Why Dojo?

Friday, October 30th, 2009
This entry is part 1 of 8 in the series Dojo Quick Start Guide

The Dojo Toolkit is an open-source JavaScript toolkit useful for building great web applications. It aims to shorten the timespan between idea and implementation by providing an exceptionally well conceived API and set of tools for assisting and fixing the issues experienced in everyday web development. It is lightning fast, extremely robust, and supplies a solid set of tools for DOM manipulation, animations, Ajax, event and keyboard normalization, internationalization (i18n) and accessibility (a11y). Dojo Base is a single, lightweight 27KB entity "across the wire." Dojo is completely free, liberally licensed (AFL or BSD), and transparently developed by an active group of developers with a strong community presence.

(more…)

SHARE

Posted in Dojo | 2 Comments

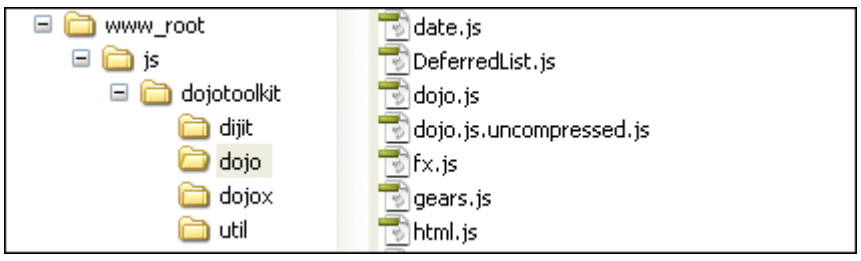
Getting the Code

Friday, October 30th, 2009
This entry is part 2 of 8 in the series Dojo Quick Start Guide

Download the newest released version of the Dojo Toolkit. Or find a previous release if necessary.

The optimized "built" version is: `dojo-release-x.y.z.zip`, where x, y, and z are major, minor, and point release numbers, respectively. Or to run the complete suite of tests, you should download the SDK or -src version of Dojo.

Unpack the contents of the archive into a folder (preferably on a web server as this is always a good case for Ajax development). Let's call it "js/". You may also name your dojo directory "dojotoolkit" as the examples here will show. If you wish to version Dojo, you may leave it as dojo-release-x.y.z. You should now have a directory structure similar to this:



(more…)



Posted in Dojo | 7 Comments

First Steps

Saturday, October 31st, 2009
This entry is part 3 of 8 in the series Dojo Quick Start Guide

Start by making a skeleton HTML file for use as a basic template for any example:

```
< !DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <title>Dojo Toolkit Test Page</title>

      <!– load the dojo toolkit base –>
      <script type="text/javascript" src="js/dojotoolkit/dojo/dojo.js"
         djConfig="parseOnLoad:true, isDebug:true"></script>

      <script type="text/javascript">
      /* our JavaScript will go here */
      </script>

      <style type="text/css">
      /* our CSS can go here */
      </style>
   </head>
   <body><!– this is a Typical WebPage starting point … –>
      <h1 id="testHeading">Dojo Skeleton Page</h1>
      <div id="contentNode">
         <p>Some Content To Replace</p>
      </div>
   </body>
</html>
```

(more…)



Posted in Dojo | 10 Comments

DOM Magic

Sunday, November 1st, 2009
This entry is part 4 of 8 in the series Dojo Quick Start Guide

A really nice tool Dojo provides is `dojo.query`. It's a great way to parse all or portions of the Document Object Model (DOM) and access selections of nodes. It really deserves its own book. Each of the following sections will touch on how to use `dojo.query` more closely, though realizing its potential is as simple as seeing it used:

```
dojo.require("dojo.NodeList-fx");
dojo.ready(function(){
    // our dom is ready, get the node:
```

```
    dojo.query("#testHeading")
      .addClass("testClass") // adds class="testClass"
      .fadeOut({ delay:500 }).play(); // and fade it out after 500 ms
});
```

Add `.testClass` CSS definitions to set color:red; and your heading will get that style before fading out:

```
.testClass {
    color:#ff0000;
}
```

