Dojo from the ground up, Part 1: Getting started with Dojo development

Joe Lennon
Lead Mobile Developer
Core International

Skill Level: Introductory

Date: 18 Jan 2011

The Dojo toolkit enables web application developers to create Rich Internet Applications by offering a wide variety of features that save development time and effort. From DOM helpers and Asynchronous JavaScript and XML (Ajax) to a full-blown widget library and object-orientation features, Dojo includes virtually everything you need to build large-scale Ajax-powered web applications. If the functions you are looking for are not included in Dojo itself, it's likely that you can find them in DojoX, a repository of extensions and experimental features that are not included in the Base or Core modules of the toolkit. In this article series, learn how to develop Dojo-powered applications from the ground up, covering the basics, Dojo's great object-orientation features, and the Dijit user interface library. By the end of this series, you will be well prepared to develop Dojo applications of your own.

View more content in this series

Introducing the Dojo Toolkit

Get started with Dojo development http://www.ibm.com/developerworks/training/kp/wa-kp-dojo/index.html
Dojo was created in 2004 to make the process of developing DHTML and JavaScript web applications easier, hiding much of the cross-browser inconsistencies that are prevalent in modern web browsers. This enabled the focus to be placed on implementing functions rather than tweaking code to make it work on every browser. Dojo is owned by the Dojo foundation, which was founded in 2005 by Alex Russell and Dylan Schiemann. Dojo is open source software (OSS) and is available under a dual-license (you can pick which one you want to adhere to) with both the Academic Free License (AFL) and a modified BSD license available.

Features at a glance

The features of the Dojo Toolkit are spread into four distinct sections. This division lets developers keep the file size of the library to a minimum, ensuring that the performance of their application is not burdened by a heavy JavaScript library download. For example, if you only need Ajax support functions, you only need to include the Base package; the extensive Dijit UI components will not be included. You will learn more about how Dojo loads different modules later in this series.

Base
: The Base package provides the foundation of the Dojo Toolkit, including functions such as DOM utility functions, CSS3 selector-based DOM querying, event handling, Ajax, basic animation, and Dojo's class-based object-oriented features. The majority of this article focuses on Base.

Core
: The Core package includes some additional functions that are not included in Base. Generally, these features tend not to be used as heavily as the functions Base provides; hence, they are loaded separately to ease the burden on the Base package. With that said, the Core package provides some really useful components including advanced animation, drag and drop, I/O, data management, internationalization (i8n), and browser history management. The Core package is not covered in this article.

Dijit
: The Dijit package includes Dojo's extensive UI library of widgets and components. Some examples of these widgets include dialog boxes, calendars, color palettes, tooltips, and trees. It also includes a set of form controls that provide much more functionality than the standard HTML form controls, as well as complete layout management options. The third part of this series will feature more in-depth coverage of Dijit.

DojoX
: Dojo eXtensions (DojoX) contains various sub-projects of the toolkit. Most of the experimental features of Dojo live in DojoX, but it also contains many stable components and features. DojoX will be covered briefly in the third part of this series.

Getting started

Enough talk about Dojo. Let's get started and actually work with the toolkit. This section explains the various ways of using Dojo in your projects, how to set up a development environment using Firefox and the Firebug plug-in, and how to write some Dojo code to make sure everything is working as expected.

Setting up Dojo

The easiest way to set up Dojo is to serve it from a Content Delivery Network (CDN), which will deliver the Dojo JavaScript files from a server near the visitor rather than from your own server. Not only does this help speed up the loading of the script, it also means that there is an increased chance that the user has already loaded the Dojo files from another website, which will lead to them being loaded from cache, speeding things up even further.

In this series, it is assumed that you are using Dojo 1.5, although any 1.x version should be compatible. Including the following `<script>` tag in your HTML page will load Dojo 1.5 from Google's CDN:

```
<script src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js"></script>
```

Alternatively, you can download Dojo to your own server and load it from there. My preferred method is to load from a CDN, but to have a local fallback should something go wrong on the CDN server. To do this, download Dojo and place the files in a suitable place relative to where you will store your web content. Assuming the relative path from your web content to the Dojo script files is "script/", the code in Listing 1 will try to load Dojo from the CDN first, and if it fails it will load the local version instead.

Listing 1. Loading Dojo from CDN with local fallback

```
<script src="https://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js">
</script>
<script>
typeof(dojo) === "undefined" && document.write(unescape('%3Cscript
src="js/libs/dojo-1.5.min.js"%3E%3C/script%3E'))
</script>
```

