

Extracted from:

Mastering Dojo

JavaScript and Ajax Tools for Great Web Experiences

This PDF file contains pages extracted from Mastering Dojo, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2008 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The logo features a large, stylized green letter 'B' on the left. To its right, the words 'Beta' and 'Book' are stacked vertically in a bold, green, sans-serif font.

Beta Book

Agile publishing for agile developers

The book you're reading is still under development. As part of our Beta book program, we're releasing this copy well before we normally would. That way you'll be able to get this content a couple of months before it's available in finished form, and we'll get feedback to make the book even better. The idea is that everyone wins!

Be warned. The book has not had a full technical edit, so it will contain errors. It has not been copyedited, so it will be full of typos and other weirdness. And there's been no effort spent doing layout, so you'll find bad page breaks, over-long lines with little black rectangles, incorrect hyphenations, and all the other ugly things that you wouldn't expect to see in a finished book. We can't be held liable if you use this book to try to create a spiffy application and you somehow end up with a strangely shaped farm implement instead. Despite all this, we think you'll enjoy it!

Throughout this process you'll be able to download updated PDFs from your account on <http://pragprog.com>. When the book is finally ready, you'll get the final version (and subsequent updates) from the same address. In the meantime, we'd appreciate you sending us your feedback on this book at <http://books.pragprog.com/titles/rgdojo/errata>, or by using the links at the bottom of each page.

Thank you for being part of the Pragmatic community!

► **Andy Hunt**

We'll see plenty more examples of function literals, but here's what we need to know for Dojo XHR. XMLHttpRequest calls are asynchronous, meaning they return control to the JavaScript program immediately while they work in the background. We need to tell XHR “this is what you do when the result comes back.” That's perfect for a function literal, because most often the function is used only for that XHR call. Defining a named function to call it only once feels like too much overhead. A function we pass to an asynchronous request is called a *handler*, or equivalently a *callback*. We want the process to “call back” our function when it's ready.

The functions-as-data concept of JavaScript turns out to be extremely useful. Dojo uses them in XHR, but also surprising places like animations and declaring subclasses. We'll discuss those later, but for now let's dig into the first project.

3.3 A Wish List with dojo.Data and dojox.Grid

The Justa Cigar Corporation is overhauling their web site, and we've just won the contract to write it. Customers on the Justa site are gung-ho about cigars, and Justa wants to help them connect to each other, share information, and (hopefully) purchase cigars.¹

Each Justa customer has a wish list of cigars with brand names, sizes, country of origin and other information. The execs want to list this in a scrollable table, and give the customer the ability to add, delete, and edit cigars in-place without leaving the page. Figure 3.1, on the following page gives the “cocktail napkin” view of what they want. Already this is impossible with the old Web technology because of the don't-leave-the-page requirement. We're going to need XHR to send add, delete and edit requests back to the server.

dojox.Grid is a good fit for the user interface part. Grid acts like a mini-spreadsheet where you can view, sort, filter, and edit tabular information. dojo.Data provides plumbing between Grid and server-based data. Together they'll form the backbone of the wish list.

Because Justa controls the web site and database, their IT department can write web services in any format you want. These web services will

1. Just for the record, we don't condone such carcinogenic activities. The point is you can use the same techniques for finding ice cream, cross-country skis, or flat screen televisions. Whatever floats your boat. Dojo will not judge you.

Wish List

	Don Carlo			?

Don Carlo (4 Stars)
Pleasing Bouquet, crisp taste,

Figure 3.1: Wish List User Interface Design (Coffee Stain Omitted)

read from the wish list database, translate the data to the right format and send. Writing these server-side scripts is beyond the scope of this book, but Dojo can generally work with any server-side language and any database because they communicate over HTTP. If this notion is foreign to you, the sidebar on the next page gives you some help on where to start.

Fortunately, we can stub out the web services while writing the client. After all, XHR simply sends a request to a URL, and gets data back in some standard format. A plain old text file will suffice—it has a URL, and we can write it in some format. That makes the job small and easier to manage. But we still need to pick a data format. `dojo.Data` has drivers for commonly used data formats like XML, comma-separated variable (CSV), JSON, and HTML tables. Seeing this list, you might think XML is the way to go. But hold up a minute!

