Crab Documentation

Release 0.5.1

Graham Bell

Sep 16, 2022

Contents

Crab is a dashboard system for monitoring cron jobs, or other scheduled tasks. The Crab server receives messages when tasks start or finish, and displays the status of all of the tasks via a web interface. It can also send notifications by email, for example to warn if a task fails, is missed or does not complete within its time-out period.

Tasks communicate with the Crab server by JSON messages sent by HTTP PUT requests. The finish message includes the status of the job, and any output from it. Further messages are used to import and export the client's crontab, which the server uses to determine the intended schedule.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Requirements

Packages

- crontab (0.15 or newer)
- CherryPy
- Mako (PyPI entry)
- jQuery
- Font Awesome (free version for web, optional)
- ansi_up (optional)
- MySQL Connector (needed only if using a MySQL database, PyPI entry)

Python Version

Crab server Has been tested on Python 2.6, 2.7 and 3.2.

Client library and utilities Works with Python 2.4 in addition to the above versions (but may require the `pytz` and `simplejson` packages also to be installed).

1.1.2 Installation

The Crab server, clients and libraries can be installed as follows:

```
python setup.py install
```

If necessary, the `--install-data` option can be used to configure the location in which the templates (`templ`), resources (`res`) and example files (`doc`) should be installed.

To run Crab without installing it, and if any of the Python dependencies listed above can not be installed, they can be symlinked into the `lib` directory in the following locations:

```
lib/cherrypy
lib/crontab
lib/mako
```

The jQuery JavaScript library should be copied or symlinked into Crab's `res` directory as:

```
res/jquery.js
```

To use Font Awesome icons, copy or symlink its `webfonts` directory into Crab's `res` directory, and also place its stylesheets inside that subdirectory, giving:

```
res/webfonts/fontawesome.css
res/webfonts/solid.css
res/webfonts/fa-*.*
```

Note that Font Awesome is not backward compatible between major version numbers. Crab now uses version 6 of Font Awesome.

To use ansi_up to interpret ANSI color commands in cron job output, copy or symlink the `ansi_up.js` file into Crab's `res` directory:

```
res/ansi_up.js
```

1.2 Server Configuration

1.2.1 Database Creation

A SQLite database file can be prepared for Crab using the schema provided:

```
% sqlite3 crab.db < doc/schema.sql
```

Alternatively if you are going to be using MySQL for your Crab database, create the database:

```
% mysqladmin -u root -p create crab
```

and create a user account for crab, changing the password (the "identified by" clause) to something suitable:

```
% mysql -u root -p mysql
> create user 'crab'@'localhost' identified by 'crab';
> grant all on crab.* to 'crab'@'localhost';
> flush privileges;
```

You can prepare a table creation script suitable for MySQL using the Makefile in the *doc* directory of the source package:

```
% make -C doc schema_mysql.sql
% mysql -u crab -p crab < doc/schema_mysql.sql
```

1.2.2 Configuration

The Crab server is configured by a `crabd.ini` file which can be placed either in `/etc/crab/` or `~/.crab/`. Note that this is a CherryPy configuration file, which is read slightly differently to typical `.ini` files which use Python's ConfigParser.

```
% cp doc/crabd.ini ~/.crab/
```

The example `crabd.ini` file should be edited to uncomment the `[crab]` and `[store]` sections. The `home` and `file` entries must point to the location of Crab's data files and the database file just created. By default the data files are installed in `share/crab` relative to the Python system prefix (`sys.prefix`).

There is also an `[outputstore]` section in the server configuration file. This allows the output from cron jobs and raw crontab files to be stored separately, and can be used to prevent the main database from becoming excessively large.

If you would like to have Crab delete the history of job events over a certain age, you can have it run a cleaning service by enabling the `[clean]` section of the server configuration file. Here you can select the cleaning schedule and length of history to keep. A fairly frequent cleaning schedule is recommended to avoid the accumulation of a large number of old events so that each cleaning operation does not take long. If the file output store is being used, the cleaning service will remove only the event records and not the output text. You can remove old output text separately, for example by running in your output store directory:

```
% find output -type f -mtime +90 -delete
% find output -type d -empty -delete
```

1.2.3 Running

The Crab server is run as `crabd`. When the server is executed directly, it will stay in the foreground:

```
% crabd
```

It can also be run in the background with the `crabd-check` script, which checks that it is not still running from a previous invocation of `crabd-check`. Therefore this is suitable for running from cron to keep the server running:

```
PYTHONPATH=/path/to/crab/lib
PATH=/path/to/crab/scripts:/bin:/usr/bin
7-57/10 * * * * CRABIGNORE=yes crabd-check
```

With the server running, the Crab dashboard should be visible from a web browser, by default on port 8000. The Crab clients will use this same web service to communicate with the server.

1.2.4 Migrating Job Information

The Crab server has the ability to export and import cron job information, including:

- The list of cron jobs.
- The configuration and notifications attached to each job.
- General host/user-based notifications.
- Raw crontabs.

You can write this information to a JSON file using the `--export` option:

```
% crabd --export job_information.json
```

Similarly you can read information with the `--import` option:

```
% crabd --import job_information.json
```

This merges the information from the file with the server's existing configuration. You can also give a file name of – to export to standard output or read from standard input.