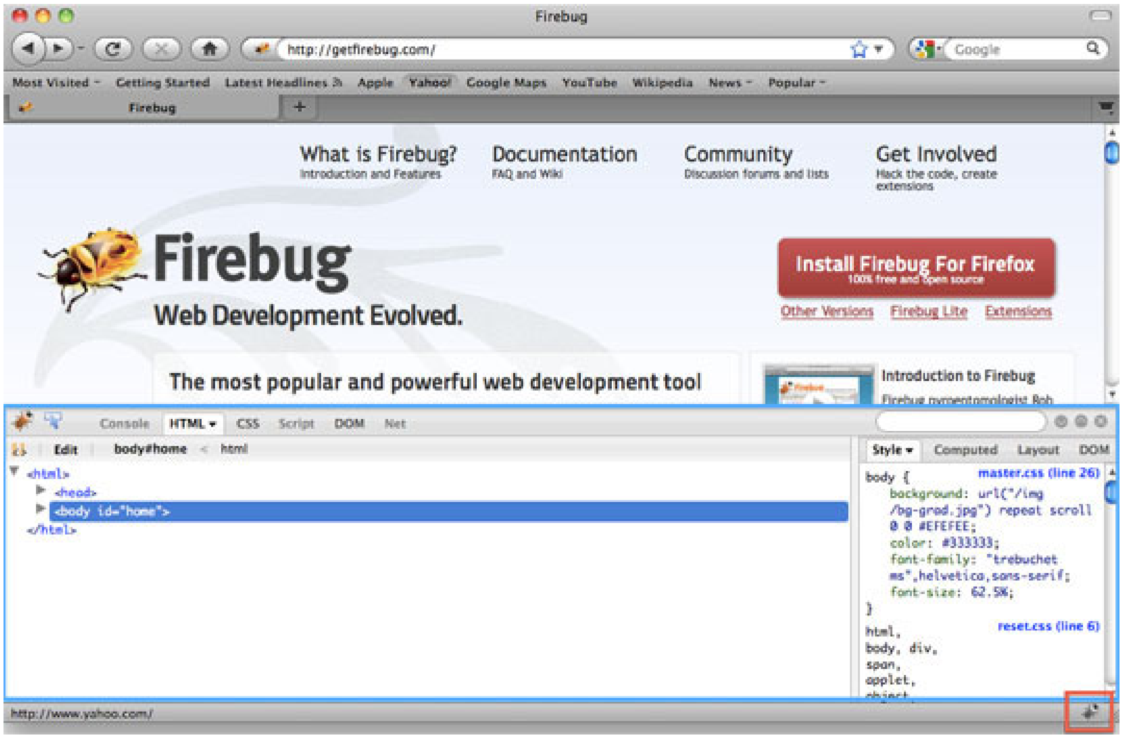
It's important that the code starting with `typeof(dojo)` is all placed on a single line; otherwise, it won't work. If you want to test that the fallback is working, simply comment out the line that loads from the CDN and test your page with the "Hello, World!" example you'll create in a few moments.

Using the Firebug console

Rather than needing to create a web page to try out Dojo, many of the examples in this article are performed using an excellent Firefox plug-in, Firebug. Firebug provides JavaScript developers with full console logging, debugging, and network monitoring facilities, which make the process of testing and fixing issues in JavaScript applications much easier. Some other web browsers actually include these features, but in this example I use Firefox and Firebug as they are available on a wide variety of platforms.

First, you must install Firefox and Firebug (see Resources for download information). After Firefox is installed, launch it and then download and install the Firebug plug-in. With Firebug installed, you should see a bug icon like the one shown in Figure 1 in the lower right corner of your Firefox window. Clicking this icon should open the Firebug window.

Figure 1. Firebug icon and window

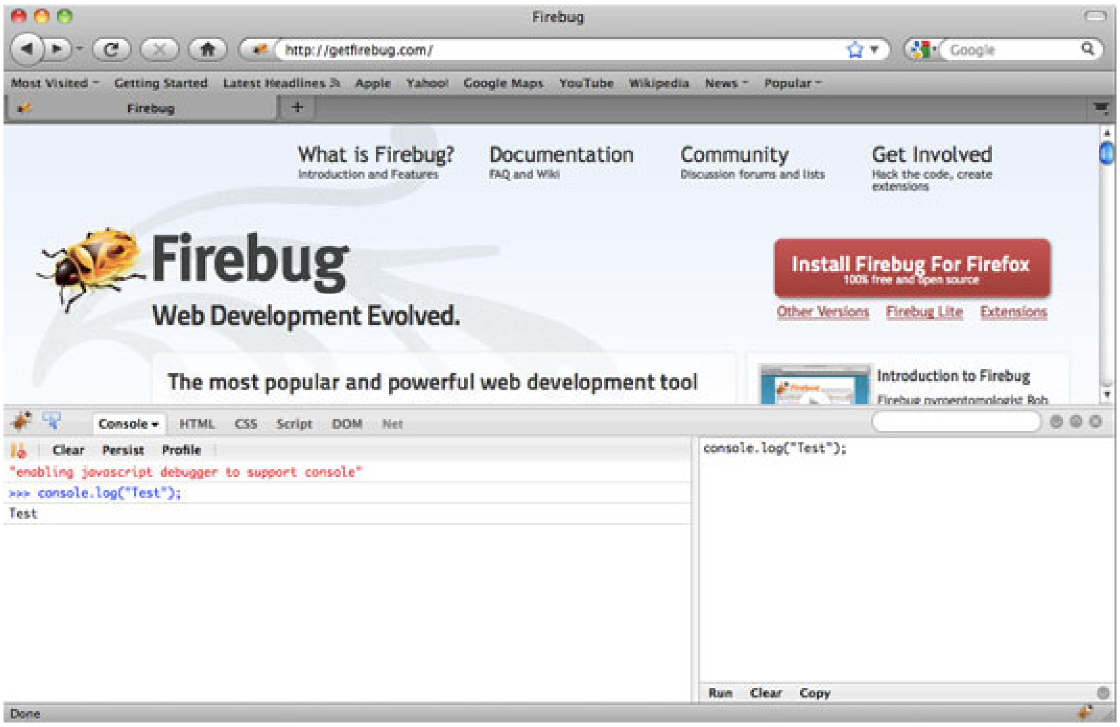

```
<html>
  <head>
  <body id="home">
</html>
```

```
body {                          master.css (line 26)
  background: url("/img
  /bg-grad.jpg") repeat scroll
  0 0 #EFEFEE;
  color: #333333;
  font-family: "trebuchet
  ms",helvetica,sans-serif;
  font-size: 62.5%;
}
html,                            reset.css (line 6)
body, div,
span,
applet,
```

