backdraft Quickstart

Rawld Gill , ALTOVISO LLC < rgill@altoviso.com >

This document was generated 2011-02-11 02:18:56.

Abstract

This tutorial gives a step-by-step procedure for downloading and installing the backdraft application framework and seeing a simple backdraft program in action.

Table of Contents

Introduction and Orientation

backdraft is a client-side JavaScript framework for writing programs in the browser. This perspective is quite different than that of almost every other web application framework available today where the program mostly executes on the server and the browser functions as little more than a fancy terminal. With a backdraft application, the server downloads a set of *static* JavaScript resources that implement the program. The word "static" is important: the server does not compute scripts and/or HTML on the fly. Usually the server will provide services to the program beyond delivering static scripts, but the user interface of the program is implemented 100% at the browser in JavaScript. There are many advantages to this design:

- Since the user interface is completely separate from the application domain logic, each part is smaller, simpler, and therefore, costs less to construct and is more likely to be constructed correctly.
- Since the user interface and application domain logic are two distinct components, they can be built in parallel.
- Testing chores are similarly smaller, simpler, cost less, or more likely correct, and can be built in parallel.
- Since the target user interface platform is the browser (compared to, for example, Microsoft Windows or X11), a single code base can potential reach multiple target devices (desktops, phones, pads, and the rest).
- Risk is diffused among two components (compared to a single monolith). In the event a component fails or becomes outdated it can replaced without affecting the other component. This is particularly relevant when considering the user interface component since interface techniques tend to evolve quickly whereas domain solutions (for example, how to arrange the database in a payroll system) tend to change quite slowly.

This design pattern, termed "browser-compute", has been formalized and is discussed in depth in browser#compute [http://www.altoviso.com/articles/browser-compute.pdf].

As usual with all engineering, these advantages have a trade off. By definition, they require more work be done at the browser, and, out of the box, the browser is a primitive, even hostile, programming environment. Modern JavaScript libraries like Dojo go a long way to mitigating this situation by providing a layer that causes all supported browsers to behave the same, and by providing a rich library that raises the level of abstraction. But, owing to the wide audience to which general-purpose JavaScript libraries are targeted, all of these libraries still view the browser as presenting "web pages for a web site", and this model is far too primitive for use building applications intended to compete with traditional desktop applications. This is largely the reason why most so-called "rich internet applications" feel clunky.

backdraft solves this problem by abstracting the browser into a programming environment suitable from writing applications, not web pages. This is the raison d'être of backdraft.

This tutorial is intended to get you going quickly and demonstrate backdraft's high level of abstraction with our version of the omnipresent "hello, world" program. The program takes control of the body element, defines a few accelerator keys, and waits for the user to press one of them. When an accelerator is detected, accelerator detection is disabled and the program displays a message box saying which accelerator was pushed. Upon dismissing the message box, accelerator detection is turned back on.

This functionality requires about *six lines of code*, no markup(!), and no server-side programming. The functionality provided would take many thousands of lines where you to start with the native browser programming environment. or, hundreds of lines with a good JavaScript library. Again, that's the purpose of backdraft–to allow you to *easily* write programs in the browser that look and behave like native applications.

This tutorial is intended to get you up fast; here's what we're going to accomplish:

1. Download the necessary software.

2. Configure an http server.

3. Describe and run the "hello, world" demonstration.

Downloading

The entry point for downloading the backdraft source code is the project download page available at http://bdframework.org/downloads.html. Point you browser to this URL and you should see a page titled "backdraft - current and past releases". This page describes the three ways to get backdraft:

1. Clone the official Fossil repository.

2. Download a packaged release.

3. Clone the github mirror.

Version control and issue tracking for the backdraft project is handled by Fossil.[1] If you decide to follow backdraft development closely, then the best way to get the project is to use Fossil, and the download page explains how to do this. But, since packaged releases are available that make downloading an installation trivial, we'll take that path for this tutorial.

