AJAX and Dojo

Status:	Draft
Version:	1.0
Authors:	Nikolai Onken Craig Riecke

Contents

- XHR Options
- XHR Callbacks
- dojo.xhrGet
- dojo.xhrPost
- dojo.xhrPut
- dojo.xhrDelete

Ajax has been a buzzword around for some time now (as far as you could call some time a lot of time ;)) and is one of the concepts which have changed the development for the web quite drastically.

Dojo provides a solid set of battle-tested XHR wrapper functions to allow you to build Ajax interactions with confidence, use a unified API, and handle forms with ease. These APIs are built into Dojo Base, so you can use them in any page that includes dojo.js. Read on to learn how easy it is to build powerful Ajax interactions with Dojo.

The XMLHTTP request object (XHR for short) is one of the basic building blocks for constructing responsive Ajax-drive interactions. By allowing you to retrieve data on the user's behalf without refreshing the whole page the XHR object provides tremendous, but cross-browser XHR usage is beset by memory leaks, divergent APIs, a lack of built-in form encoding from JavaScript, and painful corner cases when de-serializing response data.

XHR Options

All *XHR* functions follow the same pattern in the *property-bag* configuration options, passed to whichever function is called. They include:

- url - the endpoint to connect to and load data from. This must be on the same host and port as the serving page, a security limitation of XHR.
- handleAs - describes which formatter to use on the incoming data. Defaults to 'text', so any response data comes back as a plain string. Available options out of the box are: "json" (to convert the data to a JSON object), "javascript" (to load and execute JS fragments), "json-comment-optional" (to deprecate warnings about the poor security of client-side JSON parsing) and xml.
- timeout - a time in MS to wait before giving up the XHR call, and throwing an error to the error callback.
- sync - a boolean to determine if the XHR call should be synchronous or asynchronous.

Setting sync:true will cause the browser to stop the chain of execution until the data is returned. Defaults to `false`.

- form - a DOM Node of a `<form>` element, or a string ID of a `<form>` element, used to collect data to be sent along with the request. The form is passed through dojo.formToObject and is mixed into the content: attribute of the XHR call
- content - an object to be sent along with

Example usage:

```
// post some data, ignore the response
dojo.xhrPost({
    form: "someFormId", // read the url: from the action="" of the <form
    timeout: 3000, // give up after 3 seconds
    content: { part:"one", another:"part" } // creates ?part=one
another=part with GET, Sent as POST data when using xhrPos
});

// get some data, convert to JSO
dojo.xhrGet({
    url:"data.json",
    handleAs:"json",
    load: function(data){
        for(var i in data){
            console.log("key", i, "value", data[i]);
        }
    }
});
```

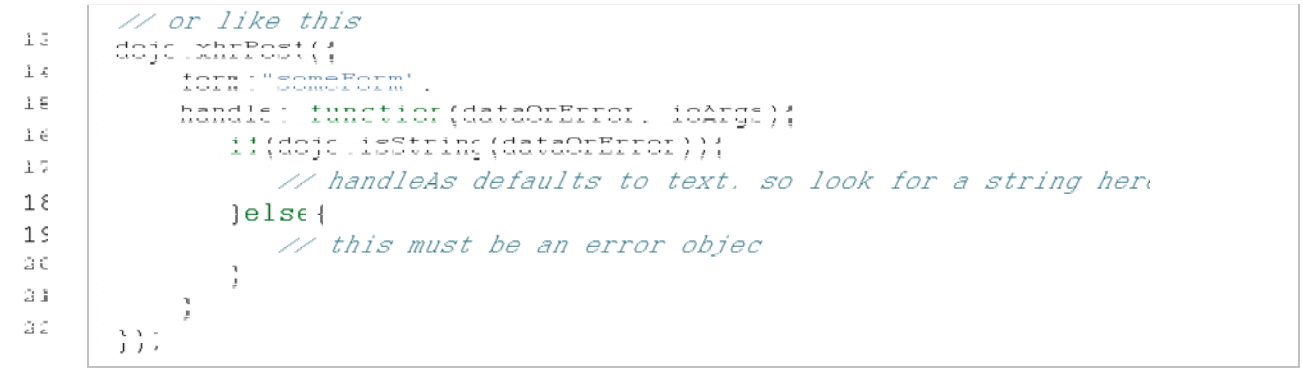
Introduced was the load: function, which is explained in the XHR Callbacks section below.

XHR Callbacks

There are three methods one can attach to the XHR Options object to determine what to do when the data comes back.

- load - executed when a successful Ajax call is complete. Is passed the data and an object of the XHR properties.
- error - executed when an Ajax call times out, or otherwise fails. Is passed the error and an object of the XHR properties.
- handle - combination of load and error callbacks, fired when either of the two conditions are met. In the success case, behaves just like load:, and in the failure case like error:

```
dojo.xhrPost({
    form:"someForm",
    load: function(data, ioArgs){
        // ioArgs is loaded with XHR information, but not useful in
simple case.
        // data is the response from the form's action="" ur
    },
    error: function(err, ioArgs){
        // again, ioArgs is useful, but not in simple case.
        console.error(err); // display the erro
    }
});
```



```
// or like this
dojo.xhrPost({
    form:"someForm",
    handle: function(dataOrError, ioArgs){
        if(dojo.isString(dataOrError)){
            // handleAs defaults to text, so look for a string her
        }else{
            // this must be an error objec
        }
    }
});
```

Alternately, you can "use plain Deferred's" to register callbacks. They are slightly more difficult to work with, but the concept is the same.

dojo.xhrGet

xhrGet will create an Ajax request using the HTTP GET method, returning some data to a callback. The callback is defined as a member of the object used to create the request (the property-bag), or by using the dojo.Deferred .addCallback method.

For complete details and examples, see the dojo.xhrGet documentation.

dojo.xhrPost

xhrPost will create an Ajax request using the HTTP POST method and is usually used to submit data to a service. It returns data to a callback. The callback is defined as a member of the object used to create the request (the property-bag), or by using the dojo.Deferred .addCallback method.

For complete details and examples, see the dojo.xhrPost documentation.

dojo.xhrPut

xhrPut will create an Ajax request using the HTTP PUT method and is usually used to submit data to a service. It returns data to a callback. The callback is defined as a member of the object used to create the request (the property-bag), or by using the dojo.Deferred .addCallback method.

For complete details and examples, see the dojo.xhrPut documentation.

dojo.xhrDelete

xhrDelete will create an Ajax request using the HTTP DELETE method, which is commonly used to signal to a service to delete a resource at a URI. Data returned is done via the form of a callback. The callback is defined as a member of the object used to create the request (the property-bag), or by using the dojo.Deferred .addCallback method.

For complete details and examples, see the dojo.xhrDelete documentation.