To use the Firebug console, click the Console tab. You may need to enable the console first by clicking the down arrow in the tab and clicking **Enable**. Now, in the console window, enter the following code: `console.log("Test");`.

You should see a result like the one shown in Figure 2, with a message that the JavaScript debugger is being enabled to support the console.

Figure 2. Using the Firebug console


Firebug
http://getfirebug.com/
Most Visited Getting Started Latest Headlines Apple Yahoo! Google Maps YouTube Wikipedia News Popular
What is Firebug? Introduction and Features
Documentation FAQ and Wiki
Community Discussion forums and lists
Get Involved Hack the code, create extensions
Firebug
Web Development Evolved.
Install Firebug For Firefox 100% free and open source
Other Versions Firebug Lite Extensions
The most popular and powerful web development tool
Introduction to Firebug
Console HTML CSS Script DOM Net
Clear Persist Profile
Run Clear Copy
Done

```
"enabling javascript debugger to support console"
>>> console.log("Test");
Test
```

```
console.log("Test");
```

Hello, World!

Next up, let's test out Dojo. For the "Hello, World!" example, you'll use Dojo to attach a function when the DOM has finished loading (see Listing 2). This function simply prints a message "Hello, World!" to the end of the page. Okay, so this won't exactly change the way you create web applications, but what it will do is allow you to ensure that Dojo is available on your page. You will also load this page throughout this article to explore Dojo features using the Firebug console.

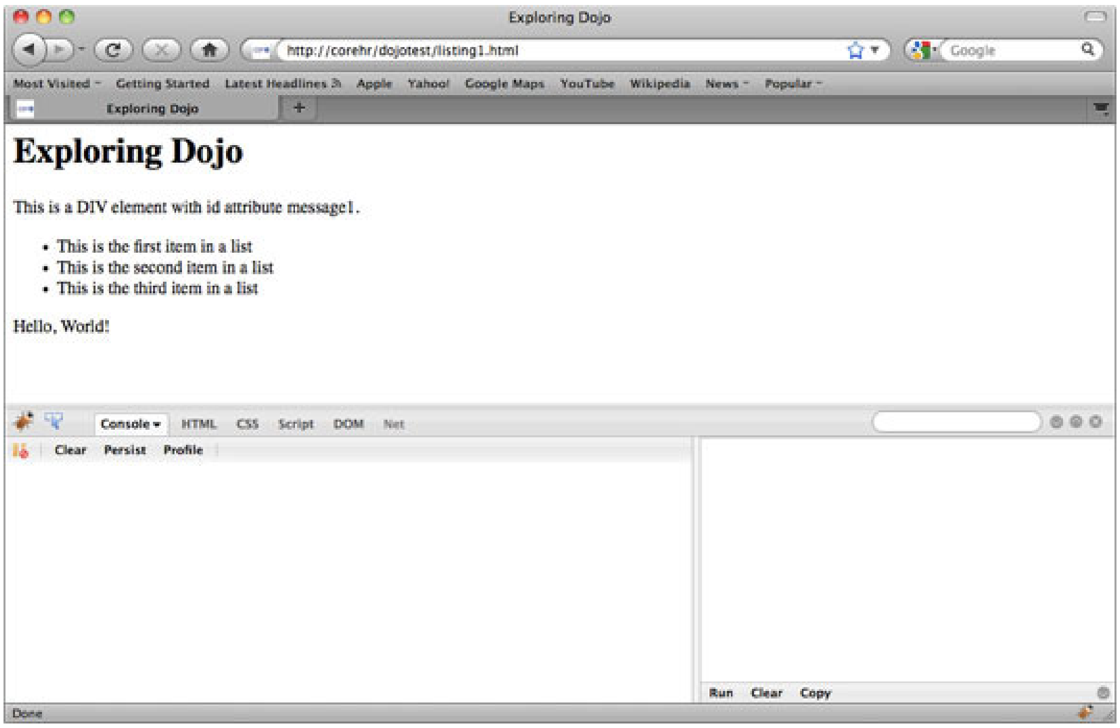
Listing 2. listing1.html: Dojo Hello World application

```
<html>
<head>
    <title>Exploring Dojo</title>
</head>
<body>
<h1>Exploring Dojo</h1>
<div id="message">This is a DIV element with id attribute message.</div>
<ul id="list">
    <li>This is the first item in a list</li>
    <li class="highlight">This is the second item in a list</li>
    <li>This is the third item in a list</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js">
</script>
<script>
dojo.addOnLoad(function() {
    dojo.create(
        "div",
        {
            "innerHTML": "Hello, World!"
        },
        dojo.body()
    );
});
```

```
</script>
</body>
</html>
```

This script will attach an `anonymous` function that fires when the DOM has finished loading. This function uses `dojo.create` to build a new DOM `<div>` element, setting its `innerHTML` property to "Hello, World!", and inserting it into the page body using the utility function `dojo.body`. The result of this is shown in Figure 3. You'll notice that the "Hello, World!" message has been appended to the end of the page body.

