NodeRunner Documentation

Release 0.1

Willliam Högman

November 30, 2016

Contents

1 Contents **3**
- 1.1 Getting started . . . 3
- 1.2 API Documentation . . . 3
- 1.3 Examples . . . 6

2 Indices and tables **7**

Python Module Index **9**

NodeRunner is a python library that lets you use Node.JS libraries from Python. NodeRunner aims to provide a feature rich pythonic interface for writing code that uses node.js in python. You can use it to create things like build-scripts etc.

You probably want to checkout the Getting started section and our API Documentation.

CHAPTER 1

Contents

1.1 Getting started

1.2 API Documentation

The API documentation covers classes and functions that you most likely will find yourself using. Reading this portion of the documentation should give you a good sense of NodeRunner's functionality. Python module for running node.js code from python

class noderunner.**Client**(*secured=False*)
Primary class for interfacing with node

The client class provides an easy to use interface for calling node functions, handling context and requirements.

call(*path*, *args*, *context*)
Calls the function at path with the passed-in args

Calls the function specified by the path using the passed in arguments. Functions are always called in the context of the next to last part of the path. This ensures that the calling ["console","log"] is called in the context of console. The arguments should be a list or tuple each containing an object that is covertable to JSON.

Parameters

- **path** (*list*) – The path to the function to be called.
- **args** – A list arguments to call the function with. All the arguments have to convertable to JSON.
- **context** (*str*) – The context containing the function

Returns The value returned by the function.

Return type A JavaScript object.

context(*name*, *reqs=[]*)
Creates a named context

Creates a named context with the passed in requirements pre-included, the context will have it's own global object making it hard to interfere with other things running in node.

Parameters

- **name** (*str*) – Name of the context to be created

- **reqs** (*list*) – The names of the requirements to be loaded into the context.

Returns a context object for the passed in name.

Return type Context

eval(*code*, *context=None*)
Evaluates the code and returns the result

Evaluates the passed in string as JavaScript code. This code may optionally be run in a context.

Parameters

- **code** (*str*) – The code to be evaluated
- **context** (*str*) – Name of the context to run the code in.

Returns The result of evaluating the expression, as best represented in python.

get(*path*, *context*)
Gets the value of a javascript variable.

Gets the value of a Javascript name/object path in the given context. The passed in path should be a list containing the path to the value that you want to read. For example a list with the elements 'console' and 'log' corresponds to "console.log".

Parameters **path** (*list*) – The path to the value to be retrived.

Returns The value of the object that the path points to.

Return type JSError or a JavaScript object.

set(*path*, *val*, *context*)
Sets the value of a javascript variable.

Sets the value of a Javascript name/object path in the given context. The passed in path should be a list containing the path to teh value that you wish to set. For example a list with the elements 'console' and 'log' corresponds to "console.log". The new value has to be a convertable to JSON and thus javascript. Therefore nested dicts are fine but other objects will fail.

Parameters

- **path** (*list*) – The path to the value to be retrived.
- **val** (*int, str, list, tuple, dict, their subclasses and nested structures comprising of these.*) – The value to that the path should be set to
- **context** (*str*) – The context to run the command in.

Returns The changed value as it is represented in JavaScript.

Return type A JavaScript object.

stop()
Stops the client and terminates the node process

class noderunner.client.**Context**(*client*, *name*)
A context in which certain commands such as eval can be run

You almost never need to create a context this way, instead use the context function on your client object.

call(**args*)
Calls the function at path with the passed-in args

Calls the function specified by the path using the passed in arguments. Functions are always called in the context of the next to last part of the path. This ensures that the calling ["console","log"] is called in the

context of console. The arguments should be a list or tuple each containing an object that is covertable to JSON. The last argument to this function should be a collection containing items, all of which are convertable to JSON, that represent the arguments to call the function with.

Parameters ***args** (*list*) – All but the last argument represent the path. The last argument should contain the arguments.

Returns The value returned by the function.

Return type A JavaScript object.

eval(*code*)

Evaluates the code in this context.

Evaluates the passed in string and returns the result, the code will be evaluated in the context named in this instance.

Parameters **code** (*str*) – The code to be evaluated

Returns The result of evaluating the expression, as best represented in python.

get(**path*)

Gets the value of a javascript variable.

Takes any number of arguments each representing one part of the path. For example calling this function with two arguments, "console" and "log" returns the value of console.log in javascript.

Parameters ***path** – The path to the value to be retrived.

Returns The value of the object that the path points to.

Return type `JSError` or a JavaScript object.

objects

Returns a handle pointing the context global object.

Retruns a handle that points the the global object or root object of this context. This is the recommended way to get a handle for interfacing with context.

set(**args*)

Sets the value of a javascript variable.

Takes any number of arguments where all but the last one represents a part of the object path. The last argument is the value to set the value to. The function returns the value as it is represented in JavaScript.

Parameters ***path** – All but the last argument represent the path. The last argument is the value to set the path to.

Returns The value that the object at the path was set to.

Return type `JSError` or A JavaScript object.

class `noderunner.client.`**Handle**(*context*, *path=[]*)

A handle is an object representing the path to some value.

Handles are used for calling JavaScript code in a pythonic way. This is acheieved using the overridable magic methods provided in Python.

Accessing an attribute that is not a member of the handle class returns a handle pointing to that object. For example accessing root.console.log, where root points to the global object in a context. Cuasing root to returns handle object pointing to console, and this object, in turn gives a reference to log.

Calling a handle will call the object that it points to returning the value returned by the javascript function.

get()
Gets the value that the handle points to

Performs a NodeRunner get operation returning the value that the handle points to.

Returns The JavaScript value that the handle points at

Return type A JavaScript object

1.3 Examples

NodeRunner comes with a number of examples to show how to best make use of its apis. If you feel that some part of the application is missing examples feel free to submit pull requests including new ones.

1.3.1 Calling the CoffeeScript compiler

```
"""Example showing how to use coffeescript with noderunner
"""
from noderunner import Client

code = """test_fn = -> console.log('foo')"""


def main():
    cli = Client()

    ctx = cli.context("example", [("cs", "coffee-script")])

    res = ctx.call("cs", "compile", (code,))

    print(res)


if __name__ == "__main__":
    main()
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

n

noderunner, 3

C

call() (noderunner.Client method), 3
call() (noderunner.client.Context method), 4
Client (class in noderunner), 3
Context (class in noderunner.client), 4
context() (noderunner.Client method), 3

E

eval() (noderunner.Client method), 4
eval() (noderunner.client.Context method), 5

G

get() (noderunner.Client method), 4
get() (noderunner.client.Context method), 5
get() (noderunner.client.Handle method), 5

H

Handle (class in noderunner.client), 5

N

noderunner (module), 3

O

objects (noderunner.client.Context attribute), 5

S

set() (noderunner.Client method), 4
set() (noderunner.client.Context method), 5
stop() (noderunner.Client method), 4