Look for the second major heading on the page that reads "Packaged Releases". Within that section, you'll see a table with a set of zip and tar.gz files. The files are named by the timestamp of the repository commit

[1]The Fossil project is hosted at http://www.fossil-scm.org/index.html/doc/tip/www/index.wiki. It's the same distributed software configuration management system used by the highly popular SQLite project, so it's robust and highly used software. We like it for several reasons, in particular because it has a built-in ticket system. http://sheddingbikes.com/posts/1276624594.html gives a good explanation for why Fossil is a good choice.

that is contained in the file. For each repository commit there is both a zip and a tar.gz version, and these two versions contain exactly the same content. Finally, notice that the packages contained in the table are listed most recent to least recent. Click on the most recent package of the kind of archive you like (zip or targ.gz) to download the file.

If you are paranoid, you can compute the SHA1 or MD5 hashes on the downloaded archives and compare these values to the those posted on the web download page. They should be the same. If they aren't, you weren't paranoid.

Once downloaded, decompress the contents to a convenient location. For all of our examples, we decompress to `~/dev/` (for all you Windows devs, ~ is the user home directory--roughly equivalent to "My Documents"). So after decompressing, you should have a directory structure that looks like this:

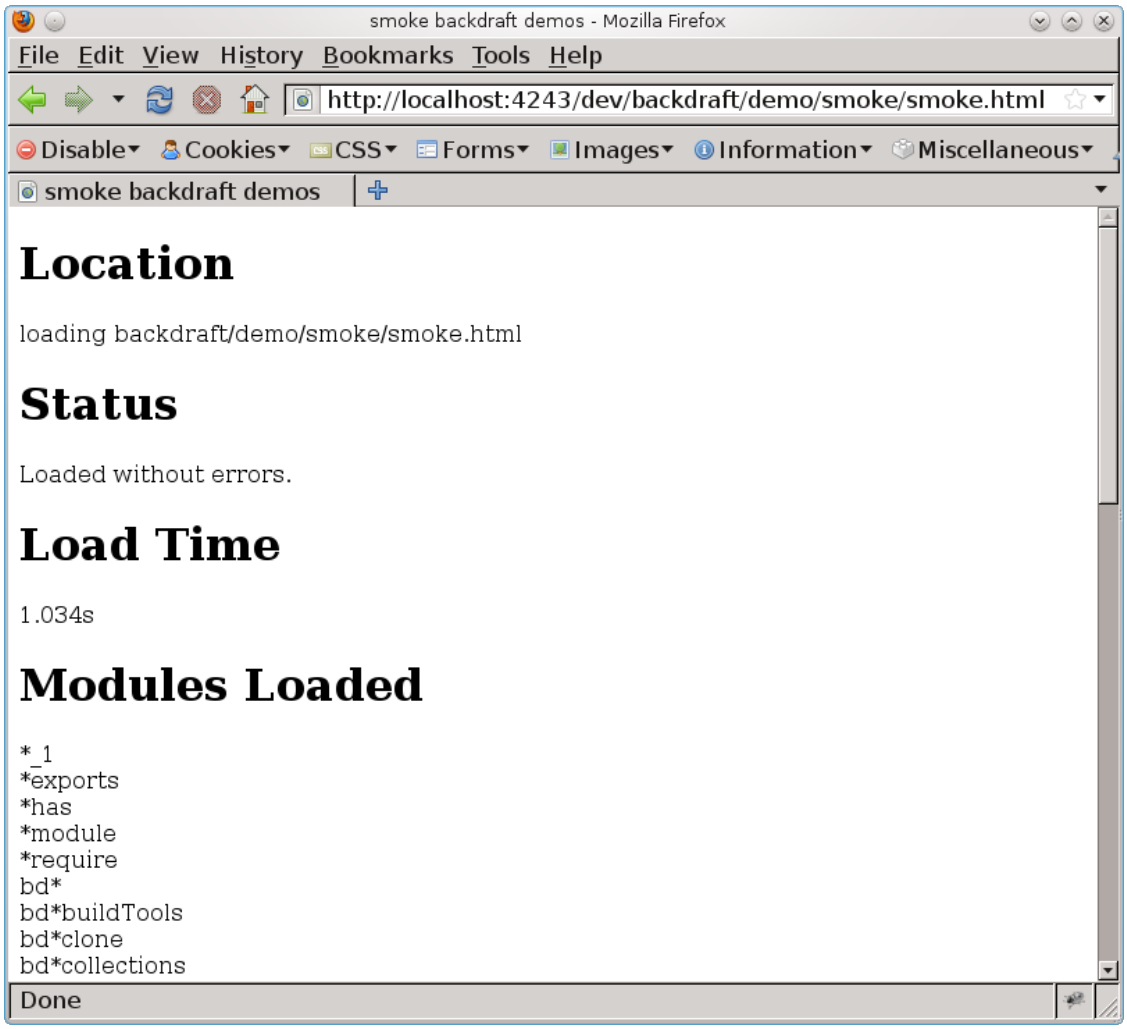
```
- ~/
  - dev/
    - backdraft/
      + demo
      + doc
      + lib
      + loader
      + test
    - dijit-v2x/
      + lib
      + tests
    - dojo-v2x/
      + lib
      + tests
    backdraft-info.txt
    dojo-sie-info.txt
    fossil-info.txt
```

