peact Documentation

Release 0.1

Matthew Spellings

Feb 09, 2019

Contents

Peact is a library for reactive programming in python.

CHAPTER 1

Installation

Installation works using distutils, for example:

```
python setup.py install --user
```

peact uses cython to build a C extension. If you update *peact/_peact.pyx*, you can trigger the cython code to rebuild with *–cython*:

```
python setup.py install --user --cython
```

CHAPTER 2

Introduction

As an analogy for peact, consider the process of building software. The predominant build method on UNIX systems involves *Makefiles*, which specify files which can be created and "recipes" to create each file. Each file has a number of dependencies, which the make system will ensure have been created before the recipe for the file is run.

Peact is a library which enables a similar method of programming inside python instead of on the filesystem. Rather than the *make* program, peact is the orchestrator of activity. Instead of files, peact deals with "quantities," each with a particular name. The recipes and file contents of *make* are replaced with python functions and python objects, respectively.

In other words, peact allows you to string together python functions which consume and produce quantities. As input values change, nodes in the graph are updated in response to these changes, potentially updating other nodes as well.

CHAPTER 3

Reactive Python API

To use peact, create a *peact.CallGraph* object and *peact.CallGraph.register() peact.CallNode* objects (representing functions) on it. Input values can come from nodes which themselves have no inputs or by calling *peact.CallGraph.inject()* to immediately set values.

After the *peact.CallGraph* has been prepared, *peact.CallGraph.pump()* can be used to step through the graph and call each registered function which needs to be updated. Values are stored in the *scope* member of a *peact.CallGraph*.

class peact.**CallNode**

CallNode objects wrap a function for use in a CallGraph.

Parameters

- **function** – The function (or callable object) to be called when the output is needed or an input changes
- **output** – A name (or list of names if the function returns a tuple) to bind the function output to. If not given, defaults to the name of the function
- **async** – True if the function can be called in a background process
- **remap** – A dictionary mapping function parameter names to scope names
- **as_needed** – True if the function should not be called when its inputs change, but only as something that needs its value is called

class peact.**CallGraph**

Handles the reactivity for a set of *CallNode* objects.

Each *CallNode* has a set of input (dependency) and output names. Nodes are added to the graph via *peact.CallGraph.register()*.

clear

Remove all modules from the call graph

inject

Puts a value or set of values into the list of stored quantities and marks it as having changed.

Example:

```
graph.inject(temperature=1.5)
graph.inject({'namespace.value': 13})
```

mark
CallGraph.mark_input(self, *args) Marks a quantity for everything that depends on it to be recomputed

mark_input
Marks a quantity for everything that depends on it to be recomputed

mark_output
Marks a quantity for the last node that computes it to be re-run

pump
Step through the graph, calling module functions whose input has changed or output is required.

Example:

```
for _ in graph.pump():
    pass
```

Parameters

- **input_names** – iterable of names for values that have changed; nodes that depend on these quantities will be re-evaluated. If None, default to the set of marked "dirty" inputs
- **output_names** – iterable of names to force computation of; nodes that provide these quantities will be re-evaluated. If None, default to the set of marked "dirty" outputs
- **async** – If True, yield intermediate results whenever an asynchronous module is encountered

pump_restore
Evaluate the graph for a set of given names. Restores the current state afterward.

Parameters

- **names** – List of quantity names to compute
- **async** – If True, compute asynchronously
- **kwargs** – List of quantities to inject into the scope before computing

pump_tick
Perform a single element of work every time it is called. Intended for embedding `peact.CallNode.pump()` into another event loop.

rebuild
Build the dependency graph for all modules currently in the graph, as well as data structures for efficient dispatch of data.

Parameters **mark_dirty** – If True, mark all properties in the graph as needing a recomputation

register
Register a function as part of this graph. Takes the same parameters as *peact.CallNode*.

Returns The given function

register_deferred
Registers a list object. This list should contain *peact.CallNode* objects and will be consulted dynamically every time *peact.CallGraph.rebuild()* is called.

Parameters **target** – List object containing *peact.CallNode* objects

register_last
Register a function as part of this graph, after the last function that supplies any quantity of the same name. Takes the same parameters as *peact.CallNode*.

Returns The given function

scope

unmark
CallGraph.unmark_input(self, *args) Voids a recomputation request for a quantity.

unmark_input
Voids a recomputation request for a quantity.

unmark_output
Voids a recomputation request for a quantity.

unregister
Remove the given function from the call graph.

Parameters

- **function** – The function which should be removed
- **rebuild** – If True, immediately rebuild the call graph

unregister_deferred
Remove the given dynamic *CallNode* provider from the graph.

Parameters

- **target** – The list object which should be removed
- **rebuild** – If True, immediately rebuild the call graph

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index