contains the Reply-To or From header of the message you’re following up (`message-insert-to`).

*C-c C-n* Insert a Newsgroups header that reflects the Followup-To or Newsgroups header of the article you’re replying to (`message-insert-newsgroups`).

*C-c C-l* Send a message to the list only. Remove all addresses but the list address from To: and Cc: headers.

*C-c M-n* Insert a request for a disposition notification. (`message-insert-disposition-notification-to`). This means that if the recipient support RFC 2298 she might send you a notification that she received the message.

*M-x message-insert-importance-high*

Insert an ‘Importance’ header with a value of ‘high’, deleting headers if necessary.

*M-x message-insert-importance-low*

Insert an ‘Importance’ header with a value of ‘low’, deleting headers if necessary.

*C-c C-f s* Change the current ‘Subject’ header. Ask for new ‘Subject’ header and append ‘(was: <Old Subject>)’. The old subject can be stripped on replying, see `message-subject-trailing-was-query` (see [\[Message Headers\]](#), page [\[undefined\]](#)).

*C-c C-f x* Ask for an additional ‘Newsgroups’ and ‘FollowUp-To’ for a cross-post. `message-cross-post-followup-to` mangles ‘FollowUp-To’ and ‘Newsgroups’

header to point to group. If `message-cross-post-default` is `nil` or if called with a prefix-argument ‘Follow-Up’ is set, but the message is not cross-posted.

- `C-c C-f t` Replace contents of ‘To’ header with contents of ‘Cc’ or ‘Bcc’ header.
- `C-c C-f w` Insert ‘To’ and ‘Cc’ headers as if you were doing a wide reply.
- `C-c C-f a` Insert ‘X-No-Archive: Yes’ in the header and a note in the body. The header and the note can be customized using `message-archive-header` and `message-archive-note`. When called with a prefix argument, ask for a text to insert. If you don’t want the note in the body, set `message-archive-note` to `nil`.

### 10.3 Movement

- `C-c C-b` Move to the beginning of the body of the message (`message-goto-body`).
- `C-c C-i` Move to the signature of the message (`message-goto-signature`).
- `C-a` If at beginning of header value, go to beginning of line, else go to beginning of header value. (The header value comes after the header name and the colon.) This behaviour can be disabled by toggling the variable `message-beginning-of-line`.

### 10.4 Insertion

- `C-c C-y` Yank the message that’s being replied to into the message buffer (`message-yank-original`).
- `C-c C-M-y` Prompt for a buffer name and yank the contents of that buffer into the message buffer (`message-yank-buffer`).
- `C-c C-q` Fill the yanked message (`message-fill-yanked-message`). Warning: Can severely mess up the yanked text if its quoting conventions are strange. You’ll quickly get a feel for when it’s safe, though. Anyway, just remember that `C-x u` (undo) is available and you’ll be all right.
- `C-c C-w` Insert a signature at the end of the buffer (`message-insert-signature`).
- `C-c M-h` Insert the message headers (`message-insert-headers`).
- `C-c M-m` Mark some region in the current article with enclosing tags. See `message-mark-insert-begin` and `message-mark-insert-end`.
- `C-c M-f` Insert a file in the current article with enclosing tags. See `message-mark-insert-begin` and `message-mark-insert-end`.

### 10.5 MIME

Message is a MIME-compliant posting agent. The user generally doesn’t have to do anything to make the MIME happen—Message will automatically add the `Content-Type` and `Content-Transfer-Encoding` headers.

The most typical thing users want to use the multipart things in MIME for is to add “attachments” to mail they send out. This can be done with the `C-c C-a` command, which will prompt for a file name and a MIME type.

You can also create arbitrarily complex multiparts using the MML language (see [section “Composing”](#) in *The Emacs MIME Manual*).

## 10.6 IDNA

Message is a IDNA-compliant posting agent. The user generally doesn't have to do anything to make the IDNA happen—Message will encode non-ASCII domain names in **From**, **To**, and **Cc** headers automatically.

Until IDNA becomes more well known, Message queries you whether IDNA encoding of the domain name really should occur. Some users might not be aware that domain names can contain non-ASCII now, so this gives them a safety net if they accidentally typed a non-ASCII domain name.

The `message-use-idna` variable control whether IDNA is used. If the variable is `nil` no IDNA encoding will ever happen, if it is set to the symbol `ask` the user will be queried (the default), and if set to `t` IDNA encoding happens automatically.

If you want to experiment with the IDNA encoding, you can invoke `M-x message-idna-to-ascii-rhs RET` in the message buffer to have the non-ASCII domain names encoded while you edit the message.

Note that you must have **GNU Libidn** installed in order to use this functionality.

## 10.7 Security

Using the MML language, Message is able to create digitally signed and digitally encrypted messages. Message (or rather MML) currently support PGP (RFC 1991), PGP/MIME (RFC 2015/3156) and S/MIME. Instructing MML to perform security operations on a MIME part is done using the `C-c C-m s` key map for signing and the `C-c C-m c` key map for encryption, as follows.

`C-c C-m s s`

Digitally sign current message using S/MIME.

`C-c C-m s o`

Digitally sign current message using PGP.

`C-c C-m s p`

Digitally sign current message using PGP/MIME.

`C-c C-m c s`

Digitally encrypt current message using S/MIME.

`C-c C-m c o`

Digitally encrypt current message using PGP.

`C-c C-m c p`

Digitally encrypt current message using PGP/MIME.

`C-c C-m C-n`

Remove security related MML tags from message.

These commands do not immediately sign or encrypt the message, they merely insert the proper MML secure tag to instruct the MML engine to perform that operation when the message is actually sent. They may perform other operations too, such as locating and retrieving a S/MIME certificate of the person you wish to send encrypted mail to. When the mml parsing engine converts your MML into a properly encoded MIME message, the

secure tag will be replaced with either a part or a multipart tag. If your message contains other mml parts, a multipart tag will be used; if no other parts are present in your message a single part tag will be used. This way, message mode will do the Right Thing (TM) with signed/encrypted multipart messages.

Since signing and especially encryption often is used when sensitive information is sent, you may want to have some way to ensure that your mail is actually signed or encrypted. After invoking the above sign/encrypt commands, it is possible to preview the raw article by using `C-u C-c RET P` (`mml-preview`). Then you can verify that your long rant about what your ex-significant other or whomever actually did with that funny looking person at that strange party the other night, actually will be sent encrypted.

*Note!* Neither PGP/MIME nor S/MIME encrypt/signs RFC822 headers. They only operate on the MIME object. Keep this in mind before sending mail with a sensitive Subject line.

By default, when encrypting a message, Gnus will use the “signencrypt” mode, which means the message is both signed and encrypted. If you would like to disable this for a particular message, give the `mml-secure-message-encrypt-*` command a prefix argument, e.g., `C-u C-c C-m c p`.

Actually using the security commands above is not very difficult. At least not compared with making sure all involved programs talk with each other properly. Thus, we now describe what external libraries or programs are required to make things work, and some small general hints.

### 10.7.1 Using S/MIME

*Note!* This section assume you have a basic familiarity with modern cryptography, S/MIME, various PKCS standards, OpenSSL and so on.