`dojo.query` returns an instance of a `dojo.NodeList`, a synthetic super-Array of domNodes. It supports most CSS3 selectors (so you can go really wild with its syntax), and execute code against the whole list of results. To demonstrate this, we're going to need something more than a single heading, so add some content to our DOM:

```
h1 id="testHeading">Dojo Skeleton Page</h1>

    <a class="link" href="#">First link</a>
    <a class="link" href="#">Second Link</a>

    <p class="para">First paragraph</p>
    <p class="para">Second paragraph</p>
    <p class="para">Third paragraph</p>
```

And use a different query:

```
dojo.require("dojo.NodeList-fx");
dojo.ready(function(){
    // get each element with class="para"
    dojo.query(".para")
      .addClass("testClass")
      .fadeOut({ delay: 1000 }).play();
});
```

All three

elements should turn red, and fade out after a second delay. The full list of things dojo.NodeList does is impressive, some of which we'll touch on in later sections of this guide.

Most dojo.query chains have standalone functions to achieve the same goals. For instance: dojo.query("#testHeading").addClass("testClass"); and dojo.addClass("testHeading","testClass") have identical results.

SHARE

Posted in Dojo | No Comments

Events

Monday, November 2nd, 2009
This entry is part 5 of 8 in the series Dojo Quick Start Guide

The next important concept we are going to cover is interacting with our page. We've already set the heading to some alternate text, but what if we wanted to do something more interesting? Perhaps change it to something else when the user clicks on it? `dojo.connect` is the one-stop solution for all your event needs:

```
dojo.ready(function(){
    var node = dojo.byId("testHeading");
```

```
    dojo.connect(node,"onclick",function(){
      node.innerHTML = "I've been clicked";
    });
});
```

A convenient way to do the above using `dojo.query` would be:

```
dojo.ready(function(){
    dojo.query("#testHeading")
      .style("cursor","pointer")
      .connect("onclick",function(){
        this.innerHTML = "I've been clicked";
      });
});
```

We added another chain .style to our example, to make the cursor a pointer when hovering over the header node. We could have done this with plain CSS, and probably should have, to avoid unnecessary code. This, however, is a convenient way to dynamically alter most any CSS property, and very useful.

This allows us to make an `onclick` function on more than one node at a time, though our `NodeList` only has one element above. We could easily find a big group of elements, and affect them all. For instance, to prevent all links on a page from leaving, utilize the normalized *event* object `dojo.connect` passes:

```
var disableLinks = function(){
    dojo.query("a").connect("onclick",function(e){
      e.preventDefault(); // stop the event
      console.log('clicked: ',e.target); // the node we clicked on
    });
};
dojo.ready(disableLinks);
```

The code `e.preventDefault` will prevent the event from "doing what it was going to do". In the example, we `preventDefault` on the click event, which would have followed the anchor link we connected to. It is common to see the event object written as e or evt when passed as a parameter.

More about the normalized event object used in Dojo can be found in the Events Quickstart at the Dojo website.

We can also connect to methods of specific objects, and execute them in the same scope. This is useful as you get into declaring classes in Dijit, or animations. Lets create a really simple object with some methods, and watch them interact:

```
var mineObj = {
    aMethod: function(){
      console.log('running A');
    },
    bMethod: function(){
      console.log('running B');
    }
};
var otherObj = {
    cMethod: function(){
      console.log('running C');
    }
};
dojo.ready(function(){
    // run bMethod() whenever aMethod() gets run
    dojo.connect(mineObj,"aMethod",mineObj,"bMethod");
```

```
    // run an entirely different object's method via a separate connection
    dojo.connect(mineObj,"bMethod",otherObj,"cMethod");

    // start chain of events
    mineObj.aMethod();
});
```

You should see "running A B C" on separate lines in the console. The full power of dojo.connect can be explored via the dojo.connect API.



Posted in Dojo | No Comments

Some Gloss: Dojo Animations

Tuesday, November 3rd, 2009
This entry is part 6 of 8 in the series Dojo Quick Start Guide

Dojo has a powerful animation system with several pre-made animations for a lot of common use cases. Adding some visual flair to you projects has never been easier, and typically makes the users experience a lot more interesting.

