bugzscout Documentation

Release 0.0.1

Thomas Van Doren

July 31, 2013

CONTENTS

Python interface for the FogBugz BugzScout API.

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 Installation

```
pip install bugzscout

# Or, with easy_install:
easy_install bugzscout

# Or, from source:
python setup.py install
```

1.2 Getting Started

```
>>> import bugzscout
>>> b = bugzscout.BugzScout('http://fogbugz/scoutSubmit.asp',
                            'fb-user',
                            'the-project',
                            'the-area')
>>> b.submit_error('An error occurred of type blah', extra='Extra info')
```

1.3 Command Line Interface

There is a command line interface for submitting errors. To simplify submitting multiple errors, the FogBugz configuration can be set in the environment.

```
# (Optional) Setup the environment.
export BUGZSCOUT_URL=http://fogbugz/scoutSubmit.asp
export BUGZSCOUT_USER=errors
export BUGZSCOUT_PROJECT='My Project'
export BUGZSCOUT_AREA=Errors

# Submit a new error.
bugzscout --extra 'Extra data for the case...' 'The description of the error.'
```

1.4 Publishing Errors Asynchronously with Celery

The Celery extension can be used to asynchronously publish errors. This is the recommended pattern for using bugzscout in production environments.

```
# Import celery extension.
import bugzscout.ext.celery_app

# Submit errors asynchronously.
bugzscout.ext.celery_app.submit_error.delay(
  'The description here...',
  extra='The extra information here...')
```

The Celery worker can use the same celery app for consuming messages.

```
celery worker --app=bugzscout.ext.celery_app
```

A `celeryconfig.py` file on the PYTHONPATH can be used to configure the celery instance. For example:

```
export CELERY_CONFIG_MODULE=celeryconfig
celery worker --app=bugzscout.ext.celery_app
```

CHAPTER

TWO

REFERENCE

2.1 BugzScout

class bugzscout.**BugzScout**(*url*, *user*, *project*, *area*)

Submit errors or issues to FogBugz via BugzScout.

__init__(*url*, *user*, *project*, *area*)

Initialize a new instance of bugzscout client.

Parameters

- **url** – string URL for bugzscout
- **user** – string fogbugz user to designate when submitting via bugzscout
- **project** – string fogbugz project to designate for cases
- **area** – string fogbugz area to designate for cases

__repr__()

String representation of this instance.

__weakref__

list of weak references to the object (if defined)

submit_error(*description*, *extra=None*, *default_message=None*)

Send an error to bugzscout.

Sends a request to the fogbugz URL for this instance. If a case exists with the **same** description, a new occurrence will be added to that case. It is advisable to remove personal info from the description for that reason. Account ids, emails, request ids, etc, will make the occurrence counting builtin to bugzscout less useful. Those values should go in the extra parameter, though, so the developer investigating the case has access to them.

When extra is not specified, bugzscout will increase the number of occurrences for the case with the given description, but it will not include an entry for it (unless it is a new case).

Parameters

- **description** – string description for error
- **extra** – string details for error
- **default_message** – string default message to return in responses

2.2 Celery Extension

Asynchronously submit errors to FogBugz via BugzScout.

bugzscout.ext.celery_app.**submit_error**(*url*, *user*, *project*, *area*, *description*, *extra=None*, *default_message=None*)

This is a proxy to an object that has not yet been evaulated.

`Proxy` will evaluate the object each time, while the promise will only evaluate it once.

Celery task for submitting errors asynchronously.

