sparkplug Documentation

Release 1.2.0

dhsavell05

Mar 18, 2018

Contents:

Sparkplug is a simple command framework designed with chat bots in mind.

CHAPTER 1

Quickstart Guide

Commands are modularized into units, like the one below. A unit can be anything with callable attributes. Any callable attribute that doesn't begin with an underscore is assumed to be a user-facing command.

The parameters of commands should be annotated in order to tell the command host how to fill them in based on the user input.

```
>>> class SomeCommands:
...     def add_one(self, number: int):
...         return number + 1
...     def say_hello(self, person: 'word', times: int):
...         return ' '.join(['Hello ' + person] * times)
...     def repeat(self, text: 'my_text_getter'):
...         return text
...     def repeat_context(self, text: 'context_text'):
...         return text
```

If, for some reason, type annotations cannot be used, a decorator is available. The following will produce the same result as above when given processed by a command host:

```
>>> class SomeCommands:
...     @parameter_types(number=int)
...     def add_one(self, number):
...         return number + 1
...     @parameter_types(person='word', times=int)
...     def say_hello(self, person: 'word', times: int):
...         return ' '.join(['Hello ' + person] * times)
...     @parameter_types(text='my_text_getter')
...     def repeat(self, text):
...         return text
...     @parameter_types(text='repeat_context')
...     def repeat_context(self, text):
...         return text
```

A command host contains units like this in which it calls commands from.

```
>>> host = CommandHost()
```

An async variant is available, but just note that it will `await` both commands and strategies.

```
>>> host = AsyncCommandHost()
```

We'll stick with the synchronous command host for this example.

Units can be added in a few different ways:

```
>>> host += 'foo', SomeCommands()      # Name specified explicitly
>>> host['foo'] = SomeCommands()       # Name specified explicitly
>>> host += SomeCommands()             # __name__ of type used to guess name
```

A command host also contains strategies, which are used to fill in the annotated types of commands. They are functions that should accept parameters in string form, and return a tuple of the object interpreted from the string *and* the remaining parameters that were not used.

Strategies can also accept an arbitrary context parameter passed to them by the host. If a strategy that needs context is called without a context parameter given to the host, then it defaults to None.

If invalid input is given, a strategy can handle this however it wants (i.e. raising an error, using a default value).

```
>>> host.add_strategy(int, strategies.int_getter) # Default strategies for common␣
↪types are available
>>> host.add_strategy('word', strategies.word_getter)
>>> host.add_strategy('text', lambda s: (s, '')) # Returns the entire input.␣
↪Everything is consumed, so '' is returned
>>> host.add_strategy('context_text', lambda s, c: (c, s)) # Returns the context␣
↪without consuming any text
```

Once all of the desired commands and strategies are set up, `call` can be used with a command string to parse and execute it.

```
>>> host.call('add_one 41')
42
>>> host.call('say_hello John 3')
'Hello John Hello John Hello John'
>>> host_call('repeat after me')
'after me'
>>> host_call('repeat_context', context='some context')
'some context'
```

In the event of a command clash (suppose we had a unit `'bar'` that also had a command named `repeat`), unit names can explicitly be specified.

```
>>> host.call('foo:repeat after me')
'after me'
```

Finally, if for some reason a unit needs to be unloaded, it can be done by subtracting its name from the host.

```
>>> host -= 'foo'
```

CHAPTER 2

Terminology

Sparkplug breaks its command management system into multiple parts. Definitions of each of the parts and how they are related are below.

2.1 Host

A host is the main part of the system. It contains units and can call the commands contained within them given a string that the user has entered.

2.2 Unit

A unit is a provider of commands. It is an object in which all of the callable attributes (that don't begin with an underscore) are assumed to be accessible by the host.

2.3 Command

A command is a function that belongs to a unit and executes a certain task. Its parameters (other than `self`) should be annotated in order to tell the host executing it how to fulfill its parameters.

2.4 Strategy

A strategy is what a host uses to resolve the parameters of a command. It is a function that should take a string (which contains the arguments passed to the command) and return a tuple of the object interpreted from it and a string containing the parameters that were not used to interpret the object. These remaining arguments are passed into the next strategy which parses the next argument, if it exists.