All animations use a single "magic object" as it's only parameter. The most important being the node: attribute, the `domNode on which to apply our animation. Some parameters are optional, and some are for advanced usage. A common setup would look something similar to:`

```
dojo.ready(function(){
    var animArgs = {
        node: "testHeading",
        duration: 1000, // ms to run animation
        delay: 250 // ms to stall before playing
    };
    dojo.fadeOut(animArgs).play();
});
```

Base Animations:

Animations included in base `dojo.js` are: fadeIn, fadeOut, and animateProperty. `dojo.animateProperty` is very powerful, and is the foundation for most advanced animations, and other animations in Dojo Core.

```
dojo.ready(function(){
    dojo.style("testHeading","opacity","0"); // hide it
    var anim1 = dojo.fadeOut({ node: "testHeading", duration:700 });
    var anim2 = dojo.animateProperty({
      node: "testHeading", delay: 1000,
      properties:{
        // fade back in and make text bigger
        opacity: { end: 1 }, fontSize: { end:19, unit:"pt"}
      }
    });
    anim1.play();
    anim2.play();
});
```

As seen, `dojo.animateProperty` will fade the element back in via it's opacity property, and simultaneously make the text larger. You can animate most any CSS property this way.

In JavaScript, when modifying multi-word properties such as font-size and border-top, you must use a mixed cased version, as hypens are illegal as keys. Use fontSize and lineHeight, instead of font-size or line-height for example.

Additional FX

A lot can be done visually with the base animations, `animateProperty` especially. To keep the size of the base `dojo.js` down, all the additional animations and tools have been packaged into a single module: `dojo.fx` to be optionally called in via `dojo.require`. Adding the module to your code provides several additional animation methods: `dojo.fx.combine`, `dojo.fx.chain`, `dojo.fx.wipeIn`, `dojo.fx.wipeOut` and `dojo.fx.slideTo`.

```
dojo.require("dojo.fx");
dojo.ready(function(){
    // slide the node to 75,75
    dojo.fx.slideTo({
      node:"testHeading",
      top:75, left:75
    }).play(); // and play it
});
```

`dojo.fx.chain` and `dojo.fx.combine` are very useful, too. They run animations in parallel or in sequence, returning a single instance of `dojo._Animation` to use:

```
dojo.require("dojo.fx");
    dojo.ready(function(){
      var anim = dojo.fadeOut({ node: "testHeading" });
      var anim2 = dojo.fadeIn({ node: "testHeading" });
      dojo.fx.chain([anim,anim2]).play();
});
```

Combining an animation to fade in and out wouldn't make sense, so lets fade it out and slide the node simultaneously using `dojo.fx.combine`:

```
dojo.require("dojo.fx");
    dojo.ready(function(){
      var anim = dojo.fadeOut({ node: "testHeading" });
      var anim2 = dojo.fx.slideTo({ node: "testHeading", top:75, left:75 });
      var result = dojo.fx.combine([anim,anim2]);
      result.play();
});
```

Animation Events

Each `dojo.Animation` has a series of "events" to tie into for more advanced usage. Going back to the one-stop-event-shop `dojo.connect`, we can connect to specific actions of the animation, and do other things. The most common are `onEnd` and `beforeBegin`:

```
dojo.ready(function(){
    var anim = dojo.fadeOut({ node: "testHeading" });
    dojo.connect(anim,"onEnd",function(){
      console.log(" the animation is done ");
    });
    dojo.connect(anim,"beforeBegin",function(){
      console.log(" the animation is about to start ");
    });
    anim.play();
});
```

These events are especially helpful when you want to do things like change some content out while a node is hidden and then fade it back in:

```
dojo.ready(function(){
    var anim = dojo.fadeOut({ node: "testHeading" });
    dojo.connect(anim,"onEnd",function(){
      dojo.byId("testHeading").innerHTML = "replaced after fade!";
      dojo.fadeIn({ node:"testHeading" }).play();
    });
    anim.play();
});
```

Conveniently, you can pass the event functions as properties to the animation. Using `dojo.connect` to setup the functions gives us a lot more power, and are typically safer for advanced uses, but sometimes it's easier to wrap it all in:

```
dojo.ready(function(){
    var anim = dojo.fadeOut({
      node: "testHeading",
      onEnd: function(){
        dojo.byId("testHeading").innerHTML = "replaced ... ";
        dojo.fadeIn({ node: "testHeading" }).play();
      }
    }).play();
});
```

The full explanation of events is available at the dojo.Animation API pages.

Note: older versions of Dojo use `dojo._Animation prior to its "promotion" as a stable public API in Dojo 1.4.`

animateProperty

Probably the most powerful of the base animations, `dojo.animateProperty` allows us to easily animate multiple css properties simultaneously.