The S/MIME support in Message (and MML) require OpenSSL. OpenSSL performs the actual S/MIME sign/encrypt operations. OpenSSL can be found at <http://www.openssl.org/>. OpenSSL 0.9.6 and later should work. Version 0.9.5a cannot extract mail addresses from certificates, and it insert a spurious CR character into MIME separators so you may wish to avoid it if you would like to avoid being regarded as someone who send strange mail. (Although by sending S/MIME messages you’ve probably already lost that contest.)

To be able to send encrypted mail, a personal certificate is not required. Message (MML) need a certificate for the person to whom you wish to communicate with though. You’re asked for this when you type `C-c C-m c s`. Currently there are two ways to retrieve this certificate, from a local file or from DNS. If you chose a local file, it need to contain a X.509 certificate in PEM format. If you chose DNS, you’re asked for the domain name where the certificate is stored, the default is a good guess. To my belief, Message (MML) is the first mail agent in the world to support retrieving S/MIME certificates from DNS, so you’re not likely to find very many certificates out there. At least there should be one, stored at the domain `simon.josefsson.org`. LDAP is a more popular method of distributing certificates, support for it is planned. (Meanwhile, you can use `ldapsearch` from the command line to retrieve a certificate into a file and use it.)

As for signing messages, OpenSSL can’t perform signing operations without some kind of configuration. Especially, you need to tell it where your private key and your certificate

is stored. MML uses an Emacs interface to OpenSSL, aptly named `smime.el`, and it contains a `custom` group used for this configuration. So, try `M-x customize-group RET smime RET` and look around.

Currently there is no support for talking to a CA (or RA) to create your own certificate. None is planned either. You need to do this manually with OpenSSL or using some other program. I used Netscape and got a free S/MIME certificate from one of the big CA's on the net. Netscape is able to export your private key and certificate in PKCS #12 format. Use OpenSSL to convert this into a plain X.509 certificate in PEM format as follows.

```
$ openssl pkcs12 -in ns.p12 -clcerts -nodes > key+cert.pem
```

The `'key+cert.pem'` file should be pointed to from the `smime-keys` variable. You should now be able to send signed mail.

*Note!* Your private key is now stored unencrypted in the file, so take care in handling it. Storing encrypted keys on the disk are supported, and Gnus will ask you for a passphrase before invoking OpenSSL. Read the OpenSSL documentation for how to achieve this. If you use unencrypted keys (e.g., if they are on a secure storage, or if you are on a secure single user machine) simply press `RET` at the passphrase prompt.

### 10.7.2 Using PGP/MIME

PGP/MIME requires an external OpenPGP implementation, such as [GNU Privacy Guard](#). Pre-OpenPGP implementations such as PGP 2.x and PGP 5.x are also supported. One Emacs interface to the PGP implementations, PGG (see [section "PGG" in PGG Manual](#)), is included, but Mailcrypt and Florian Weimer's `gpg.el` are also supported.

Note, if you are using the `gpg.el` you must make sure that the directory specified by `gpg-temp-directory` have permissions 0700.

Creating your own key is described in detail in the documentation of your PGP implementation, so we refer to it.

If you have imported your old PGP 2.x key into GnuPG, and want to send signed and encrypted messages to your fellow PGP 2.x users, you'll discover that the receiver cannot understand what you send. One solution is to use PGP 2.x instead (i.e., if you use `pgg`, set `pgg-default-scheme` to `pgp`). If you do want to use GnuPG, you can use a compatibility script called `gpg-2comp` available from <http://muppet.faveve.uni-stuttgart.de/~gero/gpg-2comp/>. You could also convince your fellow PGP 2.x users to convert to GnuPG. As a final workaround, you can make the sign and encryption work in two steps; separately sign, then encrypt a message. If you would like to change this behavior you can customize the `mml-signencrypt-style-alist` variable. For example:

```
(setq mml-signencrypt-style-alist '(("smime" separate)
                                   ("pgp" separate)
                                   ("pgpauto" separate)
                                   ("pgpmime" separate)))
```

This causes to sign and encrypt in two passes, thus generating a message that can be understood by PGP version 2.

(Refer to <http://www.gnupg.org/gph/en/pgp2x.html> for more information about the problem.)

## 10.8 Various Commands

- C-c C-r** Caesar rotate (aka. `rot13`) the current message (`message-caesar-buffer-body`). If narrowing is in effect, just rotate the visible portion of the buffer. A numerical prefix says how many places to rotate the text. The default is 13.
- C-c C-e** Elide the text between point and mark (`message-elide-region`). The text is killed and replaced with the contents of the variable `message-elide-ellipsis`. The default value is to use an ellipsis (`'[...]'`).
- C-c C-z** Kill all the text up to the signature, or if that's missing, up to the end of the message (`message-kill-to-signature`).
- C-c C-v** Delete all text in the body of the message that is outside the region (`message-delete-not-region`).
- M-RET** Insert four newlines, and then reformat if inside quoted text.
- Here's an example:
- ```
> This is some quoted text.  And here's more quoted text.
```
- If point is before 'And' and you press **M-RET**, you'll get:
- ```
> This is some quoted text.
```
- \*
- ```
> And here's more quoted text.
```
- '\*' says where point will be placed.
- C-c M-r** Rename the buffer (`message-rename-buffer`). If given a prefix, prompt for a new buffer name.
- TAB** If non-`nil` execute the function specified in `message-tab-body-function`. Otherwise use the function bound to **TAB** in `text-mode-map` or `global-map`.

## 10.9 Sending

- C-c C-c** Send the message and bury the current buffer (`message-send-and-exit`).
- C-c C-s** Send the message (`message-send`).
- C-c C-d** Bury the message buffer and exit (`message-dont-send`).
- C-c C-k** Kill the message buffer and exit (`message-kill-buffer`).

## 10.10 Mail Aliases

The `message-mail-alias-type` variable controls what type of mail alias expansion to use. Currently only one form is supported—Message uses `mailabbrev` to handle mail aliases. If this variable is `nil`, no mail alias expansion will be performed.

`mailabbrev` works by parsing the `'/etc/mailrc'` and `'~/mailrc'` files. These files look like:

```
alias lmi "Lars Magne Ingebrigtsen <larsi@ifi.uio.no>"
alias ding "ding@ifi.uio.no (ding mailing list)"
```



After adding lines like this to your ‘`~/mailrc`’ file, you should be able to just write ‘`lmi`’ in the `To` or `Cc` (and so on) headers and press *SPC* to expand the alias.

No expansion will be performed upon sending of the message—all expansions have to be done explicitly.

## 10.11 Spelling

There are two popular ways to have Emacs spell-check your messages: `ispell` and `flyspell`. `ispell` is the older and probably more popular package. You typically first write the message, and then run the entire thing through `ispell` and fix all the typos. To have this happen automatically when you send a message, put something like the following in your ‘`.emacs`’ file:

```
(add-hook 'message-send-hook 'ispell-message)
```

If you’re in the habit of writing in different languages, this can be controlled by the `ispell-message-dictionary-alist` variable:

```
(setq ispell-message-dictionary-alist
      '(("^[^Newsgroups:.*\\bde\\.]" . "deutsch8")
        (".*" . "default")))
```

`ispell` depends on having the external ‘`ispell`’ command installed.