Parameters

- **url** – string URL for bugzscout
- **user** – string fogbugz user to designate when submitting via bugzscout
- **project** – string fogbugz project to designate for cases
- **area** – string fogbugz area to designate for cases
- **description** – string description for error
- **extra** – string details for error
- **default_message** – string default message to return in responses

CHAPTER

THREE

BUGZSCOUT COMMAND LINE INTERFACE

There is a command line interface for submitting errors. It can be used for non-python application (like shell scripts).

```
usage: bugzscout [-h] [-v] [-u URL] [--user USER] [--project PROJECT] [--area AREA]
                 [-e EXTRA] [--default-message DEFAULT_MESSAGE]
                 description

Command line interface for sumbitting cases to FogBugz via BugzScout.

Environment variables can be used to set the FogBugz arguments with:

 * BUGZSCOUT_URL
 * BUGZSCOUT_USER
 * BUGZSCOUT_PROJECT
 * BUGZSCOUT_AREA

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Enable verbose output. (default: False)

FogBugz arguments:
  -u URL, --url URL     URL for bugzscout requests to be sent. Should be something like
                        .../scoutSubmit.asp. (default: None)
  --user USER           User to designate when submitting via bugzscout. (default: None)
  --project PROJECT     Fogbugz project to file cases under. (default: None)
  --area AREA           Fogbugz area to file cases under. (default: None)

error arguments:
  -e EXTRA, --extra EXTRA
                        Extra data to send with error. (default: None)
  --default-message DEFAULT_MESSAGE
                        Set default message if case is new. (default: None)
  description           Description of error. Will be matched against existing cases.
```

To simplify submitting multiple errors, the FogBugz configuration can be set in the environment.

```
# (Optional) Setup the environment.
export BUGZSCOUT_URL=http://fogbugz/scoutSubmit.asp
export BUGZSCOUT_USER=errors
export BUGZSCOUT_PROJECT='My Project'
export BUGZSCOUT_AREA=Errors
```

```
# Submit a new error.
bugzscout --extra 'Extra data for the case...' 'The description of the error.'
```

3.1 Example shell script using cli

Below is an example of a bash function that can wrap other bash calls. It reports an error to fogbugz if the call has a non-zero exit code.

```
#!/bin/bash

# Setup the environment with FogBugz configuration.
export BUGZSCOUT_URL='http://fogbugz/scoutSubmit.asp'
export BUGZSCOUT_USER='error-user'
export BUGZSCOUT_PROJECT='MyShellScript'
export BUGZSCOUT_AREA='Errors'

this_node=$(hostname --fqdn)

function bugzscout_wrap()
{
    # The call is all the arguments to this function.
    local call=$@

    # Call the function.
    bash -c "${call}"
    local exit_code=$?

    # If non-zero exit code, report error.
    if [ "${exit_code}" != "0" ] ; then
        bugzscout "${call} in ${0} failed with exit code ${exit_code} on ${this_node}" \
            || /bin/true
    fi

    return $exit_code
}
```

For example, in a bash shell, the second line would fail and report an error via BugzScout:

```
source path/to/bugzscout_wrap_function.sh
bugzscout_wrap /bin/false
```

CHAPTER

FOUR

EXAMPLES USES OF BUGZSCOUT

4.1 Simple WSGI Server Example

This example describes how to add synchronous BugScout reporting to a WSGI app. This is **not** intended for production use. It's purpose is to show how easy it is to add this kind of reporting.

An example that is suitable for production is the *celery extension example*.

This example uses the Paste python package.

4.1.1 Setup a simple WSGI server

A simple paste WSGI server that can handle HTTP requests is:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""A simple WSGI server."""

from __future__ import print_function, unicode_literals


def app(environ, start_response):
    """Simple WSGI application that returns 200 OK response with 'Hellow
    world!' in the body.

    :param environ: WSGI environ
    :param start_response: function that accepts status string and headers
    """
    start_response('200 OK', [('content-type', 'text/html')])
    return ['Hellow world!']


if __name__ == '__main__':
    import paste.httpserver
    paste.httpserver.serve(app, host='0.0.0.0', port='5000')
```

To run this server:

```
python path/to/simple_wsgi.py
```

Making requests to `http://localhost:5000/` will respond with a 200 OK and 'Hellow world!' in the body.

4.1.2 Adding BugzScout

When errors occur in this code, it would be great to report them to BugzScout. In order to so, wrap the contents of the `app` function in a try/except and call bugzscout in the except block.

This example creates a new function, `bugzscout_app`, that does just that.

