DOM Utilities

Status:	Draft
Version:	1.0
Project owner:	Peter Higgins
Available:	since V1.3

Contents

- Introduction
- Locating and Using Nodes
  - Using ID's
  - CSS Selectors
  - The <body> element
  - document vs. dojo.doc
- Manipulating Nodes
  - Node styles
  - Node attributes
  - Node placement
- Creating Nodes
  - Basic Nodes
  - Scripts
  - Cross-browser Considerations
  - Destroying Nodes
- See also

Introduction

Dojo provides a host of DOM utility functions. This guide covers combined usage of dojo.style, dojo.create, dojo.attr, and a number of other base utilities for DOM convenience.

Locating and Using Nodes

All of these functions work by doing work on a passed DomNode reference (an element in the page (such as a div, li, a tag)). In order to work with these nodes, we first must locate them in the Document Object Model (DOM). The most important note to get started is all DOM activity must be done after "onload" fired, or the DOM is otherwise "ready". This can be achieved by dojo.addOnLoad

```
1  dojo.addOnLoad(function(){
2      console.log("page ready, can modify DOM anytime after this");
3  });
```

Once we've ensured our code will only execute after the DOM is ready, we can move on to locating nodes to work on.

Using ID's

The fastest, and most common, way of locating an element is by it's `id` attribute. Dojo provides dojo.byId - a shorter to write, safer to use version of `document.getElementById`:

```
dojo.addOnLoad(function(){
    var r = dojo.byId("someId");
    r.innerHTML = "Hi, World";
});
```

Here, we locate a node with id="someId", and set it's `innerHTML` to new content. If a node is not found, dojo.byId returns undefined, and is falsy enough to work as a conditional:

```
var r = dojo.byId("someId");
if(r){ /* it exists! */ }
```

CSS Selectors

It is not always practical to attach id's to elements you intend to modify, which is why Dojo provides dojo.query - a CSS3 selector engine. It will locate nodes that match a passed CSS selector, and return a specialized Array type - dojo.NodeList - that has oodles of useful DOM manipulation functions easily available.

```
dojo.addOnLoad(function(){
    var list = dojo.query("#someId");
});
```

In this example, we locate the same node found by `dojo.byId` before, but using the CSS selector to do so. The `query()` call returns a found list of nodes (in this case, a single element as id's are unique in the DOM). This list is a standard JavaScript Array, decorated with functions common throughout Dojo, most of which have single-node variants elsewhere in the toolkit. As we'll see, the more of the Core Dojo APIs you learn, the more powerful `NodeList` instances are:

```
dojo.addOnLoad(function(){

    // find a node byId, change the id, and set the color re
    dojo.query("#someId").attr("id","newId").style("color","red");

    // find a node byId, change the id, and set the color re
    var r = dojo.byId("someId");
    dojo.attr(r, "id", "newId");
    dojo.style(r, "color", "red");
```

```
});
```

The query method is convenient and more compact and both code snippets have identical results. There is some slight additional overhead to the convenience that `dojo.query` provides, but in general using `dojo.query` can save you time make your code easier to read.

CSS selectors are a handy and powerful way to find most kinds of elements in a page.

```
// by clas
dojo.query(".someClass");

// by attribute
dojo.query("[name^='link']");

// by tag typ
dojo.query("div");

// first-childre
dojo.query("ul > li");

// odd table rows
dojo.query("table tr:nth-child(odd)");

// scoped to some other node as paren
dojo.query("a.link", "someNode");
```

Finding nodes via `dojo.query` is very helpful, though the same rules apply to either "lists of nodes" or single nodes, as found by dojo.byId. We won't cover it here, but you can even use `dojo.query` to search XML documents!

The <body> element

All DOM's should contain a <body> element. This element is a DomNode like any other, and is considered the top most DOM Node in a document's visible content. This node is available via `dojo.query`:

```
dojo.addOnLoad(function(){
    dojo.query("body").addClass("tundra");
});
```

or more quickly available as a function call:

```
dojo.addOnLoad(function(){
    dojo.addClass(dojo.body(), "tundra");
});
```

By wrapping <body> lookup in the `dojo.body()` function, we are able to redefine the

meaning of the <body> element on the fly. This is useful for situations where your script might be working with multiple documents, such as the content of an iframe or a parent window. You can change the scope of a `dojo.query` by passing a different document as the context (second) argument:

```
dojo.addOnLoad(function(){
    var ifr = document.getElementById("anIframe");
    dojo.query("body", ifr.documentElement).addClass("tundra")
});
```

document vs. dojo.doc

It is very common to see the special global 'document' used throughout JavaScript code. When using Dojo, one should reference the global dojo.doc when accessing the current document. For instance:

```
// us
dojo.doc.createElement("div");
var h = dojo.doc.getElementsByTagName("head")[0];

// instead o
document.createElement("div");
var s = document.getElementsByTagName("head")[0];

// though, to just create, this is best
dojo.create("div");
```

They have identical length in characters, but by using `dojo.doc` you are able to ensure your code will execute in switched contexts, such as iframes and popups. To switch the context for code that uses `dojo.doc`, `dojo.body()`, and `dojo.global` see the documentation for dojo.setContext.

Manipulating Nodes

Once you are comfortable with the various ways of finding nodes in the DOM, applying the actions on them are somewhat trivial. Each of the DOM manipulation functions accept a String ID or a DomNode reference as the first parameter, and apply some action on that node.

Node styles

TODOC: see dojo.style

Node attributes

Each DomNode has a series of attributes available for setting and getting. A lot of times, you can access these properties knowing you are working with a native DomNode:

```
var r = dojo.byId("foo");
console.log(r.id == "foo"); // true
```

Though for full cross browser compatibility and convenience, it is recommended you access and set attributes through dojo.attr. The API for dojo.attr is straightforward:

```
// set some node to have a new id
dojo.attr(someNode, "id", "newId");

// get the id of a node reference
var id = dojo.attr(someNode, "id");

// set multiple attributes at once
dojo.attr(someNode, {
    id:"newId",
    onclick: function(e){ /* handler code */ }
});
```

The `style` attribute is special. One can set an "inline style" by setting an attribute on the DomNode directly:

```
<div style="padding:3px; color:red; height:123px'>Lorem, baby!</div>
```

Though this isn't an `attribute` per se. Just as dojo.style (described above) accepts an object-hash of style properties, you can pass a style:{} pair to dojo.attr, and set styles in a dojo.attr call:

```
dojo.attr(someNode, {
    name:"bar",
    style:{
        color:"#ededed", fontSize:"13pt"
    },
    id:"newId"
});
```

Node placement

Where a node exists in the DOM is important.

Creating Nodes

Basic Nodes

The DOM document contains functions for creating each of the basic DOM node types.

Type	Function

Type	Function
element	createElement(tagName)
text	createTextNode(data)
comment	createComment(data)
attribute	createAttribute(name)

For example, the following fragment will create a new DOM node of for the HTML DIV tag.

```
1  dojo.doc.createElement("DIV");
```

Scripts

Cross-browser Considerations

Destroying Nodes

In order to destroy a single node or a list of nodes, Dojo Toolkit provides the following functions:

- Since V1.3:
  - dojo.destroy(node)
- From V1.0 to V1.2:
  - dojo._destroyElement(node);

See also

Here are a list of Web references on the subject of DOM

- Document Object Model (DOM) Technical Reports
  The official specifications from the W3 on the DOM.