The other popular method is using `flyspell`. This package checks your spelling while you’re writing, and marks any mis-spelled words in various ways.

To use `flyspell`, put something like the following in your ‘`.emacs`’ file:

```
(defun my-message-setup-routine ()
  (flyspell-mode 1))
(add-hook 'message-setup-hook 'my-message-setup-routine)
```

`flyspell` depends on having the external ‘`ispell`’ command installed.



## 11 Variables

### 11.1 Message Headers

Message is quite aggressive on the message generation front. It has to be – it’s a combined news and mail agent. To be able to send combined messages, it has to generate all headers itself (instead of letting the mail/news system do it) to ensure that mail and news copies of messages look sufficiently similar.

#### `message-generate-headers-first`

If `t`, generate all required headers before starting to compose the message. This can also be a list of headers to generate:

```
(setq message-generate-headers-first
      '(References))
```

The variables `message-required-headers`, `message-required-mail-headers` and `message-required-news-headers` specify which headers are required.

Note that some headers will be removed and re-generated before posting, because of the variable `message-deletable-headers` (see below).

#### `message-draft-headers`

When running Message from Gnus, the message buffers are associated with a draft group. `message-draft-headers` says which headers should be generated when a draft is written to the draft group.

#### `message-from-style`

Specifies how From headers should look. There are four valid values:

|                      |                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>nil</code>     | Just the address – ‘king@grassland.com’.                                                                                                                                         |
| <code>parens</code>  | ‘king@grassland.com (Elvis Parsley)’.                                                                                                                                            |
| <code>angles</code>  | ‘Elvis Parsley <king@grassland.com>’.                                                                                                                                            |
| <code>default</code> | Look like <code>angles</code> if that doesn’t require quoting, and <code>parens</code> if it does. If even <code>parens</code> requires quoting, use <code>angles</code> anyway. |

#### `message-deletable-headers`

Headers in this list that were previously generated by Message will be deleted before posting. Let’s say you post an article. Then you decide to post it again to some other group, you naughty boy, so you jump back to the `*post-buf*` buffer, edit the `Newsgroups` line, and ship it off again. By default, this variable makes sure that the old generated `Message-ID` is deleted, and a new one generated. If this isn’t done, the entire empire would probably crumble, anarchy would prevail, and cats would start walking on two legs and rule the world. Allegedly.

#### `message-default-headers`

This string is inserted at the end of the headers in all message buffers.

#### `message-subject-re-regexp`

Responses to messages have subjects that start with ‘**Re:**’. This is *not* an abbreviation of the English word “response”, but is Latin, and means “in response to”. Some illiterate nincompoops have failed to grasp this fact, and have

“internationalized” their software to use abominations like ‘Aw: ’ (“antwort”) or ‘Sv: ’ (“svar”) instead, which is meaningless and evil. However, you may have to deal with users that use these evil tools, in which case you may set this variable to a regexp that matches these prefixes. Myself, I just throw away non-compliant mail.

Here’s an example of a value to deal with these headers when responding to a message:

```
(setq message-subject-re-regexp
  (concat
    "^[ \t]*"
    "\\("
    "\\("
      "[Aa] [Nn] [Tt] [Ww]\\.?.\\|" ; antw
      "[Aa] [Ww]\\.\\|" ; aw
      "[Ff] [Ww] [Dd]\\.?.\\|" ; fwd
      "[Oo] [Dd] [Pp]\\.\\|" ; odp
      "[Rr] [Ee]\\.\\|" ; re
      "[Rr] [311\\351] [Ff]\\.?.\\|" ; ref
      "[Ss] [Vv]" ; sv
    "\\)"
    "\\(\\[[0-9]*\\)\\)"
    "*: [ \t]*"
    "\\)"
    "*[ \t]*"
  ))
```

#### message-subject-trailing-was-query

Controls what to do with trailing ‘(was: <old subject>)’ in subject lines. If `nil`, leave the subject unchanged. If it is the symbol `ask`, query the user what to do. In this case, the subject is matched against `message-subject-trailing-was-ask-regexp`. If `message-subject-trailing-was-query` is `t`, always strip the trailing old subject. In this case, `message-subject-trailing-was-regexp` is used.

#### message-alternative-emails

A regexp to match the alternative email addresses. The first matched address (not primary one) is used in the From field.

#### message-allow-no-recipients

Specifies what to do when there are no recipients other than `Gcc` or `Fcc`. If it is `always`, the posting is allowed. If it is `never`, the posting is not allowed. If it is `ask` (the default), you are prompted.

#### message-hidden-headers

A regexp, a list of regexps, or a list where the first element is `not` and the rest are regexps. It says which headers to keep hidden when composing a message.

```
(setq message-hidden-headers
  '(not "From" "Subject" "To" "Cc" "Newsgroups"))
```

**message-header-synonyms**

A list of lists of header synonyms. E.g., if this list contains a member list with elements Cc and To, then `message-carefully-insert-headers` will not insert a To header when the message is already Cced to the recipient.

**11.2 Mail Headers****message-required-mail-headers**

See [\[News Headers\]](#), page [\[undefined\]](#), for the syntax of this variable. It is (From Date Subject (optional . In-Reply-To) Message-ID Lines (optional . User-Agent)) by default.

**message-ignored-mail-headers**

Regexp of headers to be removed before mailing. The default is `‘^[GF]cc:\\|^Resent-Fcc:\\|^Xref:\\|^X-Draft-From:’`.

**message-default-mail-headers**

This string is inserted at the end of the headers in all message buffers that are initialized as mail.

**11.3 Mail Variables****message-send-mail-function**

Function used to send the current buffer as mail. The default is `message-send-mail-with-sendmail`. Other valid values include `message-send-mail-with-mh`, `message-send-mail-with-qmail`, `message-smtpmail-send-it`, `smtpmail-send-it` and `feedmail-send-it`.

**message-mh-deletable-headers**

Most versions of MH doesn't like being fed messages that contain the headers in this variable. If this variable is non-`nil` (which is the default), these headers will be removed before mailing when sending messages via MH. Set it to `nil` if your MH can handle these headers.

**message-qmail-inject-program**

Location of the qmail-inject program.

**message-qmail-inject-args**

Arguments passed to qmail-inject programs. This should be a list of strings, one string for each argument. It may also be a function.

For e.g., if you wish to set the envelope sender address so that bounces go to the right place or to deal with listserv's usage of that address, you might set this variable to `'("-f" "you@some.where")`.

**message-sendmail-f-is-evil**

Non-`nil` means don't add `'-f username'` to the sendmail command line. Doing so would be even more evil than leaving it out.

**message-sendmail-envelope-from**

When `message-sendmail-f-is-evil` is `nil`, this specifies the address to use in the SMTP envelope. If it is `nil`, use `user-mail-address`. If it is the symbol `header`, use the 'From' header of the message.