Server-Side Options

Dojo enables a significant architectural shift. When scripting Web 1.0 pages in a middle-tier language like PHP, ASP, or JSP, the navigation and HTML generation rests mostly in that language. With Dojo, more code executes on the client and less on the middle-tier. But you still need some middle-tier code for the following:

- Connecting to a database, and passing the results in a format Dojo understands.
- Proxying calls to external web services.

PHP is a particularly adept language for these calls because proxying is a one-line call and JSON support is built-in. (At least to 5.2 and above). But any language that can talk to HTTP servers and use databases will work.

Alternatively, you can use an Enterprise Service Bus (ESB). This software is made for proxying and translation, and many require declarative configuration in lieu of programming. Many ESB products are large and expensive, but Apache Synapse is an open-source, lightweight ESB that's a good match for Dojo.

JSON, the Language

JavaScript Object Notation, or JSON, is a better choice for sending the wish list data in our example. If you haven't seen or used it before, you might wonder why we'd pick JSON over XML. After all, practically every programming language and every browser on Earth speaks XML. It's self-descriptive, standardized, and mature.

But in browsers, XML suffers from two problems. First, browsers don't implement XML standards uniformly. Second, and this is the deal-breaker, browser-based XML implementations are slow. That's a problem because the more interactive you want your interface, the more chatty you have to be with the server, and the faster your data interchange format must be.

Enter JSON. In a nutshell, JSON data looks like the right hand side of a JavaScript assignment. For comparison, here is a snippet of XML data from our Justa wish lists:

```
<wishListItem>
```

```

    <wishId>4455</wishId>
    <description>Don Pepin Garcia Delicias</description>
    <size>7-50</size>
    ...
</wishListItem>

```

This is equivalent to the JSON:

```

"wishListItem": {
  "wishId": 4455,
  "description": "Don Pepin Garcia Delicias",
  "size": "7-50",
  ...
}

```

This looks suspiciously like an object literal. And it is! However, JSON has more restrictions placed on it:

- All strings, including the names on the left-hand side of the “:”, must be quoted. (Object literals are not as strict.)
- The only items on the right-hand side of the “:” are quoted strings, numbers, the boolean constants `true` and `false`, and the constant `null`. No expressions beyond those are allowed.

It’s really that simple. You can feed the data to a JavaScript `eval()` and get back an object, which Dojo does for you when accepting JSON from a web service. That’s why the quoting rules are so important. If they weren’t there, someone could type in a wish list item named `while(1)`; and lock up someone’s browser.

If you’re given the choice of web service output to consume by Dojo, JSON is usually preferable to XML. It’s expressive, flexible, and very easy to manipulate in JavaScript. Adapters for popular server-side languages are plentiful, as you can see at <http://www.json.org>. And it’s fast, fast, fast! Some studies have clocked it at 100 times faster than XML in a browser. This makes sense because JSON is “closer to the metal” of JavaScript, and requires less translation. When you need cigar data, those extra clock cycles count!

A Stub Data Source

Dojo.data has its own terminology, which we’ll cover completely in Chapter 10, *Dojo Data*, on page 178. But here’s enough to build our test Wish List data. A *data source* is the URL from which the data comes. In our test case, the URL will be very simple: `services/cigar_wish_list.json`. When we fill out the stubs we’ll probably send parameters along with it, as in

services/cigar_wish_list.php?userid=99555. A *data store* is the corresponding `dojo.Data` object which holds the data. Finally, an item is one data object. An item is like a database record, but can have a more complex structure.

So here's a snippet from our data source, kept in `services/cigar_wish_list.json`:

```
Download xhr_techniques/services/cigar_wish_list_abbrev.json
{
  "identifier": "wishId",
  "label": "description",
  "items":
  [
    {
      "wishId": 4455, "description": "Don Pepin Garcia Delicias",
      "size": "7-50", "origin": "Nicaragua", "wrapper": "Corojo",
      "shape": "Straight"
    },
    {
      "wishId": 4456, "description": "601 Habano Robusto",
      "size": "5-50", "origin": "Nicaragua", "wrapper": "Natural",
      "shape": "Straight"
    },
    {
      "wishId": 4457, "description": "Black Pearl Rojo Robusto",
      "size": "4 3/4-52", "origin": "Nicaragua", "wrapper": "Natural",
      "shape": "Straight"
    },
    /* ... */
  ]
}
```