Figure 3. The "Hello World" page in all its glory



Exploring Dojo

This is a DIV element with id attribute message1.

- This is the first item in a list
- This is the second item in a list
- This is the third item in a list

Hello, World!

Don't delete this page or close it for now. In the next section, you will be using this page for the sole purpose of loading Dojo, and you will try out some aspects of Dojo directly using the Firebug console.

Before moving on to the next section, it's important to point out something about the positioning of the JavaScript on the page. You may notice that rather than including Dojo and the page JavaScript in the `<head>` section of the HTML document, I have added it to the end of the page, just before the closing `<body>` element. The reason for this is that when JavaScript is placed there, it will not block other elements of the page from loading. When using JavaScript libraries in your page, it's important to ensure that they don't block other parts of your page from loading so the performance of your page is optimized. For further information on this, see Resources.

The basics

In this section, you will learn some of the useful functions Dojo provides that make it easier to work with the DOM and arrays. To try out the examples in this section, keep the page you created in the last section open in Firefox and type the code presented in the Firebug console window.

DOM utility functions

The DOM utility functions make it easier to work with the elements in the DOM by providing the ability to find items by their ID or using CSS3 selectors. There are also functions for creating and destroying elements as well as for manipulating the contents of existing elements.

`dojo.byId`

The `dojo.byId` function lets you select a DOM node by its `id` attribute. This function is an alias to the standard JavaScript `document.getElementById` function, but is obviously shorter to write and also takes care of some cross-browser inconsistencies. Let's use the Firebug console now to log the contents of the DOM element with the ID "message": `dojo.byId("message").innerHTML;`.

In the left side of the console you should see the response shown in Listing 3.

Listing 3. Response

```
>>> dojo.byId("message").innerHTML;
"This is a DIV element with id attribute message1."
```

The first line in Listing 3 simply echoes the command that was issued. The second line is the result of that command, in this case, the contents of the `<h1>` element with ID "message".

Before moving on, let's do something a bit more interesting. In the console editor area, enter the command shown in Listing 4.

Listing 4. `dojo.fadeOut` command

```
dojo.fadeOut({
    node: dojo.byId("message"),
    duration: 600
}).play();
```

You should see the element fade out and disappear from the page. To fade it back in, change the `fadeOut` reference in the command to `fadeIn` and run it again. Pretty neat, huh? This demonstrates how you can dynamically manipulate the current page using the Firebug console. Of course, these changes only apply to the current page load and won't be persisted if you navigate away from the page or refresh it.

`dojo.query`

In the last section, you learned how to get a reference to a single DOM element using its `id` attribute. As useful as this is, it isn't feasible to add this attribute to every element you want to interact with. Also, an `id` must be unique. So what if you want to reference several elements at once? This is where the `dojo.query` function steps in.

Say you want to get a reference to all of the `<li>` children of the unordered list with the ID "list" in your page and print out the contents of each of these elements to the console. Thanks to `dojo.query`, this is really simple (see Listing 5).

Listing 5. `dojo.query` command

```
dojo.query("#list li").forEach(function(item) {
    console.log(item.innerHTML);
});
```

This should produce the output shown in Listing 6 in the console.

Listing 6. Output in the console

```
>>> dojo.query("#list li").forEach(function(item) { console.log
                                                       (item.innerHTML); });
This is the first item in a list
This is the second item in a list
This is the third item in a list
[li, li.highlight, li]
```

The `dojo.query` function accepts a string argument with a CSS3 selector reference to the elements that you want to select. In this particular instance, you are saying that you want to select all `li` elements that are children of the element with the ID "list". The function returns an array of elements that match the query. In the example in Listing 6, you use the `dojo.forEach` function to iterate over this array and log the `innerHTML` property of each element found to the console. You will learn more about this function and other array functions in the next section.

Before moving on, let's use `dojo.query` to find any HTML elements with the class name `highlight` and apply some styling to make them stand out (see Listing 7).

Listing 7. Using `dojo.query` to find any HTML elements with the class name `highlight`

```
dojo.query(".highlight").style({
    backgroundColor: "yellow",
    color: "red"
});
```

You should notice that the second item in the unordered list on the page changes to have a yellow background, with the text color changing to red. You may notice that in this case the `dojo.forEach` function was not used. I will cover why this was not needed in the next section, "Arrays and NodeLists."

Other useful utility functions

In addition to the DOM querying and element selection, there are a number of other utility functions available in Dojo that make working with the DOM much easier. You have already seen a couple of these in the "Hello, World" example. The `dojo.body` function simply returns a reference to the `<body>` element of the document, `dojo.body`, and returns the document object itself. `dojo.create` lets you quickly create a new element, define its attributes, and place it in the DOM.

