Stencila Documentation

Release 0.1.0

Nokome Bentley

Aug 03, 2018

Contents

Contents:

CHAPTER 1

PythonContext

class stencila.**PythonContext**(**args*, ***kwargs*)

libraries(**args*)

list(*types=[]*)

compile(*cell*)
Compile a cell

Returns A compiled `cell`

compile_func(*func=None*, *file=None*, *dir=None*)
Compile a `func` operation

Parses the source of the function (either a string or a file path) to extract it's `description`, `param`, `return` etc properties.

```
>>> context.compile_func({
>>>     'type': 'func',
>>>     'source': 'def hello(who): return "Hello Hello %s!" % who'
>>> })
{
    'type': 'func',
    'name': 'hello',
    'source': 'def hello(): return "Hello Hello %s!" % who'
    'params': [{
        'name': 'who'
    }],
    ...
}
```

Parameters **func** (*dict or string*) – A `func` operation. If a string is supplied then an operation object is created with the `source` property set to the string.

Returns

- **func** (*dict*) – The compiled `func` operation
- **messages** (*list*) – A list of messages (e.g errors)

execute(*cell*)

```
spec = {'client':  'ContextHttpClient', 'name':  'PythonContext'}
```

CHAPTER 2

SqliteContext

class stencila.**SqliteContext**(**args*, ***kwargs*)

compile(*operation*)
Compile an operation

Returns A dictionary with `messages` and a compiled `operation`

execute(*operation*)

pack(*data*, *max_rows=30*)
Pack data into a data package or data pointer

Currently this context only deals with tables and chooses to create a package or a pointer based on the number of rows in the table

Parameters **data** – Name of the table to pack

unpack(*packed*, *name*)

fetch(*name*, *options={}*)

spec = {'client': 'ContextHttpClient', 'name': 'SqliteContext'}

CHAPTER 3

Host

class stencila.**Host**

A *Host* allows you to create, get, run methods of, and delete instances of various types. The types can be thought of a "services" provided by the host e.g. *PythonContext*, *FilesystemStorer*

The API of a host is similar to that of a HTTP server. It's methods names (e.g. *post*, *get*) are similar to HTTP methods (e.g. *POST*, *GET*) but the sematics sometimes differ (e.g. a host's *put()* method is used to call an instance method)

A *Host* is not limited to beng served by HTTP and it's methods are exposed by *HostHttpServer*. Those other classes are responsible for tasks associated with their communication protocol (e.g. serialising and deserialising objects).

This is a singleton class. There should only ever be one *Host* in memory in each process (although, for purposes of testing, this is not enforced)

id

Get the identifier of the Host

Returns An identification string

key

Get the seurity key for this Host

Returns A key string

user_dir()

Get the current user's Stencila data directory.

This is the directory that Stencila configuration settings, such as the installed Stencila hosts, and document buffers get stored.

Returns A filesystem path

temp_dir()

Get the current Stencila temporary directory.

Returns A filesystem path

environs()

types()

manifest()
Get a manifest for this host.

The manifest describes the host and it's capabilities. It is used by peer hosts to determine which "types" this host provides and which "instances" have already been instantiated.

Returns A manifest object

register()
Registration of a host involves creating a file *py.json* inside of the user's Stencila data (see *user_dir()*) directory which describes the capabilities of this host.

startup(*environ*)

shutdown(*host*)

create(*type*, *args={}*)
Create a new instance of a type

Parameters

- **type** – Type of instance
- **args** – Arguments to be passed to type constructor

Returns Name of newly created instance

get(*name*)
Get an instance

Parameters **name** – Name of instance

Returns The instance

call(*name*, *method*, *arg=None*)
Call a method of an instance

Parameters

- **name** – Name of instance
- **method** – Name of instance method
- **kwargs** – A dictionary of method arguments

Returns Result of method call

delete(*name*)
Delete an instance

Parameters **name** – Name of instance

start(*address='127.0.0.1'*, *port=2000*, *quiet=False*)
Start serving this host

Currently, HTTP is the only server available for hosts. We plan to implement a *HostWebsocketServer* soon.