Since animateProperty is a `dojo._Animation`, it uses the same arguments as other animations. With an additional object, properties we can define any style property of a node. From start to end, and optionally using a unit attribute.

Manipulating our header element to use a new font color, size, and overall opacity is as easy as:

```
dojo.ready(function(){
    var anim = dojo.animateProperty({
      node:"testHeading",
      duration:700,
      properties: {
        // javascript css names are camelCase
        // (not hyphenated)
        fontSize: { start:12, end:22, unit:"pt" },
        opacity: { start:1, end:0.5 },
        color: { start: "#000", end:"#FFE" }
      },
      delay:100 // Note! trailing commas break IE.
    });
    anim.play();
});
```

dojo.query Animations

Dojo provides another convenient module: `dojo.NodeList-fx`, which adds additional methods to `dojo.query` for the available `dojox.fx` animations. To enable these methods, simply add in the required module:

```
dojo.require("dojo.NodeList-fx");
    dojo.ready(function(){
      dojo.query("#testHeading").fadeOut().play();
});
```

The above gives us the same effect as calling `dojo.fadeOut` directly, but `dojo.query` here makes an animation for each of of the NodeList elements, and combines them into a single `dojo.Animation`. This can be useful when you have groups of like nodes you want to easily affect (in this case, all the nodes with class="fadeNode"):

```
dojo.require("dojo.NodeList-fx");
var fadeThem = function(){
    dojo.query(".fadeNode").fadeOut().play();
}
dojo.ready(function(){
    dojo.connect(dojo.byId("testHeading"),"onclick","fadeThem");
});
```

Unlike other dojo.query() chains, the NodeList-fx methods return an instance of `dojo.Animation`, preventing further chaining.

SHARE

Posted in Dojo | 1 Comment

Ajax: Simple Transports

Tuesday, November 3rd, 2009
This entry is part 7 of 8 in the series Dojo Quick Start Guide

Ajax is an acronym for "Asynchronous JavaScript and XML", a technology employed to send and receive data on the fly. It can be used to update sections of a website from any number of remote sources, send data to the server and pass responses back and forth, all without ever refreshing the web page.

Having been versed on some essential Dojo methods, we'll move on the the bread and butter of Ajax: XMLHttpRequest (or XHR for short). Dojo has several XHR methods available using common HTTP verbs: POST, GET, PUT, and DELETE.

To prepare, we need to create a file with some text to load in. Create a file named sample.txt in your js/ folder with sample text:

```
I am a remote file.
We used Ajax to put this text
in our page.
```

And modify the skeleton.html to have some basic markup and style:

```
<style type="text/css">
   #container {
      border:1px dotted #b7b7b7;
      background:#ededed;
      width:75px;
      height:55px;
   }
</style>
```

```
<div id="container" class="box">
   <div id="content">
   I am some Inner Content.
   I am going to be replaced
   </div>
</div>
```

The XHR methods use `dojo.Deferred` behind the scenes to handle *callbacks*. This is beyond the scope of a QuickStart, but extremely useful in practice. If you would like to learn more about callbacks and the various way to set them up, visit the Robust Promises with Dojo Deferred blog post or the `dojo.Deferred` API pages.

Getting data

The first stepping stone is `dojo.xhrGet`, which will return the contents of a GET call on a URL. The XHR methods share a lot of common parameters. Most important are the url: (our destination) and handleAs: (how we handle what is coming back). When the data arrives, it will be passed the the load: function we define:

```
var init = function(){
      var contentNode = dojo.byId("content");
      dojo.xhrGet({
         url: "js/sample.txt",
         handleAs: "text",
         load: function(data,args){
            // fade out the node we're modifying
            dojo.fadeOut({
               node: contentNode,
               onEnd: function(){
                  // set the data, fade it back in
                  contentNode.innerHTML = data;
                  dojo.fadeIn({ node: contentNode }).play();
               }
            }).play();
         },
         // if any error occurs, it goes here:
         error: function(error,args){
            console.warn("error!",error);
         }
      });
};
dojo.ready(init);
```

You will notice we've combined techniques above. The content will fade out, be replaced by the received data, and fade back in using methods we've learned to this point. It was almost too easy.