1.2.5 Example Configuration File

Here is the example server configuration file `crabd.ini` which is distributed with Crab:

```
# This file is read by CherryPy rather than ConfigParser
# and the following differences apply:  strings must be
# quoted, and it appears that if you include a section,
# you must include all settings in that section as the
# defaults are not kept.

# [crab]
# # Directory in which to find the res/ and templ/ directories.
# home = '/usr/share/crab'
#
# # Base URL to use when generating links to be used from
# # outside the Crab web interface, e.g. in notification
# # emails.
# base_url = 'http://crabserver.example.com:8000'
# # To generate automatically:
# base_url = None

# [store]
# # Main storage backend.
# type = 'sqlite'
# file = '/var/lib/crab/crab.db'
# # Alternatively for MySQL:
# # type = 'mysql'
# # host = 'localhost'
# # database = 'crab'
# # user = 'crab'
# # password = 'crab'

# [outputstore]
# # Storage backend to be used for storing job output
# # and raw crontabs.
# # (This is optional, unless the selected main backend
# # is not capable of storing output.)
# type = 'file'
# dir = '/var/lib/crab'

# [global]
# engine.autoreload.on = False
#
# server.socket_port = 8000
#
# # To listen on localhost only:
# server.socket_host = '127.0.0.1'
```

(continues on next page)

(continued from previous page)

```
#
# # To listen on a specific address:
# server.socket_host = '0.0.0.0'

# [email]
# # Server through which to send email notifications.
# server = 'mailhost'
#
# # Name (and address) to send email from.
# from = 'Crab Daemon'
#
# # Subjects to use for different severity levels.
# subject_ok = 'Crab notification'
# subject_warning = 'Crab notification (WARNING)'
# subject_error = 'Crab notification (ERROR)'

# [notify]
# # Cron-style schedule for sending "daily" notifications,
# # to be used for notifications without specified schedules.
# daily = '0 0 * * *'
#
# # Timezone to use for the daily notification schedule.
# timezone = 'UTC'

# # Uncomment this section if you wish to use the automated cleaning
# # service to delete the history of old events.
# [clean]
# # Cron-style schedule for cleaning operations.
# schedule = '15 0 * * *'
# # Timezone to use for the cleaning schedule.
# timezone = 'UTC'
# # Number of days for which to keep events.
# keep_days = 90

# # This section applies if crabd is run with the --accesslog option
# # giving the base access log file name (e.g. via crabd-check).
# [access_log]
# # Maximum size of log files (MiB), or 0 to disable rotation.
# max_size = 10
# # Number of past log files to keep, or 0 to disable rotation.
# backup_count = 10

# # This section applies if crabd is run with the --errorlog option
# # giving the base error log file name (e.g. via crabd-check).
# [error_log]
# # Maximum size of log files (MiB), or 0 to disable rotation.
# max_size = 10
# # Number of past log files to keep, or 0 to disable rotation.
# backup_count = 10
```

1.3 Client Configuration

There are two Crab client commands: the `crab` utility, and the `crabsh` wrapper shell. Cron jobs can either be run under `crabsh`, or they can be updated to report their own status to the Crab server.

1.3.1 Configuration

The Crab clients are configured by a `crab.ini` file which can be placed either in `/etc/crab/` or `~/.crab/`. The file specifies how to contact the Crab server, and the username and hostname which the client will use to report cron jobs.

```
% cp doc/crab.ini ~/.crab/
```

The configuration can be checked with the `crab info` command. This reports the settings, and indicates which configuration files were read. It is a useful way to check that everything is in order before importing a crontab.

1.3.2 The `crabsh` Wrapper

`crabsh` is a wrapper script designed to act like a shell. It can therefore be invoked by cron via the `SHELL` variable, for example:

```
PYTHONPATH=/path/to/crab/lib
SHELL=/path/to/crab/scripts/crabsh
0 10 * * 1-5 CRABID=test echo "Test cron job"
```

Where the rules following the `SHELL` assignment will be run with the wrapper. The `PYTHONPATH` will need to be set if Crab is not installed where the system can find it. Cron requires the full path when specifying the `SHELL`. The `CRABID` parameter is used to give the cron job a convenient and unique name. This is optional, unless there are multiple jobs with the same command, in which case they would otherwise be indistinguishable. However if it specified, then it must be unique for a given host and user, as the Crab server will use it in preference to the command string to identify cron job reports.

`crabsh` will notify the server when the job starts, and when it finishes, assuming it succeeded if the exit status was zero.

1.3.3 Crab-aware Cron Jobs

Alternatively a cron job can report its own status to the Crab server. The most straightforward way to do this is to execute the `crab` utility. So a cron job written as a shell script could include commands such as:

```
% crab start -c "$0"
% crab finish -c "$0"
% crab fail -c "$0"
```

In this way you can also report a warning with `crab warning` or an unknown status with `crab unknown`.

Python If the cron job is written in Python, it could import `crab.client` directly and make use of the `CrabClient` class.

Perl A Perl module WWW::Crab::Client is also available.

Other languages Other language libraries could be written. They would need to make HTTP PUT requests with an appropriate JSON message.

