jarjar Documentation

Release 3.0

The Austerweil Lab at UW-Madison

Jun 17, 2018

Contents

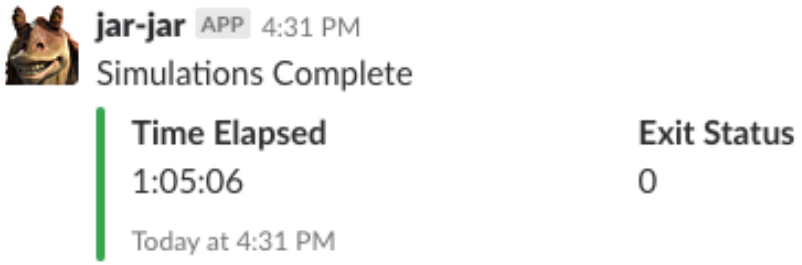
Jarjar is a python utility that makes it easy to send slack notifications to your teams. You can import it as a python module or use our command line tool.

CHAPTER 1

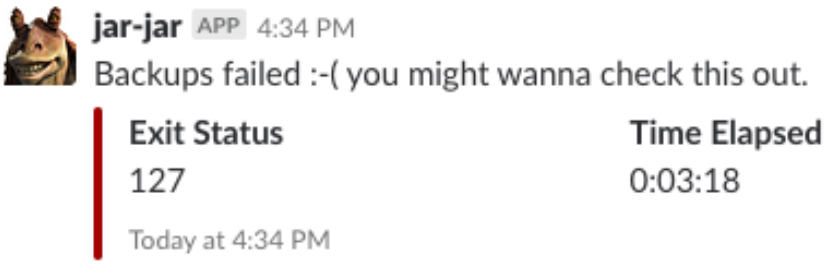
What Can Jarjar Do For Me?

Jarjar was developed at the Austerweil Lab at UW-Madison as a tool for scientists. We use it for all sorts of things, such as:

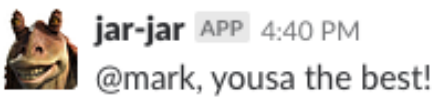
1. Sending a message so that we know when long-running processes have finished.



2. Sending notices when scheduled tasks have failed.



3. Sending out daily positive vibes.



CHAPTER 2

Quickstart

2.1 Install

Installation is simple!

```
pip install jarjar
```

My guess is that you'll want to create jarjar's config file, `~/.jarjar`. This tells jarjar what you'd like to use as a default for your slack team's webhook, the channel to post to, and the message it sends. Don't worry, you can over-ride these anytime.

Edit this snippet and add it to `~/.jarjar`:

```
channel='@username'
message='Custom message'
webhook='https://hooks.slack.com/services/your/teams/webhook'
```

If you don't know your team's webhook, you might have to make one.

2.2 Python API

Use the *jarjar python api* like:

```
from jarjar import jarjar

# initialize a jarjar object
jj = jarjar() # defaults from .jarjar
jj = jarjar(channel='#channel', webhook='slack-webhook-url')
jj = jarjar(webhook='slack-webhook-url')

# send a text message
jj.text('Hi!')
```

(continues on next page)

(continued from previous page)

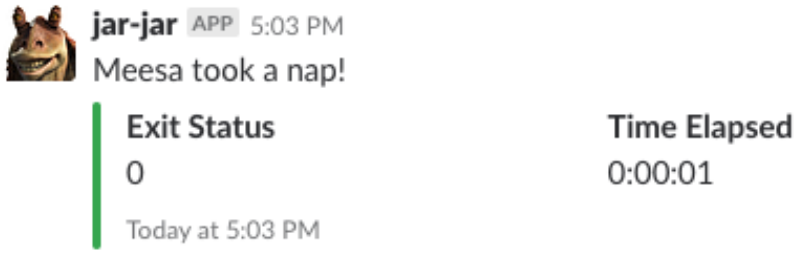
```
jj.text('Hi!', channel=["@jeffzemla", "#channel"])

# send an attachment
jj.attach({'meesa': 'jarjar binks'}), text='Hello!')
```

