# Keeping Track

Managing Messages With GNATS
The GNU Problem Report Management System
Version 4.0-beta1
December 2001

**Jeffrey M. Osier**
**Brendan Kehoe**
**Cygnus Support**
**Revised for GNATS 4 by Yngve Svendsen**

# Table of Contents

# Overview

This manual documents GNATS, the GNU Problem Report Management System, version 4.0-beta1. GNATS is a bug-tracking tool designed for use at a central *Support Site*. Users who experience problems use electronic mail, web-based or other clients communicating with the GNATS network daemon running at the support site or direct database submissions to communicate these problems to *maintainers* at that Support Site. GNATS partially automates the tracking of these *Problem Reports* (*PR*s) by:

- organizing problem reports into a database and notifying responsible parties of suspected bugs;
- allowing support personnel and their managers to edit and query accumulated bugs; and
- providing a reliable archive of problems (and their subsequent solutions) with a given program.

GNATS offers many of the same features offered by more generalized databases, including editing, querying, and basic reporting. The GNATS database itself is an ordered repository for problem reports; each PR receives a unique, incremental *PR number* which identifies it throughout its lifetime. For a discussion on the working system adopted by GNATS, see Section 1.1 [The database paradigm], page 3.

You can access the submitting, editing, and querying functions of GNATS from within GNU Emacs. See Chapter 2 [The GNATS user tools], page 13.

## Note on this version of the manual

This manual is in the process of being revised for version 4 of GNATS. The status of the different sections is as follows:

- Chapter 1 *Introducing* GNATS
  Version 4 complete, except for mention of the Cases, Keywords, Quarter and Release Note fields which may or may not make it into final version 4.
- Chapter 2 *The* GNATS *User Tools*
  Version 4 complete, except for the sections on send-pr.
- Chapter 3 *Installing* GNATS
  Version 4 complete, except for info on installing the user tools at remote locations which is yet to be written.
- Chapter 4 GNATS *Administration*
  Version 4 complete, except for treatment of the still-to-be-written gnats-passwd.
- Appendix A *Where* GNATS *lives*
  Version 4 complete.
- Appendix B *The* GNATS *network server* – `gnatsd`
  Version 4 complete.
- Appendix C *Controlling access to databases*
  Version 4 complete.
- Appendix D *Querying using regular expressions*
  Version 4 complete.

# 1 Introducing GNATS

Any support organization realizes that a large amount of data flows back and forth between the maintainers and the users of their products. This data often takes the form of problem reports and communication via electronic mail. GNATS addresses the problem of organizing this communication by defining a database made up of archived and indexed problem reports.

GNATS was designed as a tool for software maintainers. It consists of several utilities which, when used in concert, formulate and administer a database of Problem Reports grouped by site-defined *problem categories*. It allows a support organization to keep track of problems (hence the term *Problem Report*) in an organized fashion. Essentially, GNATS acts as an active archive for field-separated textual data.

## 1.1 The database paradigm

It is in your best interest as the maintainer of a body of work to organize the feedback you receive on that work, and to make it easy for users of your work to report problems and suggestions.

GNATS makes this easy by automatically filing incoming problem reports into appropriate places, by notifying responsible parties of the existence of the problem and (optionally) sending an acknowledgment to the submitter that the report was received, and by making these Problem Reports accessible to queries and easily editable. GNATS is a database specialized for a specific task.

GNATS was designed for use at a Support Site that handles a high level of problem-related traffic. It maintains Problem Reports in the form of text files with defined *fields* (see Section 1.4 [GNATS data fields], page 6). The location of the database, as well as the categories it accepts as valid, the maintainers for whom it provides service, and the submitters from whom it accepts Problem Reports, are all definable by the *Support Site*. See Chapter 4 [GNATS administration], page 43.

Each PR is a separate file within a main repository (see Appendix A [Where GNATS lives], page 73). Editing access to the database is regulated to maintain consistency. To make queries on the database faster, an index is kept automatically (see Section 4.5.1 [The `index` file], page 61).

We provide several software tools so that users may submit new Problem Reports, edit existing Problem Reports, and query the database.

- `send-pr` is used by both product maintainers and the end users of the products they support to submit PRs to the database.
- `edit-pr` is used by maintainers when editing problem reports in the database.
- Maintainers, managers and administrators can use `query-pr` to make inquiries about individual PRs or groups of PRs.

Other interfaces to GNATS include Gnatsweb, a web-based tool which provides features for submitting and editing PRs and querying the database, and TkGnats, a Tcl/Tk-based frontend. These tools are distributed together with GNATS.

At the Support Site, a GNATS *administrator* is charged with the duty of maintaining GNATS. These duties are discussed in detail in Chapter 4 [GNATS Administration], page 43,

and generally include configuring GNATS for the Support Site, editing PRs that GNATS cannot process, pruning log files, setting up new problem categories, backing up the database, and distributing `send-pr` so that others may submit Problem Reports.

Responsibility for a given Problem Report initially depends on the category of the problem. Optionally, an automated reminder can be sent to the responsible person if a problem report is neglected for a requisite time period. See Section 4.1 [GNATS configuration], page 44.

GNATS does not have the ability to decipher random text. If there is no default category set, any problem reports which arrive in a format GNATS does not recognize are placed in a separate directory pending investigation by the GNATS administrator (see Chapter 4 [GNATS Administration], page 43).

Once a problem is recorded in the database, work can begin toward a solution. A problem's initial *state* is *open* (see Section 1.3 [States of Problem Reports], page 5). An acknowledgment may be sent to the originator of the bug report (depending on whether this feature has been switched on in the GNATS configuration). GNATS forwards copies of the report to the party responsible for that problem category and to the person responsible for problems arriving from that submitter.

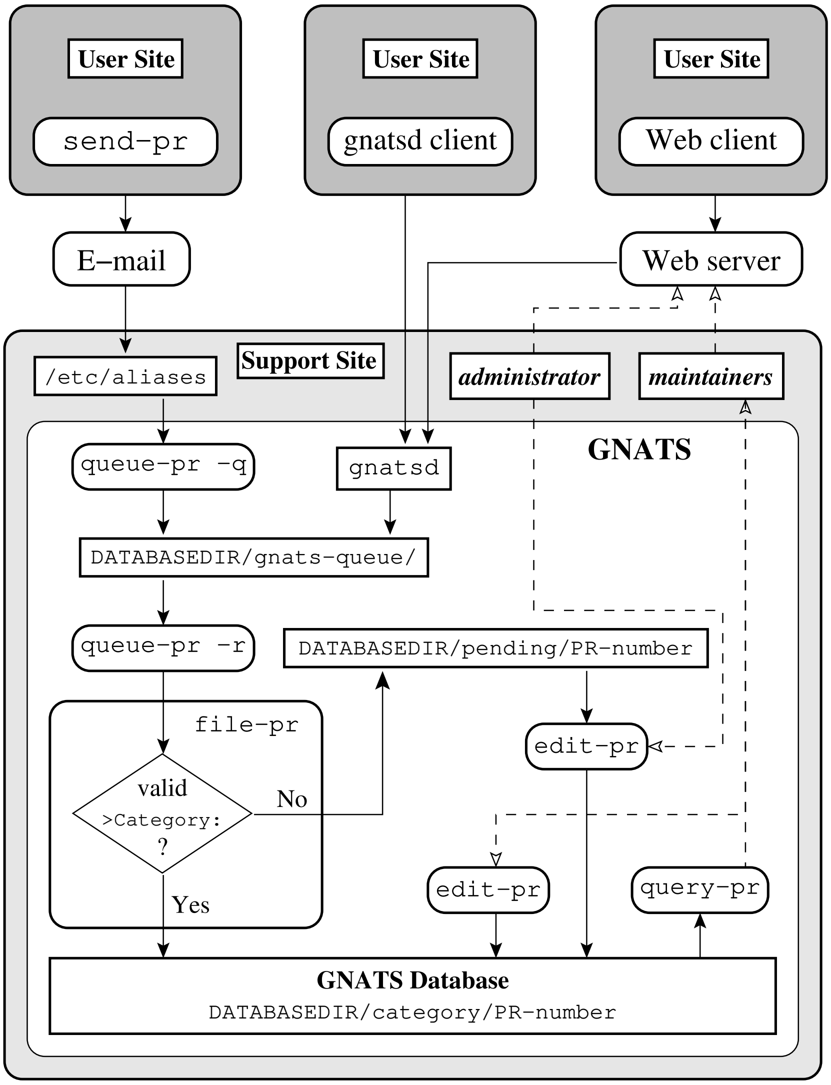
When a problem has been identified, the maintainer can change its state to *analyzed*, and then to *feedback* when a solution is found. GNATS may be configured so that each time the state of a PR changes, the submitter of the problem report is notified of the reason for the change. If the party responsible for the PR changes, the previous responsible party and the new responsible party receive notice of the change. The change and reason are also recorded in the '`Audit-Trail`' field of the Problem Report. (See Section 2.3 [Editing existing Problem Reports], page 18. For information on the '`Audit-Trail`' field, see Section 1.4 [Problem Report format], page 6.)

When the originator of the Problem Report confirms that the solution works, the maintainer can change the state to *closed*. If the PR cannot be closed, the maintainer can change its state to *suspended* as a last resort. (For a more detailed description of the standard states, see Section 1.3 [States of Problem Reports], page 5.)

It should be emphasized that what we describe here is the default way that GNATS works, but as of version 4, GNATS is extremely customizable, allowing sites to tailor almost every aspect of its behavior. See Section 4.3 [The `dbconfig` file], page 46.

## 1.2 Flowchart of GNATS activities

This informal flowchart shows the relationships of the GNATS tools to each other and to the files with which they interoperate.




## 1.3 States of Problem Reports

Each PR goes through a defined series of states between origination and closure. By default, the originator of a PR receives notification automatically of any state changes.

Unless your site has customized states (see Section 4.4.4 [states file], page 60), GNATS uses these states:

*open* The initial state of a Problem Report. This means the PR has been filed and the responsible person(s) notified.

*analyzed* The responsible person has analyzed the problem. The analysis should contain a preliminary evaluation of the problem and an estimate of the amount of time and resources necessary to solve the problem. It should also suggest possible workarounds.

*feedback* The problem has been solved, and the originator has been given a patch or other fix. The PR remains in this state until the originator acknowledges that the solution works.

*closed* A Problem Report is closed ("the bug stops here") only when any changes have been integrated, documented, and tested, and the submitter has confirmed the solution.

*suspended* Work on the problem has been postponed. This happens if a timely solution is not possible or is not cost-effective at the present time. The PR continues to exist, though a solution is not being actively sought. If the problem cannot be solved at all, it should be closed rather than suspended.

## 1.4 Problem Report format

The format of a PR is designed to reflect the nature of GNATS as a database. Information is arranged into *fields*, and kept in individual records (Problem Reports).

A Problem Report contains two different types of fields: *Mail Header* fields, which are used by the mail handler for delivery, and *Problem Report* fields, which contain information relevant to the Problem Report and its submitter. A Problem Report is essentially a specially formatted electronic mail message.

Problem Report fields are denoted by a keyword which begins with '`>`' and ends with '`:`', as in '`>Confidential:`'. Fields belong to one of eight data types as listed in Section 1.4.1 [Field datatypes reference], page 7. As of version 4 of GNATS all characteristics of fields, such as field name, data type, allowed values, permitted operations, on-change actions etc. are configurable.

For details, see see Section 4.3 [The `dbconfig` file], page 46.

### Example Problem Report

The following is an example Problem Report with the fields that would be present in a standard GNATS configuration. Mail headers are at the top, followed by GNATS fields, which begin with '`>`' and end with '`:`'. The '`Subject:`' line in the mail header and the '`>Synopsis:`' field are usually duplicates of each other.

```
Message-Id:  message-id
Date:        date
From:        address
Reply-To:    address
To:          bug-address
Subject:     subject

>Number:       gnats-id
>Category:     category
>Synopsis:     synopsis
>Confidential: yes or no
>Severity:     critical, serious, or non-critical
>Priority:     high, medium or low
>Responsible:  responsible
>State:        open, analyzed, suspended, feedback, or closed
>Class:        sw-bug, doc-bug, change-request, support,
duplicate, or mistaken
>Submitter-Id: submitter-id
>Arrival-Date: date
>Originator:   name
>Organization: organization
>Release:      release
>Environment:
   environment
>Description:
   description
>How-To-Repeat:
   how-to-repeat
>Fix:
   fix
>Audit-Trail:
appended-messages...
State-Changed-From-To: from-to
State-Changed-When: date
State-Changed-Why:
   reason
Responsible-Changed-From-To: from-to
Responsible-Changed-When: date
Responsible-Changed-Why:
   reason
>Unformatted:
   miscellaneous
```

### 1.4.1 Field datatypes reference

The following is a short reference to the characteristics of the different field types.

For details, see Section 4.3.3 [Field datatypes], page 49.

`text` A one-line text string.

`multitext`
: Multiple lines of text.

`enum`
: The value in this field is required to be from a list of specified values, defined at the Support Site.

  (See Section 4.3 [The `dbconfig` file], page 46, for details.

`multienum`
: Similar to the `enum` datatype, except that the field can contain multiple values.

`enumerated-in-file`
: Similar to `enum`, but the allowed field values are listed in a separate file on the GNATS server.

`multi-enumerated-in-file`
: Similar to `enumerated-in-file`, except that the field can contain multiple values.

`date`
: Used to hold dates.

`integer`
: Used to hold integer numbers.

### 1.4.2 Mail header fields

A Problem Report may contain any mail header field described in the Internet standard RFC-822. The `send-pr` tool can be configured either to submit PRs to the support site by e-mail or by talking directly to the `gnatsd` network daemon on the GNATS server. This is also true for other client tools such as Gnatsweb. Even when these tools are set up submit PRs directly to `gnatsd`, they will still include mail header fields that identify the sender and the subject in the submitted PR:

`To:`
: The mail address for the Support Site, automatically supplied by the tool used to submit the PR or by the originator if plain e-mail was used.

`Subject:`
: A terse description of the problem. This field normally contains the same information as the '`Synopsis`' field.

`From:`
: Supplied automatically when PRs are submitted by plain e-mail and when well-behaved tools such as `send-pr` are used; should always contain the originator's e-mail address.

`Reply-To:`
: A return address to which electronic replies can be sent; in most cases, the same address as the `From:` field.

One of the configurable options for GNATS is whether or not to retain '`Received-By:`' headers, which often consume a lot of space and are not often used. See Section 4.3 [The dbconfig file], page 46.

### 1.4.3 Problem Report fields

## Field descriptions

In a standard GNATS installation, certain fields will always be present in a Problem Report. If a PR arrives without one or more of these fields, GNATS will add them, and if they have default values set by the configuration at the Support Site, they will be filled in with these values. Certain tools such as `send-pr` are set up to provide sensible defaults for most fields.

(TODO: Make cross-reference to treatment of send-pr.conf)

In the list below, the field type (`text`, `multitext`, `enumerated`, etc.) is supplied in parantheses. The different field types are explained briefly in Section 1.4.1 [Field datatypes reference], page 7.

`Submitter-Id`
: (`enumerated-in-file`) A unique identification code assigned by the Support Site. It is used to identify all Problem Reports coming from a particular site. Submitters without a value for this field can invoke `send-pr` with the '`--request-id`' option to apply for one from the support organization. Problem Reports from those not affiliated with the support organization should use the default value of '`net`' for this field.

  See Section 4.4.3 [The `submitters` file], page 59, for details.

`Originator`
: (`text`) Originator's real name. Note that the Submitter and Originator might not be the same person/entity in all cases.

`Organization`
: (`multitext`) The originator's organization.

`Confidential`
: (`enum`) Use of this field depends on the originator's relationship with the support organization; contractual agreements often have provisions for preserving confidentiality. Conversely, a lack of a contract often means that any data provided will not be considered confidential. Submitters should be advised to contact the support organization directly if this is an issue.

  If the originator's relationship to the support organization provides for confidentiality, then if the value of this field is '`yes`' the support organization treats the PR as confidential; any code samples provided are not made publicly available.

`Synopsis`
: (`text`) One-line summary of the problem. `send-pr` copies this information to the '`Subject:`' line when you submit a Problem Report.

`Severity`
: (`enum`) The severity of the problem. By default, accepted values include:

  `critical`
  : The product, component or concept is completely non-operational or some essential functionality is missing. No workaround is known.

  `serious`
  : The product, component or concept is not working properly or significant functionality is missing. Problems that would otherwise be considered '`critical`' are usually rated '`serious`' when a workaround is known.

`non-critical`
: The product, component or concept is working in general, but lacks features, has irritating behavior, does something wrong, or doesn't match its documentation.

`Priority` (`enumerated`) How soon the originator requires a solution. Accepted values include:

- `high` A solution is needed as soon as possible.
- `medium` The problem should be solved in the next release.
- `low` The problem should be solved in a future release.

`Category` (`enumerated-in-file`) The name of the product, component or concept where the problem lies. The values for this field are defined by the Support Site. See Section 4.4.1 [The `categories` file], page 57, for details.

`Class` (`enumerated-in-file`) The class of a problem in a default GNATS installation can be one of the following:

- `sw-bug` A general product problem. ('`sw`' stands for "software".)
- `doc-bug` A problem with the documentation.
- `change-request`
  A request for a change in behavior, etc.
- `support` A support problem or question.
- `duplicate` (*pr-number*)
  Duplicate PR. *pr-number* should be the number of the original PR.
- `mistaken` No problem, user error or misunderstanding. This value can only be set by tools at the Support Site, since it has no meaning for ordinary submitters.

See Section 4.4.6 [The `classes` file], page 61, for details.

`Release` (`text`) Release or version number of the product, component or concept.

`Environment`
: (`multitext`) Description of the environment where the problem occurred: machine architecture, operating system, host and target types, libraries, pathnames, etc.

`Description`
: (`multitext`) Precise description of the problem.

`How-To-Repeat`
: (`multitext`) Example code, input, or activities to reproduce the problem. The support organization uses example code both to reproduce the problem and to test whether the problem is fixed. Include all preconditions, inputs, outputs, conditions after the problem, and symptoms. Any additional important information should be included. Include all the details that would be necessary

for someone else to recreate the problem reported, however obvious. Sometimes seemingly arbitrary or obvious information can point the way toward a solution. See also Section 2.2.5 [Helpful hints], page 18.

`Fix` (`multitext`) A description of a solution to the problem, or a patch which solves the problem. (This field is most often filled in at the Support Site; we provide it to the submitter in case he or she has solved the problem.)

GNATS adds the following fields when the PR arrives at the Support Site:

`Number` (`enumerated`) The incremental identification number for this PR. This is included in the automated reply to the submitter (if that feature of GNATS is activated; see Section 4.3 [The '`dbconfig`' file], page 46). It is also included in the copy of the PR that is sent to the maintainer.

The `Number` field is often paired with the `Category` field as

*category*/*number*

in subsequent email messages. This is GNATS' way of tracking followup messages that arrive by mail so that they are filed as part of the original PR.

`State` (`enumerated`) The current state of the PR. In default GNATS installations, accepted values are:

`open` The PR has been filed and the responsible person notified.

`analyzed` The responsible person has analyzed the problem.

`feedback` The problem has been solved, and the originator has been given a patch or other fix. Support organizations may also choose to temporarily "park" PRs that are awaiting further input from the submitter under this state.

`closed` The changes have been integrated, documented, and tested, and the originator has confirmed that the solution works.

`suspended`
Work on the problem has been postponed.

The initial state of a PR is '`open`'. See Section 1.3 [States of Problem Reports], page 5.

`Responsible`
(`text`) The person at the Support Site who is responsible for this PR. GNATS retrieves this information from the '`categories`' file (see Section 4.4.1 [The `categories` file], page 57).

`Arrival-Date`
(`date`) The time that this PR was received by GNATS. The date is provided automatically by GNATS.

`Date-Required`
(`date`) The date by which a fix is required. This is up to the maintainers at the Support Site to determine, so this field is not available until after the PR has been submitted.

`Audit-Trail`

(`multitext`) Tracks related electronic mail as well as changes in the `State` and `Responsible` fields with the sub-fields:

`State-Changed-<From>-<To>:` *oldstate*`>-<`*newstate*
The old and new `State` field values.

`Responsible-Changed-<From>-<To>:` *oldresp*`>-<`*newresp*
The old and new `Responsible` field values.

`State-Changed-By:` *name*
`Responsible-Changed-By:` *name*
The name of the maintainer who effected the change.

`State-Changed-When:` *timestamp*
`Responsible-Changed-When:` *timestamp*
The time the change was made.

`State-Changed-Why:` *reason...*
`Responsible-Changed-Why:` *reason...*
The reason for the change.

The `Audit-Trail` field also contains any mail messages received by GNATS related to this PR, in the order received. GNATS needs to find a *category*/*number* at the beginning of the Subject field of received e-mail in order to be able to file it correctly.

`Unformatted`

(`multitext`) Any random text found outside the fields in the original Problem Report.

During a Problem Report's journey from '`open`' to '`closed`', two more fields, `Last-Modified` and `Closed Date` (both of type `date`) will be added.

# 2 The GNATS User Tools

This chapter describes the user tools distributed with GNATS. The GNATS administrative and internal tools tools are described in Chapter 4 [GNATS Administration], page 43. The user tools provide facilities for initial submission, querying and editing of Problem Reports:

`send-pr` Used by anyone who has a problem with a body of work to submit a report of the problem to the maintainers of that work (see Section 2.2 [Submitting Problem Reports], page 13).

`query-pr` Used to query the GNATS database (see Section 2.4 [Querying the database], page 20).

`edit-pr` Used to edit Problem Reports (to record new data, to change the responsible party, etc.) (see Section 2.3 [Editing existing Problem Reports], page 18).

## 2.1 Environment variables and GNATS tools

All the GNATS user tools honor the `GNATSDB` environment variable which is used to determine which database to use. For a local database, it contains the name of the database to access.

For network access via gnatsd, it contains a colon-separated list of strings that describe the remote database in the form

*server:port:databasename:username:password*

Any of the fields may be omitted except for *server*, but at least one colon must appear; otherwise, the value is assumed to be the name of a local database.