A single `handle` argument can be used instead of load and error, handling both success and failure cases in a common function:

```
var init = function(){
    dojo.xhrGet({
       url: "js/sample.txt",
       handleAs: "text",
       handle: function(data,args){
      if(typeof data == "error"){
         console.warn("error!");
         console.log(args);
      }else{
         // the fade can be plugged in here, too
```

```
            dojo.byId("content").innerHTML = data;
        }
         }
    });
};
dojo.ready(init);
```

XHR has limitations. The big one being that url is not cross-domain. You can't submit the request outside of the current host (e.g. to url:"http://google.com"). It is a known limitation and a common mistake when getting excited about Ajax. Dojo provides alternatives like `dojo.io.iframe` and `dojo.io.script` for more advanced usage.

You also may experience problems with the Ajax samples if you are using Internet Explorer without a web server (from the local file system). It is a know security limitation of XHR and IE. While most of these examples do work from the file system, it is recommended you have a web server accessible to host the Dojo source, and your tests.

A full list of XHR parameters is available at the Dojo XHR API page. We are only going to skim the surface here.

Sending Data

All Dojo XHR methods are bi-directional. The only difference is the method. Using `dojo.xhrPost`, we use the POST method, embedding the data in the request (as opposed to the query string as with `dojo.xhrGet`). The data can be set directly as an object passed to the content parameter:

```
dojo.ready(function(){
    dojo.xhrPost({
        url:"submit.html",
        content: {
       "key":"value",
       "foo":42,
       "bar": {
           "baz" :"value"
       }
        },
        load: function(data,ioargs){
       console.log(data);
        }
    });
});
```

Or more commonly, conveniently converted from a `form` parameter. First, make a simple unobtrusive form in the skeleton.html:

```
<form id="mainForm" action="ajax-submit.php" method="post">
      <label for="firstName">Name: </label>
      <input type="text" name="firstName" value="Enter Name" />
</form>
```

Then, add in some JavaScript to submit the form by using `dojo.connect` to listen to the `onSubmit` event, and post the contents of the form to an alternate URL:

```
// submit the form
var formSubmit = function(e){
      // prevent the form from actually submitting
      e.preventDefault();
      // submit the form in the background
      dojo.xhrPost({
          url: "alternate-submit.php",
```

```
        form: "mainForm",
        handleAs: "text",
        handle: function(data,args){
                if(typeof data == "error"){
                    console.warn("error!",args);
                }else{
                    // show our response
                    console.log(data);
                }
        }
    });
};
dojo.ready(function(){
    var theForm = dojo.byId("mainForm");
    // another dojo.connect syntax: call a function directly
    dojo.connect(theForm,"onsubmit","formSubmit");
});
```

Notice `e.preventDefault()` being used again. The default nature of a form being submitted to to visit a new page, and we want to prevent that from happening.

An example `alternate-submit.php` would look like:

```
<?php
    print "DATA RECEIVED:";
    print "<ul>";
    foreach($_REQUEST as $key => $var){
        print "<li>".$key." = ".$var."</li>";
    }
    print "</ul>";
?>
```

Object Data

Getting text back from the server is nice, but the really great stuff comes when you start passing JavaScript objects around. Using a different handleAs: attribute, we can alter how Dojo handles the response data. Make a new file named `simple-object.json` to load:

```
{
    foo: "bar",
    name: "SitePen",
    aFunction: function(){
        alert("internal function run");
    },
    nested: {
        sub: "element",
        another: "subelement"
    }
}
```

We'll target our xhrPost url: to the new file, and supply a `handleAs: "json"` parameter to convert the response data to an actual object we can use:

```
var postData = function(){
    dojo.xhrPost({
        url: "js/simple-object.json",
        handleAs: "json",
```

```
        load: function(data,ioargs){
                // success: set heading, run function
                dojo.byId("testHeading").innerHTML += " by: "+data.name;
                if(data.aFunction && data.aFunction()){
                   // we just ran data.aFunction(). should alert() …
                }
        }
    });
};
dojo.ready(postData);
```

`dojo.xhr` exists as a wrapper for each of the method specific calls. If you use `dojo.xhr`, add another property, `method`, with the value in uppercase, e.g. GET, PUT, POST, or DELETE.

This allows us to send literally *any* kind of data back and forth across the wire, without ever interrupting the user experience.