1.3.4 Managing the Cron Job List

The Crab server needs to be given the schedule for each job so that it can detect when a job is late or missed. This is done by “importing” a user’s crontab file:

```
% crab import
```

The database entries can then be checked by “exporting” them, again using the `crab` utility:

```
% crab export
> CRON_TZ=Pacific/Honolulu
> 0 10 * * 1-5 CRABID=test echo "Test cron job"
```

The output is a set of crontab-style lines representing the entries from the database. The crontab can be retrieved exactly as last imported (from a separate database table containing the raw crontab) by giving the `--raw` option as follows:

```
% crab export --raw
```

This is useful as a backup in case a crontab is accidentally lost. However it will not contain any new jobs which have been added automatically by the Crab server since the last import.

1.3.5 Cron Job Parameters

In order to specify the Crab specific parameters of a cron job, Bourne-style shell variables at the start of a command are used. The syntax for each cron job is as follows:

```
<schedule> [CRABIGNORE=yes] [CRABID=<identifier>] <command string>
```

A command starting with CRABIGNORE set to a value other than 0/no/off/false will be ignored when importing a crontab, and `crabsh` will not report its status to the Crab server.

A CRABID specification will override any CRABID environment variable in effect, and is a better way of specifying the identifier as it can not apply to more than one cron job. There should not be multiple jobs with the same identifier for any user and host.

The Crab parameters can be placed in any order before the remainder of the command string, but they must precede any other variables.

1.3.6 Environment Variables

CRABCLIENTHOSTNAME The host name to be used by Crab clients to identify themselves, overriding any value in the configuration files.

CRABECHO If present and not set to 0/no/off/false then `crabsh` will print out the standard output and standard error it receives from the cron job. This allows the output to be sent by email via cron’s default behavior as well as being captured by the Crab system.

CRABHOME If present overrides the Crab server home directory, where the `res` and `templ` directories are to be found.

CRABHOST Specifies the Crab server to which clients should connect, overriding the setting in the configuration file.

CRABID Specifies the job identifier which `crabsh` will use to file reports if there is no `CRABID=` variable at the start of the cron command. This should be used with caution to avoid specifying the same identifier for multiple cron jobs.

CRABIGNORE Prevents Crab from acting on specific cron jobs. Jobs imported with this value present and not set to 0/no/off/false will not be entered into the database. Additionally if the `crabsh` wrapper script is used to run such a job, it will not report its status to the Crab server.

CRABPIDFILE Gives the path to a PID file which `crabsh` should use to control the execution of a cron job. When this parameter is set, it will use the file to try not to run multiple copies of the job at the same time. Each job should have a separate PID file, so this parameter is most conveniently given at the start of a command string.

CRABPORT Specifies the port on the Crab server, overriding the setting in the configuration file.

CRABQUIET If present and not set to 0/no/off/false then `crabsh` will not write fallback messages to standard output except in case of failure. Such messages would normally be sent by cron to the `MAILTO` address. This option can also be specified via the client configuration files.

CRABSHELL The shell which `crabsh` will use to invoke the cron job command. Defaults to `/bin/sh` regardless of the user's shell to replicate cron's behavior.

CRABSYSCONFIG The directory to be searched for system-level configuration files. If not set, then /etc/crab will be used.

CRABUSERCONFIG A directory to search for user-level configuration files. If not set then ~/.crab will be used.

CRABUSERNAME The user name to be used by Crab clients to identify themselves, overriding any value in the configuration files.

CRABWATCHDOG Specifies a timeout (in minutes) which `crabsh` should apply to the cron job, killing it and reporting status "watchdog" if exceeded. (Requires Python 3.3 or the subprocess32 backport module.)

CRON_TZ Cron reads this variable to know in which timezone to interpret the crontab schedule. When the server receives a crontab, it will check for this timezone and use it to override the general timezone which the `crab` utility will send with the crontab (if it is able to determine it).

MAILTO Configures the email address to which cron sends email. This is useful when `CRABECHO` is on, or if `crabsh` needs to report a failure to contact the Crab server.

SHELL Cron uses this variable to select the shell which will be used to execute the cron jobs. The full path must be specified. Crab does not use this variable itself.

TZ This can be set to the system timezone, in which case `crab import` will use it as the default timezone for the crontab.

1.3.7 Example Configuration File

Here is the example client configuration file `crab.ini` which is distributed with Crab:

```
# Configure how to connect to the server where the Crab daemon is running.
[server]
host = localhost
port = 8000
# Timeout for communication with the server (seconds).
# timeout = 30
# Number of times to attempt to connect to server.
# max_tries = 1
# Delay between connection attempts (seconds).
# retry_delay = 5

# Configuration for this client.
[client]
# Override hostname provided by OS if required.
# hostname = host

# Override username provided by OS if required.
# username = user
```

(continues on next page)

(continued from previous page)

```
# Attempt to use fully qualified domain name for client (if not specified).
# use_fqdn = false

# Configure crabsh behavior.
[crabsh]
# Choose whether to honor the inhibit message on job start.
# allow_inhibit = true
# Write fallback messages to standard output only on failure.
# quiet = false
```

1.4 The Web Interface

The Crab dashboard allows the status of the jobs to be monitored. On this page, the job status column will change color to indicate the status, and it will flash while the job is running. Clicking on the status will lead to the most recent output recorded for the job.

The host and user columns contain links leading to a summary page of the cron jobs for a given user or host. From this page, the links below each table can be used to show deleted jobs, and to display the raw crontab as last imported.

Clicking on a job ID or command link leads to the job information page, giving a summary of the job's parameters and a table of the most recent events. Clicking the status of any job finish event leads to the corresponding output.