If `GNATSDB` is not set and no command-line options are used to specify the database, it is assumed that the database is local and that its name is '`default`'.

## 2.2 Submitting Problem Reports

Use `send-pr` to submit Problem Reports to the database. `send-pr` is a shell script which composes a template for submitters to complete.

You can invoke `send-pr` from a shell prompt, or from within GNU Emacs using '`M-x send-pr`'.

### 2.2.1 Creating new Problem Reports

Invoking `send-pr` presents a PR *template* with a number of fields already filled in. Complete the template as thoroughly as possible to make a useful bug report. Submit only one bug with each PR.

A template consists of three sections:

*Comments*

The top several lines of a blank template consist of a series of comments that provide some basic instructions for completing the Problem Report, as well as a list of valid entries for the '`>Category:`' field. These comments are all preceded by the string '`SEND-PR:`' and are erased automatically when the PR is submitted. The instructional comments within ‘<’ and ‘>’ are also removed. (Only these comments are removed; lines you provide that happen to have those characters in them, such as examples of shell-level redirection, are not affected.)

*Mail Header*

`send-pr` creates a standard mail header. `send-pr` completes all fields except the ‘`Subject:`’ line with default values. (See Section 1.4 [Problem Report format], page 6.)

GNATS *fields*

These are the informational fields that GNATS uses to route your Problem Report to the responsible party for further action. They should be filled out as completely as possible. (See Section 1.4 [Problem Report format], page 6. Also see Section 2.2.5 [Helpful hints], page 18.)

The default template contains your preconfigured ‘`>Submitter-Id:`’. `send-pr` attempts to determine values for the ‘`>Originator:`’ and ‘`>Organization:`’ fields (see Section 1.4 [Problem Report format], page 6). `send-pr` will set the ‘`>Originator:`’ field to the value of the `NAME` environment variable if it has been set; similarly, ‘`>Organization:`’ will be set to the value of `ORGANIZATION`. `send-pr` also attempts to find out some information about your system and architecture, and places this information in the ‘`>Environment:`’ field if it finds any.

You may submit problem reports to different Support Sites from the default site by specifying the alternate site when you invoke `send-pr`. See Section 2.2.3 [send-pr from the shell], page 16. Each `site` has its own list of categories for which it accepts Problem Reports.

`send-pr` also provides the mail header section of the template with default values in the ‘`To:`’, ‘`From:`’, and ‘`Reply-To:`’ fields. The ‘`Subject:`’ field is empty.

The template begins with a comment section:

```
SEND-PR: -*- send-pr  -*-
SEND-PR: Lines starting with ‘SEND-PR’ will be removed
SEND-PR: automatically as well as all comments (the text
SEND-PR: below enclosed in ‘<’ and ‘>’).
SEND-PR:
SEND-PR: Please consult the document ‘Reporting Problems
SEND-PR: Using send-pr’ if you are not sure how to fill out
SEND-PR: a problem report.
SEND-PR:
SEND-PR: Choose from the following categories:
```

and also contains a list of valid `>Category:` values for the Support Site to whom you are submitting this Problem Report. One (and only one) of these values should be placed in the `>Category:` field.

The mail header is just below the comment section. Fill out the ‘`Subject:`’ field, if it is not already completed using the value of ‘`>Synopsis:`’. The other mail header fields contain default values.

```
To: support-site
Subject: complete this field
From: your-login@your-site
Reply-To: your-login@your-site
X-send-pr-version: send-pr 4.0-beta1
```

where *support-site* is an alias on your local machine for the Support Site you wish to submit this PR to.

The rest of the template contains GNATS fields. Each field is either automatically completed with valid information (such as your '`>Submitter-Id:`') or contains a one-line instruction specifying the information that field requires in order to be correct. For example, the '`>Confidential:`' field expects a value of '`yes`' or '`no`', and the answer must fit on one line; similarly, the '`>Synopsis:`' field expects a short synopsis of the problem, which must also fit on one line. Fill out the fields as completely as possible. See Section 2.2.5 [Helpful hints], page 18, for suggestions as to what kinds of information to include.

In this example, words in *italics* are filled in with pre-configured information:

```
>Submitter-Id: your submitter-id
>Originator:   your name here
>Organization:
    your organization
>Confidential:<[ yes | no ] (one line)>
>Synopsis:    <synopsis of the problem (one line)>
>Severity:    <[non-critical | serious | critical](one line)>
>Priority:    <[ low | medium | high ] (one line)>
>Category:    <name of the product (one line)>
>Class:       <[sw-bug | doc-bug | change-request | support]>
>Release:     <release number (one line)>
>Environment:
         <machine, os, target, libraries (multiple lines)>

>Description:
       <precise description of the problem (multiple lines)>
>How-To-Repeat:
       <code/input/activities to reproduce (multiple lines)>
>Fix:
       <how to correct or work around the problem, if known
        (multiple lines)>
```

When you finish editing the Problem Report, `send-pr` mails it to the address named in the '`To:`' field in the mail header. `send-pr` checks that the complete form contains a valid '`>Category:`'.

If your PR has an invalid value in one of the ENUMERATED fields (see Section 1.4 [Problem Report format], page 6), `send-pr` places the PR in a temporary file named '`/tmp/pbad`*nnnn*' on your machine. *nnnn* is the process identification number given to your current `send-pr` session. If you are running `send-pr` from the shell, you are prompted as to whether or not you wish to try editing the same Problem Report again. If you are running `send-pr` from Emacs, the Problem Report is placed in the buffer '`*gnats-send*`'; you can edit this file and then submit it with *C-c C-c*.

Any further mail concerning this Problem Report should be carbon-copied to the GNATS mailing address as well, with the category and identification number in the '`Subject:`' line of the message.

`Subject: Re: PR` *category*`/`*gnats-id*`:` *original message subject*

Messages which arrive with '`Subject:`' lines of this form are automatically appended to the Problem Report in the '`>Audit-Trail:`' field in the order received.

### 2.2.2 Using `send-pr` from within Emacs

You can use an interactive `send-pr` interface from within GNU Emacs to fill out your Problem Report. We recommend that you familiarize yourself with Emacs before using this feature (see section "Introduction" in *GNU Emacs*).

Call `send-pr` with '`M-x send-pr`'.[1] `send-pr` responds with a Problem Report template preconfigured for the Support Site to which you are going to send the report.

You may also submit problem reports to different Support Sites from the default site. To use this feature, invoke `send-pr` with

```
C-u M-x send-pr
```

`send-pr` prompts you for the name of a *site*. *site* is an alias on your local machine which points to an alternate Support Site. The Emacs interface to GNATS is described in a separate section, See Section 2.5 [Emacs], page 27.

### 2.2.3 Invoking `send-pr` from the shell

```
send-pr [ site ]
        [ -f problem-report | --file problem-report ]
        [ -t mail-address | --to mail-address ]
        [ --request-id ]
        [ -L | --list ] [ -P | --print ]
        [ -V | --version] [ -h | --help ]
```

*site* is an alias on your local machine which points to an address used by a Support Site. If this argument is not present, the default *site* is usually the site which you received `send-pr` from, or your local site if you use GNATS locally.

Invoking `send-pr` with no options calls the editor named in your environment variable `EDITOR` on a default PR template. If the environment variable `PR_FORM` is set, its value is used as a file name which contains a valid template. If `PR_FORM` points to a missing or unreadable file, or if the file is empty, `send-pr` generates an error message and opens the editor on a default template.

`-f` *problem-report*
`--file` *problem-report*
: Specifies a file, *problem-report*, where a completed Problem Report already exists. `send-pr` sends the contents of the file without invoking an editor. If *problem-report* is '`-`', `send-pr` reads from standard input.

---

[1] If typing '`M-x send-pr`' doesn't work, see your system administrator for help loading '`gnats.el`' into Emacs.

`-t` *mail-address*
`--to` *mail-address*
Sends the PR to *mail-address*. The default is preset when `send-pr` is configured. *This option is not recommended*; instead, use the argument *site* on the command line.

`-c` *mail-address*
`--cc` *mail-address*
Places *mail-address* in the `Cc:` header field of the message to be sent.

`--request-id`
Sends a request for a `>Submitter-Id:` to the Support Site.

`-L`
`--list` Prints the list of valid `>Category:` values on standard output. No mail is sent.

`-s` *severity*
`--severity` *severity*
Sets the initial value of the `>Severity:` field to *severity*.

`-P`
`--print` Displays the PR template. If the variable `PR_FORM` is set in your environment, the file it specifies is printed. If `PR_FORM` is not set, `send-pr` prints the standard blank form. If the file specified by `PR_FORM` doesn't exist, `send-pr` displays an error message. No mail is sent.

`-V`
`--version`
Displays the `send-pr` version number and a usage summary. No mail is sent.

`-h`
`--help` Displays a usage summary for `send-pr`. No mail is sent.

### 2.2.4 Submitting a Problem Report via direct e-mail

In addition to using `send-pr`, there is another way to submit a problem report. You can simply send an e-mail message to the support site.

To do this, look at the address in the '`To:`' field of the `send-pr` template. When you send unformatted e-mail to this address, GNATS processes the message as a new problem report, filling in as many fields from defaults as it can:

`Synopsis` The '`>Synopsis`' field is filled in by the '`Subject:`' header.

`Submitter ID`
GNATS will try to derive the '`>Submitter`' field from the address in the '`From:`' header.

`Description`
All of the text in the body of the e-mail message is put into the '`>Description`' field. Each line of the text is indented by one space, indicating that it is "quoted text" from the sender.

Other fields, such as category, version, severity, etc. are set to default values (if the GNATS administrator has set them).

### 2.2.5 Helpful hints

There is no orthodox standard for submitting effective bug reports, though you might do well to consult the section on submitting bugs for GNU `gcc` in section "Reporting Bugs" in *Using and Porting GNU CC*, by Richard Stallman. This section contains instructions on what kinds of information to include and what kinds of mistakes to avoid.

In general, common sense (assuming such an animal exists) dictates the kind of information that would be most helpful in tracking down and resolving problems in software.

- Include anything *you* would want to know if you were looking at the report from the other end. There's no need to include every minute detail about your environment, although anything that might be different from someone else's environment should be included (your path, for instance).
- Narratives are often useful, given a certain degree of restraint. If a person responsible for a bug can see that A was executed, and then B and then C, knowing that sequence of events might trigger the realization of an intermediate step that was missing, or an extra step that might have changed the environment enough to cause a visible problem. Again, restraint is always in order ("I set the build running, went to get a cup of coffee (Columbian, cream but no sugar), talked to Sheila on the phone, and then THIS happened. . .") but be sure to include anything relevant.
- Richard Stallman writes, "The fundamental principle of reporting bugs usefully is this: **report all the facts**. If you are not sure whether to state a fact or leave it out, state it!" This holds true across all problem reporting systems, for computer software or social injustice or motorcycle maintenance. It is especially important in the software field due to the major differences seemingly insignificant changes can make (a changed variable, a missing semicolon, etc.).
- Submit only *one* problem with each Problem Report. If you have multiple problems, use multiple PRs. This aids in tracking each problem and also in analyzing the problems associated with a given program.
- It never hurts to do a little research to find out if the bug you've found has already been reported. Most software releases contain lists of known bugs in the Release Notes which come with the software; see your system administrator if you don't have a copy of these.
- The more closely a PR adheres to the standard format, the less interaction is required by a database administrator to route the information to the proper place. Keep in mind that anything that requires human interaction also requires time that might be better spent in actually fixing the problem. It is therefore in everyone's best interest that the information contained in a PR be as correct as possible (in both format and content) at the time of submission.

## 2.3 Editing existing Problem Reports

Use `edit-pr` to make changes to existing PRs in the database. This tool can be invoked both from a shell prompt or from within GNU Emacs using '`M-x edit-pr`'.

`edit-pr` first examines the PR you wish to edit and locks it if it is not already locked. This is to prevent you from editing a PR at the same time as another user. If the PR you wish to edit is already in the process of being edited, `edit-pr` tells you the name of the person who owns the lock.

You may edit any non-readonly fields in the database. We recommend that you avoid deleting any information in the TEXT and MULTITEXT fields (such as '`>Description:`' and '`>How-To-Repeat:`' (see Section 1.4 [Problem Report format], page 6). We also recommend that you record the final solution to the problem in the '`>Fix:`' field for future reference.

After the PR has been edited, the PR is then resubmitted to the database, and the index is updated (see Section 4.5.1 [The `index` file], page 61). For information on `pr-edit`, the main driver for `edit-pr`, see Section 4.7 [Internal utilities], page 65.

If you change a field that requires a reason for the change, such as the '`>Responsible:`' or '`>State:`' fields in the default configuration, `edit-pr` prompts you to supply a reason for the change. A message is then appended to the '`>Audit-Trail:`' section of the PR with the changed values and the change reason.

Depending on how the database is configured, editing various fields in the PR may also cause mail to be sent concerning these changes. In the default configuration, any fields that generate '`>Audit-Trail:`' entries will also cause a copy of the new '`>Audit-Trail:`' message to be sent.

### 2.3.1 Invoking `edit-pr` from the shell

The usage for `edit-pr` is:

```
edit-pr [ -V | --version ] [ -h | --help ]
        [-d database | --database database] PR Number
```

Network-mode-only options:

```
[--host host | -H host] [--port port]
[--user user | -v user]
[--passwd passwd | -w passwd]
```

The options have the following meaning:

`-h, --help`
: Prints a brief usage message for edit-pr.

`-V, --version`
: Prints the version number for edit-pr.

`-d, --database`
: Specifies the database containing the PR to be edited; if no database is specified, the database named '`default`' is assumed. This option overrides the database specified in the `GNATSDB` environment variable.

`--host` *host*, `-H` *host*
: Specifies the hostname of the gnatsd server to communicate with. This overrides the value in the `GNATSDB` environment variable.

`--port` *port*
: Specifies the port number of the gnatsd server to communicate with. This overrides the value in the `GNATSDB` environment variable.

`--user` *user*, `-v` *user*
: Specifies the username to login with when connecting to the gnatsd server. This overrides the value in the `GNATSDB` environment variable.

`--passwd` *passwd*, `-w` *passwd*
: Specifies the password to login with when connecting to the gnatsd server. This overrides the value in the `GNATSDB` environment variable.

`edit-pr` calls the editor specified in your environment variable `EDITOR` on a temporary copy of that PR. (If you don't have the variable `EDITOR` defined in your environment, the default editor `vi` is used.)

Edit the PR, changing any relevant fields or adding to existing information. When you exit the editor, `edit-pr` prompts you on standard input for a reason if you have changed a field that requires specifying a reason for the change.

## 2.4 Querying the database

Obtain information from the database by using the program `query-pr`. `query-pr` uses search parameters you provide to find matching Problem Reports in the database. You can invoke `query-pr` from the shell or from within Emacs. `query-pr` uses the same arguments whether it is invoked from the shell or from Emacs.

PRs may be selected via the use of the `--expr` option, directly by number, or by the use of the (now deprecated) field-specific query operators.

By default, query options are connected with a logical AND. For example,

```
query-pr --category=foo --responsible=bar
```

only prints PRs which have a Category field of '`foo`' and a Responsible field of '`bar`'.

The `--or` option may be used to connect query options with a logical OR. For example,

```
query-pr --category=baz --or --responsible=blee
```

prints PRs which have either a Category field of '`baz`' or a Responsible field of '`blee`'.

It should be emphasized, however, that the use of these field-specific options is strongly discouraged, since they exist only for compatibility with older versions of GNATS and are likely to be deleted in the next release. The expressions specified by the `--expr` option are much more flexible (see below).

### 2.4.1 Invoking `query-pr`

From the shell, simply type *query-pr*, followed by any search parameters you wish to exercise. From Emacs, type *M-x query-pr*. `query-pr` prompts you for search parameters in the minibuffer.

`query-pr` can also be accessed by electronic mail, if your version of GNATS is configured for this. To use this feature, simply send mail to the address '`query-pr@`*your-site*' with command line arguments or options in the '`Subject:`' line of the mail header. GNATS replies to your mail with the results of your query. The default settings for the `query-pr` mail server are

```
--restricted --state="open|analyzed|feedback|suspended"
```

To override the '`--state`' parameter, specify '`--state=`*state*' in the `Subject:` line of the mail header. You can not query on confidential Problem Reports by mail.

The usage for `query-pr` is:

```
query-pr [--debug | -D] [--help | -h] [--version | -V]
         [--output file | -o file] [--list-databases]
         [--list-fields] [--list-input-fields]
         [--responsible-address address] [--field-type type]
         [--field-description description]
         [--valid-values values] [--format format | -f format]
         [--full | -F] [--summary | -q]
         [--database database | -d database] [--and | -&]
         [--or | -|] [--expr expr] [PR Number]
```

Non-network-mode options:

```
         [--print-sh-vars] [--print-directory-for-database]
```

Network-mode-only options:

```
         [--host host | -H host] [--port port]
         [--user user | -v user] [--passwd passwd | -w passwd]
```

Deprecated Options:

```
         [--list-categories | -j] [--list-states | -T]
         [--list-responsible | -k] [--list-submitters | -l]
         [--category category | -c category]
         [--synopsis synopsis | -y synopsis]
         [--confidential confidential | -C confidential]
         [--multitext multitext | -m multitext]
         [--originator originator | -O originator]
         [--release release | -A release]
         [--class class | -L class] [--cases cases | -E cases]
         [--quarter quarter | -Q quarter]
         [--keywords keywords | -K keywords]
         [--priority priority | -p priority]
         [--responsible responsible | -r responsible]
         [--restricted | -R] [--severity severity | -e severity]
         [--skip-closed | -x] [--sql | -i] [--sql2 | -I]
         [--state state | -s state]
         [--submitter submitter | -S submitter]
         [--text text | -t text]
         [--required-before date | -u date]
         [--required-after date | -U date]
         [--arrived-before date | -b date]
         [--arrived-after date | -a date]
         [--modified-before date | -B date]
         [--modified-after date | -M date]
         [--closed-before date | -z date]
         [--closed-after date | -Z date]
```

The options have the following meaning:

`--help, -h`
: Prints a help message.

`--version, -V`
: Displays the program version to stdout.

`--output` *file*, `-o` *file*
: The results of the query will be placed in this file.

`--database` *database*, `-d` *database*
: Specifies the database to be used for the query. If no database is specified, the database named default is assumed. (This option overrides the database specified in the `GNATSDB` environment variable; see Section 2.1 [Environment], page 13 for more information.)

`--list-categories, -j`
: Lists the available PR categories for the selected database.

`--list-states, -T`
: Lists the valid PR states for PRs in this database.

`--list-responsible, -k`
: Lists the users that appear in the database's responsible list.

`--list-submitters, -l`
: Lists the valid submitters for this database.

The previous –list-* options are deprecated and may be removed in future releases of GNATS; their functionality can be replaced with

```
query-pr --valid-values field
```

where *field* is one of '`Category`', '`Class`', '`Responsible`', '`Submitter-Id`', or '`State`'.

`--list-databases`
: Lists the known databases.

`--list-fields`
: Lists the entire set of field names for PRs in the selected database.

`--list-input-fields`
: Lists the fields that should be provided when creating a new PR for the currently-specified database. The fields are listed in an order that would make sense when used in a template or form.

`--field-type` *field*
: Returns the data type contained in PR field field. The current set of data types includes '`text`', '`multitext`', '`enum`', '`multienum`', '`integer`', '`date`', and '`text-with-regex-qualifier`'.

`--field-description` *field*
: Returns a human-readable description of the intended purpose of *field*.

`--valid-values` *field*
: For fields of type '`enum`', a list of valid values (one per line) is returned. Otherwise, a regular expression is returned that describes the legal values in *field*.

`--responsible-address` *name*
: The mail address of *name* is returned; *name* is assumed to be a name either appearing in the database's responsible list, or is otherwise a user on the system.

`--print-sh-vars`
: A set of '`/bin/sh`' variables is returned that describe the selected database. They include:

  `GNATSDB`

  The name of the currently-selected database.

  `GNATSDB_VALID`

  Set to 1 if the selected database is valid.

  `GNATSDBDIR`

  The directory where the database contents are stored.

  `DEBUG_MODE`

  Set to 1 if debug mode has been enabled for the database.

  `DEFAULTCATEGORY`

  The default category for PRs in the database.

  `DEFAULTSTATE`

  The default state for PRs in the database.

`--print-directory-for-database`
: Returns the directory where the selected database is located.

`--format` *format*, `-f` *format*
: Used to specify the format of the output PRs, See Section 2.4.2 [Formatting query-pr output], page 24 for a complete description.

`--full, -F`
: When printing PRs, the entre PR is displayed. This is exactly equivalent to

```
query-pr --format full
```

`--summary, -q`
: When printing PRs, a summary format is used. This is exactly equivalent to

  `query-pr --format` *summary*

`--debug, -D`
: Enables debugging output for network queries.

`--host` *host*, `-H` *host*
: Specifies the hostname of the gnatsd server to communicate with. This overrides the value in the `GNATSDB` environment variable.

`--port` *port*
: Specifies the port number of the gnatsd server to communicate with. This overrides the value in the `GNATSDB` environment variable.

`--user` *user*, `-v` *user*
: Specifies the username to login with when connecting to the gnatsd server. This overrides the value in the `GNATSDB` environment variable.

`--passwd` *passwd*, `-w` *passwd*
: Specifies the password to login with when connecting to the gnatsd server. This overrides the value in the `GNATSDB` environment variable.

`--and, -&, --or, -|`
: These options are used when connecting multiple query operators together. They specify whether the previous and subsequent options are to be logically ANDed or logically ORed.

`--expr` *expr*
: Specifies a query expression to use when searching for PRs. See Section 2.4.3 [Query expressions], page 25.

The remaining deprecated options are not described here, since their use is fairly obvious and their functionality is completely replaced by the use of the `--expr` option.

### 2.4.2 Formatting `query-pr` output

Printing formats for PRs are in one of three forms:

*formatname*
: This is a named format which is described by the database (specifically, these formats are described in the '`dbconfig`' file associated with the database). The default configuration contains five such formats: '`standard`', '`full`', '`summary`', '`sql`', and '`sql2`'.

