The unofficial

Dojo Toolkit 1.8 Quick Start Guide

Elia Contini

v1.0

Index

1. Introduction

The Dojo Toolkit (DT) is a JavaScript toolkit that gives to web developers a huge amount of functionalities and facilities to create rich web applications quickly. One of the big advantages of Dojo is to provide a uniform and coherent interface upon most famous browsers enabling developers to concentrate their efforts on application functionalities instead of solving cross browser issues. The Dojo Toolkit is divided in three projects:

1. **dojo**: which includes all basic functionalities to manage Document Object Model (DOM), arrays, events, asynchronous requests, Cascading Style Sheet (CSS) and more.
2. **dijit**: it is a rich and well tested set of widgets ready to use. Dijit contains form controls, data grid and a lot of UI utilities. All widgets are tested to guarantee accessibility.
3. **dojox**: it provides advanced functionalities and widgets. For instance, in dojox we can find widgets and utilities to build mobile web applications. Sometimes components of dojox are experimental.

The main purpose of this guide is to present the most important aspects and functionalities of Dojo Toolkit providing a guided study path in order to obtain the maximum benefit by the official Dojo tutorials. All chapters contain links to further resources. Of course, this is not an exhaustive guide, but only a good starting point to study DT.

The challenge of this guide is to provide self contained (or at least loosely coupled) chapters long no more than two pages.

Enjoy!

2. Configure and load Dojo Toolkit

Example: 00 - dojoConfig.html

In order to use DT we have to create a project structure like this (**Note**: this is only a suggestion):

```
myProject
 |- css
 |- img
 |- js
 |   |- lib
 |   |   |- dijit
 |   |   |- dojo
 |   |   `- dojox
 |   `- myWidgets
 `- index.html
```

Based on this directory layout, we can define the DT configuration in our index.html file. We will use HTML5 as markup language.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
        <meta charset="UTF-8">
        <title>Dojo Toolkit Configuration</title>
  </head>
  <body>
        <h1 id="test"></h1>
        <script>
            dojoConfig ={
              baseUrl: "js",
              isDebug: true, // enables debug
              async: true,   // enables AMD loader
              packages: [
                {
                  "name": "dojo",
                  "location": "lib/dojo"
                }
              ]
          };
        </script>
        <script src="js/lib/dojo/dojo.js"></script>
        <script>
          require(["dojo/dom", "dojo/dom-attr"], function(dom, attr){
            attr.set(dom.byId("test"), "innerHTML", "Hello, World!");
          });
        </script>
  </body>
</html>
```

In the first script tag we define the `dojoConfig` object that contains all configuration

parameters used by the Dojo core.

- `baseUrl`: specify which is the base path to find JavaScript files.
- `isDebug`: enable/disable debug messages. During development should be set on true.
- `async`: enable/disable asynchronous loading of Dojo modules. It should always set on true.
- `packages`: specify an alias and the real path (based on `baseUrl`) of a package.

In the second script tag we load the Dojo core.

In the third script tag we write our app: we need to specify which modules we need using the `require` function.

In the example, our application needs the `dojo/dom` and `dojo/dom-attr` modules. The `dojo/dom` module gives DOM functionalities such as to retrieve tags by id (`dom.byId`).

The `dojo/dom-attr` provides a unified API to deal with DOM node attribute and property values. We use it to set the `innerHTML` attribute of the given node.

2.1. Resources

- **dojoConfig**: configuring Dojo, http://dojotoolkit.org/reference-guide/1.8/dojo/_base/config.html#explicitly-creating-a-dojoconfig-object-before-including-the-dojo-core
- **require**: the Dojo AMD Loader, http://dojotoolkit.org/reference-guide/1.8/loader/amd.html#the-amd-api
- **dom.byId**: a simple alias to “document.getElementById”, http://dojotoolkit.org/reference-guide/1.8/dojo/dom.html#dojo-dom-byid
- **dojo/dom-attr**: a unified API to deal with DOM node attribute and property values, http://dojotoolkit.org/reference-guide/1.8/dojo/dom-attr.html#dojo-dom-attr-get

3. AMD: Asynchronous Module Definition

Example: 01 - AMD Module.html

A DT module is a piece of code that does something. Of course a module can use (require) other modules and a set of modules constitutes a package. First of all we have to tell to the loader in which directory our modules are. Based on the project structure exposed in the previous chapter, we will save our modules in the `myWidgets` directory. To configure the loader, we have to add a package configuration to the packages array:

```
packages: [
  {
    "name": "dojo",
    "location": "lib/dojo"
  },
  {
    "name": "my",
    "location": "lib/myWidgets"
  }
]
```

The module we are creating is called `Circle` (save the file as `Circle.js`) and calculate area and circumference. Here is the code:

```
define(["dojo/_base/declare"],
  function(declare){
    return declare("myWidgets.Circle", null, {
      _PI: null,
      _radius: null,


      constructor: function(/*integer*/ radius){
        this._radius = radius;
        this._PI = Math.PI;
      },

      getArea: function(){
        return Math.pow(this._radius, 2) * this._PI;
      },

      getCircumference: function(){
       return 2 * this._PI * this._radius;
      }
   });
 }
);
```

The `define` function allows us to define our module. The first argument contains all required modules: in our case we need only `dojo/_base/declare`, that is a module that enables to create a class in the Dojo way. The second parameter is a function that maps required modules

to their aliases to be used inside our code.

The `declare` function takes three parameters: a string as the name of the module we are creating, an array of other modules (superclasses) to be extended (in our case is `null`), an object that represent our class.

The function `constructor` defines the constructor of the class. Properties and methods that starts with _ (underscore) indicates private properties and methods by convention.

Now we are able to use our module in our page:

```
require(["dojo", "dojo/dom-attr", "my/Circle"],
  function(dojo, attr, Circle){
    var circle = new Circle(4);
    var result = "Circumference: " + circle.getCircumference() +
      "<br>Area: " + circle.getArea();
    attr.set(dojo.byId("test"), "innerHTML", result);
  }
);
```

3.1. Resources

- **define**: define AMD modules, http://dojotoolkit.org/documentation/tutorials/1.8/modules/
- **declare**: create a class, http://dojotoolkit.org/documentation/tutorials/1.8/declare/

4. AJAX: Request and Promise

Example: 02 - Request Promise.html

Modern web applications perform asynchronous requests to the server. DT provides a module called `dojo/request` to execute this kind of requests. In the example, we request information about books. The `book.json` file contains data with the following structure:

```
[
  {
    "ID": 1,
    "Title": "Book 1 Title",
    "Publisher": "Book 1 Publisher",
    "City": "Book 1 City",
    "Authors": "Author 1, Author 2",
    "Year": "2001"
  },
  ...
]
```

Our script reads the data and prints the title of each book on the page.

```
require(["dojo", "dojo/_base/array", "dojo/dom-construct",
  "dojo/request"],
  function(dojo, array, dom, request){
    var getData = request.get("book.json", {
      handleAs: "json"
    });

    getData.then(function(response){
      var container = dojo.byId("test");
      array.forEach(response, function(book, index){
        dom.create("div", {"innerHTML": book.Title}, container);
      });
    });
  }
);
```

The `get()` method returns a `Promise` object. Once the request was performed it is possible to call a function in order to manage the response. The method `then()` enables us to specify a callback to execute when data are available (**Note**: `.addCallback()` and `.addErrback()` are no longer available).

4.1. Promise

The `dojo/promise` module manages communication between asynchronous threads. When we call `request.get(url, options)`, a `promise` object is returned. Sometimes we have to request multiple resources and wait that all are retrieved. The promise module provides a useful method function called `all()` (**Note**: `dojo/promise/all` replaces

`dojo/DeferredList`). For example:

```
require(["dojo/promise/all", "dojo/request"], function(request, all){
  var promise1 = request.get("book.json", {handleAs: "json"});
  var promise2 = request.get("video.json", {handleAs: "json"});
  all([promise1, promise2]).then(function(results){
 // results will be an Array
});
```

4.2. Resources

- **dojo/request**: the Dojo IO/Request API, http://dojotoolkit.org/reference-guide/1.8/dojo/request.html
- **dojo/promise**: communication between asynchronous threads, http://dojotoolkit.org/reference-guide/1.8/dojo/promise.html
- **dojo/promise/all**: manages multiple promises, http://dojotoolkit.org/reference-guide/1.8/dojo/promise/all.html

5. Data store

Example: 03 - Data Store.html

DT gives an elegant way to manage a data collection. For example, once we retrieve data in JSON format, we can create a data store and query it as we do with a DBMS.