backdraft sits on top of dojo-sie which is contained in the dojo-v2x and dijit-v2x directory trees in the archive. dojo-sie is a separate ALTOVISO-sponsored project that has converted all modules to comply with the commonJS [http://www.commonjs.org/] Asynchronous Module Definition [http://wiki.commonjs.org/wiki/Modules/AsynchronousDefinition] specification. This work has already been rolled into dojo trunk and will be released with Dojo version 1.6. dojo-sie includes several other forward-leaning dojo enhancements including an AMD-compliant loader. dojo-sie will stay loosely aligned with Dojo's version 2.0 branch until the Dojo's 2.0 release at which point dojo-sie should be identical to Dojo 2.0. Any Dojo-enhancements that don't make it into mainline Dojo will be moved to the backdraft project. The dojo-sie project available at http://dojo-sie.org.

Setting up a Server

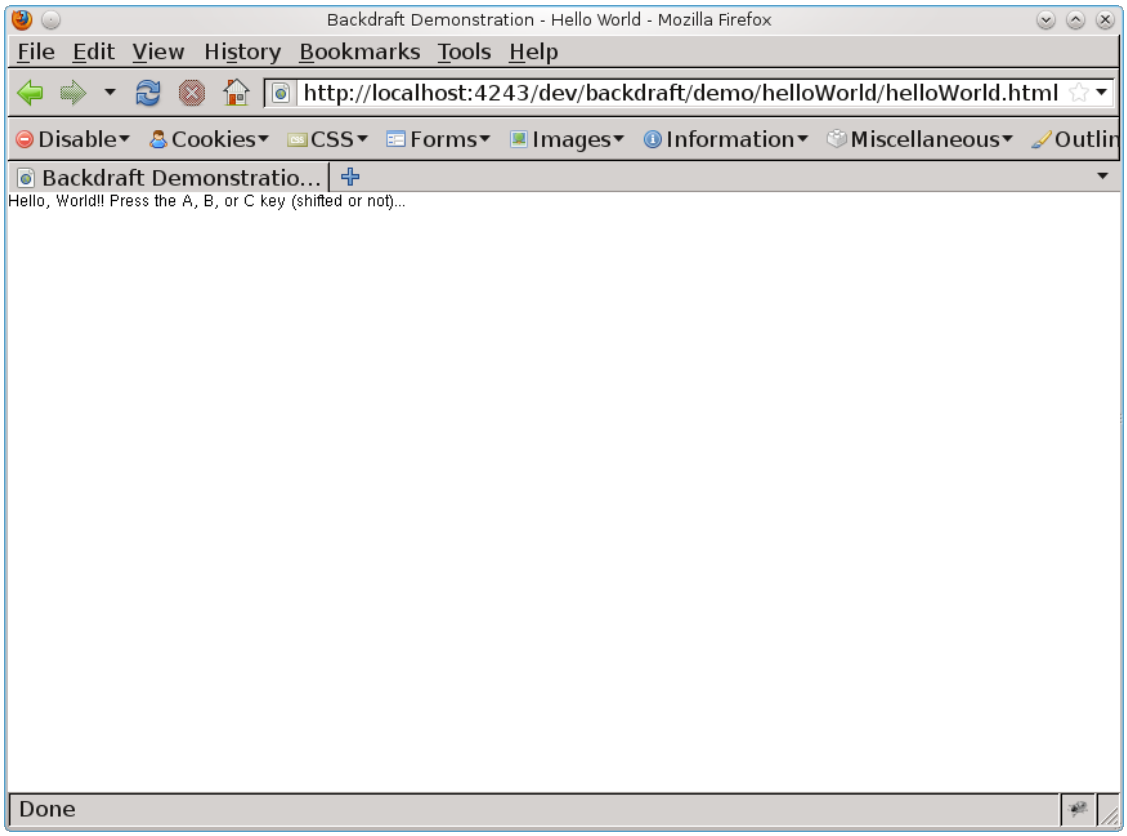
If you already have an http server that you can configure for use in your development environment, then ensure your server can resolve resources starting at the parent directly that holds the `backdraft`, `dojo-v2x`, and `dijit-v2x` trees. If you need help setting this up, see Setting up a Server. For the remainder of this tutorial, I'll assume you've unpacked the archive into a directory named `dev` and that the http server you've configured resolves `http://localhost:4243/dev/` to this directory.