The structure may look complex on first glance. Judicious use of white-space makes things a bit easier—here we use a style similar to a JavaScript program itself, lining up brackets and indenting common levels. Working from the inside out, a wish list item:

```
Download xhr_techniques/services/cigar_wish_list_abbrev.json
{
  "wishId": 4455, "description": "Don Pepin Garcia Delicias",
  "size": "7-50", "origin": "Nicaragua", "wrapper": "Corojo",
  "shape": "Straight"
},
```

is a simple object literal following the JSON rules. The brackets surrounding these objects create an array of wish list objects:

Download [xhr_techniques/services/cigar_wish_list_abbrev.json](#)

```
[
  {
    "wishId": 4455, "description": "Don Pepin Garcia Delicias",
    "size": "7-50", "origin": "Nicaragua", "wrapper": "Corojo",
    "shape": "Straight"
  },
  {
    "wishId": 4456, "description": "601 Habano Robusto",
    "size": "5-50", "origin": "Nicaragua", "wrapper": "Natural",
    "shape": "Straight"
  },
  {
    "wishId": 4457, "description": "Black Pearl Rojo Robusto",
    "size": "4 3/4-52", "origin": "Nicaragua", "wrapper": "Natural",
    "shape": "Straight"
  },
  /* ... */
]
```

Finally this array becomes the items property of the data source, as we saw above:

Download [xhr_techniques/services/cigar_wish_list_abbrev.json](#)

```
{
  "identifier": "wishId",
  "label": "description",
  "items":
  [
    {
      "wishId": 4455, "description": "Don Pepin Garcia Delicias",
      "size": "7-50", "origin": "Nicaragua", "wrapper": "Corojo",
      "shape": "Straight"
    },
    {
      "wishId": 4456, "description": "601 Habano Robusto",
      "size": "5-50", "origin": "Nicaragua", "wrapper": "Natural",
      "shape": "Straight"
    },
    {
      "wishId": 4457, "description": "Black Pearl Rojo Robusto",
      "size": "4 3/4-52", "origin": "Nicaragua", "wrapper": "Natural",
      "shape": "Straight"
    },
    /* ... */
  ]
}
```

We're going to feed this into the dojo.Data driver `dojo.data.ItemFileReadStore`.

This driver expects JSON data in a specific format with the following properties: `identifier` is the field containing an item's ID, and `label` is the field with the human-readable identifier, and `items` is the data itself, an array of objects. Not all `dojo.Data` drivers are this restrictive: CSV and XML data sources do not require an identifier field, for example.

The IT people at Justa will write a server-side component that reads database records and writes the data into this format. But this fixed data source will do for now.

The Data-Enabled Widget, `dojo.Grid`

Grid widgets are very familiar to GUI designers. A grid is a spreadsheet-like “super table” which allows editing, sophisticated display and a well-structured event system. Grids are unfamiliar to most web programmers, though, because they're difficult to construct from scratch.

The Dijit grid component is the state-of-the-art in web-enabled data grids, and it gives client/server grids a run for their money in features, performance, and stability. You can pipe `dojo.Data` datastores into it with just a few lines of code. Grid is part of DojoX because it lacks accessibility features that Dijit components must have to be part of Dijit proper. But in the meantime, Grid has been around for a few years, and the code base is solid.

Missing: an explanation of DojoX is needed in chapter 1.

We'll build the Grid first, then hook it into the data source. Every Grid needs two things:

The Model

This is the data feeding the grid. Grid's model follows the MVC architecture's definition of a model.

The Structure

This defines the look-and-feel of the grid: e.g. the column headings, the styles applied, links to cell editors, and many more.

So let's deal with the model first. We've already built our data source, and making this into a model requires just a few `<div>` tags:

[Download](#) `xhr_techniques/wish_list_grid.html`

```
<div dojoType="dojo.data.ItemFileReadStore"
    jsId="wishStore" url="services/cigar_wish_list.json">
</div>
<div dojoType="dojox.grid.data.DojoData" jsId="wishModel"
    store="wishStore" query="{ wishId: '*' }" clientSort="true">
</div>
```

These look a lot like Dijit components, but they're not. They don't display anything—usually a tip-off that they're not from Dijit. And the `dojoType=` value does not begin with `dijit`. In Section 12.1, *What Exactly is a Widget?*, on page 181, we'll learn the full story.

