

Zend_Dojo - Automated Build Layers - Matthew Weier O'Phinney

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Dojo Automated Build Layers Component Proposal

Proposed Component Name	Zend_Dojo Automated Build Layers
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Dojo Automated Build Layers
Proposers	Matthew Weier O'Phinney
Zend Liaison	Ralph Schindler
Revision	0.1 - 13 January 2009: Initial Draft. 0.2 - 25 January 2009: More comprehensive overview; modified requirements; referenced laboratory prototype (wiki revision: 8)

Table of Contents

1. Overview
 - How Dojo custom layers work
 - Proposed Zend Framework integration
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
 - Fully automated build layer
 - Aggregate javascript in the build layer
 - Only create the build layer file once
 - Manually specify where to place the build layer file
 - Manually specify where to put the build profile file
9. Class Skeletons

1. Overview

Dojo is a highly modular Javascript framework with a structure that largely mirrors the 1-class-1-file structure of Zend Framework and PEAR. During development, this is an incredibly powerful paradigm as it provides the flexibility to add new functionality as needed. However, once in production, it can be a liability: each class loaded through `dojo.require` requires at least one, and in many cases dozens, of additional requests to the server to load dependencies. Fortunately, Dojo has a solution for this via custom builds.

Creating custom builds is relatively trivial. However, at this time, it requires the developer to examine each page, aggregate the `dojo.require` and other statements, create a layer file, and create the build profile.

Zend_Dojo provides a common point of aggregation for all these areas, and could be introspected to build a custom layer file and related build profile, which would leave creating the actual build as the only step necessary by the developer. This proposal aims to address this idea.

How Dojo custom layers work

Rather than generate a large number of `dojo.require` statements for every page, a common solution in Dojo is to create what is known as a layer script. A layer script typically is namespace, and contains simply a number of `dojo.require` statements:

```
dojo.provide("foo.main");

(function() {
    dojo.require("dojo.parser");
    dojo.require("dijit.layout.BorderContainer");
    dojo.require("dijit.layout.ContentPane");
    dojo.require("dijit.form.Button");
    dojo.require("dijit.form.ValidationTextBox");
    dojo.require("dijit.form.Form");
    dojo.require("dijit.Dialog");
    dojo.require("dojox.dtl.Context");
})();
```

These scripts *may* also contain JavaScript to run on every request, such as `dojo.addOnLoad` events, definition of `dojo.data` stores, etc. Such an approach might look like this:

```
dojo.provide("foo.main");

(function(){
    dojo.require("dojo.parser");
    dojo.require("dijit.layout.BorderContainer");
    dojo.require("dijit.layout.ContentPane");
    dojo.require("dijit.form.Button");
    dojo.require("dijit.form.ValidationTextBox");
    dojo.require("dijit.form.Form");
    dojo.require("dijit.Dialog");
    dojo.require("dojox.dtl.Context");

    if (!foo.disableLayout) {
        dojo.addOnLoad(function(){
            foo.layout = new dijit.layout.BorderContainer();
        });
    }
})();
```

Within your view markup, then you would simply `dojo.require` this single layer script:

```
<script language="javascript">
    dojo.require("foo.main");
</script>
```

This approach has the benefit of making it easier to maintain your view scripts between environments, and prevents the necessity of constantly updating application code.

Dojo itself leverages this paradigm in its build system. The build system can use a layer script as a dependency, and compile all JavaScript into a single file, pulling in all dependencies recursively. Additionally, it can strip out comments, whitespace, and perform optimizations on the code.

Build profiles are simply JavaScript, and define the various build options, layers, and dependencies. An example for the above might look like the following:

```

dependencies = {
  action:      "release",
  version:     "1.3.0-foo",
  releaseName: "foo",
  loader:      "default",
  cssOptimize: "comments",
  optimize:    "shrinksafe",
  layerOptimize: "shrinksafe",
  copyTests:   false,
  layers: [
    {
      name:      "../foo/main.js",
      layerDependencies: [],
      dependencies: [
        "foo.main",
      ]
    }
  ],
  prefixes: [
    [ "dijit", "../dijit" ],
    [ "dojox", "../dojox" ],
    [ "foo",   "../foo" ]
  ]
};

```

Several things to note about it. First, note that the layer has a dependency on itself. This allows you to reference the same module in your `dojo.require` both before and after the build. Second, note that even though `dojo.parser` was referenced in a `dojo.require` statement, there is no "dojo" prefix defined; the `dojo` directory will always be included in the build.

You then pass this profile to the build script, typically located in `util/buildscripts/` of the `dojo` distribution. You need to provide the location of the profile script, and a release directory. As an example:

```
./build.sh profileFile=/abs/path/to/build/profile.js releaseDir=path/to/releases/
```

The benefits to using custom builds are tremendous:

- Decreased load times for end users. Without a build, each `dojo.require` statement will produce an XMLHttpRequest round trip with the server, leading to often dozens of calls; with a build script, these are practically eliminated. Additionally, due to the build stripping out white space and comments and doing code optimizations, the final file size is a fraction of the cumulative file size.
- Faster JavaScript responses. The build process performs a number of code optimizations, resulting in snappier response times for JS operations.
- Optionally, you can also request build operations on CSS associated with your custom Dojo namespace. This will evaluate all `@import` statements and pull the CSS from those files into a single file, and strip comments and whitespace. The result is a single, smaller CSS file – again, leading to decreased load times for end users.

But where does Zend Framework fit into this?