**message-mailer-swallows-blank-line**

Set this to non-`nil` if the system's mailer runs the header and body together. (This problem exists on SunOS 4 when sendmail is run in remote mode.) The value should be an expression to test whether the problem will actually occur.

**message-send-mail-partially-limit**

The limitation of messages sent as message/partial. The lower bound of message size in characters, beyond which the message should be sent in several parts. If it is `nil`, the size is unlimited.

## 11.4 News Headers

**message-required-news-headers** a list of header symbols. These headers will either be automatically generated, or, if that's impossible, they will be prompted for. The following symbols are valid:

**From** This required header will be filled out with the result of the **message-make-from** function, which depends on the **message-from-style**, **user-full-name**, **user-mail-address** variables.

**Subject** This required header will be prompted for if not present already.

**Newsgroups**

This required header says which newsgroups the article is to be posted to. If it isn't present already, it will be prompted for.

**Organization**

This optional header will be filled out depending on the **message-user-organization** variable. **message-user-organization-file** will be used if this variable is `t`. This variable can also be a string (in which case this string will be used), or it can be a function (which will be called with no parameters and should return a string to be used).

**Lines** This optional header will be computed by Message.

**Message-ID**

This required header will be generated by Message. A unique ID will be created based on the date, time, user name (for the local part) and the domain part. For the domain part, message will look (in this order) at **message-user-fqdn**, **system-name**, **mail-host-address** and **message-user-mail-address** (i.e. **user-mail-address**) until a probably valid fully qualified domain name (FQDN) was found.

**User-Agent**

This optional header will be filled out according to the **message-newsreader** local variable.

**In-Reply-To**

This optional header is filled out using the **Date** and **From** header of the article being replied to.

**Expires** This extremely optional header will be inserted according to the **message-expires** variable. It is highly deprecated and shouldn't be used unless you know what you're doing.

**Distribution**

This optional header is filled out according to the `message-distribution-function` variable. It is a deprecated and much misunderstood header.

**Path**

This extremely optional header should probably never be used. However, some *very* old servers require that this header is present. `message-user-path` further controls how this `Path` header is to look. If it is `nil`, use the server name as the leaf node. If it is a string, use the string. If it is neither a string nor `nil`, use the user name only. However, it is highly unlikely that you should need to fiddle with this variable at all.

In addition, you can enter conses into this list. The CAR of this cons should be a symbol. This symbol's name is the name of the header, and the CDR can either be a string to be entered verbatim as the value of this header, or it can be a function to be called. This function should return a string to be inserted. For instance, if you want to insert `Mime-Version: 1.0`, you should enter `(Mime-Version . "1.0")` into the list. If you want to insert a funny quote, you could enter something like `(X-Yow . yow)` into the list. The function `yow` will then be called without any arguments.

If the list contains a cons where the CAR of the cons is `optional`, the CDR of this cons will only be inserted if it is non-`nil`.

If you want to delete an entry from this list, the following Lisp snippet might be useful. Adjust accordingly if you want to remove another element.

```
(setq message-required-news-headers
      (delq 'Message-ID message-required-news-headers))
```

Other variables for customizing outgoing news articles:

**message-syntax-checks**

Controls what syntax checks should not be performed on outgoing posts. To disable checking of long signatures, for instance, add

```
(signature . disabled)
```

to this list.

Valid checks are:

**subject-cmsg**

Check the subject for commands.

**sender**     Insert a new **Sender** header if the **From** header looks odd.

**multiple-headers**

Check for the existence of multiple equal headers.

**sendsys**    Check for the existence of version and sendsys commands.

**message-id**

Check whether the **Message-ID** looks ok.

**from**        Check whether the **From** header seems nice.

**long-lines**

Check for too long lines.

**control-chars** Check for invalid characters.

**size** Check for excessive size.

**new-text** Check whether there is any new text in the messages.

**signature** Check the length of the signature.

**approved** Check whether the article has an **Approved** header, which is something only moderators should include.

**empty** Check whether the article is empty.

**invisible-text** Check whether there is any invisible text in the buffer.

**empty-headers** Check whether any of the headers are empty.

**existing-newsgroups** Check whether the newsgroups mentioned in the **Newsgroups** and **Followup-To** headers exist.

**valid-newsgroups** Check whether the **Newsgroups** and **Followup-to** headers are valid syntactically.

**repeated-newsgroups** Check whether the **Newsgroups** and **Followup-to** headers contains repeated group names.

**shorten-followup-to** Check whether to add a **Followup-to** header to shorten the number of groups to post to.

All these conditions are checked by default.

**message-ignored-news-headers** Regexp of headers to be removed before posting. The default is `‘^NNTP-Posting-Host:\\|^Xref:\\|^BGF cc:\\|^Resent-Fcc:\\|^X-Draft-From:’`.

**message-default-news-headers** This string is inserted at the end of the headers in all message buffers that are initialized as news.

## 11.5 News Variables

**message-send-news-function** Function used to send the current buffer as news. The default is **message-send-news**.

**message-post-method** Gnusish *select method* (see the Gnus manual for details) used for posting a prepared news message.

## 11.6 Insertion Variables

### `message-ignored-cited-headers`

All headers that match this regexp will be removed from yanked messages. The default is ‘.’, which means that all headers will be removed.

### `message-cite-prefix-regexp`

Regexp matching the longest possible citation prefix on a line.

### `message-citation-line-function`

Function called to insert the citation line. The default is `message-insert-citation-line`, which will lead to citation lines that look like:

Hallvard B Furuseth <h.b.furuseth@usit.uio.no> writes:

Point will be at the beginning of the body of the message when this function is called.

Note that Gnus provides a feature where clicking on ‘writes:’ hides the cited text. If you change the citation line too much, readers of your messages will have to adjust their Gnus, too. See the variable `gnus-cite-attribution-suffix`. See [section “Article Highlighting” in \*The Gnus Manual\*](#), for details.

### `message-yank-prefix`

When you are replying to or following up an article, you normally want to quote the person you are answering. Inserting quoted text is done by *yanking*, and each line you yank will have `message-yank-prefix` prepended to it (except for quoted and empty lines which uses `message-yank-cited-prefix`). The default is ‘>’.

### `message-yank-cited-prefix`

When yanking text from an article which contains no text or already cited text, each line will be prefixed with the contents of this variable. The default is ‘>’. See also `message-yank-prefix`.

### `message-indentation-spaces`

Number of spaces to indent yanked messages.

### `message-cite-function`

Function for citing an original message. The default is `message-cite-original`, which simply inserts the original message and prepends ‘>’ to each line. `message-cite-original-without-signature` does the same, but elides the signature. You can also set it to `sc-cite-original` to use Supercite.

### `message-indent-citation-function`

Function for modifying a citation just inserted in the mail buffer. This can also be a list of functions. Each function can find the citation between (`point`) and (`mark t`). And each function should leave point and mark around the citation text as modified.

### `message-mark-insert-begin`

String to mark the beginning of some inserted text.

### `message-mark-insert-end`

String to mark the end of some inserted text.

**message-signature**

String to be inserted at the end of the message buffer. If `t` (which is the default), the `message-signature-file` file will be inserted instead. If a function, the result from the function will be used instead. If a form, the result from the form will be used instead. If this variable is `nil`, no signature will be inserted at all.