Other functions that exist include `dojo.place`, which lets you place existing or new elements anywhere in the document. `dojo.empty` does just what you'd expect it to —it empties the content of a DOM element. `dojo.destroy` removes a node, taking any child elements along with it. For more information on any of these functions, see Resources to get a link to the Dojo reference documentation.

Arrays and NodeLists

Arrays let you store a collection of values and are available in standard JavaScript. In Dojo, arrays are extended to include several helper functions. These extended arrays are referred to as NodeLists. A NodeList can make use of any standard array functions as well as the additional Dojo-specific functions. When you use the `dojo.query` function described in the previous section, the return value is a NodeList (or `dojo.NodeList` to be specific) object. Let's look at some of the functions available in NodeLists.

`dojo.forEach`

The first function worth discussing is the `dojo.forEach` function, which you have already seen in the `dojo.query` example from the previous section of this article. This function lets you define an iterator on a NodeList, supplying a function that will be applied to each item in the NodeList. Let's look at a more basic example in Listing 8.

Listing 8. Basic example

```
var list = ['My','name','is','Joe'];
dojo.forEach(list, function(item, i) {
    console.log((i+1)+'. '+item);
});
```

This code in Listing 8 produces the output shown in Listing 9.

Listing 9. Output

```
>>> var list = ['My','name','is','Joe']; dojo.forEac...item, i)
                             { console.log((i+1)+'. '+item); });
1. My
2. name
3. is
4. Joe
```

As you can see, the `forEach` function takes each item in the array and performs the attached function on it. Above, you used an `anonymous` function, but you could also use a named function, as shown in Listing 10.

Listing 10. Iterating over an array with `dojo.forEach` using a named function

```
var list = ['My','name','is','Joe'];

var printArray = function(item, i) {
    console.log((i+1)+'. '+item);
}

dojo.forEach(list, printArray);
```

`dojo.indexOf`

The `dojo.indexOf` function lets you find what position in an array a particular value is at. The best way to illustrate it is by way of example. Using the list array created in the previous section, try to work out the index of the array the value `name` is at: `dojo.indexOf(list, "name");`.

This returns the result shown in Listing 11.

Listing 11. Result

```
>>> dojo.indexOf(list, "name");
1
```

So the `name` value is at index 1 in the array. Remember that JavaScript arrays start at index 0, so this value is the second item in the array. If you try to use this function supplying a value that isn't in the array at all, the return value is `-1`.

This function returns the first index found for the given value, so if there are multiple items in the array with the same value, it will stop at the first item. Dojo provides a similar function, `dojo.lastIndexOf`, which lets you find the last index of a particular value. This function works the exact same way as `dojo.indexOf`.

`dojo.filter`

The `dojo.filter` function enables you to create a new array that is a filtered version of another array. For example, if you wanted to create a new version of the list array you created earlier, but exclude any items with the value `is`, you could use the code shown in Listing 12.

Listing 12. Filtering an array to create a new array

```
var filteredList = dojo.filter(list, function(item) {
    return item != "is";
});

dojo.forEach(filteredList, "console.log(item)");

This results in the following output:

>>> var filteredList = dojo.filter(list,
function(it...dojo.forEach(filteredList, "console.log(item)");
My
name
Joe
```

Other NodeList functions

Dojo contains some other NodeList functions that are useful when working with arrays. The `dojo.map` function lets you create a new array that is a modified version of an existing array. For example, you might have an array of numbers that represent monetary values. You could use a map function to return an array of these values in a currency format. `dojo.some` lets you check if at least one item in an array matches specified criteria. Similarly, `dojo.every` is used to check if every item in an array matches specified criteria. For a full list of NodeList functions and their associated documentation, see Resources.

Event handling in Dojo

Most JavaScript libraries have a cross-browser implementation of native JavaScript event handling, allowing you to attach functions to be called when a DOM event is triggered. As useful as this may be, Dojo takes the concept a step further by letting you connect functions to other functions, which can be DOM events, object events, user-defined functions, or "topics," which are discussed later in this section.

DOM event handling

The first way of attaching functions to DOM events is to use the `dojo.connect` function. The best way to illustrate this is by way of example. In the Firebug console, enter the code in Listing 13.

Listing 13. Attaching functions to DOM events using `dojo.connect`

```
var message = dojo.byId("message");
dojo.connect(message, "onclick", function() {
    alert(message.innerHTML);
});
```

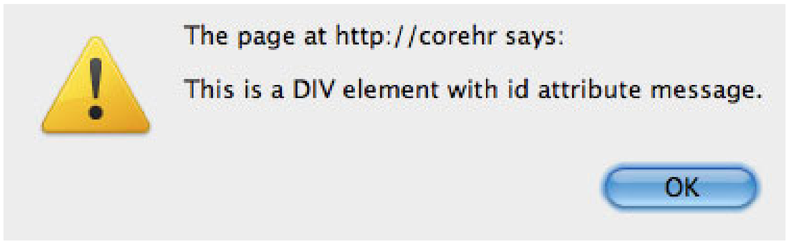
This results in the output shown in Listing 14 in the console.