1.4.1 Job Configuration

Below the summary on the job information page, there is a link allowing the job's configuration to be edited. If a job is deleted, then its configuration is considered to be orphaned. In this case, when configuring a job for which no configuration exists, the system will offer a list of orphaned configurations for re-linking. This should be used when the job is actually the continuation of a previous job. Note that notifications which are attached to specific jobs are linked via the configuration. Therefore re-linking the configuration will re-attach all associated notifications.

However this problem can generally be avoided by giving the jobs suitable names via the `CRABID` parameter. Crab will then be able to recognize jobs by name even if the command string changes.

The grace period specifies how close to the scheduled time the job must start in order not to be considered missed. The time-out is the maximum expected duration of the job. If it runs for longer than this, it will be marked as stopped with timed-out (error) status. Note that the job may actually still be running when this status is displayed. If the job is restarted, or reported as already running, during the time-out period, then the time-out is reset. If either of these timing parameters are left blank then the default values of 2 minutes grace period and 5 minutes time-out will be used.

Regular expression patterns used to determine success or failure and to identify warnings can be given. These patterns are compared to the standard output and standard error of the job when it finishes, but do not override a more severe status. For example if a job is reported as finishing with failure, then it will be logged as such even if the success or warning patterns match. If none of the patterns match then the status is logged as it was reported, unless a success pattern was defined. If the success pattern does not match then the status will be failure if the was no failure pattern or unknown if there was a failure pattern which did not match.

The "Inhibit execution" checkbox can be use to temporarily request that a job not be run. This setting is stored in the Crab server and passed to the client when it reports that a job is being started. Note that there is no guarantee that the job will not be run while this option is selected: the client could fail to connect to the server before starting the job, or it could choose to ignore the inhibit setting. The `crabsh` wrapper shell reads a configuration parameter `allow_inhibit` from the `crabsh` section of the `cran.ini` file to determine whether inhibit requests should be honored. (The default value is true, i.e. it will not run the job if it receives the inhibit flag in response to its job starting message.)

The job configuration page also allows jobs to be marked as deleted. Normally this would be done by importing a new crontab without that job in it, but having this available on the web interface is useful in situations such as the host being inaccessible. Note that if a start or finish event is received from the job, but the Crab server is still able to identify it, then the job should be automatically marked as not deleted.

There is also the option to alter the job identifier. However care must be taken to also update it in the job itself, for example via the `CRABID` parameter in the crontab. If the identifier is changed via the web server but not in the job, then the Crab server will identify it as a new job the next time it receives a start or finish report from it.

1.4.2 Notifications

Crab includes a configurable notifications system, which currently supports sending notification messages by email. Notifications can either be attached to a specific job, or configured by host name and/or by user name.

A link below the summary on the job information page allows notifications to be attached to that job. Check-boxes for each notification can be used to select which severity of events should be featured, and whether the job output should be included. The schedule box should contain a cron-style schedule specification (e.g. `0 12 * * *`), and if left blank, will default to the value given in the `crabd.ini` file, allowing all notification schedules to be managed in one place. Notifications will only be sent if there are relevant events, so it is possible to request almost-immediate error warnings by including a schedule of `* * * * *` and selecting errors only.

The add and delete links can be used to add and remove notifications, but the changes are not saved until the `Configure` button is clicked.

The drop-down menu which appears when the mouse is positioned over the Crab heading at the top of each page includes a link to the main notifications page. This allows notifications to be configured by host name and/or by user name. Notifications will include any jobs where the host and user match the specified values, but if either is left blank, then it will match all entries.

1.4.3 Additional Job Actions

Depending on the state of a job, additional links may appear below the summary on the job information page. These are:

- “Clear status”: this appears when the job is in a warning or error state. Selecting this option sets the job state to “Cleared”, which you can use to acknowledge the problem. The job’s status will then be shown in green on the dashboard.
- “Resume inhibited job”: this appears when the inhibit setting has been selected on the job configuration page. The link provides a convenient means of removing the inhibit setting.

1.5 Crab Client API

1.5.1 crab.client module

class `crab.client.`**`CrabClient`**(*command=None*, *crabid=None*)

Crab client class, used for interaction with the server.

`__init__`(*command=None*, *crabid=None*)

Constructor for CrabClient.

This causes the client to configure itself, by looking for the crab.ini file. If the environment variables CRABHOST or CRABPORT exist, these override settings from the configuration files.

If the client has been started to report on the status of a job, then the command must be supplied, and the crabid should be given if known.

start()
Notify the server that the job is starting.

Return the decoded server response, which may include an inhibit dictionary item.

finish(*status=2*, *stdoutdata=''*, *stderrdata=''*)
Notify the server that the job is finishing.

send_crontab(*crontab*, *timezone=None*)
Takes the crontab as a string, breaks it into lines, and transmits it to the server.

Returns a list of warnings.

fetch_crontab(*raw=False*)
Retrieves crontab lines from the server, and returns them as a single string.

get_info()

__module__ = 'crab.client'

1.5.2 crab module

exception crab.**CrabError**
Base class for exceptions raised internally by crab.

Library functions should re-raise expected exceptions as a CrabError to allow them to be trapped conveniently without accidentally trapping other errors.

class crab.**CrabStatus**
Helper class for status codes.

The crab libraries should refer to codes by the symbolic names given in this class.

VALUES is the set of status codes which should be accepted from a client. Other codes are for internal use, such as those generated by the monitor.