2.3 Command Line Tool

We also made a *command line tool* for use outside of python scripts. The command line tool adds functionality to execute processes and send messages when they are complete.

```
jarjar sleep 1 -m 'Meesa took a nap!'
```

And then in your slack team:



Custom attachments are not supported in the CLT at this time, but everything else is:

```
jarjar -m 'Meesa jarjar binks!'
jarjar -m 'Hi, everyone!!' --webhook '<your-url>' -c '#general'
```

CHAPTER 3

Detailed Documention

3.1 Installing jarjar

3.1.1 Just use pip.

We're on pypi.

```
pip install jarjar
```

3.1.2 Config File

You can use jarjar without a config file, but you'll need to tell it your slack webhook and channel each time.

You don't want to live that way.

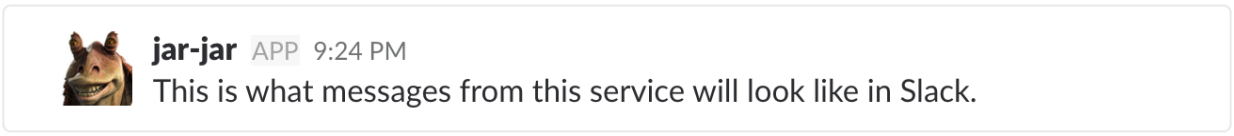
Jarjar looks for a special file in your user home `~/.jarjar` for default webhook, channel, and/or message. You can over-ride anything in there pretty much any time you want.

```
channel='@username'
message='Custom message'
webhook='https://hooks.slack.com/services/your/teams/webhook'
```

3.1.3 Configuring Slack

For this to work in the first place, you need to set up a slack webhook for your team.

While you're doing that, you can also specify a custom name and custom icon. We named our webhook robot `jar-jar`, and we used this icon, so messages look like this:



3.2 Using the jarjar command line tool

The CLT provides basic posting functionality like in the python API but it also provides a useful task execution facility.

3.2.1 Posting to your team

The jarjar CLT offers all the functionality of the python API, except for posting attachments (sorry). Posting messages is super easy though!

You can use your defaults from .jarjar

```
jarjar --message 'Meesa jarjar binks!'
```

Or not.

```
jarjar -m 'Hi, everyone!!' --webhook '<your-url>' --channel '#general'
```

3.2.2 Running processes with jarjar

We use jarjar to run a lot of longer processes when we don't want to keep our terminal sessions around. You can use jarjar for this sort of thing.

```
jarjar sleep 3600
```

Generally speaking it is safer to wrap your program in quotes so that its clear which arguments are meant for jarjar and which are meant for your task.

```
jarjar 'python3 simulations.py --niters 100 --out results.csv'
```

Now you can head out for some lunch. Here's what's going on under the hood:

1. **Start up a screen**. The screen can have a custom name (using the `-S` or `--screen_name` flags) but if you don't provide one it'll be named using the program you provide. Above, the screen was named `sleep_3600`.
2. **Run your process in that screen**. If you want you can attach to the screen (using the `-a`, `-r`, or `--attach` flags) and see the magic happen.
3. **Send a message when the process is complete**. If you specified a message (using the `-m` or `--message` flags) jarjar will send it. Jarjar will then kill your screen if:
   - You don't tell it to keep the screen (using the `--no-exit` flag).
   - You didn't attach to it (using the `-a`, `-r`, or `--attach` flags).
   - The program you ran exited with status 0.