To test your configuration, open up your browser and navigate to `http://localhost:4243/dev/backdraft/demo/smoke.html` and you should see something like this:

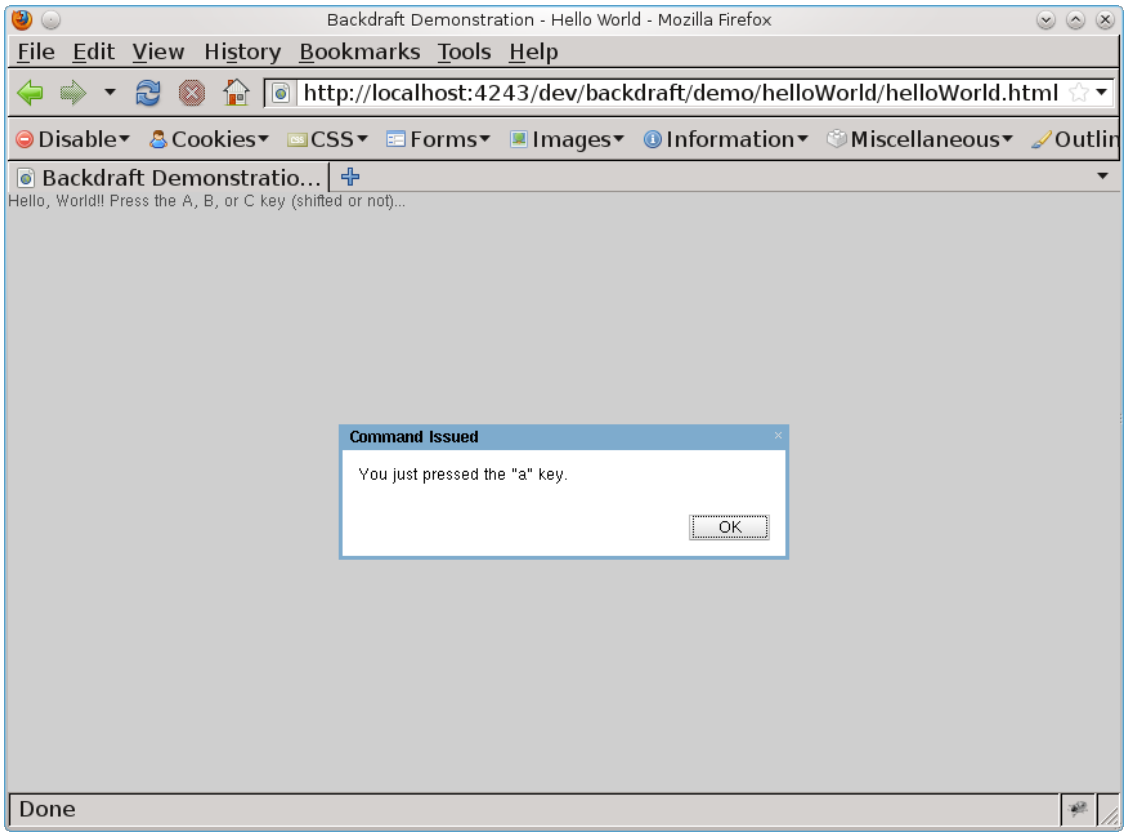


hello, world

Its time to run the "hello, world" demo. Point your browser to `http://localhost:4243/dev/backdraft/demo/helloWorld/helloWorld.html` and you should see the rather dull document:



Go ahead and press one of the **A**, **B**, or **C** keys (shifted or not). You should see a grey, semitransparent curtain put over the document and a message box pop up telling you which key you pressed. If it doesn't work initially, the browser probably doesn't have the focus. Click anywhere in the document viewport to ensure the browser has the focus and try again. The message box is movable by dragging the title bar--just like a native message box. If you try to press one of the **A**, **B**, or **C** keys while the message box is present, nothing happens since the accelerators are disabled. When you dismiss the message box by pressing the OK or X (close) button, the message box and the curtain are removed and the focus returns to the main document window. You can also exercise the program without using the mouse by pressing the **space bar** when the focus is on the OK or X button. Here's the program in action:



In the beginning, I promised that backdraft applications are executed in the browser, not at the server like most web application frameworks. Think about how this program might be implemented in a traditional framework:

1. The server would send the initial HTML page down to the browser. This page would include a JavaScript event handler to catch the keyboard events.

2. At the browser, upon detecting one of the accelerators, the event handler would execute a service request via an HTTP POST or GET transaction with the browser saying in effect "accelerator detected".

3. The server would manufacture the appropriate response to the "accelerator detected" service request by manufacturing the HTML to display the message box. Typically, this would be done by some wonderful templating system to fill in the received parameter (that is, the particular key that was pressed).

4. If the framework has some Ajax capability, the server's response would be incorporated into the current web page by appending it onto the document; otherwise, the entire page might be replaced. Indeed, that latter is exactly what happens in a very famous example Rails application.

There are lots of problems with this design: it's slow, clunky (the whole page may get refreshed), and complex (there are lot's of moving parts). Let's see how backdraft does things differently; here is the code (this code is in the file `backdraft/demo/helloWorld/main.js`.

```
1 require(["dojo", "bd", "bd/command/accelerators", "bd/widget/messageBox"], func
2  bd.forEach("A.B.C.a.b.c".split("."), function(c) {
3    bd.command.insertAccel(c, c);
4    bd.command.connect(c, function() {
5      bd.widget.messageBox("Command Issued", "You just pressed the \"" + c + "\"
```

```
 6    });
 7  });
 8  bd.start({topCreateArgs:{
 9    parent: "root",
10    descriptor: {
 1      className:"bd:widget.staticText",
 2      "class":"message",
 2      value:"Hello, World!! Press the A, B, or C key (shifted or not)..."
 4    }
 5  }});
 6 });
```

Line 1 defines a function that depends on the modules `dojo`, `bd`, `bd/command/accelerators` and `bd/widget/messageBox`. This pattern is not specific to backdraft, but rather is an example of the commonJS [http://www.commonjs.org/] Asynchronous Module Definition [http://wiki.commonjs.org/wiki/Modules/AsynchronousDefinition] module pattern. The code defined by the function comprises the entirety of the helloWorld application.

Line 2 inserts the six accelerators. For each of the characters 'A', 'a', 'B', 'b', 'C', 'c'...

1. At line 3, the backdraft command machinery is informed that the particular accelerator should be operational, and when a particular accelerator is detected, the command event with the same name should be fired. For example, when the accelerator 'A' is detected, the command event "A" is fired.

2. Line 4 connects a command event for each of the six distinct command events to a command handler.

3. Line 5 is the single-line command handler for a particular command event. It simply displays a message box with the title "Command Issued" that contains a message indicating which key was pressed.

That's it! That's really the whole program. The program is concise, simple, short, and requires zero server interaction. It's everything the traditional web application is not. **That is the point**.

The single call to bd.start at line 4 takes control of the HTML body element and appends a static text widget that displays the prompt.