Listing 14. Output

```
>>> var message = dojo.byId("message"); dojo.connect..., function()
                                              { alert(message.innerHTML); });
[div#message, "onclick", function(), 1]
```

That's great and all, but isn't the function supposed to actually do something? Absolutely. Dojo has attached a function to the `click` event handler of the element with the ID "message". To try it out, click on the message on the screen with the content "This is a DIV element with id attribute message." You should see a JavaScript alert box, as shown in Figure 4.

Figure 4. Attaching functions to DOM events


The page at http://corehr says:

This is a DIV element with id attribute message.

Nice and easy, isn't it? What if you want to attach an event to all the items in an array? For example, let's say you wanted each item in the unordered list on your page to highlight in bold when you click on them. You could easily do this with the code in Listing 15.

Listing 15. Attaching events to an array of elements

```
dojo.query("#list li").forEach(function(item) {
    dojo.connect(item, "onclick", function() {
        dojo.style(item, {
            fontWeight: "bold"
        });
    });
});
```

Try it out, it works. Dojo lets you write this piece of code in an even more concise way. Instead of using `forEach` to iterate over the array, you can use a `NodeList.connect` shortcut function to do it instead, as shown in Listing 16.

Listing 16. Attaching events to an array of elements (improved)

```
dojo.query("#list li").onclick(function(e) {
    dojo.style(e.target, {
        fontWeight: "bold"
    });
});
```

Because you have already attached an event to the list, you should refresh the page before trying the code in Listing 16 to make sure it works. The `e` argument is a reference to the `Event` object, and the `target` property of this object lets you identify the element that fired the event. You use this to identify the element that the bold styling should be applied to. Try clicking on the three list items; each of them should become bold after you click them.

Connecting functions to other functions

In the previous examples, you connected functions to DOM events. Dojo also lets you connect functions to other functions in the same manner. An example of this might be a function that shows a spinning wheel image somewhere on your page. When a user performs an Ajax function, you want to show this image. Similarly, when the function has returned a response, you want to hide the image. Without using `dojo.connect`, your code might look like Listing 17.

Listing 17. Connecting functions to other fucntions without `dojo.connect`

```
function toggleImage() {
    //Code to show/hide loading image goes here
}

function callAjax() {
    toggleImage();
    //Code to call Ajax function goes here
}

function handleResponse() {
    //Code to handle Ajax response goes here
    toggleImage();
}
```

While there's nothing wrong with this code, the `toggleImage` function call is fixed in both the `callAjax` and `handleResponse` functions. If you wanted to add another function call, you would have to modify these functions again to include the extra call. Instead of adding the function call to the functions themselves, you can use `dojo.connect` to establish the link between them. Listing 18 shows how the `dojo.connect` method might look.

Listing 18. Connecting functions to other functions with `dojo.connect`

```
function toggleImage() {
    //Code to show/hide loading image goes here
}

function callAjax() {
    //Code to call Ajax function goes here
}

function handleResponse() {
    //Code to handle Ajax response goes here
}

dojo.connect(callAjax, toggleImage);
dojo.connect(handleResponse, toggleImage);
```

This style of coding might not be to every developer's taste, but it allows you to organize your code in a manner that makes it much easier to read.

Publishing and subscribing to *topics*

The final aspect of Dojo event handling worth mentioning is the ability to publish and subscribe to topics. This lets Dojo components interact with each other, even if they are not aware of each other's existence. For example, let's say that you were defining a topic named `printName`, which expects a `message` object with a person's first name and last name. You might have a component that subscribes to this topic, which will print the name to the console any time another component publishes to this topic with a person's name. Listing 19 shows an example of such a subscription (feel free to try it in Firebug).

Listing 19. Subscription

```
dojo.subscribe("printName", function(msg) {
    console.log("The person's name is: "+msg.first_name+" "+msg.last_name);
});
```

To publish to this topic, you need to pass an array of objects that adhere to the topic's API (in this case, the objects must have a first name and last name). Listing 20 is an example.

Listing 20. Publishing to a topic

```
dojo.publish("printName", [
    {
        first_name: "Joe",
        last_name: "Lennon"
    }
]);
```

This produces the output shown in Listing 21.

Listing 21. Output

```
>>> dojo.publish("printName", [ { first_name: "Joe", last_name: "Lennon" } ]);
The person's name is: Joe Lennon
```

As you can see, by publishing this object to the `printName` topic, your subscribed function has output a corresponding message to the console.

Empowering Ajax with dojo.xhr*

Creating Ajax-powered web applications is typically done by creating an `XmlHttpRequest` (`XHR`) object, which will make an HTTP request to a given URL, passing a request header and body, and defining callback functions to define what should be done when the response returns with a successful response body or an HTTP failure response. Implementing cross-browser XHRs can be painful to say the least, but, thankfully, Dojo eases that pain significantly with a set of `dojo.xhr*` functions that let you make `GET`, `POST`, `PUT`, and `DELETE` requests.

The four functions provided are as follows:

- `xhrGet`
- `xhrPost`
- `xhrPut`
- `xhrDelete`

All of these functions follow the same syntax: accepting a single property configuration object as an argument. In this object you can define the various aspects of the Ajax request you want to make. Again, these options are the same across all of the XHR functions.

Configuration options

Some of the more useful configuration options available to the `XHR` functions are as follows:

`url`
This is the URL the HTTP request should be made to. It must live in the same domain and port combination as the page that is making the request.