A local strategy is a strategy that only applies to a certain unit.

CHAPTER 3

API Reference

This page outlines sparkplug's API.

3.1 sparkplug.host

This module contains the sparkplug command hosts and errors they might raise.

class sparkplug.host.**CommandHost**(*use_fallback_strategy=False*)
Provides a modular command system.

Parameters **use_fallback_strategy** – Whether or not a default fallback strategy (that provides None) should be used in the event of an undefined strategy.

add_local_strategy(*unit_name*, *annotation*, *strategy*)
Adds a local strategy to this CommandHandler. A local strategy functions similarly to a normal, global one, but takes takes precedence over global ones for a specific unit.

Parameters

- **unit_name** – Unit to apply this strategy to.
- **annotation** – Parameter annotation to match to the given strategy.
- **strategy** – Callable strategy to get an appropriate object from.

add_strategy(*annotation*, *strategy*)
Adds a strategy to this CommandHandler.

Parameters

- **annotation** – Parameter annotation to match to the given strategy.
- **strategy** – Callable strategy to get an appropriate object from.

call(*command_call*, *context=None*)
Calls a command and fulfills its parameters using known strategies with the given call as a string.

Parameters

- **command_call** – Name and parameters given to the command as a string.
- **context** – Optional context object that is passed to any strategies that will accept it.

Returns Return value from the command.

get_parameter(*annotation*, *text_args*, *context=None*, *domain=None*)
Gets the value for a parameter using known strategies and given arguments. If given a domain, local strategies will take priority over global ones.

Parameters

- **annotation** – Parameter annotation, determines which strategy to use.
- **text_args** – Arguments that can be used to determine parameters.
- **context** – Optional context object that is passed to the strategy. defaults to None.
- **domain** – Name of the unit that the parameter applies to, used to prioritize local strategies.

Returns Extracted parameter and remaining arguments afterwards.

get_strategy(*annotation*, *unit_name=None*)
Gets a strategy to resolve a parameter, optionally with a specific domain.

Parameters

- **annotation** – Parameter annotation find a strategy for.
- **unit_name** – Domain to check a local strategy for first.

Raises ***NonexistentStrategyError*** – if use_fallback_strategy is false and a strategy was not found.

Returns Strategy for getting a parameter with the given annotation.

class sparkplug.host.**AsyncCommandHost**(*use_fallback_strategy=False*)
An extension of CommandHost which awaits strategies and commands.

class sparkplug.host.**CommandHostError**
Represents an exception that was raised by a CommandHost.

class sparkplug.host.**NonexistentCommandError**(*attempted_command*, *available_commands*)
Represents an exception that occurred from attempting to call a command that does not exist.

class sparkplug.host.**NonexistentUnitError**(*unit_name*)
Represents an exception that occurred from attempting to access a unit that does not exist.

class sparkplug.host.**NonexistentStrategyError**(*wanted_type*)
Represents an exception that occurred from attempting to fulfill a parameter with an annotation that no strategy is defined for.

class sparkplug.host.**AmbiguityError**(*attempted_command*)
Represents an exception that occurred from attempting to call a command that is defined in multiple units without explicity specifying a domain.

3.2 sparkplug.strategies

This module contains simple strategies for use with command hosts.

sparkplug.strategies.**int_getter**(*text*)
Gets an integer from the beginning of the given text.

Parameters **text** – Parameters to work with.

Raises **ValueError** – if text does not begin with an integer.

Returns Tuple of extracted number and remaining parameters.

sparkplug.strategies.**default_int_getter**(*default*)
Returns an integer getter that returns a default value if the given parameters don't start with an integer.

Parameters **default** – Default value to return in the event of invalid input.

Returns Wrapped integer getter function.

sparkplug.strategies.**word_getter**(*text*)
Gets the first word separated by a space from text.

Parameters **text** – Parameters to work with.

Returns Tuple of the first word of the parameters and the remaining text.

sparkplug.strategies.**remaining_text_getter**(*text*)
Consumes all of the remaining parameters.

Parameters **text** – Parameters to work with.

Returns Tuple of the given parameters and an empty string.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index