  The first three are the ones most commonly used when performing queries. standard is the format used by default if no other format is specified.

  Use of the latter two are discouraged; they are merely kept for historical purposes. Other named formats may have been added by the database administrator.

*fieldname*
: A single field name may appear here. Only the contents of this field will be displayed.

"*printf string*" *fieldname fieldname ...*
: This provides a very flexible mechanism for formatting PR output. (The formatting is identical to that provided by the named formats described by the database configuration, See Section 4.3.5 [Named query definitions], page 52. The *printf string* can contain the following % sequences:

  `%[positionalspecifiers]s`: Prints the field as a string. The positional specifiers are similar to those of printf, as `+`, `-` and digit qualifiers can be used to force a particular alignment of the field contents.

  `%[positionalspecifiers]S`: Similar to `%s`, except that the field contents are terminated at the first space character.

  `%[positionalspecifiers]d`: Similar to `%s`, except that the field contents are written as a numeric value. For integer fields, the value is written as a number. For enumerated fields, the field is converted into a numeric equivalent (i.e. if the field can have two possible values, the result will be either 1 or 2). For date fields, the value is written as seconds since Jan 1, 1970.

`%F`: The field is written as it would appear within a PR, complete with field header.

`%D`: For date fields, the date is written in a standard GNATS format.

`%Q`: For date fields, the date is written in an arbitrary "SQL" format.

### 2.4.3 Query expressions

Query expressions are used to select specific PRs based on their field contents. The general form is

```
fieldname|"value" operator fieldname|"value" [booleanop ...]
```

*value* is a literal string or regular expression; it must be surrounded by double quotes, otherwise it is interpreted as a fieldname.

*fieldname* is the name of a field in the PR.

*operator* is one of:

`=` The value of the left-hand side of the expression must exactly match the regular expression on the right-hand side of the expression. See Appendix D [Querying using regular expressions], page 97.

`~` Some portion of the left-hand side of the expression must match the regular expression on the right-hand side.

`==` The value of the left-hand side must be equal to the value on the right-hand side of the expression.

The equality of two values depends on what type of data is stored in the field(s) being queried. For example, when querying a field containing integer values, literal strings are interpreted as integers. The query expression

```
Number == "0123"
```

is identical to

```
Number == "123"
```

as the leading zero is ignored. If the values were treated as strings instead of integers, then the two comparisons would return different results.

`!=` The not-equal operator. Produces the opposite result of the == operator.

`<,>` The left-hand side must have a value less than or greater than the right-hand side. Comparisons are done depending on the type of data being queried; in particular, integer fields and dates use a numeric comparison, and enumerated fields are ordered depending on the numeric equivalent of their enumerated values.

*booleanop* is either '`|`' (logical or), or '`&`' (logical and). The query expression

```
Category="baz" | Responsible="blee"
```

selects all PRs with a Category field of '`baz`' or a Responsible field of '`blee`'.

The not operator '`!`' may be used to negate a test:

```
! Category="foo"
```

searches for PRs where the category is not equal to the regular expression foo.

Parentheses may be used to force a particular interpretation of the expression:

```
!(Category="foo" & Submitter-Id="blaz")
```

skips PRs where the Category field is equal to '`foo`' and the Submitter-Id field is equal to '`blaz`'. Parentheses may be nested to any arbitrary depth.

Fieldnames can be specified in several ways. The simplest and most obvious is just a name:

```
Category="foo"
```

which checks the value of the category field for the value *foo*.

A fieldname qualifier may be prepended to the name of the field; a colon is used to separate the qualifier from the name. To refer directly to a builtin field name:

```
builtin:Number="123"
```

In this case, '`Number`' is interpreted as the builtin name of the field to check. (This is useful if the fields have been renamed. For further discussion of builtin field names, see Section 4.3 [The `dbconfig` file], page 46.

To scan all fields of a particular type, the *fieldtype* qualifier may be used:

```
fieldtype:Text="bar"
```

This searches all text fields for the regular expression '`bar`'.

Note that it is not required that the right-hand side of the expression be a literal string. To query all PRs where the PR has been modified since it was closed, the expression

```
Last-Modified != Closed-Date
```

will work; for each PR, it compares the value of its Last-Modified field against its Closed-Date field, and returns those PRs where the values differ. However, this query will also return all PRs with empty Last-Modified or Closed-Date fields. To further narrow the search:

```
Last-Modified != Closed-Date & Last-Modified != "" & Closed-Date != ""
```

In general, comparing fields of two different types (an integer field against a date field, for example) will probably not do what you want.

Also, a field specifier may be followed by the name of a subfield in braces:

```
State[type] != "closed"
```

or even

```
builtin:State[type] != "closed"
```

Subfields are further discussed in Section 4.3 [The `dbconfig` file], page 46.

### 2.4.4 Example queries

The following simple query:

```
query-pr --expr 'Category~"rats" & State~"analyzed"
                 & Responsible~"fred"'
```

yields all PRs in the database which contain the field values:

```
>Category:     rats        and
>Responsible:  fred        and
>State:        analyzed
```

The following query:

```
query-pr --expr 'State~"open|analyzed"'
```

yields all PRs in the database whose '`>State:`' values match either '`open`' or '`analyzed`' (see Appendix D [Querying using regular expressions], page 97. This search is useful as a daily report that lists all Problem Reports which require attention.

The following query:

```
query-pr --expr 'fieldtype:Text="The quick.*brown fox"'
```

yields all PRs whose TEXT fields contain the text '`The quick`' followed by '`brown fox`' within the same field. See Appendix D [Querying using regular expressions], page 97, which also contains further useful examples of query expressions.

## 2.5 The Emacs interface to GNATS

Emacs interface to GNATS provides basic access to GNATS databases, i.e. sending, editing, and querying Problem Reports. It also defines a simple major mode for editing '`dbconfig`' files.

This section provides an overview of using GNATS with Emacs. It does not describe the use of Emacs itself, for detailed instructions on using Emacs, see section "Top" in *GNU Emacs*. For installation instructions of the GNATS Emacs mode, see Section 3.2 [Installing utils], page 34.

Please note the Emacs interface was completely rewritten between GNATS 3 and GNATS 4. It now uses `gnatsd`, Appendix B [gnatsd], page 79, exclusively for its operations and uses modern Emacs features like faces. Its features are not complete though, you can send your suggestions and patches to the appropriate GNATS mailing list, Appendix E [Support], page 99.

### 2.5.1 Viewing Problem Reports

To view a particular Problem Report, use the command *`M-x view-pr`*. It asks for a Problem Report number and displays that Problem Report.

The displayed buffer is put in the view mode, section "Misc File Ops" in *GNU Emacs*. If you decide to edit the displayed Problem Report, use the command `e` (`gnats-view-edit-pr`).

`gnats-view-mode-hook`
: Hook run when `gnats-view-mode` is entered.

### 2.5.2 Querying Problem Reports

Querying the database is performed by the *`M-x query-pr`* command. The command prompts for the query expression, Section 2.4.3 [Query expressions], page 25, and displays a buffer with the list of the matching Problem Reports.

The list of the Problem Reports is displayed in the '`summary`' query format, Section 2.4.2 [Formatting query-pr output], page 24. Currently, the display format cannot be changed and it must output each Problem Report's number in the first column.

The Problem Report list buffer is put in the view mode, section "Misc File Ops" in *GNU Emacs*. You can use most of the standard view mode commands in it. Additionally, the following special commands are available:

*v*
*RET*
*mouse-2* View the current Problem Report (`gnats-query-view-pr`), Section 2.5.1 [Emacs viewing], page 27.

*e* Edit the current Problem Report (`gnats-query-edit-pr`), Section 2.5.4 [Emacs editing], page 29.

*g* Update the Problem Report list (`gnats-query-reread`). The last performed query is executed again and the buffer is filled with the new results.

*G* Perform new query (`query-pr`).

*s* Send new Problem Report (`send-pr`), Section 2.5.3 [Emacs submitting], page 28.

*D* Change the current database (`gnats-change-database`), Section 2.5.6 [Emacs and databases], page 30.

*q* Bury buffer, the buffer is put at the end of the list of all buffers. This is useful for quick escape of the buffer, without killing it.

If the value of the variable *gnats-query-reverse-listing* is non-`nil`, the listing appears in the reversed order, i.e. with the Problem Reports of the highest number first, in the buffer.

Similarly to other GNATS Emacs modes, there is a hook available for the Problem Report list.

`gnats-query-mode-hook`
Hook run when `gnats-query-mode` is entered.

### 2.5.3 Submitting new Problem Reports

You can submit new Problem Reports with the command *M-x send-pr*. The command puts you to the problem editing buffer, Section 2.5.4 [Emacs editing], page 29. The buffer is prefilled with the initial report fields and their default values, if defined. You can use the usual Problem Report editing commands, Section 2.5.4 [Emacs editing], page 29. When you have filled in all the fields, you can send the Problem Report by presing *C-c C-c*.

If you run *M-x send-pr* with a prefix argument, it runs the `gnats-change-database` command before putting you to the editing buffer, Section 2.5.6 [Emacs and databases], page 30.

You can set the following variables to get some fields pre-filled:

*gnats-default-organization*
Default value of the '`Organization`' field used in new Problem Reports.

*gnats-default-submitter*
Default value of the '`Submitter-Id`' field used in new Problem Reports.

### 2.5.4 Editing Problem Reports

To edit a particular Problem Report, use the command *M-x* `edit-pr`. It asks for a Problem Report number and puts the given Problem Report in the editing buffer. See Section 2.5.5 [Emacs editing buffer], page 29, for information how to edit the Problem Report in the buffer and how to submit your changes.

Note you can also start editing of a selected Problem Report directly from within the viewing buffer, Section 2.5.1 [Emacs viewing], page 27, or the query result buffer, Section 2.5.2 [Emacs querying], page 27.

### 2.5.5 The Problem Report editing buffer

When you invoke a Problem Report editing command, the Problem Report is put into a special editing buffer. The Problem Report is formatted similarly to the `query-pr -F` output, Section 2.4.2 [Formatting query-pr output], page 24. Field identifiers are formatted as

```
>Field:
```

with the text of the field following the identifier on the same line for single-line fields or starting on the next line for multi-line fields.

The Problem Report editing mode tries to prevent you from violating the Problem Report format and the constraints put on the possible field values. Generally, you can use usual editing commands, some of them have a slightly modified behavior though. (If you encounter a very strange behavior somewhere, please report it as a bug, Appendix E [Support], page 99.)

You can move between fields easily by pressing the *TAB* (`gnats-next-field`) or *M-TAB* (`gnats-previous-field`) keys.

The field tags are read-only and you cannot edit them nor delete them. If you want to "remove" a field, just make its value empty.

Editing a field value depends on the type of the edited field, Section 4.3.3 [Field datatypes], page 49. For text fields, you can edit the value directly, assuming you preserve the rule about single-line and multi-line values mentioned above.

For enumerated fields, you cannot edit the value directly. You can choose it from the list of the allowed values, either from the menu popped up by pressing the middle mouse button or from within minibuffer by pressing any key on the field's value. If the pressed key matches any of the allowed field values, that value is put as the default value after the minibuffer prompt. You can also cycle through the allowed field values directly in the editing buffer using the `SPACE` key. Enumerated field values are marked by a special face to not confuse you; you must have enabled font lock mode to benefit from this feature, section "Font Lock" in *GNU Emacs*.

Some field values can be read-only, you cannot edit them at all.

Once you have edited the Problem Report as needed, you can send it to the server with the *C-c C-c* command (`gnats-apply-or-submit`). Successful submission is reported by a message and the buffer modification flag in mode line is cleared. Then you can either kill the buffer or continue with further modifications.

`gnats-edit-mode-hook`
: Hook run when `gnats-edit-mode` is entered.

### 2.5.6 Changing the database

By default, the Emacs interface connects to the default database, Section 4.2 [databases file], page 45. If you want to connect to another database, use the command *M-x gnats-change-database*. It will ask you for the database name to use, server and port it can be accessed on, and your login name.

If you want to use the `gnatsd` command, Appendix B [gnatsd], page 79, directly, without connecting to a remote server or the localhost connection port, provide your local file system full path to `gnatsd` as the server name. Port number does not matter in this case.

If the database requires a password to allow you the access to it, you are prompted for the password the first time you connect to the database. If you provide an invalid password, you cannot connect to the database anymore and you have to run the *M-x gnats-change-database* command again.

### 2.5.7 dbconfig mode

The Emacs interface defines a simple major mode `gnats-dbconfig-mode` for editing '`dbconfig`' files, Section 4.3 [dbconfig file], page 46. It defines basic mode attributes like character syntax and font lock keywords, it does not define any special commands now.

`gnats-dbconfig-mode-hook`
: Hook run when `gnats-dbconfig-mode` is entered.

### 2.5.8 Other commands

*M-x unlock-pr*
: Ask for a Problem Report number and unlock that Problem Report. This function is useful if connection to a GNATS server was interrupted during an editing operation and further modifications of the Problem Report are blocked by a stealth lock.

### 2.5.9 Customization

All the user variables can be customized in the customization group `gnats`, section "Easy customization" in *GNU Emacs*.

# 3 Installing GNATS

See also Appendix A [Where the tools and utilities reside], page 73.

There are several steps you need to follow to fully configure and install GNATS on your system. You need `root` access in order to create a new account for `gnats` and to install the GNATS utilities. You may need `root` access on some systems in order to set up mail aliases and to allow this new account access to `cron` and `at`.

If you are updating an older version of GNATS rather than installing from scratch, see Section 3.8 [Upgrading from older versions], page 38.

GNATS installation relies on two other freely available software packages, which should be installed before you go on to configure and build GNATS. These are GNU `make` and `Texinfo`. Both are available from the GNU FTP site at `ftp://ftp.gnu.org`.

To build GNATS, you must:

- Run `configure`, with correct options if the defaults are unsuitable for your site. See Section 3.1 [Configuring and compiling the software], page 32. Default installation locations are in Appendix A [Where GNATS lives], page 73.
- Compile the GNATS programs on your system. See Section 3.1 [Configuring and compiling the software], page 32.
- Create an initial database by using the `mkdb` command. See Section 3.3 [Setting up the default database], page 35.
- Set up periodic jobs, using cron, to handle Problem Reports arriving by mail. See Section 3.4 [Setting up periodic jobs], page 36.
- Set up mail aliases for GNATS. See Section 3.5 [Setting up mail aliases], page 36.
- Install the GNATS tools and utilities locally, and install the user tools (`query-pr`, `edit-pr`, `send-pr`) on every machine in your local network. See Section 3.7 [Installing the user tools], page 38.
- Install the GNATS daemon '`gnatsd`'. See Section 3.6 [Installing the daemon], page 37.
- Update the local configuration files

  ```
  dbconfig       categories     submitters
  responsible    gnatsd.access  states
  classes      addresses
  ```

  in '*DATABASEDIR*/`gnats-adm`', as well as the files

  ```
  gnatsd.host_access    databases
  ```

  in '*SYSCONFDIR*/`gnats`' (usually '`/usr/local/etc/gnats`').
- If there are people outside your organization who will be submitting PRs or who are supposed to be able to query and/or edit PRs, you may need to instruct them to obtain and build the GNATS tools `query-pr`, `edit-pr` and `send-pr` for their systems. However, for many sites, setting up a remote access interface to GNATS, such as Gnatsweb is a better solution since this requires no configuration on the remote side.

## 3.1 Configuring and compiling the software

1. First, unpack your distribution. We provide source code in a `tar` file which was compressed using `gzip`. The code can be extracted into a directory *unpackdir* using

   ```
   cd unpackdir
   gunzip gnats-4.0-beta1.tar.gz
   tar xvf gnats-4.0-beta1.tar
   ```

   The sources reside in a directory called '`gnats-4.0-beta1`' when unpacked. We call this the *top level* of the source directory, or *srcdir*. The sources for the GNATS tools are in the subdirectory '`gnats-4.0-beta1/gnats/*`'. Lists of files included in the distribution are in each directory in the file '`MANIFEST`'.

2. As of GNATS version 4, having Emacs installed on the GNATS server is no longer a requirement. If you do not have Emacs installed, disregard this step altogether.

   You may wish to alter the installation directory for the Emacs lisp files. If your Emacs lisp library is not in '*prefix*`/share/emacs/site-lisp`', edit the file *srcdir*`/gnats/Makefile.in`. Change the variable `lispdir` from '*prefix*`/emacs/site-lisp`' to the directory containing your Emacs lisp library. For information on *prefix*, see Section A.1 [prefix], page 73.

3. Create an account for the `gnats` user. You can actually name this user whatever you want to, as long as it is a valid username on your system, but we strongly recommend that you call the user `gnats`. If you do decide to give it some other name, remember to use the option `--with-gnats-user` when running `configure` below. Below, we will anyway refer to this user by the name `gnats`.

   This user must have an entry in the file '`/etc/passwd`'. As for ordinary users, create a standard home directory for the `gnats` user. The default `PATH` for this user should contain '*exec-prefix*`/bin`' and '*exec-prefix*`/libexec/gnats`'. The *exec-prefix* value is configurable with the `--exec-prefix` configure option described below, but for standard installations, these two directories correspond to '`/usr/local/bin`' and '`/usr/local/libexec/gnats`'.

4. Run `configure`. You can nearly always run `configure` with the simple command

   ```
   ./configure
   ```

   and the "Right Thing" happens:

   - GNATS is configured in the same directory you unpacked it in;
   - when built, GNATS runs on the machine you're building it on;
   - when installed, files are installed under '`/usr/local`' (see Appendix A [Where GNATS lives], page 73).
   - all GNATS utilities operate on the *default database*, assumed to be named *default* and to be located in '`/usr/local/com/default`', unless you invoke the utilities with `-d` *databasename* or `--directory=`*databasename*, or set the *GNATSDB* environment variable to point to some other database.

   The most common options to `configure` are listed below:

   ```
   configure [ --prefix=prefix ]
             [ --exec-prefix=exec-prefix ]
             [ --with-gnats-service=service-name ]
   ```

```
[ --with-gnats-user=username ]
[ --with-gnatsd-user-access-file=path ]
[ --with-gnatsd-host-access-file=path ]
[ --with-gnats-dblist-file=path ]
[ --with-gnats-default-db=path ]
[ --with-kerberos ] [ --with-krb4 ]
[ --verbose ]
```

`--prefix=`*prefix*
: All host-independent programs and files are to be installed under *prefix*. (Host-dependent programs and files are also installed in *prefix* by default.) The default for *prefix* is '`/usr/local`'. See Appendix A [Where GNATS lives], page 73.

`--exec-prefix=`*exec-prefix*
: All host-dependent programs and files are to be installed under *exec-prefix*. The default for *exec-prefix* is *prefix*. See Appendix A [Where GNATS lives], page 73.

`--with-gnats-service=`*service-name*
: Set *service-name* to be the GNATS network service. Default name is *support*.

`--with-gnats-user=`*username*
: Set *username* to be the user name for GNATS. Default username is *gnats*.

`--with-gnatsd-user-access-file=`*path*
: Set global (across all databases) gnatsd user access file to *path*. Default is '`/usr/local/etc/gnats/gnatsd.access`'. Per-database user access permissions are set in a '`gnatsd.access`' file in the '`gnats-adm`' subdirectory of each database.

`--with-gnatsd-host-access-file=`*path*
: Set global (across all databases) gnatsd host access file to *path*. Default is '`/usr/local/etc/gnats/gnatsd_host.access`'. There is currently no way to specify host access permissions on a per-database basis.

`--with-gnats-dblist-file=`*path*
: Specify the file containing the list of databases.

  Default is '*prefix*`/etc/gnats/databases`'.

`--with-gnats-default-db=`*path*
: Specify the default database to use when GNATS tools are invoked without the `-d` or `--databasename` option, and when the *GNATSDB* envrionment variable hasn't been set. Default is '`/`*prefix*`/com/gnatsdb`'.

`--with-kerberos`
: Include code for Kerberos authentication.

`--with-krb4`
: Support Kerberos 4.

`--verbose`
: Give verbose output while `configure` runs.

`configure` supports several more options which allow you to specify in great detail where files are installed. For a complete list of options, run `./configure --help` in the source directory.

You can build GNATS in a different directory (*objdir*) from the source code by calling the `configure` program from the new directory, as in

```
mkdir objdir
cd objdir
srcdir/configure ...
```

By default, `make` compiles the programs in the same directory as the sources (*srcdir*).

5. Make sure you have GNU `make`, then run

   ```
   make all info
   ```

   from the directory where `configure` created a '`Makefile`' (this is *objdir* if you used it, otherwise *srcdir*.) These targets indicate:

   `all` Compile all programs

   `info` Create '`info`' files using `makeinfo`.

## 3.2 Installing the utilities

The following steps are necessary for a complete installation. You may need `root` access for these.

1. Install the utilities by invoking

   ```
   make install install-info
   ```

   These targets indicate:

   `install` Installs all programs into their configured locations (see Appendix A [Where GNATS lives], page 73).

   `install-info`
   Installs '`info`' files into their configured locations (see Appendix A [Where GNATS lives], page 73).

   After you have installed GNATS, you can remove the object files with

   ```
   make clean
   ```

2. If you do not have Emacs installed on your GNATS server, this step should be skipped.

   Place the following lines in the file '`default.el`' in your Emacs lisp library, or instruct your local responsible parties to place the lines in their '`.emacs`':

   ```
   (autoload 'send-pr "gnats"
      "Command to create and send a problem report." t)
   (autoload 'edit-pr "gnats"
      "Command to edit a problem report." t)
   (autoload 'view-pr "gnats"
      "Command to view a problem report." t)
   (autoload 'query-pr "gnats"
   ```

```
   "Command to query information about problem reports." t)
(autoload 'unlock-pr "gnats"
   "Unlock a problem report." t)
(autoload 'gnats-dbconfig-mode "gnats"
  "Major mode for editing the `dbconfig' GNATS configuration file." t)
(add-to-list 'auto-mode-alist '("\\<dbconfig$" . gnats-dbconfig-mode))
```

## 3.3 Installing the default database

For the following steps, log in as the user `gnats`.

