kTBS Bench Manager Documentation

Release 0.1

Vincent

April 24, 2014

Contents

Contents:

CHAPTER 1

Bench Manager

class `bench_manager.`**`BenchManager`**`(`*set_log_info=False*`)`

Manage benchmarks by timing function against different contexts.

General concept

A BenchManager is instantiated in order to collect functions to benchmark, like so:

```
>>> my_bench_manager = BenchManager()
```

In order to add functions to bench, one flag them for bench by using the `bench()` decorator. For example:

```
>>> @my_bench_manager.bench
... def add_one(n):
...     return n + 1
```

Each flagged function is then called against contexts. A context is a function with optional setup and teardown, and it must *yield* the parameter that benchmarked functions need:

```
>>> @my_bench_manager.context
... def three():
...     # optional setup
...     try:
...         # yield the parameter
...         yield 3
...     finally:
...         # optional teardown
...         pass
```

Finally, to perform the benchmarks, one must call:

```
>>> my_bench_manager.run('/tmp/my_results.csv')
```

The result of the two examples above is to time `add_one(3)`.

Technical details

Each context is stored in the list `_contexts`. Each function to benchmark is stored in the list `_bench_funcs`.

When `run()` is called, it will iterate over functions and contexts to call each function against each context.

Variables

- **_contexts** (*list*) – contexts to apply
- **_bench_funcs** (*list*) – functions to benchmark

- **_results** (*dict*) – collected benchmark results
- **_logger** – global logger

`bench`(*func*)
Prepare a function to be benched and add it to the list to be run later.

Parameters **func** (*function*) – the function to bench

`context`(*func*)
Decorate a function to act as a context.

Parameters **func** (*function*) – the function that describes the context

`run`(*output_filename*)
Benchmark functions against contexts.

Parameters **output_filename** (*str*) – filename of the CSV output

`write_output`(*output_filename*)
Write results of the BenchManager to a nicely formatted CSV file.

Parameters **output_filename** (*str*) – filename of the CSV output

CHAPTER 2

Benchable Graph

class `benchable_graph.`**`BenchableGraph`**(*store*, *graph_id*, *store_config*, *graph_create=False*)

Provides a convenient way to use a graph for benchmarks.

`connect`()

Connect to the store.

Note: For some configurations, RDFlib will postpone the actual connection to the store until needed (when doing a graph.query() or graph.add()).

This behaviour comes from RDFbib implementation of graph.open().

`close`(*commit_pending_transaction=True*)

Close a connection to a store.

Parameters **commit_pending_transaction** (*bool*) – True if to commit pending transaction before closing, False otherwise.

Note: The graph.close() method is not implemented for SPARQL Store in RDFLib

CHAPTER 3

Indices and tables

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

Python Module Index