MotionJS – User Manual

Prerequisites

You need *NodeJS* in version 0.4.0 or higher. To install or update *NodeJS*, refer to the documentation at https://github.com/joyent/node/wiki/Installation.
To install the modules *MotionJS* depends on, it is best to use the package manager npm. The current version (1.0 or higher required) of npm can be installed simply via:

```
curl http://npmjs.org/install.sh | sh
```

This command should be executed as root. After npm is installed, you can install the dependencies jake and nodeunit[1] via npm install. As both should be accessible on the command line, install them globally. Again, execute these commands as root.

```
npm install -g jake

npm install -g nodeunit
```

After this, you should be able to invoke node, npm, jake and nodeunit from the command line. If so, your system is ready to develop applications with *MotionJS*.

Installation

An actual installation is not necessary. Copy the *MotionJS* distribution to your disk. Additional code (contained in a *bundle*) can now be placed inside the bundles directory.

Usage

MotionJS can be used both server-side and client-side.

Server

In order to use *MotionJS*, you need to require the module located at bundles/org.motionjs.core/core.js in the script in which you want to use *MotionJS*, e.g.:

```
var motionjs = require('./../org.motionjs.core/core')
```

Client

On the client, you need to include the script containing *MotionJS* into all webpages in which you want to use *MotionJS*. There are two ways to do this: in development context (when you start to develop), you let the development server assemble that script:

```
<script src="http://localhost:8080/motion?port=8080" type="text/javascript"
charset="utf-8"></script>
```

See web/demoDevelopment.html for an example. Before you access the page in your browser, you need to start the server listening at the specified port (here 8080). See the section Build System on how to do this.

[1] https://github.com/caolan/nodeunit

When switching to production context, you need an assembled script ready for use without the development server. Generate this script by running the deploy command (see Build System). Then reference the generated script (which is placed in web/scripts):

```
<script src="scripts/motion.min.js" type="text/javascript" charset="utf-
8"></script>
```

See web/production.html for an example.

Reference

Objects

Objects are referenced by *identifiers*, which reflect the physical location and the filename in which the object is defined (e.g. a filename bundles/org.motionjs.demo/point.js refers to the *identifier* org.motionjs.demo/point). Each such file consists of two parts: an *object definition* and an *object configuration*. An example *object definition* is shown below:

```
exports.definition =
{ x: 0
, init: function (x) {
    this.x = x
  }
}
```

In this definition, you define the properties (variables and functions) of the object. If the object should be instantiable, you must provide an init function.
While the *object definition* is required, the *object configuration* is optional. There, you may specify

- which *identifiers* should be inherited (by giving an array of *identifiers*),
- on which *identifiers* this *identifier* depends (by giving an object which keys represent the dependencies and which values define the properties on which to set the dependencies) and
- if the object should be shared by the framework (defaults to false).

```
exports.configuration =
{ inherits: ['identifier1', 'identifier2']
, dependencies:
 { 'identifier3' = '_dependency' }
, shared: true
}
```

Global Configuration

The configuration is located in configuration.js. Below is a sample configuration:

```
exports.configuration =
{ mode: 'development'
, objects:
  { 'org.motionjs.demo/model/engine':
      { defaultArguments: ['BMW'] }
  , 'org.motionjs.event/manager':
      { providedBy: 'org.motionjs.demo/eventManager' }
  }
}
```

mode is either development or production. This refers to the context in which the framework runs. In development context, the development server (see section Build System) needs to run in order for *MotionJS* to work.

In the objects object, for each *identifier*, it is possible to set the defaultArguments and/or the providedBy option. defaultArguments is an array of arguments to pass to the init function if the *identifier* if the argument is not given via create (see API reference). In this example, the init function of org.motionjs.demo/model/engine will be passed BMW if no argument is given for create.

providedBy contains the *identifier* which should be loaded instead of another *identifier*. In this example, whenever org.motionjs.event/manager is requested, org.motionjs.demo/eventManager will be loaded.

API

The *MotionJS* object exposes to functions: create and clone. With create, a new object can be created from an *identifier*. However, the object is not returned, but passed to the given callback. Alternatively, an error is passed if something went wrong. An example:

```
motionjs.create('org.motionjs.demo/model/engine, function (error, engine) {
  if (error) throw error
  console.log(engine)
})
```

clone returns a clone of the passed object. clone will recursively clone contained objects. Optionally, it is possible to pass a built-in type as a second parameter to filter properties. In the following examples, only properties that are functions will be cloned.

```
var functionsOfObj = motionjs.clone(obj, ‘function’)
```

Build System

Tasks can be executed via jake from the root directory, e.g.:

```
jake devserver 8080
```

The following tasks are available:

- **devserver**
  Starts the development server. The port can be given as an optional argument and defaults to 8080.

- **deploy**
  Generates the files needed for the production context from the *cached requests*.

- **cache**
  This task will need a second parameter to run. The argument show will display all *cached requests*. Use delete and either a hash as third parameter to delete a single cached requests or all to delete all requests. New requests can be added via add (the requested *identifier* is specified as the first parameter, the already loaded *identifiers* in a comma separated list as the second parameter).

- **demo**
  This command will run the server side demo from the org.motionjs.demo bundle.

- **interface**
  Given an *identifier*, interface will show the public methods (not prefixed with an underscore) of the *identifier.*

- **test**
  The framework uses nodeunit as a testing framework. nodeunit has the advantage of being usable both client-side and server-side. This enables you to write unit tests for both sides in the same style. With this task, server-side tests are executable via jake test server path/to/file.js (multiple files can be executed if a folder is specified containing many unit tests). On the client, test need to run in the browser. The framework provides a simple test server (again a simple *NodeJS* server) which sole purpose it is to dynamically serve a HTML response composed of the unit tests to run. The test server can be invoked via jake test client path/to/file. The test will run on each request to the testserver (e.g. accessible at http://localhost:8081).