SUCCESS = 0

FAIL = 1

UNKNOWN = 2

COULDNOTSTART = 3

ALREADYRUNNING = 4

WARNING = 5

INHIBITED = 6

WATCHDOG = 7

VALUES = set([0, 1, 2, 3, 4, 5, 6, 7])

LATE = -1

MISSED = -2

TIMEOUT = -3

CLEARED = -4

INTERNAL_VALUES = set([-4, -3, -2, -1])

static get_name(*status*)
Returns a readable name for the given status code.

static is_trivial(*status*)
Determines whether a status code is trivial and should mostly be ignored.

static is_ok(*status*)
Returns true if the code does not indicate any kind of problem.

static is_warning(*status*)
True if the given status is some kind of warning.

static is_error(*status*)
True if the given status is an error, i.e. not OK and not a warning.

class `crab.`**CrabEvent**
Helper class for crab events.

Currently just provides symbolic names for the event types.

START = 1

ALARM = 2

FINISH = 3

static get_name(*event*)
Returns a readable name for the given event type code.

1.6 Internal API

1.6.1 Crab Server

crab.server

class `crab.server.`**CrabServer**(*bus*)
Crab server class, used for interaction with the client.

crontab(*host*, *user*, *raw=False*)
CherryPy handler for the crontab action.

Allows the client to PUT a new crontab, or use a GET request to see a crontab-style representation of the job information held in the the storage backend.

start(*host*, *user*, *crabid=None*)
CherryPy handler allowing clients to report jobs starting.

finish(*host*, *user*, *crabid=None*)
CherryPy handler allowing clients to report jobs finishing.

crab.server.config

`crab.server.config.`**read_crabd_config**()
Determine Crab server configuration.

This returns a CherryPy configuration dictionary, with values read from the config files and environment variables.

`crab.server.config.`**construct_log_handler**(*filename*, *log_config*)

`crab.server.config.`**`construct_store`**(*storeconfig*, *outputstore=None*)
Constructs a storage backend from the given dictionary.

crab.server.io

`crab.server.io.`**`import_config`**(*store*, *file_*)
Read job and configuration information from a JSON file.

`crab.server.io.`**`export_config`**(*store*, *file_*)
Write job and configuration information to a JSON file.

1.6.2 Notifications

crab.notify

`class` `crab.notify.`**`CrabNotifyJob`**(*n*, *start*, *end*)

`end`
Alias for field number 2

`n`
Alias for field number 0

`start`
Alias for field number 1

`class` `crab.notify.`**`CrabNotify`**(*config*, *store*)
Class for sending notification messages.

crab.notify.email

`class` `crab.notify.email.`**`CrabNotifyEmail`**(*crab_home*, *base_url*, *config_email*)
Class to send notification messages by email.

crab.report

`class` `crab.report.`**`CrabReportJob`**(*id_*, *start*, *end*, *skip_ok*, *skip_warning*, *skip_error*, *include_output*)

`end`
Alias for field number 2

`id_`
Alias for field number 0

`include_output`
Alias for field number 6

`skip_error`
Alias for field number 5

`skip_ok`
Alias for field number 3

skip_warning
Alias for field number 4

start
Alias for field number 1

class crab.report.**CrabReport**(*num*, *error*, *warning*, *ok*, *info*, *events*, *stdout*, *stderr*)

error
Alias for field number 1

events
Alias for field number 5

info
Alias for field number 4

num
Alias for field number 0

ok
Alias for field number 3

stderr
Alias for field number 7

stdout
Alias for field number 6

warning
Alias for field number 2

class crab.report.**CrabReportGenerator**(*store*, ***kwargs*)
Class for generating reports on the operation of cron jobs.

This class maintains a cache of job information and events to allow it to handle multiple report requests in an efficient manner. This depends on a single configuration, so methods for adjusting the filtering are not provided.

crab.report.text

crab.report.text.**report_to_text**(*report*, *event_list=True*)

crab.report.html

crab.report.html.**report_to_html**(*report*, *home*, *base_url*)

1.6.3 Services

crab.service

class crab.service.**CrabMinutely**
A thread which will call its run_minutely method for each minute which passes.

The problem which this class seeks to address is that other methods of pausing a program (eg. threading.Timer, time.sleep) can't guarantee not to pause for longer than expected. Therefore in the context of cron jobs, it might be possible to miss a cron scheduling point.

run()
Thread run function.

This calls _check_minute on regular intervals.

run_minutely(*datetime_*)
This is the method which will be called each minute. It should be overridden by subclasses.

Will be called with a datetime object for the relevant minute with the seconds and microseconds set to zero.

`crab.service.`**minute_equal**(*a*, *b*)
Determine whether one time is in the same minute as another.

`crab.service.`**minute_before**(*a*, *b*)
Determine whether one time is in a minute before another.

crab.service.clean

class `crab.service.clean.`**CrabCleanService**(*config*, *store*)
Service to clean the store by removing old events.

run_minutely(*datetime_*)
Performs cleaning if scheduled for the given minute.

crab.service.monitor

exception `crab.service.monitor.`**JobDeleted**
Exception raised by _initialize_job if the job can not be found.

class `crab.service.monitor.`**CrabMonitor**(*store*, *passive=False*)
A class implementing the crab monitor thread.

run()
Monitor thread main run function.