We are now going to initialize the default GNATS database. Run the following command:

```
mkdb default
```

This creates a database named `default`, with all its data stored below the directory '*prefix*/com/gnatsdb', in a default installation this corresponds to '/usr/local/com/gnatsdb'. If you specified the `--with-gnats-default-db` option when running configure, the default database will be created under the directory you specified instead. `mkdb` creates the database directory itself, together with three different subdirectories[1]:

- A directory for the mandatory GNATS category *pending*.
- A 'gnats-queue' directory for queueing new messages to GNATS before they are processed by `file-pr`.
- The administrative directory 'gnats-adm'. This directory is populated with default configuration files from the '*prefix*/etc/gnats/defaults' directory.

The next configuration step is to edit the default files copied to the database's 'gnats-adm' directory by `mkdb`.

The default 'dbconfig' file installed by `mkdb` provides a good basis for many GNATS databases. The default file causes similar behaviour to the 3.x versions of GNATS. However, even if this might be precisely what you want, you should still go through the file and check that the default settings suit your needs. See Section 4.3 [The 'dbconfig' file], page 46.

Then edit the files 'categories', 'responsible', and 'submitters' in the 'gnats-adm' directory (see Section 4.4 [Other database-specific config files], page 57) to reflect your local needs. For special configurations, you may also have to edit the 'states' and 'classes' files.

If you used the `--with-gnats-default-db` option in the pre-build configure to change the location of the default database, you need to edit the 'databases' config file, see Section 4.2 [The 'databases file'], page 45. This file is by default located in the '*prefix*/etc/gnats' directory, but may have been changed by the option `--with-gnats-dblist-file` option during configure.

---

[1] Upgraders from older versions of GNATS should note that category directories are now created "on-the-fly" as needed by default.

## 3.4 Setting up periodic jobs

Allow the new user `gnats` access to `cron` and `at`. To do this, add the name `gnats` to the files '`cron.allow`' and '`at.allow`', which normally reside in the directory '`/var/spool/cron`'. If these files do not exist, make sure `gnats` does not appear in either of the files '`cron.deny`' and '`at.deny`' (in the same directory). If you changed the name of the GNATS user during configure, remember to substitute as appropriate in the previous steps.

Create a `crontab` entry that periodically runs the program `queue-pr` with the '`--run`' option (see Section 4.7.1 [queue-pr], page 65). For example, to run '`queue-pr --run`' every ten minutes, create a file called '`.mycron`' in the home directory of the user `gnats` which contains the line:

```
0,10,20,30,40,50 * * * * exec-prefix/libexec/gnats/queue-pr --run
```

(Specify the full path name for `queue-pr`.) Then run

```
crontab .mycron
```

See the `man` pages for `cron` and `crontab` for details on using `cron`.

## 3.5 Setting up mail aliases

The following mail aliases must be added on the machine where the GNATS server is installed. The instructions below are for Sendmail or Sendmail-like mail systems. If these instructions don't fit your system, particularly if you do not have an '`aliases`' file, ask your mail administrator for advice.

The following aliases should be placed in the file '`/etc/aliases`'. Yoy may need `root` access to add these aliases:

- Create an alias for the GNATS administrator. This address should point to the address of the person in charge of administrating GNATS:

  ```
  gnats-admin:  address
  ```

- Create an alias to redirect incoming Problem Reports. This alias should redirect incoming mail via a *pipe* to the program '`queue-pr -q`'. For example, if Problem Reports coming to your site are to arrive at the address '`bugs@your.company.com`', create an alias to the effect of:

  ```
  bugs:  "| exec-prefix/libexec/gnats/queue-pr -q"
  ```

  This places incoming Problem Reports in the '*gnats-queue*' directory of your database. Remember to fill in the full path of the `queue-pr` command as appropriate for your installation.

- You may also wish to forward a copy of each incoming Problem Report to a log. This can be accomplished with something like:

  ```
  bug-q: "| exec-prefix/libexec/gnats/queue-pr -q"
  bug-log:  /some/path/bugs.log
  bugs:      bug-q, bug-log
  ```

  This configuration archives incoming Problem Reports in the file '`bug.log`', and also feeds them to the program `queue-pr`. (Remember, '`bug.log`' needs to be world-writable, and should be pruned regularly; see Chapter 4 [GNATS Administration],

page 43.) In order for the log file to protect fully against data loss in case a disk runs full, try to place it on a different disk volume than the GNATS database.

- If you want your users to be able to query the database by mail, use the following alias:

  ```
  query-pr: "| exec-prefix/libexec/gnats/mail-query"
  ```

  The `mail-query` program uses '`--restricted`' to search on the database, and by default only searches for PRs that aren't closed (see Section 2.4 [Querying the database], page 20).

## 3.6 Installing the daemon

By default, the daemon and clients are set to use port 1529. Add the line

```
support 1529/tcp # GNATS
```

to your '`/etc/services`' file. If you want a different service name, configure GNATS with

```
--with-gnats-service=servicename
```

In your '`inetd.conf`' file, add the line

```
support stream tcp nowait gnats /usr/local/libexec/gnats/gnatsd gnatsd
```

adjusting the path accordingly if you used configure options to make changes to the defaults. To make `inetd` start spawning the GNATS daemon when connected on that port, send it a hangup signal (`HUP`).

Some operating systems have replaced `inetd` with the more modern `xinetd`. Instead of editing '`inetd.conf`', you should create the file '`/etc/xinetd.d/support`', containing something like the following:

```
service support
{
   disable=no
   socket_type=stream
   protocol=tcp
   wait=no
   user=gnats
   server=/usr/local/libexec/gnats/gnatsd
   server_args=gnatsd
}
```

If you specified a different service name when running `configure`, you need to give the file the same name as the service name, and you need to adjust the `service` line above. If the `--prefix` or `--exec-prefix` options were passed to `configure`, adjust the `server` line above, and if you used the `--with-gnats-user` option, adjust the `user` line.

Then restart `xinetd` to make the new configuration current.

If you use an Internet superserver different from `inetd` or `xinetd`, please refer to its documentation for information how to configure it.

At this point, you will probably want to set the access permissions of the different hosts that are going to be accessing your databases. The access permissions can currently only be set on a global scale (that is, across all the databases on a GNATS server). The location and name of the global host access configuration file can be set during the pre-build configure as shown above, but by default the file is '`/usr/local/etc/gnats/gnatsd_host.access`'. It lists the hosts allowed to access your server, and what their default access levels are. Each line in the file denotes one server, or one part of a network domain. There are three fields on each line, but only two are currently used. To grant all hosts from the domain *site.com* edit access, use this line:

*site.com*:*edit*

If you run a GNATS web interface or similar tool on the same machine as the server is running on, you probably want to grant *localhost* edit access:

*localhost*:*edit*

If you are using Kerberos, the '`gnatsd_host.access`' file shows the sites that don't require Kerberos authentication.

The third field might in the future be used for things like controlling what categories, submitter-id's PRs, etc., can be accessed from that site. Access attempts that are denied are logged to the syslog messages file ('`/var/adm/messages`' on many systems).

## 3.7 Installing the user tools

When you install the GNATS utilities, the user tools are installed by default on the host machine. If your machine is part of a network, however, you may wish to install the user tools on each machine in the network so that responsible parties on those machines can submit new Problem Reports, query the database, and edit existing PRs. To do this, follow these steps *on each new host*. You may need `root` access on each machine.

(still to be written...)

## 3.8 Upgrading from older versions

The following procedure covers an upgrade from all GNATS 3 versions newer than 3.108. If your installation is an older 3.10x version, or even the ancient 3.2 version, you need to review the '`UPGRADING.old`' file in the GNATS distribution before carrying out the steps detailed here.

### 3.8.1 Overview

Although almost all of the GNATS internals have been redesigned and rewritten for GNATS 4, little has changed in the format and structure of the database data. The only change that needs to be taken into account when upgrading is the fact that the database index format is binary in a default installation of GNATS 4. Thus, you will need to regenerate your database index by using the `gen-index` tool.

Apart from building and installing new binaries, the major changes which impinge on the upgrade procedure are all on the configuration side. The main database configuration file, '`dbconfig`', is far more complex and powerful than the old '`config`' file, and while the installation process creates a sensible set of default values which are similar to GNATS 3.11x's defaults, you still need to migrate any changes you may have made to your own local configuration.

Another aspect which needs consideration are remote submitter sites. Such sites either need to be instructed to upgrade their locally installed copies of the GNATS user tools (send-pr, edit-pr and query-pr), or they should be given access through interfaces such as Gnatsweb.

Since the GNATS network daemon has been completely reworked, with an entirely new command set, all network-based interfaces, such as Gnatsweb and TkGnats need to be upgraded to versions that support GNATS 4. The 'contrib' directory of this distribution contains some third-party interfaces, and the 'README' file contains pointers to where you can obtain the newest versions of these tools.

This document only deals with upgrading GNATS itself. Third-party tools should have separate upgrading instructions in their distributions.

### 3.8.2 Upgrading

1. Before you begin, make a backup of your entire GNATS database directory hierarchy, the GNATS executables directory and the GNATS user tools (send-pr, query-pr etc.) The locations of these may vary, but in a default GNATS 3 installation, the database(s) reside under '/usr/local/share/gnats', the executables are located in '/usr/local/libexec/gnats' and the user tools reside in '/usr/local/bin'.
2. (optional) In order to avoid confusing your users, you may want to remove the old GNATS 3 executables and tools, escpecially if you plan to install GNATS 4 in a different location than version 3.
3. Build and install GNATS 4. See Chapter 3 [Installing GNATS], page 31. It is recommended that you use the --with-gnats-default-db option when running configure, in order to set the default database to be one of your already existing GNATS 3 databases.
4. Edit the GNATS 'databases' file, located in '*SYSCONFDIR*/gnats'. In a default GNATS 4 installation this is '/usr/local/etc/gnats'. Add entries for all your old GNATS 3 databases. See Section 4.2 [The 'databases' file], page 45.
5. In GNATS 3, the file 'gnatsd.conf' specifies minimum access levels for the different hosts accessing the GNATS daemon, gnatsd. There is one 'gnatsd.conf' for each database. In GNATS 4, these files have been replaced by a single file named 'gnatsd.host_access' which contains settings that apply across all the databases on the server. This file is located in the same directory as the 'databases' file. You need to combine the host access settings from all your GNATS 3 databases and add them to the 'gnatsd.host_access' file. Note that you are no longer able to control host access on a per-database basis. Optionally, you may delete the old 'gnatsd.conf' files. See Appendix C [Controlling access to GNATS databases], page 93.
6. Next, you need to migrate the settings in the old 'config' files of your databases to corresponding 'dbconfig' files. The database you specified with the --with-gnats-default-db configure option got a default 'dbconfig' installed. This default file contains field definitions etc. which makes this version of GNATS behave almost exactly like older versions. Copy this default file to the 'gnats-adm' directories of any other GNATS databases that you may have on your host before you proceed to migrate your old configuration settings.

   The following is a list of the configuration directives that may be present in a 'config' file and their counterparts (if any) in GNATS 4.

*GNATS_ADDR*
: This setting has no counterpart in GNATS 4, since GNATS no longer needs to know its own mail address.

*GNATS_ADMIN*
: This setting is now set in the '`responsible`' file in the '`gnats-adm`' directory of your database(s).

*GNATS_SITE*
: GNATS 4 has no concept of a named 'site', so this directive is obsolete.

*SUBMITTER*
: Obsolete, since it relates to *GNATS_SITE*.

*DEFAULT_RELEASE*
*DEFAULT_ORGANIZATION*
: The GNATS 4 '`dbconfig`' file has separate configuration sections for each defined field. Field defaults are set with the `default` keyword in these sections. See Section 4.3 [The '`dbconfig`' file], page 46.

*NOTIFY*
: Controlled by the `notify-about-expired-prs` setting in the '`dbconfig`' file.

*ACKNOWLEDGE*
: Controlled by the `send-submitter-ack` setting in the '`dbconfig`' file.

*DEFAULT_SUBMITTER*
: The default submitter is now always the first entry in the '`submitters`' file of your database.

*KEEP_RECEIVED_HEADERS*
: Controlled by the `keep-all-received-headers` setting in the '`dbconfig`' file.

*DEBUG_MODE*
: Controlled by the `debug-mode` setting in the '`dbconfig`' file.

*BDAY_START*
*BDAY_END*
*BWEEK_START*
*BWEEK_END*
: Controlled by the settings `business-day-hours` and `business-week-days` in the '`dbconfig`' file.

*DEFINE_CATEGORY*
: The default category for PRs that arrive without one is now the first category listed in the '`categories`' file of your database.

After your are done migrating the settings, you may optionally delete the old '`config`' files. Since there are many more configuration settings available in the GNATS 4 '`dbconfig`' file, you should take some time to review them all before proceeding. See Section 4.3 [The '`dbconfig`' file], page 46.

7. GNATS 4 uses a different password format in the '`gnatsd.access`' file than older versions, since it supports `crypt()` and MD5 passwords (see Appendix C [Controlling

access to GNATS databases], page 93). You need to translate your old '`gnatsd.access`' files to the new format by using the `gnats-pwconv` tool which was installed in the '*EXEC-PREFIX*`/libexec/gnats`' directory, typically '`/usr/local/libexec/gnats`'. See Section 4.6.6 [Managing user passwords], page 64.

8. The final step involves regenerating the indexes of your databases. For this, log in as the user `gnats`. Then run the `gen-index` command for each of your databases. See Section 4.6 [Administrative Utilities], page 62 for details on how to use `gen-index`.
9. Sit back and enjoy your new GNATS 4 setup...

# 4 GNATS Administration

In daily usage, GNATS is self-maintaining. However, there are various administrative duties which need to be performed periodically. Also, requirements may change with time, so it may be necessary to make changes to the GNATS configuration at some point:

*emptying the* `pending` *directory*
: If a Problem Report arrives with a '`>Category:`' value that is unrecognized by the '`categories`' file, or if that field is missing, GNATS places the PR in the '`pending`' directory (see Appendix A [Where the tools and utilities reside], page 73). PRs submitted in free-form by email will always be filed in the '`pending`' directory. If so configured, GNATS sends a notice to the `gnats-admin` and to the party responsible for that submitter (as listed in the '`submitters`' file) when this occurs.

  To have these "categoryless" PRs filed correctly, you can then use a GNATS tool such as `edit-pr` to set the correct category of each PR in the '`pending`' directory.

  In order to protect yourself from problems caused by full disks, you should arrange to have all mail that is sent to the GNATS database copied to a log file (Section 3.5 [Setting up mail aliases], page 36). Then, should you run out of disk space, and an empty file ends up in the database's '`pending`' directory, you need only look in the log file, which should still contain the full message that was submitted.

*adding another database*
: GNATS supports multiple databases. If you find at some point that you need to add another database to your server, the `mkdb` tool does most of the work for you. See Section 4.6.1 [Adding another database], page 62.

*adding new categories*
: Most installations of GNATS will only require you to add a new line to the '`categories`' file. The category directory will then be created automatically as needed. However, if automatic directory creation has been switched off in the '`dbconfig`' file (see Section 4.3 [The `dbconfig` file], page 46), you need to use the '`mkcat`' program.

*removing categories*
: To remove a category, you need to make sure the relevant subdirectory is empty (in other words, make sure no PRs exist for the category you wish to remove). You can then remove the category listing from the '`categories`' file, and invoke

  `rmcat` *category*...

  to remove *category* (any number of categories may be specified on the command line to `rmcat`, so long as they abide by the above constraints).

*adding and removing maintainers*
: Edit the '`responsible`' file to add a new maintainer or to remove an existing maintainer. See Section 4.4.2 [The `responsible` file], page 58.

*building a new index*

If your index becomes corrupted, or if you wish to generate a new one for some reason, use the program `gen-index` (see Section 4.6.4 [Regenerating the index], page 63).

*pruning log files*

Log files often grow to unfathomable proportions. As with gardening, it is best to prune these as they grow, lest they take over your disk and leave you with no room to gather more Problem Reports. If you keep log files, be sure to keep an eye on them. (See Section 3.5 [Setting up mail aliases], page 36.)

*BACKING UP YOUR DATA*

Any database is only useful if its data remains uncorrupted and safe. Performing periodic backups ensures that problems like disk crashes and data corruption are reversible.

See Appendix A [Where GNATS lives], page 73.

## 4.1 Overview of GNATS configuration

See Appendix A [Where GNATS lives], page 73.

GNATS has two different kinds of configuration file. The *site-wide* configuration files determine overall behaviour across all the databases on your machine, while the *database-specific* configuration files determine how GNATS behaves when dealing with a specific database. These files can be edited at any time — the next time a GNATS tool is invoked, the new parameters will take effect.

These are the site-wide configuration files used by GNATS:

`databases`

Specifies database names and their associated parameters, such as in which directory they are located. This file is used by the GNATS clients to determine the location of a database referred to by name. See Section 4.2 [The `databases` file], page 45.

`gnatsd.host_access`

Controls access levels for the different machines that will do lookups in the databases on this machine. See Appendix C [GNATS access control], page 93.

The database-specific configuration is determined by the following files in the '`gnats-adm`' subdirectory of the database directory.

`dbconfig` Controls most aspects of how GNATS behaves when dealing with your database. See Section 4.3 [The `dbconfig` file], page 46.

`categories`

The list of categories that GNATS accepts as valid for the '`>Category:`' field, and the maintainers responsible for each category. Update this file whenever you have a new category, or whenever a category is no longer valid. You must also update this file whenever responsibility for a category changes, or if a maintainer is no longer valid. See Section 4.4.1 [The `categories` file], page 57.

`responsible`
: An alias list mapping names to their associated mailing addresses. The names in this list can have multiple email addresses associated with them. If a responsible user does not show up in this list, they are assumed to be a user local to the system. This list is not associated with just the responsible user field; all email addresses are mapped through this file whenever mail is sent from GNATS. See Section 4.4.2 [The `responsible` file], page 58.

`submitters`
: Lists sites from whom GNATS accepts Problem Reports. The existence of this file is mandatory, although the feature it provides is not; see Section 4.4.3 [The `submitters` file], page 59.

`addresses`
: Mappings between submitter IDs and submitters' e-mail addresses. Use of this file is optional. If you get Problem reports where the '`>Submitter`' field is not filled in, GNATS will use entries in this file to try to derive the submitter ID from the e-mail headers. See Section 4.4.5 [The `addresses` file], page 60.

`states`
: Lists the possible states for Problem Reports, typically ranging from *open* to *closed*. See Section 4.4.5 [The `states` file], page 60.

`classes`
: Lists the possible classes of Problem Report. This provides an easy way to have "subcategories", for example by setting up classes such as `sw-bug`, `doc-bug`, `change-request` etc. See Section 4.4.6 [The `classes` file], page 61.

`gnatsd.access`
: Specify the access levels for different users to your database. See Appendix C [GNATS access control], page 93.

## 4.2 The `databases` file

The '`databases`' configuration file is a site-wide configuration file containing the list of GNATS databases that are available either on the host itself or remotely over the network, together with some parameters associated with each database.

The file contains one line for each database. For databases located on the host itself, each line consists of three fields separated by colons:

*database name:short description of database:path/to/database*

The first field is the database name. This is the name used to identify the database when invoking programs such as `query-pr` or `send-pr`, either by using the `--database` option of the program or by setting the *GNATSDB* environment variable to the name of the database. The second field is a short human-readable description of the database contents, and the final field is the directory where the database contents are kept.

For a database that is located across a network, but which should be accessible from this host, the entry for the database should look like this:

*database name:short description of database::hostname:port*

The first two fields are the same as for local databases, the third field is empty (notice the two adjacent '`:`' symbols, indicating an empty field), the fourth field is the hostname of the remote GNATS server, and the fifth field is the port number that the remote GNATS is running on.

If GNATS was built with default options, the ‘`databases`’ file will be located in the ‘`/usr/local/etc/gnats`’ directory. However, if the option `--with-gnats-dblist-file` was used during building of GNATS, the ‘`databases`’ file has the name and location given to this option. A sample ‘`databases`’ file is installed together with GNATS.

Note that if you add a new local database, you must create its data directory, including appropriate subdirectories and administrative files. This is to be best done using the `mkdb` tool, See Section 4.6.1 [mkdb], page 62.

## 4.3 The `dbconfig` file

The ‘`dbconfig`’ configuration file controls the configuration of a GNATS database. Each database has its own individual copy of this file, which is located in the ‘`gnats-adm`’ subdirectory of the database.

The file consists of standard plain text. Whitespace is completely optional and is ignored. Sets of braces ‘`@`’ are used to delimit the different sections, and all non-keyword values must be surrounded with double quotes. The values in ‘`dbconfig`’ can be changed at any time; the new values take effect for all subsequent iterations of GNATS tools.

The ‘`dbconfig`’ file contains 6 major sections, which must appear in the following order:

- Overall database configuration
- Individual field configuration
- Named query definitions
- Audit-trail and outgoing email formats
- Index file description
- Initial Problem Report input fields

The different sections are described below. While reading the following sections, it will be useful to refer to the sample ‘`dbconfig`’ file which is installed when a new database is initialized with the `mkdb` tool. In fact, the sample file provides a configuration that should be usable for a great range of sites, since it reproduces the behaviour of the older, less customizable 3.x versions of GNATS.

### 4.3.1 Overall database configuration

The overall database options are controlled by settings in the `database-info` section of the ‘`dbconfig`’ file. The following is the general format of this section:

```
database-info {
  [options]
}
```

The following options and values may be used in the `database-info` section:

`debug-mode` *true* | *false*

If set to `true`, the database is placed into debug mode. This causes all outgoing email to be sent to the *gnats-admin* user listed in the ‘`responsible`’ file of the database. The default value is `false`.

`keep-all-received-headers` *true* | *false*
: If set to `true`, all of the Received: headers for PRs submitted via email are kept in the PR. Otherwise, only the first one is kept. The default value is `false`.