Proposed Zend Framework integration

The `dojo()` view helper aggregates all the information needed by a build layer: the various `dojo.require` statements, `dojo.addOnLoad` events (if desired), and Dojo-related initialization javascript (if desired).

We propose a class that can take the metadata contained in the `dojo()` view helper in order to generate both a layer script and a build profile. This will simplify the deployment story for developers using Zend Framework and Dojo.

This class will also allow aggregating the results of visiting several pages with differing Dojo requirements, allowing developers to create

comprehensive build layers for their site.

2. References

- [Dojo Build System documentation](#)
- [My blog entry on layers](#)
- [Prototype code](#)

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **MUST** provide the ability to generate Dojo custom layer files.
 - The component **MUST** allow specifying the layer name
 - The component **MUST** allow specifying whether or not to include `dojo.addOnLoad` events and/or Dojo-related JavaScript aggregated in the helper
- The component **MUST** allow creating the build profile
 - The generated layer file **MUST** allow specifying the layer script path (relative to `dojo.js`)
 - The generated layer file **MUST** determine the dependent namespaces from the `dojo.require` statements
 - The component **MUST** allow specifying arbitrary build options
 - The component **MUST** provide default recommended build options
- This component **SHOULD** allow parsing an existing layer file into tokens
 - The component **SHOULD** check to see if new `dojo.require` statements and/or `javascript` have been registered; if so, it **MUST** create a composite file

4. Dependencies on Other Framework Components

- `Zend_Dojo_View_Helper_Dojo`
- `Zend_Dojo_View_Helper_Dojo_Container`

5. Theory of Operation

When configuring the `dojo()` view helper, you will be able to optionally specify an automated build layer. When doing so, you will be able to specify a variety of options:

- The layer namespace (e.g., "custom", "bug", "contact")
- The filesystem path to the layer (e.g., `APPLICATION_PATH . '/../public/js/custom'`); by default, this will be auto-discovered based on the path to `dojo.js` and the public directory (assumed to be `APPLICATION_PATH . '/../public'`).
- The path to the layer, relative to `dojo.js` (e.g., `'../custom'`); by default, this will be assumed to be the `'../$layerNamespace'`.
- The path to the build profile (e.g., `APPLICATION_PATH . '/../misc/$layerNamespace.profile.js'`); by default, this will be auto-discovered based on the path to `dojo.js`, and will be in `../util/buildscripts/profiles/$layerNamespace.profile.js`.
- Flag indicating whether only `dojo.require` statements should be part of the layer, or if all `javascript` captured by the `dojo()` view helper should be included. By default, only the `dojo.require` statements will be included.
- Flag indicating whether or not the helper should update an existing layer file.

Once you have, anytime a page is hit where the `dojo()` view helper is rendered, the layer file will either be created or updated, and a corresponding build profile will as well. It is then up to the developer to either manually generate the build from the build profile, or create an automated task for doing so.

6. Milestones / Tasks

- Milestone 1: Creation of proposal
- Milestone 2: Working prototype created targetting all **MUST** and **WILL** requirements, including unit tests
- Milestone 3: Updated prototype supporting all **SHOULD** and **COULD** requirements, including unit tests

- Milestone 4: Full documentation of new functionality
- Milestone 5: Review for inclusion in trunk

7. Class Index

- Zend_Dojo_View_Helper_Dojo_Layer
- Zend_Dojo_View_Helper_Dojo_BuildProfile
- Zend_Dojo_View_Helper_Dojo_Container (updates)

8. Use Cases

UC-01

Fully automated build layer

The example below creates a fully automated build layer in ../custom/main.js (relative to dojo.js). As the developer browses to pages that utilize different Dojo functionality, it will be updated. Additionally, it will create a build profile in ../util/buildscripts/profiles/custom.profile.js.

```
$view->dojo()->setLocalPath('/js/dojo/dojo.js')
->setAutoLayer(array('layerNamespace' => 'custom'));
```

UC-02

Aggregate javascript in the build layer

The example below builds on UC-01, and sets a flag telling the dojo() view helper to aggregate javascript to the build layer.

```
$view->dojo()->setLocalPath('/js/dojo/dojo.js')
->setAutoLayer(array(
    'layerNamespace' => 'custom',
    'includeJavascript' => true,
));
```

UC-03

Only create the build layer file once

The next example builds on the previous, and sets a flag indicating that the build layer file should only be created once, and never updated.

```
$view->dojo()->setLocalPath('/js/dojo/dojo.js')
->setAutoLayer(array(
    'layerNamespace' => 'custom',
    'includeJavascript' => true,
    'allowUpdates' => false,
));
```

UC-04

Manually specify where to place the build layer file

```
$view->dojo()->setLocalPath('/js/dojo/dojo.js')
->setAutoLayer(array(
    'layerNamespace' => 'custom',
    'includeJavascript' => true,
    'allowUpdates' => false,
    'layerSystemPath' => APPLICATION_PATH . '/public/js-src/custom',
    'layerWebPath' => '/js/custom/',
));
```

UC-05

Manually specify where to put the build profile file

```
$view->dojo()->setLocalPath('/js/dojo/dojo.js')
->setAutoLayer(array(
    'layerNamespace' => 'custom',
    'includeJavascript' => true,
    'allowUpdates' => false,
    'layerSystemPath' => APPLICATION_PATH . '/public/js-src/custom',
    'layerWebPath' => '../js-src/custom/',
    'buildProfilePath' => APPLICATION_PATH . '/misc/',
));
```

9. Class Skeletons

TBD

```
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>
```