When the thread is started, this function will run. It begins by fetching a list of jobs and using them to populate its data structures. When this is complete, the Event status_ready is fired.

It then goes into a loop, and every few seconds it checks for new events, processing any which are found. The new_event Condition is fired if there were any new events.

We call _check_minute from CrabMinutely to check whether the minute has changed since the last time round the loop.

run_minutely(*datetime_*)
Every minute the job scheduling is checked.

At this stage we also check for new / deleted / updated jobs.

get_job_status(*id_=None*)
Fetches the status of all jobs as a dict.

For efficiency this returns a reference to our job status dict. Callers should not modify it. If a job ID is specified, the status entry for that job is returned, or a dummy entry if it is not in the status dict.

wait_for_event_since(*startid*, *alarmid*, *finishid*, *timeout=120*)
Function which waits for new events.

It does this by comparing the IDs with our maximum values seen so far. If no new events have already be seen, wait for the new_event Condition to fire.

A random time up to 20s is added to the timeout to stagger requests.

crab.service.notify

class `crab.service.notify.`**`CrabNotifyService`**(*config*, *store*, *notify*)

Service to send notifications as required.

Currently only a single daily schedule is implemented.

`run_minutely`(*datetime_*)

Issues notifications if any are scheduled for the given minute.

1.6.4 Storage

crab.store

class `crab.store.`**`CrabStore`**

`get_jobs`(*host=None*, *user=None*, ***kwargs*)

Fetches a list of all of the cron jobs, excluding deleted jobs by default.

Optionally filters by host or username if these parameters are supplied. Other keyword arguments (e.g. include_deleted, crabid, command, without_crabid) are passed to the _get_jobs method.

`delete_job`(*id_*)

Mark a job as deleted.

`undelete_job`(*id_*)

Remove deletion mark from a job.

`update_job`(*id_*, ***kwargs*)

Updates job information.

Keyword arguments are passed on to the private _update_job method, and can include: crabid, command, time, timezone.

`log_start`(*host*, *user*, *crabid*, *command*)

Inserts a job start record into the database.

Returns a dictionary including a boolean value indicating whether the job inhibit setting is active or not.

`log_finish`(*host*, *user*, *crabid*, *command*, *status*, *stdout=None*, *stderr=None*)

Inserts a job finish record into the database.

The output will be passed to the write_job_output method, unless both stdout and stderr are empty.

`get_job_config`(*id_*)

Retrieve configuration data for a job by ID number.

`write_job_output`(*finishid*, *host*, *user*, *id_*, *crabid*, *stdout*, *stderr*)

Writes the job output to the store.

This will use the outputstore's corresponding method if it is defined, otherwise it writes to this store.

`get_job_output`(*finishid*, *host*, *user*, *id_*, *crabid*)

Fetches the standard output and standard error for the given finish ID.

The result is returned as a two element list. Returns a pair of empty strings if no output is found.

This will use the outputstore's corresponding method if it is defined, otherwise it reads from this store.

get_crontab(*host*, *user*)
Fetches the job entries for a particular host and user and builds a crontab style representation.

Please see crab.util.crontab.write_crontab for more details of how the crontab is constructed.

save_crontab(*host*, *user*, *crontab*, *timezone=None*, *allow_filter=True*)
Takes a list of crontab lines and uses them to update the job records.

It looks for the CRABID and CRON_TZ variables, but otherwise ignores everything except command lines. It also checks for commands starting with a CRABID= definition, but otherwise inserts them into the database as is.

If "allow_filter" is True (as is the default) then cron jobs are skipped if they have a specified user name or client host name which does not match the given host or user name.

Returns a list of warning strings.

check_job(**args*, ***kwargs*)
Ensure that a job exists in the store.

Acquires the lock and then calls the private _check_job method.

write_raw_crontab(*host*, *user*, *crontab*)

get_raw_crontab(*host*, *user*)

crab.store.db

class crab.store.db.**CrabDBLock**(*conn*, *error_class*, *cursor_args={}*, *ping=False*)

class crab.store.db.**CrabStoreDB**(*lock*, *outputstore=None*)
Crab storage backend using a database.

Currently written for SQLite but since it uses the Python DB API it should be possible to generalize it by altering the queries based on the database type where necessary.

log_alarm(*id_*, *status*)
Inserts an alarm regarding a job into the database.

This is for alarms generated interally by crab, for example from the monitor thread. Such alarms are currently stored in an separate table and do not have any associated output records.

get_job_info(*id_*)
Retrieve information about a job by ID number.

write_job_config(*id_*, *graceperiod=None*, *timeout=None*, *success_pattern=None*, *warning_pattern=None*, *fail_pattern=None*, *note=None*, *inhibit=False*)
Writes configuration data for a job by ID number.

Returns the configuration ID number.

disable_inhibit(*id_*)
Disable the inhibit setting for a job.

This is a convenience routine to simply disable the inhibit job configuration parameter without having to read and write the rest of the configuration.

get_orphan_configs()
Make a list of orphaned job configuration records.

relink_job_config(*configid*, *id_*)

get_job_finishes (*id_*, *limit=100*, *finishid=None*, *before=None*, *after=None*, *include_alreadyrunning=False*)
Retrieves a list of recent job finish events for the given job, most recent first.

Can optionally find a particular finish, or finishes before or after a certain finish. In the case of finishes after a certain finish, the most recent event will be last.