These act more like object assignment statements. Each of these tags has a `jsId=` attribute which declares a JavaScript variable to hold the object. You can use these variables in your own JavaScript code or, as we do here, you can feed the contents of one object into another. The first tag sets up a `dojo.data.ItemFileReadStore`, a datastore using JSON in the special format we used for `cigar_wish_list.json`.

The second `<div>` is the *dojo.Data Grid adapter*. It's an object of type `dojox.grid.data.DojoData` which takes in the datastore `wishStore`, defined above. The adapter can apply sorting and filtering to the datastore, designated by the attributes `clientSort=` and `query=`. The former is straightforward. The latter involves another object literal defining filter criteria. In this case, `{ wishId: '*' }` means match every item that has a `wishId` property. In our case, that's all of the records in the store.

With the model taken care of, we can define the structure. A *structure* is composed of *views*, each of which is a horizontally-independent scrollable region. By that we mean the views are laid out from left to right and when you use the up/down scrollbar all views scroll up or down in lockstep. But each view has a left/right scrollbar that only acts within its view. These group together to form a structure, of which every Grid has one and only one. A structure is a lot like a spreadsheet. You can split the screen vertically into a non-scrollable section and scrollable section. This works well for rows having a unique ID. If your rows are long, you can keep the ID at the left, while scrolling through the rest of the record on the right. In `dojox.Grid` terminology, this is a structure with two views.

We define the view in JavaScript. Our example requires only one:

[Download](#) `xhr_techniques/wish_list_grid.html`

```
var view1 = {
  cells:
    [ /* Array of rows */
      [ /* Array of subrows */

        /* Each object is a cell */
        {name: 'Cigar', field: "description", width: "15em"},
        {name: 'Length/Ring', field: "size"},
        {name: 'Origin', field: "origin"},
```

```

        {name: 'Wrapper', field: "wrapper"},
        {name: 'Shape', field: "shape"}
    ]
}];

```

Again, here's a compound JavaScript object that requires study. A view is a two dimensional array of *cells*, each of which is an object defining column characteristics. Each cell defines a *name* property, used in the column heading, and a *field* property which points to a field in our data source. (More on that later.) In addition, some cells have additional properties, like *width* in the first cell.

You might ask, "Where are the quotation marks around name and field?" It's important to distinguish between JSON and JavaScript here. In JSON, you *must* use "name" and "field". In JavaScript object literals, you can leave them out. Most people omit the quotes because it's easier to distinguish property names and values.

Working outwards in our code, the innermost array of cells is called a *subrow*, or *physical row*. In Grid, you can wrap a record's data over more than one subrow. These subrows bind together in a *row*, also called a *logical row*. Logical rows are selectable as a unit. This is a nice way to fold data into a narrow vertical space, so the user doesn't have to scroll left and right.

We have just one subrow, so the cells property has one array element nested in the outer array. That was easy. Finally we get to the outermost component, the structure, which is trivial:

[Download](#) xhr_techniques/wish_list_grid.html

```
var wishStructure = [ view1 ];
```

The structure here is just one view, but you can use many views in a complex structure if you want.

All right. We have the model and the structure. All that's left is to pull these together into the Grid tag:

[Download](#) xhr_techniques/wish_list_grid.html

```
<div id="grid" dojoType="dojox.Grid" model="wishModel"
    structure="wishStructure"></div>
```

This seemingly trivial statement has a lot of powerful JavaScript behind it, and does amazing things without any extra work. We'll see that in a minute.

First, we'll step back, review our work and fold in some last minute touches. We add a load statement for the Tundra Grid stylesheet. Since Grid is a part of DojoX, the styles are kept separate from the Dijit Tundra theme. Our own CSS styles set the border, width and height of the grid. And finally there are some `dojo.require()` calls to load everything. Note the module names here: `dojox.grid.Grid` provides the `dojox.Grid` widget and `dojox.grid._data.model` provides the `dojox.grid.data.DojoData` adapter.