```
import bugzscout
import sys
import traceback

# Create an instance of BugzScout to use for all errors thrown in
# bugzscout_app.
b = bugzscout.BugzScout(
    'http://fogbugz/scoutSubmit.asp', 'error-user', 'MyAppProject', 'Errors')


def bugzscout_app(environ, start_response):
    """Simple WSGI application that returns 200 OK response with 'Hellow
    world!' in the body. If an uncaught exception is thrown, it is reported to
    BugzScout.

    :param environ: WSGI environ
    :param start_response: function that accepts status string and headers
    """
    try:
        start_response('200 OK', [('content-type', 'text/html')])
        return ['Hellow world!']
    except Exception as ex:
        # Set the description to a familiar string with the exception
        # message. Add the stack trace to extra.
        b.submit_error('An error occurred in MyApp: {0}'.format(ex.message),
                       extra=traceback.extract_tb(*sys.exc_info()))

        # Reraise the exception.
        raise ex
```

The same `__main__` block from above can be used, substituting `bugzscout_app` for `app`. When an error is thrown during a request, it will be recorded to FogBugz via BugzScout (assuming the BugzScout configuration is correctly set). Note that the request will still fail in the same way as it would above, since the exception is re-raised.

4.2 Example WSGI server using celery extension

The *simple WSGI example* provides a concise example of how to use bugzscout, but it is not suitable for production use. In a production environment, the bugzscout reporting should not make an HTTP request to a FogBugz server during another request. It is much more performant to do so asynchronously.

Celery is an asynchronous task queue that supports several backend implementations, like AMQP. The bugzscout package comes with a Celery extension that has a flexible configuration.

The celery extension is the recommended way to submit errors in production environments. This example describes how to setup and use the celery extension in the context of a WSGI server application.

4.2.1 Prerequisite

The celery package needs to be installed in the current environment before this will work. Celery is not declared as a dependency, since it can be bulky and it is not needed to use the core bugzscout functionality.

```
pip install celery
```

4.2.2 Celery setup

The rest of this example will assume the code is being added to a module called, `myapp.py`. The module first needs to import the celery extension and set a global `celery` variable.

```
import bugzscout.ext.celery_app
celery = bugzscout.ext.celery_app.celery
```

Setting a direct reference to the celery instance in bugzscout will be important when consuming the messages in the celery worker.

Next, create a WSGI app. This example provides a slightly more interesting app, which returns different responses based on the HTTP request method.

```
def app(environ, start_response):
    """Simple WSGI application. Returns 200 OK response with 'Hellow world!' in
    the body for GET requests. Returns 405 Method Not Allowed for all other
    methods.

    Returns 500 Internal Server Error if an exception is thrown. The response
    body will not include the error or any information about it. The error and
    its stack trace will be reported to FogBugz via BugzScout, though.

    :param environ: WSGI environ
    :param start_response: function that accepts status string and headers
    """
    try:
        if environ['REQUEST_METHOD'] == 'GET':
            start_response('200 OK', [('content-type', 'text/html')])
            return ['Hellow world!']
        else:
            start_response(
                '405 Method Not Allowed', [('content-type', 'text/html')])
            return ['']
    except Exception as ex:
        # Call start_response with exception info.
        start_response(
            '500 Internal Server Error',
            [('content-type', 'text/html')],
            sys.exc_info())

        # Record the error to FogBugz and this will return the body for the
        # error response.
        return _handle_exc(ex)
```

The `_handle_exc` function, invoked when an exception is caught, will call the celery extension. That will publish a task to be consumed by a celery worker. By returning `[”]`, the response body will be of length zero.

```
def _handle_exc(exception):
    """Record exception with stack trace to FogBugz via BugzScout,
    asynchronously. Returns an empty string.
```

```
    Note that this will not be reported to FogBugz until a celery worker
    processes this task.