ALREADYRUNNING events are only reported if the include_alreadyrunning argument is set.

get_job_events (*id_*, *limit=100*, *start=None*, *end=None*)
Fetches a combined list of events relating to the specified job.

Return events, newest first (with finishes first for the same datetime). This ordering allows us to apply the SQL limit on number of result rows to find the most recent events. It gives the correct ordering for the job info page.

get_events_since (*startid*, *alarmid*, *finishid*)
Extract minimal summary information for events on all jobs since the given IDs, oldest first.

get_fail_events (*limit=40*)
Retrieves the most recent failures for all events, combining the finish and alarm tables.

This method has to include a list of status codes to exclude since the filtering is done in the SQL. The codes skipped are CLEARED, LATE, SUCCESS, ALREADYRUNNING and INHIBITED.

delete_old_events (*datetime_*)
Delete events older than the given datetime.

get_notifications ()
Fetches a list of notifications, combining those defined by a config ID with those defined by user and/or host.

get_job_notifications (*configid*)
Fetches all of the notifications configured for the given configid.

get_match_notifications (*host=None*, *user=None*)
Fetches matching notifications which are not tied to a configuration entry.

write_notification (*notifyid*, *configid*, *host*, *user*, *method*, *address*, *time*, *timezone*, *skip_ok*, *skip_warning*, *skip_error*, *include_output*)
Adds or updates a notification record in the database.

delete_notification (*notifyid*)
Removes a notification from the database.

crab.store.file

class crab.store.file.**CrabStoreFile** (*dir*)
Store class for cron job output.

This backend currently implements only the write_job_output and get_job_output methods, to allow it to be used as an "outputstore" along with CrabStoreDB.

write_job_output (*finishid*, *host*, *user*, *id_*, *crabid*, *stdout*, *stderr*)
Write the cron job output to a file.

The only parameter required to uniquely identify the event associated with this output is the "finishid", but the host, user and job identifiers are also provided to allow hierarchical storage.

Only writes a stdout file (extension set by self.outext, by default txt), and a stderr file (extension self.errext, default err) if they are not empty.

get_job_output (*finishid*, *host*, *user*, *id_*, *crabid*)
Find the file containing the cron job output and read it.

As for write_job_output, only the "finishid" is logically required, but this method makes use of the host, user and job identifiers to read from a directory hierarchy.

Requires there to be an stdout file but allows the stderr file to be absent.

write_raw_crontab (*host*, *user*, *crontab*)
Writes the given crontab to a file.

get_raw_crontab (*host*, *user*)
Reads the given user's crontab from a file.

crab.store.mysql

crab.store.sqlite

class crab.store.sqlite.**CrabStoreSQLite** (*filename*, *outputstore=None*)

1.6.5 Utilities

crab.util.bus

crab.util.bus.**priority** (*level*)
Decorator to set the priority attribute of a function.

class crab.util.bus.**CrabStoreListener** (*bus*)
Base class for plugins which require a store.

Listens on the "crab-store" channel, setting the instance value "store" to the store object received.

subscribe ()

class crab.util.bus.**CrabPlugin** (*bus*, *name*, *class_*, ***kwargs*)
Class to launch Crab services as CherryPy plugins.

This class subscribes to the "start" channel. When it recieves a message, it constructs an instance of the service class and publishes it on the "crab-service" channel.

subscribe ()

start ()

notify (*notify*)

crab.util.compat

crab.util.compat.**restore_signals** ()
Restore signals which Python otherwise ignores.

For more information about this issue, please see: http://bugs.python.org/issue1652

crab.util.compat.**subprocess_communicate** (*p*, *input=None*, *timeout=None*)

crab.util.compat.**subprocess_call** (*args*, *timeout=None*, ***kwargs*)

exception crab.util.compat.**TimeoutExpired**

crab.util.filter

class crab.util.filter.**CrabEventFilter**(*store*, *timezone=None*)

Class implementing an event filtering action.

default_timezone = <UTC>

set_timezone(*timezone*)

Sets the timezone used by the filter.

classmethod set_default_timezone(*timezone*)

Sets the default timezone for all filters.

in_timezone(*datetime_*)

Convert the datetime string as output by the database to a string in the specified timezone.

Includes the zone code to indicate that the conversion has been performed.

crab.util.guesstimezone

crab.util.guesstimezone.**guess_timezone**()

Function to try to determine the operating system's timezone setting.

Currently this checks for a TZ environment variable. Otherwise it checks if /etc/localtime is a link or tries to find the file in /usr/share/zoneinfo which matches. It uses pytz to get a list of common timezones to try.

crab.util.pid

crab.util.pid.**pidfile_write**(*pidfile*, *pid*)

Attempt to write a key for the given process into the file specified.

crab.util.pid.**pidfile_running**(*pidfile*)

Read the pidfile specified and check if the process is running.

If the pidfile was found then its timestamps are updated (using *os.utime*). This is to try to avoid the pidfile being automatically removed from temporary directories if a job runs for a very long time.

crab.util.pid.**pidfile_delete**(*pidfile*)

Attempt to delete the specified pidfile.

crab.util.schedule

class crab.util.schedule.**CrabSchedule**(*specifier*, *timezone*)

Class handling the schedule of a cron job.

match(*datetime_*)

Determines whether the given datetime matches the scheduling rules stored in the class instance.