`notify-about-expired-prs` *true* | *false*
: If set to `true`, notification email about expired PRs is sent via the `at-pr` command. Otherwise, required times for PR fixes are not used. The default value is `false`.

`send-submitter-ack` *true* | *false*
: When new PRs are submitted to the database, an acknowledgment email will be sent to the submitter of send-submitter-ack is set to `true`. This is in addition to the normal notification mail to the person(s) responsible for the new PR. The default value is `false`.

`libexecdir` "*directory*"
: Specifies the directory where the GNATS administrative executables are located. In particular, `at-pr` and `mail-pr` are invoked from this directory. The default value is the empty string, which is unlikely to be useful.

`business-day-hours` *day-start - day-end*
: Used to specify the hours that define a business day. The values are inclusive and should be given in 24-hour format, with a dash separating them. GNATS uses these values to determine whether the required completion time for a PR has passed. The default values are 8 for `day-start` and 17 for `day-end`.

`business-week-days` *week-start - week-end*
: Specifies the start and ending days of the business week, where 0 is Sunday, 1 is Monday, etc. The days are inclusive, and the values should be given with a dash separating them. GNATS uses these values to determine whether the required completion time for a PR has passed. The default values are 1 for `week-start` and 5 for `week-end`.

`create-category-dirs` *true* | *false*
: If set to `true`, database directories for categories are automatically created as needed. Otherwise, they must be created manually (usually with the `mkcat` script). It is recommended that the default value of `true` be kept.

`category-dir-perms` *mode*
: Standard octal mode-specification specifying the permissions to be set on auto-created category directories. Default is `0750`, yielding user read, write and execute, and group read and execute.

### 4.3.2 Individual field configuration

Each type of field in a PR must be described with a `field` section in the '`dbconfig`' file. These sections have the following general structure:

```
field "fieldname" {
  description "string"
  [ field-options ... ]
  datatype [ datatype-options ... ]
```

```
    [ on-change { edit-options ... } ]
  }
```

`fieldname` is used as the field header in the PR. The characters `>` and `:` are used internally as field markers by GNATS, so they must not be used in fieldnames.

The order in which the `field` sections appear in the '`dbconfig`' file determines the order in which they appear in the PR text. There is no required order, unlike previous versions of GNATS — the Unformatted field and multitext fields may appear anywhere in the PR.

The following `field-options` may be present within a `field` section:

`builtin-name` "*name*"
: Indicates that this field corresponds to one of the GNATS built-in fields.

  GNATS has several fields which are required to be present in a PR, and this option is used to map their external descriptions to their internal usage. The external field names are:

  `number` The PR's unique numeric identifier

  `category` The category that the PR falls into

  `synopsis` The one-line description of the PR

  `confidential`
  If set to `yes`, the PR is confidential

  `severity` Severity of the problem described by the PR

  `priority` Priority of the PR

  `responsible`
  The person responsible for handling the PR

  `state` The current state of the PR

  `submitter`
  The user that submitted the PR

  `arrival-date`
  The arrival date of the PR

  `last-modified`
  The date the PR was last modified

  `audit-trail`
  The audit-trail recording changes to the PR

  For these built-in fields, a matching field description must appear in the '`dbconfig`' file. Otherwise, the configuration will be considered invalid, and errors will be generated from the GNATS clients and gnatsd.

`description` "*description text*"
: A one-line human-readable description of the field. Clients can use this string to describe the field in a help dialog. The string is returned from the FDSC command in gnatsd and is also available via the `--field-description` option in `query-pr`.

  This entry must be present in the field description, and there is no default value.

`query-default` *exact-regexp* | *inexact-regexp*
: Used to specify the default type of searches performed on this field. This is used when the ^ search operator appears in a query, and is also used for queries in `query-pr` that use the old `--field` query options.

  If the option is not given, the default search is `exact-regexp`.

`textsearch`
: If this option is present, the field will be searched when the user performs a `--text` search from `query-pr`. The field is also flagged as a `textsearch` field in the set of field flags returned by the `FIELDFLAGS` command in gnatsd.

  By default, fields are not marked as `textsearch` fields.

`read-only`
: When this option is present, the field contents may not be edited — they must be set when the PR is initially created. In general, this should only be used for fields that are given as internal values rather than fields supplied by the user.

  By default, editing is allowed.

### 4.3.3 Field datatypes

Each field description has to contain a datatype declaration which describes what data are to be store in the field. The general format for such a declaration is

```
datatype  [ options ... ]
```

The available datatypes are:

`text  [ matching { "regexp" [ "regexp" ... ] } ]`
: The `text` datatype is the most commonly used type; it is a one-line text string.

  If the `matching` qualifier is present, the data in the field must match at least one of the specified regexps. This provides an easy and flexible way to limit what text is allowed in a field. If no `matching` qualifier is present, no restriction is placed on what values may appear in the field.

`multitext  [ { default "string" } ]`
: The field can contain multiple lines of text.

  If the `default` option is present, the field will default to the specified `string` if the field is not given a value when the PR is initially created. Otherwise, the field will be left empty.

```
enum  {
  values {
    "string" [ "string" ... ]
  }
  [ default "string" ]
}
```
: Defines an enumerated field, where the values in the PR field must match an entry from a list of specified values. The list of allowed values is given with the `values` option. The `values` option is required for an enumerated field.

If a `default` option is present, it is used to determine the initial value of the field if no entry for the field appears in an initial OR (or if the value in the initial PR is not one of the acceptable values). However, the value in the `default` statement is not required to be one of the accepted values; this can be used to allow the field to be initially empty, for example.

If no `default` option is specified, the default value for the field is the first value in the `values` section.

```
multienum {
  values {
    "string" [ "string" ... ]
  }
  [ separators "string" ]
  [ default "string" ]
}
```

The `multienum` datatype is similar to the `enum` datatype, except that the field can contain multiple values, separated by one or more characters from the `separators` list. If no `separators` option is present, the default separators are space (' ') and colon (':').

The values in the `default` string for this field type should be separated by one of the defined separators. An example clarifies this. If we have a field named `ingredients` where the default values should be '`sugar`', '`flour`' and '`baking powder`' and the separator is a colon ':', the following sets these defaults:

```
default "sugar:flour:baking powder"
```

```
enumerated-in-file {
  path "filename"
  fields {
    "name" [ "name" ... ]
  } key "name"
  [ allow-any-value ]
}
```

The `enumerated-in-file` type is used to describe an enumerated field with an associated *administrative file* which lists the legal values for the field, and may optionally contain additional fields that can be examined by query clients or used for other internal purposes. It is similar to the `enum` datatype, except that the list of legal values is stored in a separate file.

`filename` is the name of a file in the '`gnats-adm`' administrative directory for the database.

The format of the administrative file should be simple ASCII. *Subfields* within the file are separated with colons (':'). Lines beginning with a hash sign ('#') are ignored as comments. Records within the file are separated with newlines.

The `field` option is used to name the subfields in the administrative file. There must be at least one subfield, which is used to list the legal values for the field. If the administrative file is empty (or does not contain any non-empty non-comment lines), the PR field must be empty.

The `key` option is used to designate which field in the administrative file should be used to list the legal values for the PR field. The value must match one of the field names in the `field` option.

If the `allow-any-value` option is present, the value of the PR field is not required to appear in the administrative file — any value will be accepted.

Note that there is no `default` keyword for `enumerated-in-file`. These fields get their default value from the *first* entry in the associated administrative file.

```
multi-enumerated-in-file {
  path "filename"
  fields {
    "name" [ "name" ... ]
  } key "name"
  [ default "string" ]
  [ allow-any-value ]
  [ separators "string" ]
}
```

`multi-enumerated-in-file` is to `multienum` what `enumerated-in-file` is to `enum`. Its options have the same meaning as their counterparts in the `multienum` and `enumerated-in-file` fields.

*NOTE*: Keywords may appear in any sequence, with one exception – the `separators` keyword, if present, has to come last. This rule only applies to fields of type `multi-enumerated-in-file`.

`date` The `date` datatype is used to hold dates. Date fields must either be empty or contain a correctly formatted date.

No defaults or other options are available. The field is left empty if no value for the field is given in the initial PR.

`integer [ { default "integer" } ]`

Integer fields are used to hold numbers. They must either be empty or contain a value composed entirely of digits, with an optional leading sign.

If the `default` option is present, the field will have the value of `integer` if the field is not given a value when the PR is initially created. Otherwise, the field will be left empty.

### 4.3.4 Edit controls

The `on-change` subsection of a `fields` section specifies one or more actions to be performed when the field value is edited by the user. It has the general form

```
on-change [ "query-expression" ] {


    [ add-audit-trail ]


    [ audit-trail-format {
      format "formatstring"
      [ fields { "fieldname" ... } ]
    } ]
```

```
    [ require-change-reason ]


    [ set-field | append-to-field "fieldname" {
      "format-string" [ fieldlist ]
    } ]
  }
```

The optional `query-expression` controls whether or not the actions in the `on-change` section are taken. If the expression fails to match, the actions are skipped.

The `add-audit-trail` option indicates that an entry should be appended to the PR's audit-trail when this field is changed. The format of the entry is controlled by the optional `audit-trail-format` section within the field, or by the global `audit-trail-format` section. See Section 4.3.6 [Audit-trail formats], page 53 and Section 4.3.7 [Outgoing email formats], page 53.

The `require-change-reason` option specifies that a change reason must be present in the PR when this field is edited. This option only makes sense if an audit-trail entry is required, as the change reason is otherwise unused.

The `set-field` and `append-to-field` options are used to change the value of the field `fieldname` in the PR. The supplied format is used to format the value that will be placed in the field.

`append-to-field` appends the resulting formatted string to the existing, while `set-field` completely replaces the contents.

Any field may be edited by the `set-field` or `append-to-field` option (the `read-only` option on a field is ignored). However, the changes are subject to the usual field content checks.

There is also a global `on-change` section that is executed once for each PR edit. A typical use for such a section is to set the last-modified date of the PR.

### 4.3.5 Named query definitions

When queries are performed via `query-pr`, they can refer to a query format described by a `query` section in the '`dbconfig`' file:

```
query "queryname" {
  format "formatstring"
  [fields { "fieldname" [ "fieldname" ... ] } ]
}
```

`formatstring` is as described in Section 2.4.2 [Formatting query-pr output], page 24. It basically contains a string with printf-like % escapes. The output of the query is formatted as specified by this format string.

The `fields` option lists the fields to be used with the format string. If the `fields` option is present without a `format` option, the contents of the listed fields are printed out, separated by newlines.

The named query formats *full*, *standard* amd *summary* must be present in the '`dbconfig`' file. *full* and *summary* correspond to the `query-pr` options `--full` and `--summary`, while *standard* is used when no format option is given to `query-pr`.

### 4.3.6 Audit-trail formats

These formats are similar to the named query formats, but they include more options. They are used for formatting audit-trail entries and for outgoing email messages.

There is currently only one audit-trail format, defined by the `audit-trail-format` option:

```
audit-trail-format {
  format "formatstring"
  [ fields { "fieldname" [ "fieldname" ... ] } ]
}
```

For those fields that require an audit-trail entry, the audit-trail text to be appended is formatted as described by this format. The per-field `audit-trail-format` is used in preference to this one, if it exists.

`formatstring` and `fieldname` are similar to those used by the named query format. `fieldname` may also be a *format parameter*, which is a context-specific value. (Format parameters are distinguished from fieldnames by a leading dollar sign ('`$`')).

The following format parameters are defined for `audit-trail-format` entries:

`$Fieldname`
: The name of the field for which an audit-trail entry is being created.

`$OldValue`
: The old value of the field.

`$NewValue`
: The new field value.

`$EditUserEmailAddr`
: The email address of the user editing the field. Set by the `EDITADDR` `gnatsd` command or from the '`responsible`' file; if not available, user's local address is used.

`$CurrentDate`
: The current date.

`$ChangeReason`
: The reason for the change; may be blank if no reason was supplied.

These parameters may be used anywhere a `fieldname` can appear.

### 4.3.7 Outgoing email formats

During the life of a PR, GNATS can be configured to send out a range of email messages. When a PR first arrives, an acknowledgment message is sent out if the `send-submitter-ack` parameter above is set to `true`. Certain edits to the PR may also cause email to be sent out to the various parties, and if a PR is deleted, GNATS may send notification email.

The formats of the email messages are controlled by `mail-format` sections in the '`dbconfig`' file. The general structure of a `mail-format` section is as follows:

```
mail-format "format-name" {
  from-address {
    [ fixed-address "address" ]
    [ email-header-name | [ mail-header-name | ... ] ]
  }
  to-address {
    [ fixed-address "address" ]
    [ "email-header-name" | [ "mail-header-name" | ... ] ]
  }
  reply-to {
    [ fixed-address "address" ]
    [ "email-header-name" | ... ] | [ "gnats-field-name" | ... ]
  }
  header {
    format "formatstring"
    [ fields { "fieldname" [ "fieldname" ... ] } ]
  }
  body {
    format "formatstring"
    [ fields { "fieldname" [ "fieldname" ... ] } ]
  }
}
```

`gnats` recognizes and uses 6 different `format-name` values:

`initial-response-to-submitter`
: Format of the message used when mailing an initial response back to the PR submitter. This message will only be sent if `send-submitter-ack` in the overall database configuration is set to `true`.

`initial-pr-notification`
: Format of the message sent to the responsible parties when a new PR with category different from "pending" arrives.

`initial-pr-notification-pending`
: Format of the message sent to the responsible parties when a new PR that ends up with category "pending" arrives.

`appended-email-response`
: Format of the notification message sent out when a response to a PR is received via email.

`audit-mail`
: Format of the message sent out when a PR edit generates an Audit-Trail entry.

`deleted-pr-mail`
: Format of the message sent out when a PR is deleted.

The `from-address`, `to-address` and `reply-to` subsections of a mail-format section specify the contents of the `To:`, `From:` and `Reply-To:` headers in outgoing email. There are two ways to specify the contents: by using a `fixed-address` specification, or by specifying `email-header-names` or `gnats-field-names`.

When an `email-header-name` or `gnats-field-name` value is given, GNATS will attempt to extract an email address from the specified location. If several values are given on the same line, separated by '`|`' characters, GNATS will try to extract an address from each location in turn until it finds a header or field which is nonempty. The following example should clarify this:

```
mail-format "initial-response-to-submitter" {
  from-address {
    fixed-address "gnats-admin"
  }
  to-addresses {
    "Reply-To:" | "From:" | "Submitter-Id"
  } ...
```

This partial `mail-format` section specifies the format of the address headers in the email message that is sent out as an acknowledgment of a received PR. The `From:` field of the message will contain the email address of the GNATS administrator, as specified by the `gnats-admin` line in the '`responsible`' file. To fill in the `To:` header, GNATS will first look for the mail header `Reply-To:` in the PR and use the contents of that, if any. If that header doesn't exist or is empty, it will look for the contents of the `From:` email header, and if that yields nothing, it will look for the GNATS `Submitter-Id` field and use the contents of that.

Other email headers to be included in messages sent out by GNATS can be specified by `header` subsections of the `mail-header` section. `formatstring` and `fieldname` are similar to those used by the named query format. Each header line must have a newline character ('`\n`') at the end.

The email message body is specified in the `body` subsection of the `mail-format` section. Just as for a `header` section, the `body` section must contain a `formatstring` and `fieldname` values.

For some of the formats that GNATS recognizes, special variables are available for use. The following table lists the formats that provide special variables. See the example below for an illustration of how they are used.

`appended-email-response`

- `$MailFrom`
  The From: line of the original message.
- `$MailTo`
  The To: line of the original message.
- `$MailSubject`
  The Subject: line of the original message.
- `$MailCC`
  The CC: line of the original message.
- `$NewAuditTrail`
  The text of the new audit trail entry (corresponds to the body of the message).

`audit-mail`

`$EditUserEmailAddr`
: The email address of the user editing the PR. Set by the `EDITADDR` `gnatsd` command or from the '`responsible`' file; if not available, user's local address is used.

`$OldResponsible`
: The previous Responsible field entry, if it was changed.

`$NewAuditTrail`
: The Audit-Trail: entries that have been appended by the edits.

`deleted-pr-mail`

`$EditUserEmailAddr`
: The email address of the user deleting the PR. Set by the `EDITADDR` `gnatsd` command or from the '`responsible`' file; if not available, user's local address is used.

`$PRNum`
: The number of the PR that was deleted.

The following example illustrates the use of these special variables:

```
mail-format "deleted-pr-mail" {
  from-address {
    "$EditUserEmailAddr"
  }
  to-addresses {
    fixed-address "gnats-admin"
  }
  header {
    format "Subject: Deleted PR %s\n"
    fields { "$PRNum" }
  }
  body {
    format "PR %s was deleted by user %s.\n"
    fields { "$PRNum" "$EditUserEmailAddr" }
  }
}
```

This `mail-format` section specifies the format of the email message that is sent out when a PR is deleted. The `From:` field is set to the email address field of the user who deleted the PR, the subject of the message contains the number of the deleted PR, and the message body contains both the PR number and the user's email address.

### 4.3.8 Index file description

The `index` section of the '`dbconfig`' file lists the fields that appear in the database index. The index is always keyed by PR number. The general format for the `index` section is

```
index {
  path "file"
  fields { "fieldname" [ "fieldname" ... ] }
  binary-index true | false
  [ separator "symbol" ]
```

```
}
```

The path parameter gives the name of the index file in the 'gnats-adm' directory of the database. Only one index is allowed per database, so only one path entry is allowed.

The fields parameter controls what fields will appear, and in what order, in the index. Fields are listed by their names, separated by spaces (' '). Fields will appear in the order they are listed.

The binary-index parameter controls whether the index is supposed to be in plaintext or binary format. Binary format is recommended, as it avoids potential problems when field separators appear as bona-fide field contents.

When plaintext format is used, by setting binary-index false, the symbol ('|') is used as a field separator in the index, unless the optional separator parameter is used to redefine the separator character.

### 4.3.9 Initial PR input fields

An initial-entry section in the 'dbconfig' file is used to describe which fields should be present on initial PR entry; this is used by tools such as send-pr to determine which fields to include in a "blank" PR template. The general format for the initial-entry section is

```
initial-entry {
  fields { "fieldname" [ "fieldname" ... ] }
}
```

## 4.4 Other database-specific config files

### 4.4.1 The categories file

The 'categories' file contains a list of problem categories, specific to the database, which GNATS tracks. This file also matches responsible people with these categories. You must edit this file initially, creating valid categories. In most installations, GNATS is configured to create directories on disk for valid categories automatically as needed (see Section 4.3.1 [Overall database configuration], page 46). If GNATS isn't set up to do this, you need to run mkcat to create the corresponding subdirectories of the database directory. For instructions on running mkcat, see Section 4.6.2 [Adding a problem category], page 62.

To create a new category, log in as GNATS, add a line to this file, and run mkcat if applicable. Lines beginning with '#' are ignored.

A line in the 'categories' file consists of four fields delimited by colons, as follows:

*category*:*description*:*responsible*:*notify*

*category* A unique category name, made up of text characters. This name cannot contain spaces or any of the following characters:

```
! $ & * ( ) { } [ ] ` ' " ; : < > ~
```

Ideally, category names should not contain commas or begin with periods. Each line has a corresponding subdirectory in the database directory.

*description*
A terse textual description of the category.

*responsible*
The name tag of the party responsible for this category of problems, as listed in the '`responsible`' file (see Section 4.4.2 [The `responsible` file], page 58).

*notify* One or more other parties which should be notified when a Problem Report with this category arrives, such as a project manager, other members of the same project, other interested parties, or even log files. These should be separated with commas.

A good strategy for configuring this file is to have a different category for each product your organization supports or wishes to track information for.

```
rock:ROCK program:me:myboss,fred
stone:STONE utils:barney:fred
iron:IRON firewall:me:firewall-log
```

In the above example, the nametags '`myboss`', '`me`', '`fred`', and '`barney`' must be defined in the '`responsible`' file (see Section 4.4.2 [The `responsible` file], page 58).

Problem Reports with a category of '`rock`' are sent to the local mail address (or alias) '`me`', and also sent to the addresses '`myboss`' and '`fred`'. PRs with a category of '`stone`' are sent to the local addresses '`barney`' and '`fred`' only, while PRs with the category '`iron`' are sent only to '`me`', and are also filed in `firewall-log` (in this case, a mail alias should be set up, see Section 3.5 [Setting up mail aliases], page 36.

If you want to separate PRs in each problem category into specific subsets, say *documentation* and *software bugs*, using the '`classes`' file is recommended. See Section 4.4.6 [The '`classes`' file], page 61.

Only one category *must* be present for GNATS to function:

```
pending:Non-categorized PRs:gnats-admin:
```

The '`pending`' directory is created automatically when you run `mkdb` to initialize a new database. (see Section 3.1 [Configuring and compiling the software], page 32).

### 4.4.2 The `responsible` file

This file contains a list of the responsible parties. Lines beginning with '`#`' are ignored. Each entry contains three fields, separated by colons:

*responsible*:*full-name*:*mail-address*

*responsible*
A name-tag description of the party in question, such as her or his user name, or the name of the group. This name is listed in the PR in the '`>Responsible:`' field.

*full-name* The full name of the party ("Charlotte Bronte"; "Compiler Group").

*mail-address*
The full, valid mail address of the party. This field is only necessary if the responsible party has no local mail address or alias.

A sample '`responsible`' listing might be:

```
ren:Ren Hoek:
stimpy:Stimpson J. Cat:stimpy@lederhosen.org
```

Here, `ren` is a local user. `stimpy` is remote, so his full address must be specified.

The following entry *must* be present for GNATS to function:

```
gnats-admin:GNATS administrator:
```

`gnats-admin` is usually defined as a mail alias when GNATS is installed, so for this purpose `gnats-admin` is a local address. However, this line can alos be used to redefine the email address of the GNATS administrator, by adding the desired address at the end of the line.

### 4.4.3 The `submitters` file

This is a database of sites which submit bugs to your support site. It contains six fields delineated by colons. Lines beginning with '`#`' will be ignored.