**message-signature-file**

File containing the signature to be inserted at the end of the buffer. The default is `'~/signature'`.

**message-signature-insert-empty-line**

If `t` (the default value) an empty line is inserted before the signature separator.

Note that RFC1036bis says that a signature should be preceded by the three characters `'-- '` on a line by themselves. This is to make it easier for the recipient to automatically recognize and process the signature. So don't remove those characters, even though you might feel that they ruin your beautiful design, like, totally.

Also note that no signature should be more than four lines long. Including ASCII graphics is an efficient way to get everybody to believe that you are silly and have nothing important to say.

## 11.7 Various Message Variables

**message-default-charset**

Symbol naming a MIME charset. Non-ASCII characters in messages are assumed to be encoded using this charset. The default is `nil`, which means ask the user. (This variable is used only on non-MULE Emacsen. See [section "Charset Translation" in \*Emacs MIME Manual\*](#), for details on the MULE-to-MIME translation process.

**message-signature-separator**

Regexp matching the signature separator. It is `'^-- *$'` by default.

**mail-header-separator**

String used to separate the headers from the body. It is `'--text follows this line--'` by default.

**message-directory**

Directory used by many mailey things. The default is `'~/Mail/'`. All other mail file variables are derived from `message-directory`.

**message-auto-save-directory**

Directory where Message auto-saves buffers if Gnus isn't running. If `nil`, Message won't auto-save. The default is `'~/Mail/drafts/'`.

**message-signature-setup-hook**

Hook run when initializing the message buffer. It is run after the headers have been inserted but before the signature has been inserted.

**message-setup-hook**

Hook run as the last thing when the message buffer has been initialized, but before yanked text is inserted.



**message-header-setup-hook**

Hook called narrowed to the headers after initializing the headers.

For instance, if you're running Gnus and wish to insert a 'Mail-Copies-To' header in all your news articles and all messages you send to mailing lists, you could do something like the following:

```
(defun my-message-header-setup-hook ()
  (let ((group (or gnus-newsgroup-name "")))
    (when (or (message-fetch-field "newsgroups")
              (gnus-group-find-parameter group 'to-address)
              (gnus-group-find-parameter group 'to-list))
      (insert "Mail-Copies-To: never\n"))))

(add-hook 'message-header-setup-hook
          'my-message-header-setup-hook)
```

**message-send-hook**

Hook run before sending messages.

If you want to add certain headers before sending, you can use the `message-add-header` function in this hook. For instance:

```
(add-hook 'message-send-hook 'my-message-add-content)
(defun my-message-add-content ()
  (message-add-header "X-In-No-Sense: Nonsense")
  (message-add-header "X-Whatever: no"))
```

This function won't add the header if the header is already present.

**message-send-mail-hook**

Hook run before sending mail messages. This hook is run very late – just before the message is actually sent as mail.

**message-send-news-hook**

Hook run before sending news messages. This hook is run very late – just before the message is actually sent as news.

**message-sent-hook**

Hook run after sending messages.

**message-cancel-hook**

Hook run when canceling news articles.

**message-mode-syntax-table**

Syntax table used in message mode buffers.

**message-strip-special-text-properties**

Emacs has a number of special text properties which can break message composing in various ways. If this option is set, message will strip these properties from the message composition buffer. However, some packages requires these properties to be present in order to work. If you use one of these packages, turn this option off, and hope the message composition doesn't break too bad.

**message-send-method-alist**

Alist of ways to send outgoing messages. Each element has the form

*(type predicate function)*

*type*        A symbol that names the method.

*predicate*   A function called without any parameters to determine whether the message is a message of type *type*.

*function*    A function to be called if *predicate* returns non-*nil*. *function* is called with one parameter – the prefix.

```
((news message-news-p message-send-via-news)
 (mail message-mail-p message-send-via-mail))
```

## 11.8 Sending Variables

**message-fcc-handler-function**

A function called to save outgoing articles. This function will be called with the name of the file to store the article in. The default function is **message-output** which saves in Unix mailbox format.

**message-courtesy-message**

When sending combined messages, this string is inserted at the start of the mailed copy. If the string contains the format spec '%s', the newsgroups the article has been posted to will be inserted there. If this variable is *nil*, no such courtesy message will be added. The default value is "The following message is a courtesy copy of an article\\nthat has been posted to %s as well.\\n\\n".

**message-fcc-externalize-attachments**

If *nil*, attach files as normal parts in Fcc copies; if it is non-*nil*, attach local files as external parts.

**message-interactive**

If non-*nil* wait for and display errors when sending a message; if *nil* let the mailer mail back a message to report errors.

## 11.9 Message Buffers

Message will generate new buffers with unique buffer names when you request a message buffer. When you send the message, the buffer isn't normally killed off. Its name is changed and a certain number of old message buffers are kept alive.

**message-generate-new-buffers**

If non-*nil*, generate new buffers. The default is *t*. If this is a function, call that function with three parameters: The type, the to address and the group name. (Any of these may be *nil*.) The function should return the new buffer name.

**message-max-buffers**

This variable says how many old message buffers to keep. If there are more message buffers than this, the oldest buffer will be killed. The default is 10. If this variable is *nil*, no old message buffers will ever be killed.

**message-send-rename-function**

After sending a message, the buffer is renamed from, for instance, ‘**\*reply to Lars\***’ to ‘**\*sent reply to Lars\***’. If you don’t like this, set this variable to a function that renames the buffer in a manner you like. If you don’t want to rename the buffer at all, you can say:

```
(setq message-send-rename-function 'ignore)
```

**message-kill-buffer-on-exit**

If non-*nil*, kill the buffer immediately on exit.

## 11.10 Message Actions

When Message is being used from a news/mail reader, the reader is likely to want to perform some task after the message has been sent. Perhaps return to the previous window configuration or mark an article as replied.

The user may exit from the message buffer in various ways. The most common is *C-c C-c*, which sends the message and exits. Other possibilities are *C-c C-s* which just sends the message, *C-c C-d* which postpones the message editing and buries the message buffer, and *C-c C-k* which kills the message buffer. Each of these actions have lists associated with them that contains actions to be executed: **message-send-actions**, **message-exit-actions**, **message-postpone-actions**, and **message-kill-actions**.

Message provides a function to interface with these lists: **message-add-action**. The first parameter is the action to be added, and the rest of the arguments are which lists to add this action to. Here’s an example from Gnus:

```
(message-add-action
 '(set-window-configuration ,(current-window-configuration))
 'exit 'postpone 'kill)
```

This restores the Gnus window configuration when the message buffer is killed, postponed or exited.

An *action* can be either: a normal function, or a list where the CAR is a function and the CDR is the list of arguments, or a form to be **eval**ed.



## 12 Compatibility

Message uses virtually only its own variables—older `mail-` variables aren't consulted. To force Message to take those variables into account, you can put the following in your `.emacs` file:

```
(require 'messcompat)
```

This will initialize many Message variables from the values in the corresponding mail variables.