`handleAs`
Lets you define the format that the response should be treated as. The default is `text`, but `json`, `javascript`, `xml`, and a couple of other options are also available. You will see an example of creating an Ajax request with a callback function that handles a JSON response format later in this section.

`form`
Either a reference to or the string ID representation of a `<form>` element. The values of each field in the form will be sent along with the request as the request body.

`content`
: An object containing the parameters you want to pass to the resource in the request body. This object will be mixed in with the values taken from the `form` property if both are provided.

An example of an `XHR` function is shown in Listing 22.

Listing 22. Example of an XHR function call

```
dojo.xhrGet({
    url: "save_data.php",
    content: {
        id: "100",
        first_name: "Joe",
        last_name: "Lennon"
    }
});
```

This will asynchronously make an HTTP `GET` request to the save_data.php file in the same location as the document itself. It will pass the properties of the content object to the PHP script as parameters. In PHP, you would then use the `$_GET` variable to retrieve these values and perhaps save them to a database.

Callback functions

In the previous example, you learned how you might call an Ajax request using `dojo.xhrGet`. While the example was sufficient for actually making the request, it provided no facility to handle any response. Callback functions are also passed to the configuration object. The following options are available:

`load`
: This function will be executed when an Ajax request returns a successful response message. The response data and the request object are passed to this function as arguments.

`error`
: This function will be executed if there is a problem with the Ajax request. This can occur if the URL specified in the Ajax request is invalid, if the request times out, or if some other HTTP error occurs. The error message and the request object are passed as arguments.

`handle`
: This function allows you to combine the load and error callback functions into a single function (useful if you don't really care whether the request results in a success or error).

In the next example, you will create an Ajax call with a load callback function that will load some data from a JSON file and print it to the page.

Working with JSON data

Let's put the `dojo.xhr*` functions to a better test by creating a more real example. First, create a new file—put this file in the same directory as the listing1.html file—and add some JSON data to it. The contents of the file are shown in Listing 23.

Listing 23. data.json — JSON data to be processed by an Ajax request

```
{
    count: 4,
    people: [
        {
            first_name: "Joe",
            last_name: "Lennon",
            age: 25
        },{
            first_name: "Darragh",
            last_name: "Duffy",
            age: 33
        },{
            first_name: "Jonathan",
            last_name: "Reardon",
            age: 30
        },{
            first_name: "Finian",
            last_name: "O'Connor",
            age: 23
        }
    ]
}
```

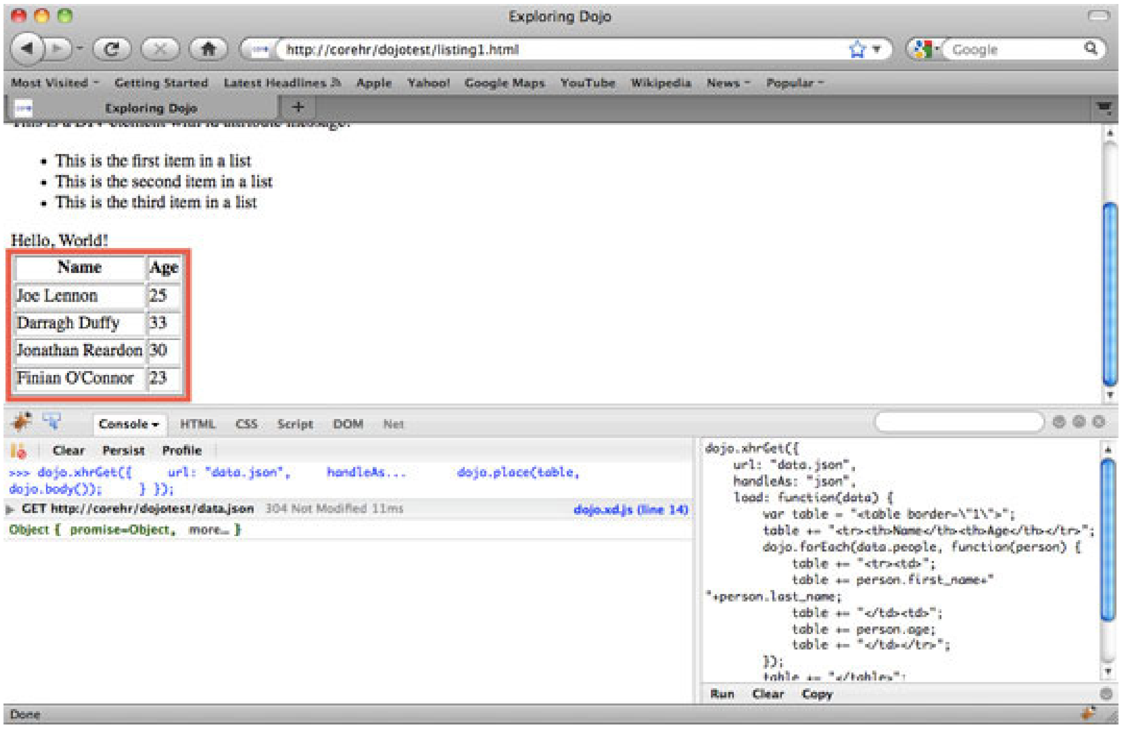
Now create an Ajax request in Firebug (make sure the listing1.html page is loaded in Firefox so that Dojo is loaded). This request uses the `load` callback function to process the JSON response and print a table to the page (see Listing 24).