Posted in Dojo | 1 Comment

Dijit: Prepackaged

Thursday, November 5th, 2009
This entry is part 8 of 8 in the series Dojo Quick Start Guide

Dojo's widget system is called Dijit. *Dijits* are the official, accessible, themed components shipped with the Dojo Toolkit. It has its own namespace, and likewise its own collection of utility functions:

```
dijit.byId("firstWidget"); // is a reference to the actual widget.
dijit.byId("firstWidget").domNode; // is the domNode the widget uses
// as opposed to:
dojo.byId("testHeading"); // is a domNode in our page
```

Using Dijits

There are two ways to make *Dijits*: via markup, or programatically. The markup route breaks W3C validation because Dojo conveniently uses customized attributes in the markup to configure the widget. If this concerns you, it can all be done with script. We'll do both. (Note: Dojo 1.6 adds support for HTML5 data attributes making the validation point moot).

Start by making a new skeleton file, including a couple changes for Dijit styling: the default theme `claro`'s CSS, and setting to enable it:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Dijit Test Page</title>

    <link rel="stylesheet"
            href="js/dojotoolkit/dijit/themes/claro/claro.css" />

    <!– load the dojo toolkit base –>
    <script type="text/javascript" src="js/dojotoolkit/dojo/dojo.js"
      djConfig="parseOnLoad:true, isDebug:true"></script>

    <script type="text/javascript">
```

```
    // our code, and dojo.requires()
    </script>

  </head>
  <body class="claro"><!– this is a Typical WebPage starting point … –>
    <h1 id="testHeading">Dijit Skeleton Page</h1>

    <!– empty placeholder nodes –>
    <div id="sampleNode"></div>
    <div id="anotherNode"></div>

  </body>
</html>
```

Claro was added to Dojo at version 1.5. If you are using Dojo prior to version 1.5, replace all instances of Claro with Tundra.

Dijit uses Dojo's package system to track dependencies via `dojo.require`. Simply call in the modules you need in a script tag. For instance, to use a `dijit.Dialog` and `dijit.form.Button`, you need the following calls:

```
<script type="text/javascript">
    dojo.require("dijit.Dialog");
    dojo.require("dijit.form.Button");
</script>
```

The dijit.Dialog is a modal dialog box. It takes the node's content, and displays it front-and-center on the viewport, awaiting user interaction. It can also act as a form element. To explore beyond this guide, visit the `dijit.Dialog` API Pages, or the reference guide.

From markup

You can specify all the attributes needed to setup your widget directly in markup, the most important being the dojoType. The Dojo parser finds the `dojoType` attribute, and turns the node into a Dijit with the matching `classname`. `title` is a common attribute used by many widgets with headings:

```
<div dojoType="dijit.Dialog" id="sampleNode" title="The First Widget">
    <p>I am the Content inside the dialog.</p>
</div>
<button dojoType="dijit.form.Button" id="myButton"
  onClick="console.log('clicked')">
  And Button
</button>
```

If `parseOnLoad` is *true*, the widgets will be created, then `dojo.ready` code will be executed. If you want to execute code *before* widgets are parsed, set `parseOnLoad:false`, and put your code inside a `dojo.ready` function as before. Issuing the command `dojo.parser.parse();` will create the widgets when you are ready.

If `parseOnLoad` is true, the parser is loaded automatically. Otherwise, you must issue a `dojo.require("dojo.parser");` call to include the required functions. All Dijits use the parser, so it is included automatically.

From JavaScript

The same results can be achieved using valid HTML and JavaScript. Our markup is simple, valid HTML:

```
<div id="sampleNode">
    <p>I am the Content inside the dialog.</p>
</div>
```

```
<button id="myButton">
      Show Button
</button>
```

And our script is standard Dojo code. We pass all the attributes as an object into our *constructor*, and tell it to use the node "sampleNode" for its content. All Dijits (or declared classes) can be created using the JavaScript `new` function.