## 13 Appendices

### 13.1 Responses

To determine where a message is to go, the following algorithm is used by default.

*reply* A *reply* is when you want to respond *just* to the person who sent the message via mail. There will only be one recipient. To determine who the recipient will be, the following headers are consulted, in turn:

**Reply-To**

**From**

*wide reply* A *wide reply* is a mail response that includes *all* entities mentioned in the message you are responded to. All mailboxes from the following headers will be concatenated to form the outgoing **To/Cc** headers:

**From** (unless there's a **Reply-To**, in which case that is used instead).

**Cc**

**To**

If a **Mail-Copies-To** header is present, it will also be included in the list of mailboxes. If this header is '**never**', that means that the **From** (or **Reply-To**) mailbox will be suppressed.

*followup* A *followup* is a response sent via news. The following headers (listed in order of precedence) determine where the response is to be sent:

**Followup-To**

**Newsgroups**

If a **Mail-Copies-To** header is present, it will be used as the basis of the new **Cc** header, except if this header is '**never**'.





## 14 Index

(Index is nonexistent)



## 15 Key Index

(Index is nonexistent)



## Short Contents

|                            |     |
|----------------------------|-----|
| The Gnus Newsreader .....  | 1   |
| 1 Starting Gnus .....      | 3   |
| 2 Group Buffer .....       | 13  |
| 3 Summary Buffer .....     | 43  |
| 4 Article Buffer .....     | 111 |
| 5 Composing Messages ..... | 119 |
| 6 Select Methods .....     | 129 |
| 7 Scoring .....            | 211 |
| 8 Message .....            | 235 |
| Message .....              | 1   |
| 9 Interface .....          | 3   |
| 10 Commands .....          | 9   |
| 11 Variables .....         | 17  |
| 12 Compatibility .....     | 29  |
| 13 Appendices .....        | 31  |
| 14 Index .....             | 33  |
| 15 Key Index .....         | 35  |



# Table of Contents

|                                           |           |
|-------------------------------------------|-----------|
| <b>The Gnus Newsreader .....</b>          | <b>1</b>  |
| <b>1 Starting Gnus .....</b>              | <b>3</b>  |
| 1.1 Finding the News .....                | 3         |
| 1.2 The First Time .....                  | 4         |
| 1.3 The Server is Down .....              | 4         |
| 1.4 Slave Gnusae .....                    | 4         |
| 1.5 New Groups .....                      | 5         |
| 1.5.1 Checking New Groups .....           | 5         |
| 1.5.2 Subscription Methods .....          | 6         |
| 1.5.3 Filtering New Groups .....          | 7         |
| 1.6 Changing Servers .....                | 7         |
| 1.7 Startup Files .....                   | 8         |
| 1.8 Auto Save .....                       | 9         |
| 1.9 The Active File .....                 | 9         |
| 1.10 Startup Variables .....              | 10        |
| <b>2 Group Buffer .....</b>               | <b>13</b> |
| 2.1 Group Buffer Format .....             | 13        |
| 2.1.1 Group Line Specification .....      | 13        |
| 2.1.2 Group Mode Line Specification ..... | 15        |
| 2.1.3 Group Highlighting .....            | 15        |
| 2.2 Group Maneuvering .....               | 16        |
| 2.3 Selecting a Group .....               | 16        |
| 2.4 Subscription Commands .....           | 18        |
| 2.5 Group Data .....                      | 18        |
| 2.6 Group Levels .....                    | 19        |
| 2.7 Group Score .....                     | 20        |
| 2.8 Marking Groups .....                  | 20        |
| 2.9 Foreign Groups .....                  | 21        |
| 2.10 Group Parameters .....               | 23        |
| 2.11 Listing Groups .....                 | 28        |
| 2.12 Sorting Groups .....                 | 29        |
| 2.13 Group Maintenance .....              | 31        |
| 2.14 Browse Foreign Server .....          | 31        |
| 2.15 Exiting Gnus .....                   | 32        |
| 2.16 Group Topics .....                   | 32        |
| 2.16.1 Topic Commands .....               | 33        |
| 2.16.2 Topic Variables .....              | 35        |
| 2.16.3 Topic Sorting .....                | 35        |
| 2.16.4 Topic Topology .....               | 36        |
| 2.16.5 Topic Parameters .....             | 36        |

|          |                                     |           |
|----------|-------------------------------------|-----------|
| 2.17     | Misc Group Stuff .....              | 37        |
| 2.17.1   | Scanning New Messages .....         | 38        |
| 2.17.2   | Group Information .....             | 39        |
| 2.17.3   | Group Timestamp .....               | 39        |
| 2.17.4   | File Commands .....                 | 40        |
| 2.17.5   | Sieve Commands .....                | 40        |
| <b>3</b> | <b>Summary Buffer .....</b>         | <b>43</b> |
| 3.1      | Summary Buffer Format .....         | 43        |
| 3.1.1    | Summary Buffer Lines .....          | 43        |
| 3.1.2    | To From Newsgroups .....            | 46        |
| 3.1.3    | Summary Buffer Mode Line .....      | 47        |
| 3.1.4    | Summary Highlighting .....          | 48        |
| 3.2      | Summary Maneuvering .....           | 48        |
| 3.3      | Choosing Articles .....             | 49        |
| 3.3.1    | Choosing Commands .....             | 49        |
| 3.3.2    | Choosing Variables .....            | 50        |
| 3.4      | Scrolling the Article .....         | 51        |
| 3.5      | Reply, Followup and Post .....      | 51        |
| 3.5.1    | Summary Mail Commands .....         | 51        |
| 3.5.2    | Summary Post Commands .....         | 53        |
| 3.5.3    | Summary Message Commands .....      | 54        |
| 3.5.4    | Canceling Articles .....            | 54        |
| 3.6      | Delayed Articles .....              | 55        |
| 3.7      | Marking Articles .....              | 56        |
| 3.7.1    | Unread Articles .....               | 56        |
| 3.7.2    | Read Articles .....                 | 57        |
| 3.7.3    | Other Marks .....                   | 58        |
| 3.7.4    | Setting Marks .....                 | 59        |
| 3.7.5    | Generic Marking Commands .....      | 60        |
| 3.7.6    | Setting Process Marks .....         | 60        |
| 3.8      | Limiting .....                      | 61        |
| 3.9      | Threading .....                     | 63        |
| 3.9.1    | Customizing Threading .....         | 63        |
| 3.9.1.1  | Loose Threads .....                 | 63        |
| 3.9.1.2  | Filling In Threads .....            | 66        |
| 3.9.1.3  | More Threading .....                | 66        |
| 3.9.1.4  | Low-Level Threading .....           | 67        |
| 3.9.2    | Thread Commands .....               | 68        |
| 3.10     | Sorting the Summary Buffer .....    | 69        |
| 3.11     | Asynchronous Article Fetching ..... | 70        |
| 3.12     | Article Caching .....               | 71        |
| 3.13     | Persistent Articles .....           | 72        |
| 3.14     | Article Backlog .....               | 73        |
| 3.15     | Saving Articles .....               | 73        |
| 3.16     | Decoding Articles .....             | 76        |
| 3.16.1   | Uuencoded Articles .....            | 77        |
| 3.16.2   | Shell Archives .....                | 77        |