[Download](#) xhr_techniques/wish_list_grid.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.or...*TRUNC*
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Justa Cigar Wish List</title>
<style type="text/css">
    @import "/dojoroot/dijit/themes/tundra/tundra.css";
    @import "/dojoroot/dojo/resources/dojo.css";
    @import "/dojoroot/dojox/grid/_grid/tundraGrid.css";
</style>
<script type="text/javascript" src="/dojoroot/dojo/dojo.js" djConfig="parseOnLo...*TRUNC*
<script>
    dojo.require("dojo.parser");
    dojo.require("dojo.data.ItemFileReadStore");
    dojo.require("dojox.grid.Grid");
    dojo.require("dojox.grid._data.model");

var view1 = {
    cells:
        [ /* Array of rows */
            [ /* Array of subrows */

                /* Each object is a cell */
                {name: 'Cigar', field: "description", width: "15em"},
                {name: 'Length/Ring', field: "size"},
                {name: 'Origin', field: "origin"},
                {name: 'Wrapper', field: "wrapper"},
                {name: 'Shape', field: "shape"}

            ]
        ]
    };
    var wishStructure = [ view1 ];
</script>
<style>
#grid {
    border: 1px solid #333;
    width: 550px;
    margin: 10px;
    height: 200px;
    font-size: 0.9em;
```

Justa Cigar Corporation

"Sometimes a cigar is a Justa Cigar!"

Cigar	Length/Ring	Origin	Wrapper	Shape
Don Pepin Garcia Delicias	7-50	Nicaragua	Corojo	Straight
601 Habano Robusto	5-50	Nicaragua	Natural	Straight
Black Pearl Rojo Robusto	4 3/4-52	Nicaragua	Natural	Straight
Padron Churchill	6 7/8-46	Nicaragua	Natural	Straight
Davidoff Aniversario No. 2	7-48	Dominican Republic	Natural	Straight
Cohiba Churchill	7-49	Dominican Republic	Sungrown	Straight

Figure 3.2: The Wish List: Scrollable, Sortable, and Full of Tasty Cigars

```

    font-family: Geneva, Arial, Helvetica, sans-serif;
}
</style>

</head>
<body class="tundra">

<h1>Justa Cigar Corporation</h1>
<h3>"Sometimes a cigar is a Justa Cigar!"</h3>

    <div dojoType="dojo.data.ItemFileReadStore"
        jsId="wishStore" url="services/cigar_wish_list.json">
    </div>
    <div dojoType="dojox.grid.data.DojoData" jsId="wishModel"
        store="wishStore" query="{ wishId: '*' }" clientSort="true">
    </div>

    <div id="grid" dojoType="dojox.Grid" model="wishModel"
        structure="wishStructure"></div>

</body>
</html>

```

Missing: Code truncated

Run this script and the Grid pops up as shown in Figure 3.2. And dig the functionality! The grid is:

- Alternately striped: odd/even colors are automatically applied for



```

GET http://localhost:8080/using_dojo/xhr_techniques/services/cigar_wish_list.json (16ms)
Headers Response
{
  "identifier": "wishId",
  "label": "description",
  "items": [ {
    "identifier": "4455", "label": "The Best Cigar Selection", "items": "

```

Figure 3.3: Firebug View of Getting the Wish List over XHR

easy reading.

- Scrollable: scroll bars automatically appear if needed for horizontal or vertical scrolling.
- Column sizable: point between columns on the top and drag left or right.
- Row-selectable: just click anywhere on a row to select it. The row changes color.
- Sortable: just click on a column header to sort by that field. Click again to switch the sort direction.

If you have the Firebug debugger installed in your browser, you can watch the XHR packets go over the network, as shown in Figure 3.3

dojo.data and dojox.grid provide a quick way to get XHR up-and-running against your domain's web services. What if you want to use services outside your network? You can write your own proxy in a server-side language which calls the outside service on your behalf. But for some specially-written web services, there's an even easier way.

3.4 Researching Cigars using JSONP

The second part adds lists of cigar-specific hyperlinks to the grid. This would be an arduous task, were it not for the Yahoo Search web service and `dojo.io.script()`. `dojo.io.script()` calls JSONP web services, and we'll use it to stitch the Yahoo search to our application. The user asks for a list of hyperlinks relevant to the cigar name, and Dojo builds a list of titles and links.

<http://developer.yahoo.com> is a good starting place for researching Yahoo web services. From there, the Search API documentation is a few clicks

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Mastering Dojo's Home Page

<http://pragprog.com/titles/rgdojo>

Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/rgdojo.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragprog.com/catalog
Customer Service:	orders@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com