```
dojo.require("dijit.Dialog");
dojo.require("dijit.form.Button");
dojo.ready(function(){
   // make the button
   var theButton = new dijit.form.Button({
            onClick:function(){
               console.log("clicked");
            }
   },"myButton");

   // make our Dialog
   var theDijit = new dijit.Dialog({
            title:"The First Widget"
   },"sampleNode");
   // make sure its started. parser does this if using markup
   theDijit.startup();
});
```

When the button is clicked, you should see the word "clicked" in your Firebug (or Firebug Lite) console.

Manipulating The Widget

With our dialog successfully loaded and parsed (no errors were thrown, and the content of the Dialog is hidden), we need to explore some of the ways to manipulate the widgets. The function `dijit.byId` gives us a reference to our widget. The `dijit.Dialog` has an `id` of `sampleNode`.

To make the button the button control the dialog, modify the button's `onClick` attribute to do more than print text:

```
<div dojoType="dijit.Dialog" id="sampleNode" title="The First Widget">
      <p>I am the Content inside the dialog.</p>
</div>
<button dojoType="dijit.form.Button" id="myButton"
      onClick="dijit.byId('sampleNode').show()">
      Show Dialog
</button>
```

If using the programmatic method, modify the lines that create the button:

```
// make the button
var theButton = new dijit.form.Button({
      onClick:function(){
         dijit.byId("sampleNode").show();
      }
},"myButton");
```

The `dijit.Dialog` inherits from a `dijit.layout.ContentPane` which provides a few content-handling methods, including `setHref`. Add a new button outside the dialog with a new `onClick` function:

```
<div dojoType="dijit.Dialog" id="sampleNode" title="The First Widget">
      <p>I am the Content inside the dialog.</p>
```

```
</div>
<button dojoType="dijit.form.Button" id="myButton"
   onClick="dijit.byId('sampleNode').show()">
   Show Dialog
</button>
<button dojoType="dijit.form.Button" id="otherButton"
   onClick="dijit.byId('sampleNode').setHref('sample.txt')">
   Change Content
</button>
```

Or programatically by adding another button to our HTML:

```
<div id="sampleNode">
      <p>I am the Content inside the dialog.</p>
</div>
<button id="myButton">
      Show Button
</button>
<button id="otherButton">
      Change Dialog
</button>
```

And an additional new call:

```
// make the button
var theButton = new dijit.form.Button({
      onClick:function(){
         dijit.byId("sampleNode").show();
      }
},"myButton");
var theButton = new dijit.form.Button({
      onClick:function(){
            dijit.byId("sampleNode").setHref("sample.txt");
      }
},"otherButton")
```

Adding an `id` attribute to the paragraph inside the Dialog is an easy way to demonstrate another useful Dijit tool, `dijit.getEnclosingWidget`, to find which widget contains a passed `domNode`:

```
// show the dialog onLoad, without knowing it's id
dojo.addOnLoad(function(){
      // add <p id="myPara"> to the dialog content
      var p = dojo.byId("myPara");
      var theDijit = dijit.getEnclosingWidget(p);
      theDijit.show();
});
```

SHARE

Posted in Dojo | 2 Comments

Most Popular Stories

Most Popular

- Dojo 1.2 Grid (160)

- The Dojo Offline Toolkit (136)
- RESTful JSON + Dojo Data (89)
- DojoX FileUploader Upgrade to Support Flash 10 (73)
- New Features in Dojo Grid 1.2 (70)

Archives

- January 2011
- December 2010
- November 2010
- October 2010
- September 2010
- August 2010
- July 2010
- June 2010
- May 2010
- April 2010
- March 2010
- January 2010
- December 2009
- November 2009
- October 2009
- September 2009
- August 2009
- June 2009
- May 2009
- April 2009
- March 2009
- February 2009
- January 2009
- December 2008
- November 2008
- October 2008
- September 2008
- August 2008
- July 2008
- June 2008
- May 2008
- April 2008
- March 2008
- February 2008
- January 2008
- December 2007
- November 2007
- October 2007
- September 2007
- August 2007
- July 2007
- June 2007
- May 2007
- April 2007
- March 2007
- February 2007
- January 2007
- December 2006

- November 2006
- October 2006
- September 2006
- August 2006
- July 2006
- September 2001

Follow Us Subscribe

Dojo Training and Support

Ajax and Dojo workshops by core dojo contributors and committers.

Enterprise support for the Dojo Toolkit and Persevere by the developers the projects.

Learn how to leverage the full power of the Dojo Grid and Charting.

Ajax Consulting

RIA Design and Development

Design is more than a pretty picture. It's about creating an inviting experience that allows users to complete their tasks quickly and without frustration. Has your app made a user smile today?

Efficient Development

Our expert development team is intelligent, thorough, and quick, allowing us to nimbly jump into any project and meet with success on your terms.

Search