|          |                                       |            |
|----------|---------------------------------------|------------|
| 3.16.3   | PostScript Files .....                | 77         |
| 3.16.4   | Other Files .....                     | 77         |
| 3.16.5   | Decoding Variables .....              | 78         |
| 3.16.5.1 | Rule Variables .....                  | 78         |
| 3.16.5.2 | Other Decode Variables .....          | 78         |
| 3.16.5.3 | Uuencoding and Posting .....          | 79         |
| 3.16.6   | Viewing Files .....                   | 80         |
| 3.17     | Article Treatment .....               | 80         |
| 3.17.1   | Article Highlighting .....            | 80         |
| 3.17.2   | Article Fontisizing .....             | 82         |
| 3.17.3   | Article Hiding .....                  | 82         |
| 3.17.4   | Article Washing .....                 | 84         |
| 3.17.5   | Article Header .....                  | 87         |
| 3.17.6   | Article Buttons .....                 | 87         |
| 3.17.6.1 | Related variables and functions ..... | 88         |
| 3.17.7   | Article button levels .....           | 89         |
| 3.17.8   | Article Date .....                    | 90         |
| 3.17.9   | Article Display .....                 | 91         |
| 3.17.10  | Article Signature .....               | 92         |
| 3.17.11  | Article Miscellanea .....             | 92         |
| 3.18     | MIME Commands .....                   | 92         |
| 3.19     | Charsets .....                        | 95         |
| 3.20     | Article Commands .....                | 96         |
| 3.21     | Summary Sorting .....                 | 96         |
| 3.22     | Finding the Parent .....              | 97         |
| 3.23     | Alternative Approaches .....          | 98         |
| 3.23.1   | Pick and Read .....                   | 98         |
| 3.23.2   | Binary Groups .....                   | 99         |
| 3.24     | Tree Display .....                    | 99         |
| 3.25     | Mail Group Commands .....             | 101        |
| 3.26     | Various Summary Stuff .....           | 103        |
| 3.26.1   | Summary Group Information .....       | 104        |
| 3.26.2   | Searching for Articles .....          | 104        |
| 3.26.3   | Summary Generation Commands .....     | 105        |
| 3.26.4   | Really Various Summary Commands ..... | 105        |
| 3.27     | Exiting the Summary Buffer .....      | 106        |
| 3.28     | Crosspost Handling .....              | 107        |
| 3.29     | Duplicate Suppression .....           | 107        |
| 3.30     | Security .....                        | 108        |
| 3.31     | Mailing List .....                    | 109        |
| <b>4</b> | <b>Article Buffer .....</b>           | <b>111</b> |
| 4.1      | Hiding Headers .....                  | 111        |
| 4.2      | Using MIME .....                      | 112        |
| 4.3      | Customizing Articles .....            | 114        |
| 4.4      | Article Keymap .....                  | 116        |
| 4.5      | Misc Article .....                    | 117        |

|          |                                    |            |
|----------|------------------------------------|------------|
| <b>5</b> | <b>Composing Messages .....</b>    | <b>119</b> |
| 5.1      | Mail .....                         | 119        |
| 5.2      | Posting Server .....               | 119        |
| 5.3      | POP before SMTP .....              | 120        |
| 5.4      | Mail and Post .....                | 121        |
| 5.5      | Archived Messages .....            | 121        |
| 5.6      | Posting Styles .....               | 124        |
| 5.7      | Drafts .....                       | 125        |
| 5.8      | Rejected Articles .....            | 126        |
| 5.9      | Signing and encrypting .....       | 127        |
| <b>6</b> | <b>Select Methods .....</b>        | <b>129</b> |
| 6.1      | Server Buffer .....                | 129        |
| 6.1.1    | Server Buffer Format .....         | 129        |
| 6.1.2    | Server Commands .....              | 130        |
| 6.1.3    | Example Methods .....              | 130        |
| 6.1.4    | Creating a Virtual Server .....    | 131        |
| 6.1.5    | Server Variables .....             | 132        |
| 6.1.6    | Servers and Methods .....          | 132        |
| 6.1.7    | Unavailable Servers .....          | 132        |
| 6.2      | Getting News .....                 | 133        |
| 6.2.1    | NNTP .....                         | 133        |
| 6.2.1.1  | Direct Functions .....             | 136        |
| 6.2.1.2  | Indirect Functions .....           | 137        |
| 6.2.1.3  | Common Variables .....             | 138        |
| 6.2.2    | News Spool .....                   | 138        |
| 6.3      | Getting Mail .....                 | 139        |
| 6.3.1    | Mail in a Newsreader .....         | 139        |
| 6.3.2    | Getting Started Reading Mail ..... | 140        |
| 6.3.3    | Splitting Mail .....               | 141        |
| 6.3.4    | Mail Sources .....                 | 142        |
| 6.3.4.1  | Mail Source Specifiers .....       | 142        |
| 6.3.4.2  | Function Interface .....           | 148        |
| 6.3.4.3  | Mail Source Customization .....    | 148        |
| 6.3.4.4  | Fetching Mail .....                | 149        |
| 6.3.5    | Mail Back End Variables .....      | 149        |
| 6.3.6    | Fancy Mail Splitting .....         | 150        |
| 6.3.7    | Group Mail Splitting .....         | 153        |
| 6.3.8    | Incorporating Old Mail .....       | 155        |
| 6.3.9    | Expiring Mail .....                | 156        |
| 6.3.10   | Washing Mail .....                 | 158        |
| 6.3.11   | Duplicates .....                   | 160        |
| 6.3.12   | Not Reading Mail .....             | 160        |
| 6.3.13   | Choosing a Mail Back End .....     | 161        |
| 6.3.13.1 | Unix Mail Box .....                | 161        |
| 6.3.13.2 | Rmail Babyl .....                  | 161        |
| 6.3.13.3 | Mail Spool .....                   | 161        |
| 6.3.13.4 | MH Spool .....                     | 163        |