Entries are of the format:

*submitter-id*:*name*:*type*:*resp-time*:*contact*:*notify*

*submitter-id*
: A unique identifier for a specific site or other entity who submits Problem Reports. The first `submitter-id` listed in the file will be the default for PRs that arrive with invalid or empty submitter fields.

*name*
: The full name or a description of this entity.

*type*
: Optional description for the type of relationship of this submitter to your support site. This could indicate a contract type, a level of expertise, etc., or it can remain blank.

*resp-time*
: Optional quoted response time in *business hours*. If the database '`dbconfig`' file has the `notify-about-expired-prs` entry set to *true* (see Section 4.3.1 [Overall database configuration], page 46, GNATS will use this field to schedule when GNATS should notify the gnats-admin, responsible person and submitter contact that the PR wasn't analyzed within the agreed response time. Business hours and business-week days are set in the '`dbconfig`' file. For information on `at-pr`, the program which sends out this reminder, see Section 4.7.3 [Timely Reminders], page 67.

*contact*
: The name tag of the main *contact* at the Support Site for this submitter. This contact should be in the '`responsible`' file (see Section 4.4.2 [The `responsible` file], page 58). Incoming bugs from *submitter* are sent to this contact. Optionally, this field can be left blank.

*notify*
: Any other parties who should receive copies of Problem Reports sent in by *submitter*. They need not be listed in the '`responsible`' file.

A few example entries in the '`submitters`' file:

```
univ-hell:University of Hades:eternal:3:beelzebub:lucifer
tta:Telephones and Telegraphs of America:support:720:dave:
```

In this example, when a PR comes in from the University of Hades (who has an eternal contract), it should have '`univ-hell`' in its '`Submitter-Id`' field. This Problem Report goes to `beelzebub` (who should be in the '`responsible`' file), and if it is not analyzed within three business hours a reminder message is sent. `lucifer` also receives a copy of the bug, and a copy of the reminder message as well (if it is sent). When Telephones and Telegraphs of America utilizes their support contract and submits a bug, a copy is sent only to `dave`, who has 720 business hours to return an analysis before a reminder is sent.

To disable the feature of GNATS which tracks the '`>Submitter-Id:`', simply alter the '`submitters`' file to only contain one *submitter-id* value, and instruct your submitters to ignore the field.

### 4.4.4 The `states` file

This file lists the possible states for Problem Reports. Each line consists of a state, followed optionally by colon and a one-line description of what the state means. Lines beginning with '`#`' will be ignored.

*state*`:`*optional descriptive text*

State names can contain any alphanumeric character, '`-`' (hyphen), '`_`' (underscore), or '`.`' (period), but no other characters. The state name ends with a newline, or else with a colon followed by optional descriptive text. The descriptive text, if present, can contain any character except newline, which marks the end of the description. Empty or all-whitespace descriptions are allowed. This is not recommended, however, since although GNATS doesn't use this field, external tools might.

The first state listed will be the state automatically assigned to Problem Reports when they arrive; by default this is named "open". The last state listed is the end state for Problem Reports — one should usually assume that a PR in this state is not being actively worked on; by default this state is named "closed".

It is probably best to leave "open" as the first state and "closed" as the last, otherwise some external tools looking for those two states by name may be fooled.

### 4.4.5 The `addresses` file

This file contains mappings between submitter IDs and corresponding e-mail addresses.

When a PR comes in without a submitter ID (if someone sends unformatted e-mail to the PR submission email address), GNATS will try to derive the submitter ID from the address in the "From:" header. The entries in this file consist of two fields, separated by a colon:

*submitter-id*`:`*address-fragment*

*submitter-id*
: A valid submitter ID

*address-fragment*
: Part of all of the e-mail address to be matched

Here is an example of an `addresses` file:

```
# Addresses for Yoyodine Inc
yoyodine:yoyodine.com
yoyodine:yoyodine.co.uk
```

```
# Addresses for Foobar Inc.
foobar1:sales.foobar.com
foobar2:admin.foobar.com
foobar3:clark@research.foobar.com
```

GNATS checks each line in the `addresses` file, comparing *address-fragment* to the end of the "From:" header, until it finds a match. If no match is found, GNATS uses the default submitter ID.

You can only have one address fragment per line, but you can have more than one line for a given submitter ID. An address fragment can be a domain (i.e. yoyodine.com), a machine location (admin.foobar.com), or a full e-mail address (clark@research.foobar.com).

GNATS can match addresses in three e-mail formats:

'`From: name@address.com`'
: The address by itself without a full name, not enclosed in brackets

'`From: Real Person <name@address.com>`'
: A full name (optional, with or without quotation marks), followed by the address enclosed in angle brackets

'`From: name@address.com (Real Person)`'
: An address, followed by a name or comment in parentheses

If GNATS sees other e-mail address formats, it uses the default submitter ID.

### 4.4.6 The `classes` file

This file lists the possible classes of Problem Reports. Each line consists of a class name, followed by a colon and an optional class type name (the class type name is not used for anything in the current implementation of GNATS, so it may be left blank. The `class` and `class-type-name` fields may only contain alphanumerics, '`-`', '`_`', and '`.`', but no other characters.

Then comes another colon, followed by an optional one-line description of the class. GNATS itself does not use the class description, but external tools such as Gnatsweb and TkGnats may use it. Therefore, a line in this file should at least contain the following:

```
class::class description
```

Lines beginning with '`#`' will be ignored, and the first listed class is the default class for an incoming Problem Report.

## 4.5 Administrative data files

The following files are database-specific and are located in the '`gnats-adm`' subdirectory of the database directory. These files are maintained by GNATS; you should never need to touch them.

### 4.5.1 The `index` file

The index is used to accelerate searches on the database by `query-pr` and `edit-pr`. This file is not created until the first PR comes in. It is then kept up to date by GNATS; you should never touch this file.

Searches on subjects contained in the index are much faster than searches which depend on data not in the index. Inexes come in two different formats: *binary* and *plain-text*. Binary indexes are safer, in that they avoid certain problems that may crop up if the field separators used by plain-text indexes appear in field data.

A plain-text index contains single-line entries for all PR fields except for the multitext fields such as Description, How-To-Repeat, etc. Fields are separated by bars ('`|`') except for '`>Category:`' and '`>Number:`', which are separated by a slash ('`/`').

Binary indexes are not meant to be human-readable, but they are safer than the plain-text variety, in that they avoid certain problems that may crop up if the field separators used by plain-text indexes appear in field data.

The format of the index for a database is set in the '`dbconfig`' file. See Section 4.3.8 [Index file description], page 56.

Should the '`index`' file become corrupted, the `gen-index` utility can be used to regenerate it. See Section 4.6.4 [Regenerating the index], page 63.

### 4.5.2 The `current` file

This file contains the last serial number assigned to an incoming PR. It is used internally by GNATS; you need never touch this file.

## 4.6 Administrative utilities

These tools are used by the GNATS administrator as part of the periodic maintenance and configuration of GNATS. See Chapter 4 [GNATS Administration], page 43.

### 4.6.1 Adding another database

To initialize a new GNATS database:

1. Add a line for the new database in the '`databases`' file (see Appendix A [Where GNATS lives], page 73.
2. Run `mkdb`, using

   `mkdb` *database*

   where *database* is the database you specified in the '`databases`' file. `mkdb` creates the database directory and populates it with the directories '`pending`', '`gnats-queue`' and '`gnats-adm`'. A full set of sample configuration files is copied to the '`gnats-adm`' directory.

### 4.6.2 Adding a problem category

To add new categories to the database:

1. Add a line to for each new category to the '`categories`' file in the gnats-adm directory of the database. See Section 4.4.1 [The `categories` file], page 57.
2. Run `mkcat` If applicable. If `create-category-dirs` is set to `false` in the database '`dbconfig`' file, you need to create category directories with `mkcat`. `mkcat` creates a subdirectory under the database directory for any new categories which appear in the '`categories`' file.

### 4.6.3 Removing a problem category

To remove a category from the database:

1. Remove the Problem Reports from the subdirectories corresponding to the categories you wish to remove, or assign the PRs to new categories. All PRs for a given category reside in a subdirectory of the database directory, with the same name as the category.
2. Run `rmcat` using

   `rmcat` *category* `[` *category*... `]`

   where *category* is the category you wish to remove. You can specify as many categories as you wish as long as each category has no PRs associated with it. `rmcat` removes the directory where the Problem Reports for that category had been stored.

### 4.6.4 Regenerating the index

If your '`index`' file becomes corrupted, or if you need a copy of the current index for some reason, use

```
gen-index [ -n | --numeric ]
          [ -d databasename | --database=databasename ]
  [ -o filename | --outfile=filename ]
          [ -i | --import ] [ -e | --export ]
          [ -h | --help] [ -V | --version ]
```

With no options, `gen-index` generates an index that is sorted by the order that the categories appear in the '`categories`' file. The index is generated in plaintext or binary format according to the value of `binary-index` in the '`dbconfig`' file (see Section 4.3.8 [Index file description], page 56). The results are printed to standard output. The options are:

`-n`
`--numeric`
: Sorts index entries numerically.

`-d` *databasename*
`--database=`*databasename*
: Specifies the database to index. If this option is left out, `gen-index` attempts to index the database with name taken from the the *GNATSDB* environment variable, and if that is undefined, the default database, as set when GNATS was built (usually `default`).

`-o` *filename*
`--outfile=`*filename*
: Places output in *filename* rather than sending it to standard output.

`-i`
`--import`
: Import the existing index file instead of re-indexing the database.

`-e`
`--export`
: Force plaintext output.

`-h`
`--help`
: Prints the usage for `gen-index`.

`-V`
`--version`
Prints the version number for `gen-index`.

### 4.6.5 Checking database health

The '`check-db`' script is useful for performing periodic checks on database health. It accepts the following options:

`-d` *databasename*
`--database=`*databasename*
Determines the database which to operate on.

`--all-databases`
Check all GNATS databases on the system. This option takes precedence over the `--database` option.

If no option is given, the default database is checked.

During its operation, `check-db` first attempts to lock *database*. If this is not possible, it repeats the locking attempts for five minutes; if it fails, it sends a mail message notifying the administrator of the failure and exits.

Once the database is locked, the script searches the database for lock files that are more than 24 hours old. Any old lock files are reported to the administrator in a mail message.

After checking for old lock files, it calls `gen-index` (see Section 4.6.4 [Regenerating the index], page 63) and compares the results with the current '`index`' file of the database; any inconsistencies are reported to the administrators in a mail message.

After checking the index file for inconsistencies, the script unlocks the database and exits.

### 4.6.6 Managing user passwords

Older versions of GNATS, up to and including version 3.x, stored user passwords in plaintext in the '`gnatsd.access`' files. Version 4 has the options of storing the password as MD5 or standard DES `crypt()` hashes (as most UNIX versions do in the system password files) as well as in plaintext. Since the password strings require a prefix to indicate how they are encrypted, one is forced to convert the old password files to a new format when upgrading to GNATS version 4. See Section 3.8 [Upgrading from older versions], page 38.

The `gnats-pwconv` tool takes care of converting the old password files to the new format:

```
gnats-pwconv [ -c | --crypt ] [ -m | --md5 ] [ -p | --plaintext ]
          [ -h | --help] [ -V | --version ] INFILE [OUTFILE]
```

Unless the `--version` or `--help` options are given, exactly one encryption method must be specified, as well as an input file. The output file parameter is optional. If one is not specified, results will be printed on standard output.

## 4.7 Internal utilities

These tools are used internally by GNATS. You should never need to run these by hand; however, a complete understanding may help you locate problems with the GNATS tools or with your local implementation.

### 4.7.1 Handling incoming traffic

The program `queue-pr` handles traffic coming into GNATS. `queue-pr` queues incoming Problem Reports in the '`gnats-queue`' directory of the database, and then periodically (via `cron`) passes them on to `file-pr` to be filed in the GNATS database. See Chapter 3 [Installing GNATS], page 31.

The usage for `queue-pr` is as follows:

```
queue-pr [ -q | --queue ] [ -r | --run ]
         [ -f filename | --file=filename ]
         [ -d databasename | --database=databasename ]
         [ -h | --help] [ -V | --version ]
```

One of '`-q`' or '`-r`' (or their longer-named counterparts) must be present upon each call to `queue-pr`. These options provide different functions, as described below.

`-q`
`--queue`
: Accepts standard input as an incoming mail message, placing this message in an incrementally numbered file in the '`gnats-queue`' directory under *DATABASEDIR* (see Appendix A [Where GNATS lives], page 73).

`-r`
`--run`
: Redirects files in the '`gnats-queue`' directory into the program `file-pr` one by one.

`-f` *filename*
`--file=`*filename*
: Used with '`-q`' (or '`--queue`'), accepts the file denoted by *filename* as input rather than reading from standard input.

`-d` *databasename*
`--directory=`*databasename*
: Specifies database to operate on. If this option is left out, the value of the *GNATSDB* environment variable is used, and if that is undefined, the default database name set when GNATS was built is used (usually `default`).

`-h`
`--help`
: Prints the usage for `gen-index`.

`-V`
`--version`
: Prints the version number for `gen-index`.

### 4.7.2 Processing incoming traffic

`queue-pr` hands off queued Problem Reports to `file-pr` one at a time. `file-pr` checks each Problem Report for correct information in its fields (particularly a correct

'>Category:'), assigns it an identification number, and files it in the database under the appropriate category.

If the '>Category:' field does not contain a valid category value (i.e., matching a line in the categories file; see Section 4.4.1 [The categories file], page 57), the PR is assigned to the default category, as set in the dbconfig file. If there is no default category defined, the PR is given a '>Category:' value of 'pending' and is placed in the 'pending' directory. The GNATS administrator is notified of the unplaceable PR.

file-pr assigns the Problem Report an identification number, files it in the GNATS database (under the default, if the '>Category:' field contains an invalid category), and sends acknowledgments to appropriate parties. For the default GNATS configuration, the person responsible for that category of problem (see Section 4.4.1 [The categories file], page 57) and the person responsible for the submitter site where the PR originated (see Section 4.4.3 [The submitters file], page 59) receive a copy of the PR in its entirety. Optionally, the originator of the PR receives an acknowledgment that the PR arrived and was filed (see Section 4.1 [Changing your GNATS configuration], page 44).

The usage for file-pr is as follows:

```
file-pr [ -f filename | --file=filename ]
        [ -d databasename | --database=databasename ]
     [ -h | --help ] [ -V | --version ]

          network options:
           [ -H host | --host=host ]
           [ -P port | --port=port ]
           [ -v username | --user=username ]
           [ -w password | --passwd=password ]
```

file-pr requires no options in order to operate, and takes input from standard input (normally, the output of 'queue-pr -r') unless otherwise specified. The options include:

-f *filename*
--filename=*filename*
: Uses *filename* as input rather than standard input.

-d *databasename*
--database=*databasename*
: Performs refiling operations on *database*. If this option is left out, the value of the *GNATSDB* environment variable is used, and if that is undefined, the default database name set when GNATS was built is used (usually default).

-h
--help
: Prints the usage for file-pr.

-V
--version
: Prints the version number for file-pr.

file-pr can file PRs across a network, talking to a remote gnatsd. The following options relate to network access:

`-H` *host*
`--host=`*host*
    Hostname of the GNATS server.

`-P` *port*
`--port=`*port*
    The port that the GNATS server runs on.

`-v` *username*
`--username=`*username*
    Username used to log into the GNATS server.

`-w` *password*
`--passwd=`*password*
    Password used to log into the GNATS server.

### 4.7.3 Timely reminders

`at-pr` creates a queued job using `at` which, after an allotted *response time* is past, checks the PR to see if its state has changed from '`open`'. When the PR is originally filed, `file-pr` checks the `notify-about-expired-prs` parameter in the '`dbconfig`' file. If this parameter is set to `true`, `file-pr` calls `at-pr`, which sets up the expiry check.

The '`submitters`' file contains the response time for each `>Submitter-Id:` (see Section 4.4.3 [The `submitters` file], page 59). The time is determined in *business hours*, which are defined in the database's '`dbconfig`' file (see Section 4.3.1 [Overall database configuration], page 46).

If the PR is urgent and is still open after the requisite time period has passed, `at-pr` sends a reminder to the GNATS administrator, to the maintainer responsible for that submitter, and to the maintainer responsible for the PR with the following message:

```
To: submitter-contact responsible gnats-admin
Subject: PR gnats-id not analyzed in #hours hours

PR gnats-id was not analyzed within the acknowledgment period
of #hours business hours.  The pertinent information is:

 Submitter-Id: submitter
 Originator: full name of the submitter
 Synopsis: synopsis
 Person responsible for the PR: responsible

--
The GNU Problem Report Management System (GNATS)
```

The PR is *urgent* if its priority is '`critical`' or if its priority is '`serious`' and the severity is '`high`'.

### 4.7.4 The `edit-pr` driver

`pr-edit` does the background work for `edit-pr`, including error-checking and refiling newly edited Problem Reports, handling file and database locks and deletion of PRs. It can be called interactively, though it has no usable editing interface.

The usage for `pr-edit` is:

```
pr-edit   [ -l username | --lock=username ] [ -u | --unlockdb ]
          [ -L | --lockdb ] [ -U | --unlockdb ] [ -c | --check ]
          [ -C | --check-initial ] [ -s | --submit ]
          [ -a field | --append field=field ]
          [ -r field | --replace=field ] [ --delete-pr ]
          [ -R reason | --reason=reason ]
          [ -p process-id | --process=process-id ]
          [ -d databasename | --database=databasename ]
          [ -f filename | --filename=filename ]
          [ -V | --version ]
          [ -h | --help ] [ -v username | --user=username ]
          [ -w passwd | --passwd=passwd ]
          [ -H host | --host=host ]
          [ -P port | --port=port ]
          [ -D | --debug ] [ PR number ]
```

A *lock* is placed on a Problem Report while the PR is being edited. The lock is simply a file in the '`locks`' subdirectory of the '`gnats-adm`' directory of the database, with the name '*gnats-id*`.lock`', which contains the name of the user who created the lock. *user* then "owns" the lock, and must remove it before the PR can be locked again, even by the same *user*[1]. If a PR is already locked when you attempt to edit it, `pr-edit` prints an error message giving the name of the user who is currently editing the PR.

If you do not specify *PR number*, `pr-edit` reads from standard input. You must specify *PR number* for the functions which affect PR locks, '`--lock=`*username*' and '`--unlock`'.

`-L`
`--lockdb`
: Locks the database specified with the `--database` or `-d` option. No PRs may be edited, created or deleted while the database is locked. This option is generally used when editing the index file.

`-U`
`--unlockdb`
: Unlocks the specified database. No check is made that the invoking user actually had locked the database in the first place; hence, it is possible for anyone to steal a database lock.

`-c`
`--check`
`-C`
`--check-initial`
: The `--check` options are used to verify that a proposed PR's field contents are valid. The PR is read in (either from stdin or a file specified with `--filename`), and its fields are compared against the rules specified by the database configuration of the selected database. Warnings are given for enumerated fields whose contents do not contain one of the required values or fields that do not match required regexps. `--check-initial` is used to verify initial PRs, rather than proposed edits of existing PRs.

[1] This approach may seem heavy-handed, but it ensures that changes are not overwritten.

`-s`
`--submit` Used to submit a new PR to the database. The PR is read in and verified for content; if the PR is valid as an initial PR, it is then added to the database. A zero exit code is returned if the submission was successful. Otherwise, the reason(s) for the PR being rejected are printed to stdout, and a non-zero exit code is returned.

The following options require a PR number to be given.

`--delete-pr`
Deletes the specified PR from the database. The PR must be in a closed state, and not locked. Only the user *gnats* (or the user name specified instead of *gnats* during the building of GNATS) is permitted to delete PRs.

`-l` *username*
`--lock=`*username*
Locks the PR. *username* is associated with the lock, so the system administrator can determine who actually placed the lock on the PR. However, anyone is permitted to remove locks on a PR. If the optional `--process` or `-p` option is also given, that process-id is associated with the lock.

`-u`
`--unlock` Unlocks the specified PR.

`-a` *field*
`--append=`*field*
`-r` *field*
`--replace=`*field*
`--append` and `--replace` are used to append or replace content of a specific field within a PR. The new field content is read in from stdin (or from the file specified with the `--filename` option), and either appended or replaced to the specified field. The field contents are verified for correctness before the PR is rewritten. If the edit is successful, a zero exit status is returned. If the edit failed, a non-zero exit status is returned, and the reasons for the failure are printed to stdout.

`-R` *reason*
`--reason=`*reason*
Certain PR fields are configured in the database configuration to require a short text describing the reason of every change that happens to them, See Section 4.3 [dbconfig file], page 46. If you edit a problem and change any of such fields, you must issue a short text, the *reason* of the change, through this option. If the option is used and no change-reason requiring field is actually changed, the option has no effect.

`PR number` If only a `PR number` is specified with no other options, a replacement PR is read in (either from stdin or the file specified with `--filename`). If the PR contents are valid and correct, the existing PR is replaced with the new PR contents. If the edit is successful, a zero exit status is re turned. If the edit failed, a non-zero exit status is returned, and the reasons for the failure are printed to stdout.

`-d` *database*
`--database=`*database*
Specifies the database which is to be manipulated. If no database is specified, the default database name set when GNATS was built is used (usually `default`). This option overrides the database specified in the GNATSDB environment variable.

`-f` *filename*
`--filename=`*filename*
For actions that require reading in a PR or field content, this specifies the name of a file to read. If `--filename` is not specified, the PR or field content is read in from stdin.

`-h`
`--help` Prints the usage for `pr-edit`.

`-V`
`--version`
Prints the version number for `pr-edit`.

`pr-edit` can edit PRs across a network, talking to a remote gnatsd. The following options relate to network access:

`-H` *host*
`--host=`*host*
Hostname of the GNATS server.

`-P` *port*
`--port=`*port*
The port that the GNATS server runs on.

`-v` *username*
`--username=`*username*
Username used to log into the GNATS server.

`-w` *password*
`--passwd=`*password*
Password used to log into the GNATS server.

`-D`
`--debug` Used to debug network connections.