Examples

```
# send a custom message
jarjar python run-simulations.py --message 'Simulations Complete!'

# name your screen
jarjar sleep 1 --screen-name 'snooze'

# watch the magic happen
jarjar <program> --attach

# keep the screen around for debugging
jarjar <program> --no-exit

# show jarjar version
jarjar --version

# get help
jarjar --help
```

3.2.3 Argument Reference

- `-h`, `--help`. Show help message.
- `-v`, `--version`. Show jarjar version.
- `-m`, `--message`. Specify message to send. This best done in single-quotes (`jarjar -m 'hi'`) but jarjar rolls with the punches (like `jarjar -m hi`).
- `-w`, `--webhook`. Specify webhook to post to.
- `-c`, `--channel`. Specify channel to post to. Unlike in the python module, only one channel can be supplied at a time. Since `#` is interpreted as a shell comment, you'll want to put this in single quotes (`jarjar -c '#general'`).
- `-a`, `-r`, `--attach`. Attach to the screen once the program has started running. If you didn't provide a program jarjar will think you are weird.
- `-S`, `--screen_name`. Specify the name of the screen created for the program. If you didn't provide a program jarjar will think you are weird.
- `--no-exit`. Don't exit the screen even if the program exited successfully. If you didn't provide a program jarjar will think you are weird.
- `--no-jarjar`. Run a program but don't send a slack message about it. In this case jarjar is just acting as a screen generator. If you didn't provide a program jarjar will think you are weird.

3.3 API Documentation

class jarjar.**jarjar**(*message=None*, *channel=None*, *webhook=None*)

A jarjar slack messenger.

This is largely a wrapper around functionality in `requests.post()` with facilities to store and set default values for the desired message to send, channel to post within, and slack team webhook.

Inference for these values proceeds as follows.

1. Any argument provided to *text()* or *attach()* supersedes all defaults.
2. Defaults can be provided at initialization or via a config file (`~/.jarjar`), which looks like:

```
channel="@username"
message="Custom message"
webhook="https://hooks.slack.com/services/your/teams/webhook"
```

3. Arguments provided at initialization supersede those in `~/.jarjar`. If the channel or webhook arguments are never provided, an error is raised. If the channel and webhook are provided but not a message or attachment, jarjar will make something up.

Parameters

message [str] Optional. Default message to send.

channel [str, list] Optional. Name of the default channel to post within.

webhook [str] Optional. Webhook URL for the default slack team.

Methods

attach(attach, channel=None, webhook=None, message=None)	Send an attachment. User may also include a text message.
text(message, channel=None, webhook=None, attach=None)	Send a text message. User may also include an attachment.
set_webhook(webhook)	Set jarjar's default webhook.
set_channel(channel)	Set jarjar's default channel.
set_message(message)	Set jarjar's default message.

attach (*attach=None*, ***kwargs*)
Send an attachment.

This method is largely identical to *text()*, only differing in the first argument (`attach`), which is expected to be a dictionary.

Parameters

attach [dict] Attachment data. Optional *but weird if you don't provide one*. All values are converted to string for the slack payload so don't sweat it.

message [str] Text to send. Optional. If attach is None and there is no default *and* you don't provide one here, jarjar just wings it.

channel [str, list] Optional. Name of the channel to post within. Can also be a list of channel names; jarjar will post to each.

webhook [str] Optional. Webhook URL for the slack team.

Returns

response [requests.models.Response] Requests response object for the POST request to slack.

set_channel (*channel*)
Set default channel.

Parameters

channel [str] Name of the channel to post within.

set_message(*message*)
Set default message.

Parameters

message [str] Default message to send.

set_webhook(*webhook*)
Set default webhook.

Parameters

webhook [str] Webhook URL for the slack team.

text(*message=None*, ***kwargs*)
Send a text message.

This method is largely identical to `attach()`, only differing in the first argument (`message`), which is expected to be a string.

Parameters

message [str] Text to send. Optional *but weird if you don't provide one*. If attach is None and there is no default *and* you don't provide one here, jarjar just wings it.

attach [dict] Attachment data. Optional. All values are converted to string for the slack payload so don't sweat it.

channel [str, list] Optional. Name of the channel to post within. Can also be a list of channel names; jarjar will post to each.

webhook [str] Optional. Webhook URL for the slack team.

Returns

response [requests.models.Response] Requests response object for the POST request to slack.

Python Module Index

j

Index