|           |                                 |     |
|-----------|---------------------------------|-----|
| 6.3.13.5  | Maildir .....                   | 163 |
| 6.3.13.6  | Group parameters .....          | 165 |
| 6.3.13.7  | Article identification .....    | 166 |
| 6.3.13.8  | NOV data .....                  | 166 |
| 6.3.13.9  | Article marks .....             | 167 |
| 6.3.13.10 | Mail Folders .....              | 167 |
| 6.3.13.11 | Comparing Mail Back Ends .....  | 168 |
| 6.4       | Browsing the Web .....          | 171 |
| 6.4.1     | Archiving Mail .....            | 171 |
| 6.4.2     | Web Searches .....              | 172 |
| 6.4.3     | Slashdot .....                  | 173 |
| 6.4.4     | Ultimate .....                  | 174 |
| 6.4.5     | Web Archive .....               | 174 |
| 6.4.6     | RSS .....                       | 175 |
| 6.4.7     | Customizing w3 .....            | 176 |
| 6.5       | IMAP .....                      | 176 |
| 6.5.1     | Splitting in IMAP .....         | 180 |
| 6.5.2     | Expiring in IMAP .....          | 182 |
| 6.5.3     | Editing IMAP ACLs .....         | 183 |
| 6.5.4     | Expunging mailboxes .....       | 183 |
| 6.5.5     | A note on namespaces .....      | 183 |
| 6.5.6     | Debugging IMAP .....            | 184 |
| 6.6       | Other Sources .....             | 184 |
| 6.6.1     | Directory Groups .....          | 184 |
| 6.6.2     | Anything Groups .....           | 185 |
| 6.6.3     | Document Groups .....           | 186 |
| 6.6.3.1   | Document Server Internals ..... | 187 |
| 6.6.4     | SOUP .....                      | 189 |
| 6.6.4.1   | SOUP Commands .....             | 190 |
| 6.6.4.2   | SOUP Groups .....               | 191 |
| 6.6.4.3   | SOUP Replies .....              | 192 |
| 6.6.5     | Mail-To-News Gateways .....     | 192 |
| 6.7       | Combined Groups .....           | 193 |
| 6.7.1     | Virtual Groups .....            | 193 |
| 6.7.2     | Kibozed Groups .....            | 194 |
| 6.8       | Gnus Unplugged .....            | 195 |
| 6.8.1     | Agent Basics .....              | 195 |
| 6.8.2     | Agent Categories .....          | 196 |
| 6.8.2.1   | Category Syntax .....           | 197 |
| 6.8.2.2   | Category Buffer .....           | 201 |
| 6.8.2.3   | Category Variables .....        | 201 |
| 6.8.3     | Agent Commands .....            | 202 |
| 6.8.3.1   | Group Agent Commands .....      | 202 |
| 6.8.3.2   | Summary Agent Commands .....    | 202 |
| 6.8.3.3   | Server Agent Commands .....     | 203 |
| 6.8.4     | Agent Visuals .....             | 203 |
| 6.8.5     | Agent as Cache .....            | 204 |
| 6.8.6     | Agent Expiry .....              | 204 |

|          |                                 |            |
|----------|---------------------------------|------------|
| 6.8.7    | Agent Regeneration .....        | 204        |
| 6.8.8    | Agent and IMAP .....            | 205        |
| 6.8.9    | Outgoing Messages .....         | 206        |
| 6.8.10   | Agent Variables .....           | 206        |
| 6.8.11   | Example Setup .....             | 207        |
| 6.8.12   | Batching Agents .....           | 208        |
| 6.8.13   | Agent Caveats .....             | 208        |
| <b>7</b> | <b>Scoring .....</b>            | <b>211</b> |
| 7.1      | Summary Score Commands .....    | 211        |
| 7.2      | Group Score Commands .....      | 213        |
| 7.3      | Score Variables .....           | 214        |
| 7.4      | Score File Format .....         | 216        |
| 7.5      | Score File Editing .....        | 220        |
| 7.6      | Adaptive Scoring .....          | 221        |
| 7.7      | Home Score File .....           | 223        |
| 7.8      | Followups To Yourself .....     | 224        |
| 7.9      | Scoring On Other Headers .....  | 224        |
| 7.10     | Scoring Tips .....              | 225        |
| 7.11     | Reverse Scoring .....           | 225        |
| 7.12     | Global Score Files .....        | 226        |
| 7.13     | Kill Files .....                | 227        |
| 7.14     | Converting Kill Files .....     | 228        |
| 7.15     | GroupLens .....                 | 228        |
| 7.15.1   | Using GroupLens .....           | 228        |
| 7.15.2   | Rating Articles .....           | 229        |
| 7.15.3   | Displaying Predictions .....    | 229        |
| 7.15.4   | GroupLens Variables .....       | 230        |
| 7.16     | Advanced Scoring .....          | 230        |
| 7.16.1   | Advanced Scoring Syntax .....   | 230        |
| 7.16.2   | Advanced Scoring Examples ..... | 231        |
| 7.16.3   | Advanced Scoring Tips .....     | 232        |
| 7.17     | Score Decays .....              | 232        |
| <b>8</b> | <b>Message .....</b>            | <b>235</b> |
|          | <b>Message .....</b>            | <b>1</b>   |

|           |                                                    |           |
|-----------|----------------------------------------------------|-----------|
| <b>9</b>  | <b>Interface .....</b>                             | <b>3</b>  |
| 9.1       | New Mail Message .....                             | 3         |
| 9.2       | New News Message .....                             | 3         |
| 9.3       | Reply .....                                        | 3         |
| 9.4       | Wide Reply .....                                   | 4         |
| 9.5       | Followup .....                                     | 4         |
| 9.6       | Canceling News .....                               | 4         |
| 9.7       | Superseding .....                                  | 5         |
| 9.8       | Forwarding .....                                   | 5         |
| 9.9       | Resending .....                                    | 5         |
| 9.10      | Bouncing .....                                     | 6         |
| 9.11      | Mailing Lists .....                                | 6         |
| 9.11.1    | Composing a correct MFT header automagically ..... | 6         |
| 9.11.2    | Honoring an MFT post .....                         | 7         |
| <b>10</b> | <b>Commands .....</b>                              | <b>9</b>  |
| 10.1      | Buffer Entry .....                                 | 9         |
| 10.2      | Header Commands .....                              | 9         |
| 10.2.1    | Commands for moving to headers .....               | 9         |
| 10.2.2    | Commands to change headers .....                   | 10        |
| 10.3      | Movement .....                                     | 11        |
| 10.4      | Insertion .....                                    | 11        |
| 10.5      | MIME .....                                         | 11        |
| 10.6      | IDNA .....                                         | 12        |
| 10.7      | Security .....                                     | 12        |
| 10.7.1    | Using S/MIME .....                                 | 13        |
| 10.7.2    | Using PGP/MIME .....                               | 14        |
| 10.8      | Various Commands .....                             | 15        |
| 10.9      | Sending .....                                      | 15        |
| 10.10     | Mail Aliases .....                                 | 15        |
| 10.11     | Spelling .....                                     | 16        |
| <b>11</b> | <b>Variables .....</b>                             | <b>17</b> |
| 11.1      | Message Headers .....                              | 17        |
| 11.2      | Mail Headers .....                                 | 19        |
| 11.3      | Mail Variables .....                               | 19        |
| 11.4      | News Headers .....                                 | 20        |
| 11.5      | News Variables .....                               | 22        |
| 11.6      | Insertion Variables .....                          | 23        |
| 11.7      | Various Message Variables .....                    | 24        |
| 11.8      | Sending Variables .....                            | 26        |
| 11.9      | Message Buffers .....                              | 26        |
| 11.10     | Message Actions .....                              | 27        |
| <b>12</b> | <b>Compatibility .....</b>                         | <b>29</b> |

|           |                         |           |
|-----------|-------------------------|-----------|
| <b>13</b> | <b>Appendices .....</b> | <b>31</b> |
| 13.1      | Responses .....         | 31        |
| <b>14</b> | <b>Index .....</b>      | <b>33</b> |
| <b>15</b> | <b>Key Index.....</b>   | <b>35</b> |