### 4.7.5 The `diff-prs` tool

The `diff-prs` tool is invoked as follows:

`diff-prs` *prfile1 prfile2*

`diff-prs` simply reads the PRs contained in *prfile1* and *prfile2* and returns a list of the fields that are different between the two. No output is produced if the PRs are identical.

### 4.7.6 The `pr-age` tool

The `pr-age` tool reports the time, in days and hours, since the PR arrived. Usage is

```
pr-age  [ -d databasename | --database=databasename ]
        [ -H host | --host=host ]
        [ -P port | --port=port ]
        [ -v username | --user=username ]
        [ -w password | --passwd=password ]
        [ -h | --help ] [ -V | --version ]
```

For an explanation of the arguments listed above, please refer to the usage description for `file-pr` (Section 4.7.2 [`file-pr`], page 65).

# Appendix A Where GNATS lives

We use a few conventions when referring to the installation structure GNATS uses. These values are adjustable when you build and install GNATS (see Chapter 3 [Installing GNATS], page 31).

## A.1 *prefix*

*prefix* corresponds to the variable '`prefix`' for `configure`, which passes it on to the '`Makefile`' it creates. *prefix* sets the root installation directory for *host-independent* files as follows:

database-specific configuration and database contents
: '*prefix*`/com`'

site-wide configuration files
: '*prefix*`/etc/gnats`'

`man` pages
: '*prefix*`/man`'

`info` documents
: '*prefix*`/info`'

`include` files
: '*prefix*`/include`'

*etc. . .*

The default value for *prefix* is '`/usr/local`', which can be changed on the command line to `configure` using

```
configure --prefix=prefix ...
```

See Section 3.1 [Configuring and compiling the software], page 32.

## A.2 *exec-prefix*

*exec-prefix* corresponds to the variable '`exec-prefix`' for `configure`, which passes it on to the '`Makefile`' it creates. *exec-prefix* sets the root installation for *host-dependent* files as follows:

GNATS user tools
: '*exec-prefix*`/bin`'

administrative and support utilities
: '*exec-prefix*`/libexec/gnats`'

compiled libraries
: '*exec-prefix*`/lib`'

*etc. . .*

`configure` supports several more options which allow you to specify in great detail where different files are installed. The locations given in this appendix do not take into account highly customized installations, but fairly ordinary GNATS installations should be covered by the material here. For a complete list of options accepted by `configure`, run `./configure --help` in the '`gnats`' subdirectory of the distribution.

Since most installations are not intended to be distributed around a network, the default value for *exec-prefix* is the value of '`prefix`', i.e., '`/usr/local`'. However, using *exec-prefix* saves space when you are installing a package on several different platforms for which many files are identical; rather than duplicate them for each host, these files can be shared in a common repository, and you can use symbolic links on each host to find the host-dependent files.

Use *exec-prefix* in conjunction with *prefix* to share host-independent files, like libraries and `info` documents. For example:

```
  for each host:
configure --prefix=/usr/gnu --exec-prefix=/usr/gnu/H-host
make all install ...
```

Using this paradigm, all host-dependent binary files are installed into '`/usr/gnu/H-`*host*`/bin`', while files which do not depend on the host type for which they were configured are installed into '`/usr/gnu`'.

You can then use a different symbolic link for '`/usr/gnu`' on each host ('`/usr`' is usually specific to a particular machine; it is always specific to a particular architecture).

```
  on host-1:
ln -s /usr/gnu/H-host-1 /usr/gnu
  on host-2:
ln -s /usr/gnu/H-host-2 /usr/gnu
```

To the end user, then, placing '`/usr/gnu/bin`' in her or his `PATH` simply works transparently for each *host* type.

You can change *exec-prefix* on the command line to `configure` using

```
configure --exec-prefix=exec-prefix ...
```

We recommend that you consult section "Using `configure`" in *Cygnus configure*, before attempting this.

## A.3 The '`gnats-adm`' directory

Each GNATS database located on a server has its own directory, as listed in the '`databases`' (see Section 4.2 [The `databases` file], page 45 and given when the `mkdb` utility is invoked to initialize the database (see Section 4.6.1 [Initializing a new database], page 62). This directory has several subdirectories, one of which is named '`gnats-adm`'. This directory contains all configuration files related to this specific database, including the '`categories`', '`submitters`', '`responsible`', '`states`', '`classes`', '`dbconfig`', '`addresses`', '`states`' and '`gnatsd.access`', as well as two files generated and maintained by GNATS, '`index`' and '`current`'.

## A.4 Default installation locations

*prefix* defaults to '`/usr/local`'; change using `configure` (see Section 3.1 [Configuring and compiling the software], page 32).

*exec-prefix*

defaults to *prefix*; change using `configure` (see Section 3.1 [Configuring and compiling the software], page 32).

GNATS installs tools, utilities, and files into the following locations.

*exec-prefix*`/bin`

`send-pr` See Section 2.2 [Submitting Problem Reports], page 13.

`edit-pr` See Section 2.3 [Editing existing Problem Reports], page 18.

`query-pr` See Section 2.4 [Querying the database], page 20.

*exec-prefix*`/libexec/gnats`

`at-pr` See Section 4.7.3 [Timely reminders], page 67.

`check-db` See Section 4.6.5 [Checking database health], page 64.

`delete-pr`

Tool for deleting PRs. Deprecated. Use the –delete-pr option of `pr-edit` instead (see Section 4.7.4 [The edit-pr driver], page 67).

`diff-prs` See Section 4.7.5 [The `diff-prs` tool], page 70.

`file-pr` See Section 4.7.2 [Interface to pr-edit for filing new PRs], page 65.

`gen-index`

See Section 4.6.4 [Regenerating the index], page 63.

`gnatsd` The GNATS daemon.

`gnats-pwconv`

See Section 4.6.6 [Converting old password files], page 64.

`mail-query`

See Section 3.5 [Setting up mail aliases], page 36.

`mkcat` See Section 4.6.2 [Adding a problem category], page 62.

`mkdb` See Section 4.6.1 [Script for creating new databases], page 62.

`pr-age` See Section 4.7.6 [The `pr-age` tool], page 71.

`pr-edit` See Section 4.7.4 [The main PR processor], page 67.

`queue-pr` See Section 4.7.1 [Handling incoming traffic], page 65.

`rmcat` See Section 4.6.3 [Removing categories], page 63.

*exec-prefix*`/lib/libiberty.a`

The GNU `libiberty` library.

*prefix*`/etc/gnats`

'databases'
: The list of valid databases. For local databases it contains the pathname of the database directory, and for remote databases it has the hostname and port of the gnatsd server to connect to. (see Section 4.2 [The databases file], page 45).

'defaults'
: A directory containing the set of default files used when a new database is created with mkdb. (see Section 4.3 [The dbconfig file], page 46).

'gnatsd.host_access'
: The access-level settings for the different hosts that will be accessing the databases on your server. The settings in this file apply across all databases on the server. See Section C.3 [The 'gnatsd.host_access' file], page 93.

'gnatsd.access'
: The access-level settings for the users that will be accessing the databases on your server. The settings in this file apply across all databases on the server. See Section C.4 [The 'gnatsd.access' file], page 94.

*prefix*/share/emacs/site-lisp

'gnats.el'
'gnats.elc'
: The Emacs versions of the programs send-pr, query-pr, edit-pr, and view-pr. See Chapter 2 [The GNATS user tools], page 13. To change this directory you must change the lispdir variable in 'Makefile.in'; see Section 3.1 [Configuring and compiling the software], page 32.

*prefix*/info

'gnats.info'
'send-pr.info'
: The GNATS manuals, in a form readable by info (the GNU hypertext browser). See section "Reading GNU Online Documentation" in *GNU Online Documentation*.

*prefix*/man/man1
*prefix*/man/man8

man pages for all the GNATS tools and utilities.
: See Chapter 2 [The GNATS user tools], page 13.

Per-database directory

'gnats-adm'
: Administration and configuration data files that define behaviour of the particular database. The files

'categories'

'submitters'

'responsible'

'states'

'classes'

'dbconfig'

'addresses'

'states'

'gnatsd.access'

'index' (*This file is created by* GNATS.)

'current' (*This file is created by* GNATS.)

exist here. See Section 4.1 [GNATS configuration], page 44, Section 4.5 [Administrative data files], page 61 and Appendix C [Controlling access to databases], page 93.

'gnats-queue'
: Incoming Problem Reports are queued here until the next iteration of 'queue-pr -r' (see Section 4.7.1 [Handling incoming traffic], page 65).

'pending'
: If no default category is set, problem reports without a category are reassigned to the category 'pending' and placed here pending intervention by GNATS administrators. See Chapter 4 [GNATS administration], page 43.

'*category*'
: Each valid category has a corresponding subdirectory in the database. All Problem Reports associated with that category are kept in that subdirectory.

# Appendix B The GNATS network server – gnatsd

This section describes in details how the GNATS network daemon works. This information is mainly assumed to be useful for developers of GNATS client software.

## B.1 Description of gnatsd

The gnatsd network daemon is used to service remote GNATS requests such as querying PRs, PR creation, deletion, and editing, and miscellaneous database queries. It uses a simple ASCII-based command protocol (similar to SMTP or POP3) for communicating with remote clients.

It also provides a security model based either on IP-based authentication (generally considered very weak) or username/passwords, where passwords may be in cleartext, UNIX crypt or MD5 hash format. Access through gnatsd is granted according to certain predefined *access levels*. Access levels are further discussed in Appendix C [Controlling access to databases], page 93. It should be emphasized that security has not been a focus of development until now, but future versions are expected to address this more thoroughly.

All of the GNATS clients are capable of communicating via the GNATS remote protocol to perform their functions.

gnatsd is usually started from the inetd facility and should run as the 'gnats' user (the actual username of this user is configurable during installation, see Section 3.1 [Configuring and compiling the software], page 32 for details.)

## B.2 gnatsd options

The daemon supports the following command-line options:

```
gnatsd [--database database | -d database]
       [--not-inetd | -n] [--max-access-level level | -m level]
       [--version | -V] [--help | -h]
```

`-V, --version`
: Prints the program version to stdout and exits.

`-h, --help`
: Prints a short help text to stdout and exits.

`-d, --database`
: Specifies the default database which is to be serviced by this invocation of gnatsd. (The selected database may be changed via the CHDB command; the name set with this option is simply the default if no CHDB command is issued.) If no database is specified, the database named default is assumed. This option overrides the database specified in the GNATSDB environment variable.

`-n, --not-inetd`
: As its name suggests, indicates that gnatsd is not being invoked from inetd. This can be used when testing gnatsd, or if it is being run via ssh or some other mechanism.

This has the effect of using the local hostname where `gnatsd` is being invoked for authentication purposes, rather than the remote address of the connecting client.

`--max-access-level, -m`
: Specifies the maximum access level that the connecting client can authenticate to. Authentication is as normal but if the user or host authenticates at a higher level, access level is still forced to this level. See Appendix C [Controlling access to databases], page 93 for details on access levels.

## B.3 gnatsd command protocol

Commands are issued to `gnatsd` as one or more words followed by a carriage-return/linefeed pair. For example, the `CHDB` (change database) command is sent as '`CHDB database<CR><LF>`' (the `CRLF` will not be explicitly written for future examples.)

Replies from `gnatsd` are returned as one or more response lines containing a 3-digit numeric code followed by a human-readable string; the line is terminated with a `<CR><LF>` pair. For example, one possible response to the `CHDB` command above would be:

```
210 Now accessing GNATS database 'database'.
```

The three-digit code is normally followed by a single ASCII space (character 0x20). However, if additional response lines are to be returned from the server, there will be a single dash '`-`' instead of the space character after the three-digit code.

Response code values are divided into ranges. The first digit reflects the general type of response (such as "successful" or "error"), and the subsequent digits identify the specific type of response.

*Codes 200-299*
: Positive response indicating that the command was successful. No subsequent data will be transmitted with the response. In particular, code 210 (`CODE_OK`) is used as the positive result code for most simple commands.

  Commands that expect additional data from the client (such as `SUBM` or `VFLD`) use a two-step mechanism for sending the data. The server will respond to the initial command with either a 211 (`CODE_SEND_PR`) or 212 (`CODE_SEND_TEXT`) response line, or an error code if an error occurred with the initial command. The client is then expected to send the remaining data using the same quoting mechanism as described for server responses in the 300-349 range. The server will then send a final response line to the command.

*Codes 300-399*
: Positive response indicating that the query request was successful, and that a PR or other data will follow. Codes 300-349 are used when transmitting PRs, and 350-399 are used for other responses.

  Codes in the 300-349 range are followed by a series of `CRLF`-terminated lines containing the command response, usually a PR. The final line of the result is a single period '`.`'. Result lines that begin with a period have an extra period prepended to them.

Codes in the 350-399 range use a different scheme for sending their responses. The three-digit numeric code will be followed by either a dash '-' or a single space. If the code is followed by a dash, that indicates that another response line will follow. The final line of the response has a single space after the three-digit code.

In previous versions of the protocol the first line of a `CODE_INFORMATION` (310) response was to be ignored. This is no longer the case. Instead, any lines marked with code `CODE_INFORMATION_FILLER` (351) are to be ignored. This allows the server to transmit additional headers or other human-readable text that can be safely ignored by the clients.

*Codes 400-599*

An error occurred, usually because of invalid command parameters or invalid input from the client, missing arguments to the command, or a command was issued out of sequence. The human-readable message associated with the response line describes the general problem encountered with the command.

Multiple error messages may be returned from a command; in this case the '-' continuation character is used on all but the last response line.

*Codes 600-799*

An internal error occurred on the server, a timeout occurred reading data from the client, or a network failure occurred. These errors are of the "this should not occur" nature, and retrying the operation may resolve the problem. Fortunately, most GNATS transactions are idempotent; unfortunately, locking the database or a PR are not repeatable actions (we cannot determine if an existing lock is the one we originally requested, or someone else's).

## B.4 `gnatsd` commands

Note that the set of GNATS commands and their responses is somewhat inconsistent and is very much in flux. At present the GNATS clients are rather simple-minded and not very strict about processing responses. For example, if the server were to issue a code 300 (`CODE_PR_READY`) response to a `CHDB` command, the client would happily expect to see a PR appear (and would print it out if one was sent).

It is thus suggested that any clients that use the GNATS protocol be equally flexible about the way received responses are handled; in particular, only the first digit of the response code should be assumed to be meaningful, although subsequent digits are needed in some cases (codes 300-399). No attempt should be made to parse the message strings on error response lines; they are only intended to be read by humans, and will be changed on a regular basis.

Almost every command may result in the response 440 (`CODE_CMD_ERROR`). This indicates that there was a problem with the command arguments, usually because of insufficient or too many arguments being specified.

Access to most `gnatsd` commands requires a certain *access level*. For details of this, see Section C.5 [Privileged `gnatsd` commands], page 95.

`USER` [*userid password*]

Specifies the userid and password for database access. Either both a username and password must be specified, or they both may be omitted; in the latter case, the current access level is returned.

The possible server responses are:

`350 (CODE_INFORMATION)`
The current access level is specified.

`422 (CODE_NO_ACCESS)`
A matching username and password could not be found.

`200 (CODE_OK)`
A matching username and password was found, and the login was successful.

`QUIT` Requests that the connection be closed. Possible responses:

`201 (CODE_CLOSING)`
Normal exit.

The `QUIT` command has the dubious distinction of being the only command that cannot fail.

`LIST` *list type*

Describes various aspects of the database. The lists are returned as a list of records, one per line. Each line may contain a number of colon-separated fields.

Possible values for *list type* include

`Categories`
Describes the legal categories for the database.

`Submitters`
Describes the set of submitters for the database.

`Responsible`
Lists the names in the responsible administrative file, including their full names and email addresses.

`States`
Lists the states listed in the state administrative file, including the state type (usually blank for most states; the closed state has a special type).

`Classes`
Lists the set of PR classes and their associated human-readable descriptions.

`FieldNames`
Lists the entire set of PR fields.

`InitialInputFields`
Lists the fields that should be provided when a PR is initially entered.

`Databases`
Lists the set of databases.

The possible responses are:

`301 (CODE_TEXT_READY)`
Normal response, followed by the records making up the list as described above.

`416 (CODE_INVALID_LIST)`
The requested list does not exist.

`FTYP` *field* `[`*field* `...]`

Describes the type of data held in the field(s) specified with the command. The currently defined data types are:

`Text`
A plain text field, containing exactly one line.

`MultiText`
A text field possibly containing multiple lines of text.

`Enum`
An enumerated data field; the value is restricted to one entry out of a list of values associated with the field.

`MultiEnum`
The field contains one or more enumerated values. Values are separated with spaces or colons '`:`'.

`Integer`
The field contains an integer value, possibly signed.

`Date`
The field contains a date.

`TextWithRegex`
The value in the field must match one or more regular expressions associated with the field.

The possible responses are:

`350 (CODE_INFORMATION)`
The normal response; the supplied text is the data type.

`410 (CODE_INVALID_FIELD_NAME)`
The specified field does not exist.

If multiple field names were given, multiple response lines will be sent, one for each field, using the standard continuation protocol; each response except the last will have a dash '`-`' immedately after the response code.

`FTYPINFO` *field property*

Provides field-type-related information. Currently, only the property '`separators`' for MultiEnum fields is supported. When '`separators`' is specified, the possible return codes are:

`350 (CODE_INFORMATION)` A proper MultiEnum *field* was specified and the returned text is the string of separators specified for the field in the dbconfig file (see Section 4.3.3 [Field datatypes], page 49) quoted in `'`'s.

`435 (CODE_INVALID_FTYPE_PROPERTY)` The '`separators`' property is not defined for this field, i.e. the specified *field* is not of type MultiEnum.

Currently, specifying a different property than '`separators`' results in return code 435 as above.

`FDSC` *field* `[`*field* `... ]`

Returns a human-readable description of the listed field(s). The possible responses are:

`350 (CODE_INFORMATION)` The normal response; the supplied text is the field description.

`410 (CODE_INVALID_FIELD_NAME)` The specified field does not exist.

Like the `FVLD` command, the standard continuation protocol will be used if multiple fields were specified with the command.

`FIELDFLAGS` *field* `[`*field* `... ]`

Returns a set of flags describing the specified field(s). The possible responses are either

`410 (CODE_INVALID_FIELD_NAME)`
meaning that the specified field is invalid or nonexistent, or

`350 (CODE_INFORMATION)`
which contains the set of flags for the field. The flags may be blank, which indicate that no special flags have been set for this field.

Like the `FDSC` and `FTYP` commands, multiple field names may be listed with the command, and a response line will be returned for each one in the order that the fields appear on the command line.

The flags include:

`textsearch`
The field will be searched when a text field search is requested.

`allowAnyValue`
For fields that contain enumerated values, any legal value may be used in the field, not just ones that appear in the enumerated list.

`requireChangeReason`
If the field is edited, a reason for the change must be supplied in the new PR text describing the reason for the change. The reason must be supplied as a multitext PR field in the new PR whose name is `field-Changed-Why` (where `field` is the name of the field being edited).

`readonly`
The field is read-only, and cannot be edited.

`FVLD` *field*

Returns one or more regular expressions or strings that describe the valid types of data that can be placed in field. Exactly what is returned is dependent on the type of data that can be stored in the field. For most fields a regular expression is returned; for enumerated fields, the returned values are the list of legal strings that can be held in the field.

The possible responses are:

`301 (CODE_TEXT_READY)`
The normal response, which is followed by the list of regexps or strings.

`410 (CODE_INVALID_FIELD_NAME)` The specified field does not exist.

`VFLD` *field* `VFLD` can be used to validate a given value for a field in the database. The client issues the `VFLD` command with the name of the field to validate as an argument. The server will either respond with `212 (CODE_SEND_TEXT)`, or `410 (CODE_INVALID_FIELD_NAME)` if the specified field does not exist.

Once the 212 response is received from the server, the client should then send the line(s) of text to be validated, using the normal quoting mechanism described for PRs. The final line of text is followed by a line containing a single period, again as when sending PR text.

The server will then either respond with 210 (CODE_OK), indicating that the text is acceptable, or one or more error codes describing the problems with the field contents.

`INPUTDEFAULT` *field* `[`*field* `... ]`
: Returns the suggested default value for a field when a PR is initially created. The possible responses are either 410 (CODE_INVALID_FIELD_NAME), meaning that the specified field is invalid or nonexistent, or 350 (CODE_INFORMATION) which contains the default value for the field.

  Like the FDSC and FTYP commands, multiple field names may be listed with the command, and a response line will be returned for each one in the order that the fields appear on the command line.

`RSET`
: Used to reset the internal server state. The current query expression is cleared, and the index of PRs may be reread if it has been updated since the start of the session. The possible responses are:

  `200 (CODE_OK)`
  The state has been reset.

  `440 (CODE_CMD_ERROR)`
  One or more arguments were supplied to the command.

  `6xx (internal error)`
  There were problems resetting the state (usually because the index could not be reread). The session will be immediately terminated.

`LKDB`
: Locks the main GNATS database. No subsequent database locks will succeed until the lock is removed. Sessions that attempt to write to the database will fail. The possible responses are:

  `200 (CODE_OK)` The lock has been established.

  `440 (CODE_CMD_ERROR)` One or more arguments were supplied to the command.

  `431 (CODE_GNATS_LOCKED)` The database is already locked, and the lock could not be obtained after 10 seconds.

  `6xx (internal error)` An internal error occurred, usually because of permission or other filesystem-related problems. The lock may or may not have been established.

`UNDB`
: Unlocks the database. Any session may steal a database lock; no checking of any sort is done. The possible responses are:

  `200 (CODE_OK)`
  The lock has been removed.

  `432 (CODE_GNATS_NOT_LOCKED)`
  The database was not locked.

`440 (CODE_CMD_ERROR)`
One or more arguments were supplied to the command.

`6xx (internal error)`
The database lock could not be removed, usually because of permissions or other filesystem-related issues.

`LOCK` *PR user* `[`*pid*`]`
Locks the specified *PR*, marking the lock with the *user* name and the optional *pid*. (No checking is done that the *user* or *pid* arguments are valid or meaningful; they are simply treated as strings.)