The datetime is converted to the stored timezone, and then the components of the time are checked against the matchers in the CronTab superclass.

next_datetime(*datetime_*)

return a datetime rather than number of seconds.

previous_datetime(*datetime_*)

return a datetime rather than number of seconds.

crab.util.statuspattern

crab.util.statuspattern.**check_status_patterns**(*status*, *config*, *output*)
Function to update a job status based on the patterns.

Compares the given output with the patterns in the job configuration, and returns the updated status.

crab.util.string

crab.util.string.**remove_quotes**(*value*)
If the given string starts and ends with matching quote marks, remove them from the returned value.

```
>>> remove_quotes('alpha')
'alpha'
>>> remove_quotes('"bravo"')
'bravo'
>>> remove_quotes("'charlie'")
'charlie'
```

If the quotes are mismatched it should not remove them.

```
>>> remove_quotes('"delta')
'"delta'
>>> remove_quotes("echo'")
"echo'"
```

crab.util.string.**quote_multiword**(*value*)
If the given string contains space characters, return it surrounded by double quotes, otherwise return the original string.

```
>>> quote_multiword('alpha')
'alpha'
>>> quote_multiword('bravo charlie')
'"bravo charlie"'
```

crab.util.string.**split_quoted_word**(*value*)
Splits the given string on the first word boundary, unless it starts with a quote.

If quotes are present it splits at the first matching quote. Eg.:

```
>>> split_quoted_word('alpha bravo charlie delta echo')
['alpha', 'bravo charlie delta echo']
>>> split_quoted_word('"alpha bravo" charlie delta echo')
('alpha bravo', 'charlie delta echo')
```

Does not handle escaped quotes within the string.

crab.util.string.**split_crab_vars**(*command*)
Looks for Crab environment variables at the start of a command.

Bourne-style shells allow environment variables to be specified at the start of a command. This function takes a string corresponding to a command line to be executed by a shell, and looks for environment variables in the 'CRAB namespace', i.e. those consisting of CRAB followed by a number of upper case characters.

```
>>> split_crab_vars('some command')
('some command', {})
>>> split_crab_vars('CRABALPHA=bravo another command')
('another command', {'CRABALPHA': 'bravo'})
```

Returns: a tuple consisting of the remainder of the command and a dictionary of Crab's environment variables.

`crab.util.string.`**`alphanum`**`(`*value*`)`

Removes all non-alphanumeric characters from the string, replacing them with underscores.

```
>>> alphanum('a3.s_?x9!t')
'a3_s__x9_t'
```

`crab.util.string.`**`mergelines`**`(`*text*`)`

Merges the lines of a string by removing newline characters.

```
>>> mergelines('alpha' + chr(10) + 'bravo')
'alphabravo'
```

`crab.util.string.`**`true_string`**`(`*text*`)`

Tests whether the string represents a true value.

The following strings are false:

```
>>> [true_string(x) for x in ['0', 'no', 'false', 'off']]
[False, False, False, False]
```

And anything else is taken to be true, for example:

```
>>> [true_string(x) for x in ['1', 'yes', 'true', 'on']]
[True, True, True, True]
```

The results are case insensitive.

```
>>> [true_string(x) for x in ['no', 'NO', 'True', 'Off']]
[False, False, True, False]
```

crab.util.web

`crab.util.web.`**`abbr`**`(`*text*, *limit=60*, *tolerance=10*`)`

Returns an abbreviated and HTML-escaped version of the specified text.

The text is trimmed to the given length limit, but if a space is found within the preceeding 'tolerance' number of characters, then it is trimmed there. The result is an HTML span element with the full text as the title, unless it was not necessary to trim it.

```
>>> abbr('alpha bravo', 15, 5)
'alpha bravo'
>>> abbr('alpha bravo charlie', 15, 5)
'<span title="alpha bravo charlie">alpha bravo&hellip;</span>'
```

1.6.6 Web

crab.web.web

`crab.web.web.`**`empty_to_none`**`(`*value*`)`

`class` `crab.web.web.`**`CrabWebBase`**`(`*bus*`)`

`subscribe``()`

class crab.web.web.**CrabWebQuery**

CherryPy handler class for the JSON query part of the crab web interface.

jobstatus(*startid*, *alarmid*, *finishid*)

CherryPy handler returning the job status dict fetched from the monitor thread.

jobinfo(*id_*)

CherryPy handler returning the job information for the given job.

class crab.web.web.**CrabWeb**(*crab_home*, *options*)

CherryPy handler for the HTML part of the crab web interface.

subscribe()

index()

Displays the main crab dashboard.

job(*id_*, *command=None*, *finishid=None*, *barerows=None*, *unfiltered=None*, *limit=None*, *enddate=None*, *submit_config=None*, *submit_relink=None*, *submit_confirm=None*, *submit_cancel=None*, *orphan=None*, *graceperiod=None*, *timeout=None*, *success_pattern=None*, *warning_pattern=None*, *fail_pattern=None*, *note=None*, *inhibit=None*, *crabid=None*, *submit_notify=None*, ***kwargs*)

Displays information about a current job.

Currently also supports showing the job output. If command='output' but the finishid is not provided, then it will find the most recent output for the given job.

user(*user*)

Displays crontabs belonging to a particular user.

host(*host*)

Displays crontabs belonging to a particular user.

notify(*submit_notify=None*, ***kwargs*)

Allows match-based notifications to be viewed and configured.

dynres(*name*)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index