```
require(["dojo/store/Memory", "dojo/request"],
  function(Memory, request){
    var books = null;
    var getData = request.get("book.json", {handleAs: "json"});

    getData.then(function(response){
    books = new Memory({"data": response, "idProperty": "ID"});

    // retrieves all books
    var allBooks = books.query({});

    // retrives book with ID 1
    var book1 = books.get(1); // ID = 1

    // save updated data on the store
    book1.Title = "Title updated";
    books.put(book1);

    var newBook = {
      "ID": 10,
      "Title": "Unofficial Dojo Toolkit 1.8 Quick Start Guide",
      "Publisher": "Elia Contini",
      "City": "Giubiasco",
      "Authors": "Elia Contini",
      "Year": "2012"
    }

    // insert a new book
    books.add(newBook);

    // remove book with id 9
    books.remove(9);
  });
});
```

When we create a Memory store we have to specify two parameters:

- `data`: an array of objects;
- `idProperty`: the name of the property to use as the identifier (key).

5.1. Resources

- **dojo/store**: an uniform interface for the access and manipulation of stored data, http://dojotoolkit.org/reference-guide/1.8/dojo/store.html
- **dojo/store/Memory**: an object store wrapper for JavaScript/JSON available directly with an array, http://dojotoolkit.org/reference-guide/1.8/dojo/store/Memory.html

6. Managing events

Example: 04 - Events.html

From version 1.8, Dojo Toolkit provides a new unified module to manage events. In the example we can see how to use the on module. The HTML code is:

```
<button id="myButton">Click me!</button>
```

and the Dojo stuff are:

```
require(["dojo/dom", "dojo/on"],
  function(dom, on){
    var btn = dom.byId("myButton");
    on(btn, "click", function(event){
      console.log("You click me!", event);
    });
  }
);
```

6.1. Resources

- **Events**: the official tutorial, http://dojotoolkit.org/documentation/tutorials/1.8/events/
- **dojo/on**: general-purpose event handler, http://dojotoolkit.org/reference-guide/1.8/dojo/on.html

7. Creating widgets

Example: 05 - Custom Widget.html

From version 1.6, the Dojo team started the refactoring of Dojo Toolkit in order to make it compliant with the CommonJS AMD API. AMD allows to create more efficient scripts, to use different toolkits in the same web application avoiding conflicts and much more. In version 1.7, the refactoring has been completed and this release is also the base to start the migration to Dojo 2.0. Of course, in Dojo 1.8 we have to write AMD module.

We will create a simple widget that displays the classical Hello, world!

Based on the directory layout that we saw in chapter 2, we create a file called `HelloWorld.js` and save it in `myProject/lib/myWidgets/` directory.

Inside `HelloWorld.js` we can write these lines of code:

```
define([
 "dijit/_WidgetBase",
  "dijit/_TemplatedMixin",
 "dojo/_base/declare",
  "dojo/dom-attr"
 ],
 function(_Widget, _Templated, declare, domAttr){
   return declare("myWidgets.HelloWorld", [_Widget, _Templated],{
     templateString: '<div data-dojo-attach-point="_box"></div>',

     postCreate: function(){
       domAttr.set(this._box, "innerHTML", "Hello, World!");
     }
   });
});
```

Now we have to set the `dojoConfig` properly.

```
packages: [
  {
    "name": "dojo",
    "location": "lib/dojo"
  },
  {
    "name": "dijit",
    "location": "lib/dijit"
  },
  {
    "name": "my",
    "location": "lib/myWidgets"
  }
]
```

And to see the result, we have to write this piece of code in our page:

```
require(["dojo", "dojo/dom", "my/HelloWorld"],
  function(dojo, dom, HelloWorld){
    var helloWorld = new HelloWorld();
    helloWorld.placeAt(dom.byId("test"));
    helloWorld.startup();
  }
);
```

7.1. Resources

- **dojo/dom-attr**: the core Dojo DOM attributes API, http://dojotoolkit.org/reference-guide/1.8/dojo/dom-attr.html
- **Asynchronous Module Definition (AMD)**: Notes on Dojo, http://blog.eliacontini.info/post/16352021810/notes-on-dojo-asynchronous-module-definition-amd
- **Creating Template-based Widgets**: the official tutorial, http://dojotoolkit.org/documentation/tutorials/1.8/templated/