The `EDIT` command requires that the PR be locked before it may be successfully executed. However, it does not require that the lock is owned by the editing session, so the usefulness of the lock is simply as an advisory measure.

The `APPN` and `REPL` commands lock the PR as part of the editing process, and they do not require that the PR be locked before they are invoked.

The possible responses are:

`440 (CODE_CMD_ERROR)`
Insufficient or too many arguments were specified to the command.

`300 (CODE_PR_READY)`
The lock was successfully obtained; the text of the PR (using the standard quoting mechanism for PRs) follows.

`400 (CODE_NONEXISTENT_PR)`
The PR specified does not exist.

`430 (CODE_LOCKED_PR)`
The PR is already locked by another session.

`6xx (internal error)`
The PR lock could not be created, usually because of permissions or other filesystem-related issues.

`UNLK` *PR* Unlocks *PR*. Any user may unlock a PR, as no checking is done to determine if the requesting session owns the lock.

The possible responses are:

`440 (CODE_CMD_ERROR)`
Insufficient or too many arguments were specified to the command.

`200 (CODE_OK)`
The PR was successfully unlocked.

`433 (CODE_PR_NOT_LOCKED)`
The PR was not locked.

`6xx (internal error)`
The PR could not be unlocked, usually because of permission or other filesystem-related problems.

`DELETE` *PR*
Deletes the specified *PR*. The user making the request must have admin privileges (see Appendix C [Controlling access to databases], page 93). If successful,

the PR is removed from the filesystem and the index file; a gap will be left in the numbering sequence for PRs. No checks are made that the PR is closed.

The possible responses are:

`200 (CODE_OK)`
The PR was successfully deleted.

`422 (CODE_NO_ACCESS)`
The user requesting the delete does not have admin privileges.

`430 (CODE_LOCKED_PR)`
The PR is locked by another session.

`431 (CODE_GNATS_LOCKED)`
The database has been locked, and no PRs may be updated until the lock is cleared.

`6xx (internal error)`
The PR could not be successfully deleted, usually because of permission or other filesystem-related problems.

`CHEK [initial]`
Used to check the text of an entire PR for errors. Unlike the `VFLD` command, it accepts an entire PR at once instead of the contents of an individual field.

The `initial` argument indicates that the PR text to be checked is for a PR that will be newly created, rather than an edit or replacement of an existing PR.

After the `CHEK` command is issued, the server will respond with either a `440 (CODE_CMD_ERROR)` response indicating that the command arguments were incorrect, or a `211 (CODE_SEND_PR)` response code will be sent.

Once the `211` response is received from the server, the client should send the PR using the normal PR quoting mechanism; the final line of the PR is then followed by a line containing a single period, as usual.

The server will then respond with either a `200 (CODE_OK)` response, indicating there were no problems with the supplied text, or one or more error codes listing the problems with the PR.

`EDIT` *PR*
Verifies the replacement text for *PR*. If the command is successful, the contents of *PR* are completely replaced with the supplied text. The PR must previously have been locked with the `LOCK` command.

The possible responses are:

`431 (CODE_GNATS_LOCKED)`
The database has been locked, and no PRs may be updated until the lock is cleared.

`433 (CODE_PR_NOT_LOCKED)`
The PR was not previously locked with the `LOCK` command.

`400 (CODE_NONEXISTENT_PR)`
The specified PR does not currently exist. The `SUBM` command should be used to create new PRs.

`211 (CODE_SEND_PR)`
The client should now transmit the replacement PR text using the normal PR quoting mechanism. After the PR has been sent, the server will respond with either `200 (CODE_OK)` indicating that the edit was successful, or one or more error codes listing problems either with the replacement PR text or errors encountered while updating the PR file or index.

`EDITADDR` *address*
Sets the e-mail address of the person communicating with `gnatsd`. The command requires at least the `edit` access level.

The possible responses are:

`200 (CODE_OK)`
The address was successfully set.

`440 (CODE_CMD_ERROR)`
Invalid number of arguments were supplied.

`APPN` *PR field*
`REPL` *PR field*
Appends to or replaces the contents of *field* in *PR* with the supplied text. The command returns a `201 (CODE_SEND_TEXT)` response; the client should then transmit the new field contents using the standard PR quoting mechanism. After the server has read the new contents, it then attempts to make the requested change to the PR.

The possible responses are:

`200 (CODE_OK)`
The PR field was successfully changed.

`400 (CODE_NONEXISTENT_PR)`
The PR specified does not exist.

`410 (CODE_INVALID_FIELD_NAME)`
The specified field does not exist.

`402 (CODE_UNREADABLE_PR)`
The PR could not be read.

`431 (CODE_GNATS_LOCKED)`
The database has been locked, and no PRs may be updated until the lock is cleared.

`430 (CODE_LOCKED_PR)`
The PR is locked, and may not be altered until the lock is cleared.

`413 (CODE_INVALID_FIELD_CONTENTS)`
The supplied (or resulting) field contents are not valid for the field.

`6xx (internal error)`
An internal error occurred, usually because of permission or other filesystem-related problems. The PR may or may not have been altered.

`SUBM` Submits a new PR into the database. The supplied text is verified for correctness, and if no problems are found a new PR is created.

The possible responses are:

`431 (CODE_GNATS_LOCKED)`
The database has been locked, and no PRs may be submitted until the lock is cleared.

`211 (CODE_SEND_PR)`
The client should now transmit the new PR text using the normal quoting mechanism. After the PR has been sent, the server will respond with either a `200 (CODE_OK)` response indicating that the new PR has been created (and mail sent to the appropriate persons), or one or more error codes listing problems with the new PR text.

`CHDB` *database*
Switches the current database to the name specified in the command.

The possible responses are:

`422 (CODE_NO_ACCESS)`
The user does not have permission to access the requested database.

`417 (CODE_INVALID_DATABASE)`
The database specified does not exist, or one or more configuration errors in the database were encountered.

`220 (CODE_OK)`
The current database is now *database*. Any operations performed will now be applied to *database*.

`DBLS` Lists the known set of databases.

The possible responses are:

`6xx (internal error)`
An internal error was encountered while trying to obtain the list of available databases, usually due to lack of permissions or other filesystem-related problems, or the list of databases is empty.

`301 (CODE_TEXT_READY)`
The list of databases follows, one per line, using the standard quoting mechanism. Only the database names are sent.

The `gnatsd` access level '`listdb`' denies access until the user has authenticated with the USER command. The only other command available at this access level is `DBLS`. This access level provides a way for a site to secure its GNATS databases while still providing a way for client tools to obtain a list of the databases for use on login screens etc. See Appendix C [Controlling access to databases], page 93.

`DBDESC` *database*
Returns a human-readable description of the specified *database*.

Responses include:

`6xx (internal error)`
An internal error was encountered while trying to read the list of available databases, usually due to lack of permissions or other filesystem-related problems, or the list of databases is empty.

`350 (CODE_INFORMATION)`
The normal response; the supplied text is the database description.

`417 (CODE_INVALID_DATABASE)`
The specified database name does not have an entry.

`EXPR` *query expression*

Specifies a *query expression* used to limit which PRs are returned from the `QUER` command. The expression uses the normal query expression syntax, (see Section 2.4.3 [Query expressions], page 25).

Multiple `EXPR` commands may be issued; the expressions are boolean ANDed together.

Expressions are cleared by the `RSET` command.

Possible responses include:

`415 (CODE_INVALID_EXPR)`
The specified expression is invalid, and could not be parsed.

`200 (CODE_OK)`
The expression has been accepted and will be used to limit the results returned from `QUER`.

`QFMT` *query format*

Use the specified *query format* to format the output of the `QUER` command. The query format may be either the name of a query format known to the server (see Section 4.3.5 [Named query definitions], page 52), or an actual query format (see Section 2.4.2 [Formatting query-pr output], page 24). The possible responses are:

`200 (CODE_OK)`
The normal response, which indicates that the query format is acceptable.

`440 (CODE_CMD_ERROR)`
No query format was supplied.

`418 (CODE_INVALID_QUERY_FORMAT)`
The specified query format does not exist, or could not be parsed.

`QUER [`*PR*`] [`*PR*`] [...]`

Searches the contents of the database for PRs that match the (optional) specified expressions with the `EXPR` command. If no expressions were specified with `EXPR`, the entire set of PRs is returned.

If one or more PRs are specified on the command line, only those PRs will be searched and/or output.

The format of the output from the command is determined by the query format selected with the `QFMT` command.

The possible responses are:

`418 (CODE_INVALID_QUERY_FORMAT)`
A valid format was not specified with the `QFMT` command prior to invoking `QUER`.

`300 (CODE_PR_READY)`
One or more PRs will be output using the requested query format. The PR text is quoted using the normal quoting mechanisms for PRs.

`220 (CODE_NO_PRS_MATCHED)`
No PRs met the specified criteria.

`ADMV` *field key* `[`*subfield*`]`

Returns an entry from an administrative data file associated with *field*. *key* is used to look up the entry in the data file. If *subfield* is specified, only the value of that subfield is returned; otherwise, all of the fields in the adm data file are returned, separated by colons '`:`'.

The responses are:

`410 (CODE_INVALID_FIELD_NAME)` The specified field does not exist.

`221 (CODE_NO_ADM_ENTRY)` An adm entry matching the key was not found, or the field does not have an adm file associated with it.

`350 (CODE_INFORMATION)` The normal response; the supplied text is the requested field(s).

## B.5 gnatsd environment variables

`gnatsd` supports the `GNATSDB` environment varable in almost the same way as the GNATS tools do. This variable is used to determine which database to use. For a local database, it contains the name of the database to access. `gnatsd` cannot service remote databases (though it might be interesting if it could) so the database is always assumed to be local.

If `GNATSDB` is not set and the `--database` option is not supplied, it is assumed that the database is local and that its name is '`default`'.

# Appendix C Controlling access to databases

## C.1 Overview

GNATS supports granting various levels of access to the GNATS databases served by the network daemon, `gnatsd`.

GNATS access can be controlled at these levels:

`deny` gnatsd closes the connection

`none` no further access until userid and password given

`listdb` only listing of available databases is allowed

`view` query and view PRs with Confidential=no only

`viewconf` query and view PRs with Confidential=yes

`edit` full edit access

`admin` full admin access

These access levels are used in the following settings:

- overall gnatsd access level
- overall access level set by host name or IP address
- overall access level set by userid and password
- per-database access level set by userid and password

## C.2 Overall `gnatsd` access level

The overall `gnatsd` access level is set by starting `gnatsd` with the option

`-m` *level* `or --maximum-access-level=`*level*`,`

where *level* is one of the six access levels listed above. This restricts any access to the GNATS daemon to levels up to and including *level*, regardless of the settings in the access control files discussed below. If this option is left out, any access levels set in the access control files will be allowed.

The discussion below assumes that the pre-build configure of GNATS was done without altering the default values for the `--with-gnatsd-user-access-file` and `--with-gnatsd-host-access-file` options. If non-default values were given, substitute as appropriate below.

## C.3 Overall access levels per host

The file '*SYSCONFDIR*`/gnats/gnatsd.host_access`' (usually '`/etc/gnats/gnatsd.host_access`') controls overall access levels on a per-host basis, meaning that settings in this file apply across all databases on the server. Entries in this file are on the following format:

*host:access-level:whatever*

*host* is the hostname or IP address of the host contacting gnatsd. Wildcard characters are supported: '`*`' matches anything; '`?`' matches any single character. By using wildcards, you can specify access levels for entire network subnets and domains.

The second field is the access level of *host*. The default is `deny`. If the user's hostname isn't in the file or its access level is set to `deny`, the connection is closed immediately.

GNATS currently doesn't make use of the third field. Remember to still include the second '`:`' on the line if you choose to leave the third field empty.

Whenever a `CHDB` command is processed (or defaulted), the user's access level is set to the level for their host, as determined by the values in the '`gnatsd.host_access`' file. However, even if a host is given the `none` access level, an individual can still give the `USER` command to possibly gain a higher (but never lower) access than is set for their host. The gnatsd `USER` command takes two arguments: `USER <userid> <passwd>`.

## C.4 Access levels per user

Access levels per user can be set both across all databases on the server or on a per-database basis. The '`gnatsd.access`' file in a database's '`gnats-adm`' directory specifies the user access rules for that database. If it doesn't exist, or doesn't contain the user name given to `gnatsd`, then the file '*SYSCONFDIR*`/gnats/gnatsd.access`', usually '`/etc/gnats/gnatsd.access`', specifying the per-user access levels across all the databases on the server is checked.

The user access files can only *increase* the access level defined in the host access files for the given host, they can never lower it.

If the access level is `none` after processing the userid and password, the connection is closed.

The '`gnatsd.access`' files can contain plain text passwords, in such a case they should be owned by the GNATS user with file permission 600.

Wildcard characters are supported for the userid and password with plain text passwords. A null string or '`*`' matches anything; '`?`' matches any one character.

Entries in the database-specific '`gnatsd.access`' user access file in the '`gnats-adm`' directory of the database have the following general format:

*userid:password:access-level*

*password* should either be in plain text, DES `crypt()`[1] or MD5 hash format[2].

If the password is in plain text format, it must be prefixed by '`$0$`' and if it is in MD5 format, it needs to be prefixed by the string '`$1$`'. Passwords encrypted by `crypt()` should have no prefix.

A `gnats-passwd` tool to manage '`gnatsd.access`' files is planned. In the meantime, `crypt()` passwords can be generated by using standard UNIX passwords tools, while MD5 passwords can be generated with the following little Perl snippet:

[1] DES crypt is the standard password encryption format used by most UNIX systems

[2] MD5 is only supported on platforms that have a `crypt()` function that supports MD5. Among others, this currently includes GNU Linux and OpenBSD.

```
perl -e 'use Crypt::PasswdMD5 ; print Crypt::PasswdMD5::unix_md5_crypt
"password" , time() % 100000000'
```

If your Perl installation doesn't have the Crypt module installed, you need to install it. On most systems, the following command achieves this:

```
perl -MCPAN -e 'install Crypt::PasswdMD5'
```

A tool for conversion of pre-version 4 '`gnatsd.access`' files is distributed with GNATS 4. See Section 4.6.6 [Converting old password files], page 64.

The *access-level* field should contain one of the values listed at the beginning of this appendix. This overrides (increases but never lowers) the access level given as the default for the user's host in the global gnatsd.host_access file.

The following shows an example gnatsd.access file with plain text passwords:

```
rickm:$0$ruckm:edit
pablo:$0$pueblo:view
*:*:none
```

And this is the same file with MD5-encrypted passwords:

```
rickm:$1$92388613$D7ZIYikzTUqd./dODTFrI.:edit
pablo:$1$92388652$QRfAhIBG5elT.FQjQKhj80:view
*:*:none
```

In these examples, anybody other than rickm and pablo get denied access, assuming that the host access level is also `none`. You could set the catch-all rule at the end to be `*:*:view` to allow view access to anyone.

The overall user access file '*SYSCONFDIR*/`gnats/gnatsd.access`', usually '`/etc/gnats/gnatsd.access`', adds a fourth *database* field. This file contains a comma-separated list of database names, as defined in the '`databases`' file (see Section 4.2 [The `databases` file], page 45. Wildcard characters are supported. The databases listed in this field are the ones to which the other settings on the same line will be applied.

## C.5 Privileged `gnatsd` commands

Every `gnatsd` command has a minimum access level attached to it. If your access level is too low for a command, you get this response:

```
LOCK 12
422 You are not authorized to perform this operation (LOCK).
```

The commands `CHDB`, `USER` and `QUIT` are unrestricted.

The `DBLS` command requires at least `listdb` access.

A user must have at least `edit` access for these commands:

`LKDB` lock the main GNATS database.

`UNDB` unlock the main GNATS database.

`LOCK` *PR user pid*
lock *PR* for *user* and optional *pid* and return PR text.

`UNLK` *PR* unlock *PR*.

`EDIT` *PR* check in edited *PR*.

`APPN` *PR field*, `REPL` *PR field*
: Appends to or replaces the contents of *field* in *PR*.

The `DELETE` *PR* command is special in that it requires `admin` access. All other commands require `view` access.

`edit-pr` and `query-pr` accept the command line arguments `-v|--user` and `-w|--passwd`. See Chapter 2 [The GNATS User Tools], page 13.

# Appendix D Querying using regular expressions

See also Section 2.4.3 [Query expressions], page 25.

Unfortunately, we do not have room in this manual for a complete exposition on regular expressions. The following is a basic summary of some regular expressions you might wish to use.

*NOTE: When you use query expressions containing regular expressions as part of an ordinary query-pr shell command line, you need to quote them with* `''`*, otherwise the shell will try to interpret the special characters used, yielding highly unpredictable results.*

See section "Regular Expression Syntax" in *Regex*, for details on regular expression syntax. Also see section "Syntax of Regular Expressions" in *GNU Emacs Manual*, but beware that the syntax for regular expressions in Emacs is slightly different.

All search criteria options to `query-pr` rely on regular expression syntax to construct their search patterns. For example,

```
query-pr --expr 'State="open"' --format full
```

matches all PRs whose '`>State:`' values match with the regular expression '`open`'.

We can substitute the expression '`o`' for '`open`', according to GNU regular expression syntax. This matches all values of '`>State:`' which begin with the letter '`o`'.

We see that

```
query-pr --expr 'State="o"' --format full
```

is equivalent to

```
query-pr --expr 'State="o"' --format full
```

in this case, since the only value for '`>State:`' which matches the expression '`o`' is '`open`'. '`State="o"`' also matches '`o`', '`oswald`', and even '`oooooo`', but none of those values are valid states for a Problem Report in default GNATS installations.

We can also use the expression operator '`|`' to signify a logical `OR`, such that

```
query-pr --expr 'State="o|a"' --format full
```

matches all '`open`' or '`analyzed`' Problem Reports.

Regular expression syntax considers a regexp token surrounded with parentheses, as in '`(`*regexp*`)`', to be a *group*. This means that '`(ab)*`' matches any number (including zero) of contiguous instances of '`ab`'. Matches include '', '`ab`', and '`ababab`'.

Regular expression syntax considers a regexp token surrounded with square brackets, as in '`[`*regexp*`]`', to be a *list*. This means that '`Char[(ley)(lene)(broiled)`' matches any of the words '`Charley`', '`Charlene`', or '`Charbroiled`' (case is significant; '`charbroiled`' is not matched).

Using groups and lists, we see that

```
query-pr --expr 'Category="gcc|gdb|gas"' --format full
```

is equivalent to

```
query-pr --expr 'Category="g(cc|db|as)"' --format full
```

and is also very similar to

```
query-pr --expr 'Category="g[cda]"' --format full
```

with the exception that this last search matches any values which begin with 'gc', 'gd', or 'ga'.

The '.' character is known as a *wildcard*. '.' matches on any single character. '*' matches the previous character (except newlines), list, or group any number of times, including zero. Therefore, we can understand '.*' to mean "match zero or more instances of any character."

```
query-pr --expr 'State=".*a"' --format full
```

matches all values for '>State:' which contain an 'a'. (These include 'analyzed' and 'feedback'.)

Another way to understand what wildcards do is to follow them on their search for matching text. By our syntax, '.*' matches any character any number of times, including zero. Therefore, '.*a' searches for any group of characters which end with 'a', ignoring the rest of the field. '.*a' matches 'analyzed' (stopping at the first 'a') as well as 'feedback'.

*Note:* When using 'fieldtype:Text' or 'fieldtype:Multitext' (see Section 2.4.3 [Query expressions], page 25), you do not have to specify the token '.*' at the beginning of your expression to match the entire field. For the technically minded, this is because these queries use 're_search' rather than 're_match'. 're_match' *anchors* the search at the beginning of the field, while 're_search' does not anchor the search.

For example, to search in the >Description: field for the text

```
The defrobulator component returns a nil value.
```

we can use

```
query-pr --expr 'fieldtype:Multitext="defrobulator.*nil"' --format full
```

To also match newlines, we have to include the expression '(.|^M)' instead of just a dot ('.'). '(.|^M)' matches "any single character except a newline ('.') *or* ('|') any newline ('^M')." This means that to search for the text

```
The defrobulator component enters the bifrabulator routine
and returns a nil value.
```

we must use

```
query-pr --expr 'fieldtype:Multitext="defrobulator(.|^M)*nil"'
        --format full
```

To generate the newline character '^M', type the following depending on your shell:

csh
: '*control*-V *control*-M'

tcsh
: '*control*-J *control*-J'

sh (*or* bash)
: Use the ⟨RETURN⟩ key, as in

```
(.|
)
```

Again, see section "Regular Expression Syntax" in *Regex*, for a much more complete discussion on regular expression syntax.

# Appendix E GNATS support

The GNATS home page is located at `http://www.gnu.org/software/gnats`. It contains all the important references to the available information about GNATS and the related software.

There is also a special page dedicated to the GNATS development at `http://savannah.gnu.org/projects/gnats`.

There are several GNATS mailing lists. The most important ones are:

`info-gnats@gnu.org`
: Announcements and other important information about GNATS and the related software. This is a very low volume moderated list.

`bug-gnats@gnu.org`
: The bug reporting mailing list on the GNATS itself. Please note that the preferred way to report GNATS bugs is to submit them via the web interface at `http://bugs.gnu.org/cgi-bin/gnatsweb.pl?database=gnats`. New bug reports submitted via the web interface are copied to the mailing list automatically.

`help-gnats@gnu.org`
: General discussion about GNATS. Anything related to GNATS (user questions, development, suggestions, etc.) can be discussed there.

The complete list of GNATS related mailing lists is available from the web page at `http://savannah.gnu.org/project/gnats`.

When you report problems concerning GNATS itself, please do not forget to provide especially the following information:

- The GNATS version you are using.
- The *exact* way how to reproduce the bug.
- Your configuration.
- If you encounter a compilation or build problem, it is especially important to mention the operating system, compiler and possibly other build utilities you use.

Providing this information in the initial report avoids further unnecessary communication, saves our limited development resources and helps to track down and fix the problem soon.

# Index

## D



## E



## F



## G

## H



## I



## K



## L



## M



## N



## O

## P



## Q



## R



## S



## T

## U



## V



## W