    :param exception: uncaught exception thrown in app
    """
    # Set the description to a familiar string with the exception
    # message. Add the stack trace to extra.
    bugzscout.ext.celery_app.submit_error.delay(
        'http://fogbugz/scoutSubmit.asp',
        'error-user',
        'MyAppProject',
        'Errors',
        'An error occurred in MyApp: {0}'.format(exception.message),
        extra=traceback.extract_tb(*sys.exc_info()))

    # Return an empty body.
    return ['']
```

Finally, add a main block to run the Paste httpserver.

```
if __name__ == '__main__':
    import paste.httpserver
    paste.httpserver.serve(app, host='0.0.0.0', port='5000')
```

The *full myapp.py module* is below.

4.2.3 Overview of WSGI app

When an error is thrown while processing a request, this app will:

1. Catch the error.
2. Call `start_response` with exception info, which is a WSGI convention for returning error responses.
3. Call `_handle_exc`.
4. `_handle_exc` will publish a new task with the error message and stack trace. The description and extra are the same as the *simple WSGI example*. The call to `bugzscout.ext.celery_app.submit_error.delay` will do different things depending on how celery is configured. If an AMQP broker is used, the call will publish a message to an exchange. Later, a celery worker will consume that message.
5. Finally, `_handle_exc` returns `["]` so the response body will be empty. Since this is designed to work in production environments, exposing internal stack traces is not be ideal.

At this point, each error is creating a new celery task. There needs to be a celery worker present to consume those tasks.

4.2.4 Configuring celery

The celery instance can be configured as described in the celery docs. For example, it can be configured to use a rabbitmq-server as a broker with:

```
celery.conf.update(
    BROKER_URL='amqp://user:pass@rabbitmq-server:5672//'
)
```

4.2.5 Celery worker setup

Since a `celery` variable is set in the module, the celery executable can use it to start a worker. For example:

```
celery --app=myapp worker --loglevel=DEBUG
```

Running celery in this way is suitable for development. For production environments, see the Running the worker as a daemon section of the celery docs.

4.2.6 Full myapp.py source

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""A simple WSGI server that reports errors to FogBugz via BugzScout
asynchronously.
"""

from __future__ import print_function, unicode_literals

import bugzscout.ext.celery_app
import sys
import traceback

# Set celery here, so this module can be designated as the app when running the
# celery work.
celery = bugzscout.ext.celery_app.celery


def _handle_exc(exception):
    """Record exception with stack trace to FogBugz via BugzScout,
    asynchronously. Returns an empty string.

    Note that this will not be reported to FogBugz until a celery worker
    processes this task.

    :param exception: uncaught exception thrown in app
    """
    # Set the description to a familiar string with the exception
    # message. Add the stack trace to extra.
    bugzscout.ext.celery_app.submit_error.delay(
        'http://fogbugz/scoutSubmit.asp',
        'error-user',
        'MyAppProject',
        'Errors',
        'An error occurred in MyApp: {0}'.format(exception.message),
        extra=traceback.extract_tb(*sys.exc_info()))

    # Return an empty body.
    return ['']


def app(environ, start_response):
    """Simple WSGI application. Returns 200 OK response with 'Hellow world!' in
    the body for GET requests. Returns 405 Method Not Allowed for all other
    methods.

```

```
47      Returns 500 Internal Server Error if an exception is thrown. The response
48      body will not include the error or any information about it. The error and
49      its stack trace will be reported to FogBugz via BugzScout, though.
50
51      :param environ: WSGI environ
52      :param start_response: function that accepts status string and headers
53      """
54      try:
55          if environ['REQUEST_METHOD'] == 'GET':
56              start_response('200 OK', [('content-type', 'text/html')])
57              return ['Hellow world!']
58          else:
59              start_response(
60                  '405 Method Not Allowed', [('content-type', 'text/html')])
61              return ['']
62      except Exception as ex:
63          # Call start_response with exception info.
64          start_response(
65              '500 Internal Server Error',
66              [('content-type', 'text/html')],
67              sys.exc_info())
68
69          # Record the error to FogBugz and this will return the body for the
70          # error response.
71          return _handle_exc(ex)
72
73
74  if __name__ == '__main__':
75      import paste.httpserver
76      paste.httpserver.serve(app, host='0.0.0.0', port='5000')
```

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

b