Returns self

stop(*quiet=False*)
Stop serving this host

Returns self

run(*address='127.0.0.1'*, *port=2000*)
Start serving this host and wait for connections indefinitely

spawn()

servers

Get a list of servers for this host.

Currenty, only a *HostHttpServer* is implemented but in the future onther servers for a host may be added (e.g. *HostWebsocketServer*)

Returns A dictionary of server details

generate_token(*host=None*)

Generate a request token.

Returns A JWT token string

authorize_token(*token*)

Authorize a request token.

Throws an error if the token is invalid.

Parameters **token** – A JWT token string

view()

View this host in the browser

Opens the default browser at the URL of this host

CHAPTER 4

HostHttpServer

class stencila.**HostHttpServer**(*host*, *address='127.0.0.1'*, *port=2000*)

A HTTP server for a Host

Provides access to a `Host` via a REST-like HTTP protocol using `POST` to create new instance and `PUT` to run one of it's methods. Implements authorization using single-, or multi-, use "tickets" and session tokens.

The following example illustrates creating a `PythonContext` and then running it's `execute` method. It uses the `http` command line tool (https://httpie.org/) for brevity and session management but you could also use `curl` or other HTTP client.

```
# Start the server
> python -m stencila
Host has started at: http://127.0.0.1:2000/?ticket=w8Z0ZkuWlz8Y
Use Ctrl+C to stop

# Then in another shell create a new PythonContext (using the above ticket
# to obtain access authorization and a session token) using POST
> http --session=/tmp/session.json  POST :2000/PythonContext?ticket=w8Z0ZkuWlz8Y
HTTP/1.0 200 OK
Content-Length: 21
Content-Type: application/json
Date: Wed, 28 Feb 2018 21:36:37 GMT
Server: Werkzeug/0.12.2 Python/2.7.12
Set-Cookie:␣
↪token=PjvskQ38vtuJQg2hNYEHwPppvw8RKbs0AaYcA9uStannZkGfRr3I0g9jyeQD3L3f; Path=/

"pythonContext1"

# Then use the returned name of the PythonContext instance to run it's "execute"␣
↪method
# using PUT
> http --session=/tmp/session.json  PUT :2000/pythonContext1!execute code='sys.
↪version'
HTTP/1.0 200 OK
Content-Length: 153
```

(continues on next page)

(continued from previous page)

```
Content-Type: application/json
Date: Wed, 28 Feb 2018 21:39:54 GMT
Server: Werkzeug/0.12.2 Python/2.7.12

{
    "messages": [],
    "value": {
        "data": "2.7.12 (default, Nov 20 2017, 18:23:56) [GCC 5.4.0 20160609]",
        "type": "string"
    }
}
```

url
Get the URL of the server

Returns A URL string

start(*real=True*)
Start the server

stop(*real=True*)
Stop the server

handle(*request*)
Handle a HTTP request

route(*verb*, *path*, *authorized=False*)
Route a HTTP request

static(*request*, *response*, *path*)
Handle a GET request for a static file

run(*request*, *response*, *method*, **args*)
Run a host method

error(*request*, *response*, *code*, *name*, *what=''*)

error400(*request*, *response*, *what=''*)

error401(*request*, *response*, *what=''*)

error403(*request*, *response*, *what=''*)

error404(*request*, *response*, *what=''*)

error500(*request*, *response*)

CHAPTER 5

Value

A module for packing and unpacking values so that they can be transferred between languages and hosts.

Type and packaing and unpacking of data values

`stencila.value.`**`type`**(*value*)
Get the type code for a value

Parameters **`value`** – A Python value

Returns Type code for value

`stencila.value.`**`pack`**(*value*)
Pack an object into a value package

Parameters **`value`** – A Python value

Returns A value package

`stencila.value.`**`unpack`**(*pkg*)
Unpack a value package into a Python value

Parameters **`pkg`** – The value package

Returns A Python value

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

U

V