Listing 24. Using Ajax to load and process the JSON data

```
dojo.xhrGet({
    url: "data.json",
    handleAs: "json",
    load: function(data) {
        var table = "<table border=\"1\">";
        table += "<tr><th>Name</th><th>Age</th>
</tr>";
        dojo.forEach(data.people, function(person) {
            table += "<tr><td>";
            table += person.first_name+" "+person.last_name;
            table += "</td><td>";
            table += person.age;
            table += "</td></tr>";
        });
        table += "</table>";
        dojo.place(table, dojo.body());
    }
});
```

Try the code shown in Listing 24 in Firebug. A table should be dynamically added to your page with the data loaded from the JSON file. This is shown in Figure 5.

Figure 5. Ajax-loaded table from JSON data



- This is the first item in a list
- This is the second item in a list
- This is the third item in a list

Hello, World!

Name	Age
Joe Lennon	25
Darragh Duffy	33
Jonathan Reardon	30
Finian O'Connor	23

```
>>> dojo.xhrGet({     url: "data.json",     handleAs...      dojo.place(table,
dojo.body());     } });
GET http://corehr/dojotest/data.json   304 Not Modified 11ms      dojo.xd.js (line 14)
Object { promise=Object,  more... }
```

```
dojo.xhrGet({
    url: "data.json",
    handleAs: "json",
    load: function(data) {
        var table = "<table border=\"1\">";
        table += "<tr><th>Name</th><th>Age</th></tr>";
        dojo.forEach(data.people, function(person) {
            table += "<tr><td>";
            table += person.first_name+"
"+person.last_name;
            table += "</td><td>";
            table += person.age;
            table += "</td></tr>";
        });
        table += "</table>";
```

In a real-world example you would use a server-side language like PHP, Python, ASP.NET, or Java to generate the JSON data dynamically based on the parameters passed to it by the Ajax request.

Conclusion

In this part of the Dojo from the ground up article series, you learned about Dojo and the basics of how to use it. In particular, you learned about the various DOM utility functions, array functions, event handling, and XHR features. In the next part of the series, you will learn how to use Dojo's Java™-like class-based object orientation features.

Downloads

Description	Name	Size	Download method
Article source code	dojo.ground.1.zip	1KB	HTTP

Information about download methods

Resources

Learn

- Visit the home page for the Dojo Toolkit.
- Check out some Dojo Toolkit Demos.
- Introducing The Dojo Toolkit: Read an excellent introduction to the Dojo Toolkit from the Opera developer site.
- Introduction to the Dojo toolkit, Part 1: Setup, core, and widgets: Read another fine introduction to the Dojo toolkit from Javaworld.
- Dojo 1.5: Ready to power your web app: Learn about some of the new features in Dojo 1.5 from this article on Sitepen.
- Introduction to the Dojo Toolkit: Tutorial: Read an introductory tutorial from Ajax Matters.
- "Internationalizing web applications using Dojo" (developerWorks, August 2008): Discover a way to perform native language support in the context of web sites and web applications using the i18n feature of the Dojo toolkit.
- "Consuming web services with the Dojo Toolkit" (developerWorks, September 2010): Learn how to consume services using the Dojo Toolkit to enable Ajax on a web page.
- Take a look at *Dojo: Using the Dojo JavaScript Library to Build Ajax Applications* by James E. Harmon.
- "Writing a custom Dojo application"(developerWorks, December 2008): Find out much more about Dojo in this developerWorks article.
- "Develop HTML widgets with Dojo" (developerWorks, October 2006) explores Dojo's extensibility.
- "Using the Dojo Toolkit with WebSphere Portal"(developerWorks, November 2007) describes how to install, configure, use, and leverage the Dojo Toolkit in WebSphere Portal applications.
- The developerWorks Web development zone specializes in articles covering various web-based solutions.

Get products and technologies

- Download the Dojo Toolkit. Version 1.5 was used in this article.
- Access the Dojo Toolkit API documentation.
- Get Firefox.
- Get Firebug.
- IBM - Dojo Extension sample: The Dojo Extension Feature Set can be used to enable a IBM WebSphere Portlet Factory model to leverage functionality provided by the Dojo JavaScript Toolkit.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Create your My developerWorks profile today and set up a watchlist on Dojo. Get connected and stay connected with My developerWorks.
- Find other developerWorks members interested in web development.
- Share what you know: Join one of our developerWorks groups focused on web topics.
- Roland Barcia talks about Web 2.0 and middleware in his blog.
- Follow developerWorks' members' shared bookmarks on web topics.
- Get answers quickly: Visit the Web 2.0 Apps forum.

About the author

Joe Lennon

Joe Lennon is a 25-year-old mobile and web application developer from Cork, Ireland. Joe works for Core International, where he leads the development of Core's mobile HR self service solutions. Joe is also a keen technical writer, having written many articles and tutorials for IBM developerWorks on topics such as DB2 pureXML, Flex, JavaScript, Adobe AIR, .NET, PHP, Python and much more. Joe's first book, *Beginning CouchDB* was published in late 2009 by Apress. In his spare time, Joe enjoys travelling, reading and video games.