Ex2 specification: **Batcher Service**

eran.amar@gmail.com

Recap and Introduction

JS event loop

JavaScript runs as a single threaded process, meaning that on each point in time, only one part of our code can be executed, and only upon its completion, other parts of our code will run. That is, functions never interfere with each other, and in this exercise we are going to exploit that fact.

Why to batch?

- Performance, performance, performance.
- To prevent processing on the data structure in an intermediate phase (with only part of the changes to the data, instable state)

Goal

In this exercise we will intertwine the callback pattern and the closure pattern to form a very powerful service - a batcher. We will take advantage of the setTimeout function in order to break the synchronous execution of our code.

Task Overview

Your goal is to implement a batcher service for a specific operation upon an abstract data structure. Our data structure supports a `setData` method that gets an object with key-value pairs, and we want that several calls to `setData` with different arguments will be combined into a single invocation (i.e `setData(allPairs)`). Pay attention - we know that in order to do that the actual setData execution must be delayed (or 'deferred') a little - until when? - until the next tick of the event loop.

Public API: the Batcher Service

Initialization:

```
var batcher = buildBatcher(dataStructure);
```

Methods:

/**
dataMap - regular object that contains key-value pairs, that is the data to be set into our data structure
*/

```
batcher.setData(dataMap){ … }
```

/**
Returns: the number of flushed operations.
Note: your code should call `dataStructure.setData` only once per invocation.
The completion callbacks shouldn't be called when there is nothing to flush.
*/

```
batcher.flush() { … }
```

/**
callback - a function to invoke after each flush. Can be called several times with different callbacks (no need to check that they are different) and should remember all previous callbacks.
*/

```
batcher.uponCompletion(callback) { … }
```

Recommended Steps

1) Implement `setData` that is not self-flushing (i.e must be flushed "manually" by calling `flush()`)
2) Implement the `flush` and `uponCompletion` methods
3) Use `setTimeout` to make your `setData` flushing itself upon the next tick of the event loop

Getting Started

Fork the following git repository, it already contains the outline for this exercise.

Links

- Elaborate explanation on JS event loop in this post.
- Super basic information about timers in JS. Please read!
- More about timers: actual delay, queuing and cancelling.

Enjoy()