Here is the associated HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Backdraft Demonstration - Hello World</title>
    <link href="css.css" rel="stylesheet" type="text/css" />
    <script src="../config.js"></script>
    <script src="../../loader/require.js"></script>
  </head>
  <body class="tundra">
  </body>
</html>
```

As promised, *zero markup*! The first script element configures the loader and the second script element loads the loader which then loads the application contained in main.js described above. This is typical of a backdraft application, although real applications tend to include a pacifier to occupy the user while the program loading.

That concludes this tutorial. To see a typically application frame program demonstration, go to http://bdframework.org/demo-appFrame.html. The backdraft reference manual is available at http://docder.altoviso.com/docder.html?doc=/docs/com.altoviso.backdraft.api.ref.js. Coincidently, the backdraft reference manual is presented in a backdraft example application named "docder" (that is, "document reader"). This application is available as a full ALTOVISO-sponsored project at http://www.altoviso.com/projects/docder/ [] and is another good example to study. Although the backdraft framework is several years old, the public-facing release of the software is new as of 4Q2010, and several additional resources are planned for release is rapid order--be sure to revisit the top-level project pages frequently.

backdraft is an ALTOVISO-sponsored project. Commercial support for backdraft (as well as Dojo, rich internet application construction, and other technologies and services) is available from ALTOVISO; go to http://www.altoviso.com/services for more information. Of course donations (http://bdframework.org#donate) are always welcome to help keep this project going!

Supplemental: Setting up a Server

This supplemental section explains how to configure lighttpd because its a great server and is super-easy to configure. The simplicity of the lighttpd configuration files should help you translate the configuration discussed below to your server of choice if you prefer something else.

The lighttpd project page is at http://www.lighttpd.net. If you don't already have lighttpd installed and you're on a Unix or Linux system, follow the instructions at http://redmine.lighttpd.net/projects/lighttpd/wiki/GetLighttpd. For Windows users, there's a compiled binary at http://en.wlmp-project.net/downloads.php?cat=lighty. Finally, for OS X users, you can install lighttpd by issuing the following commands:

```
$ wget http://download.lighttpd.net/lighttpd/releases-1.4.x/lighttpd-1.4.28.tar.gz
$ tar xzvf lighttpd-1.4.28.tar.gz
$ cd lighttpd-1.4.28
$ ./configure
$ make
$ make install
$ sudo make install
```

Next we need to configure lighttpd to point to the directory trees we just installed. The easiest thing to do here is simply point the server root to ~. lighttpd won't recognize the directory ~, so you'll have to use the full path. On my machine that's `/home/rcgill`; yours will be different, so adjust accordingly. Here is what a lighttpd configuration file would look like to do that:[2]

```
server.modules = ( "mod_alias", "mod_accesslog", "mod_proxy",
                   "mod_cgi", "mod_fastcgi")
server.bind          = "127.0.0.1"
server.port          = 4243
server.document-root ="/home/rcgill/"
server.errorlog      ="/home/rcgill/dev/lighttpd.error.log"
accesslog.filename   ="/home/rcgill/dev/lighttpd.access.log"
index-file.names     =("index.html", "index.htm", "default.htm" )
```

[2]There are lots of different ways to set up your development environment. The suggested configuration is reasonable and simple, but by no means the best or only way.

```
dir-listing.activate = "enable"

# mimetype mapping
mimetype.assign             = (
  ".pdf"          =>      "application/pdf",
  ".sig"          =>      "application/pgp-signature",
  ".spl"          =>      "application/futuresplash",
  ".class"        =>      "application/octet-stream",
  ".ps"           =>      "application/postscript",
  ".torrent"      =>      "application/x-bittorrent",
  ".dvi"          =>      "application/x-dvi",
  ".gz"           =>      "application/x-gzip",
  ".pac"          =>      "application/x-ns-proxy-autoconfig",
  ".swf"          =>      "application/x-shockwave-flash",
  ".tar.gz"       =>      "application/x-tgz",
  ".tgz"          =>      "application/x-tgz",
  ".tar"          =>      "application/x-tar",
  ".zip"          =>      "application/zip",
  ".mp3"          =>      "audio/mpeg",
  ".m3u"          =>      "audio/x-mpegurl",
  ".wma"          =>      "audio/x-ms-wma",
  ".wax"          =>      "audio/x-ms-wax",
  ".ogg"          =>      "application/ogg",
  ".wav"          =>      "audio/x-wav",
  ".gif"          =>      "image/gif",
  ".jar"          =>      "application/x-java-archive",
  ".jpg"          =>      "image/jpeg",
  ".jpeg"         =>      "image/jpeg",
  ".png"          =>      "image/png",
  ".xbm"          =>      "image/x-xbitmap",
  ".xpm"          =>      "image/x-xpixmap",
  ".xwd"          =>      "image/x-xwindowdump",
  ".css"          =>      "text/css",
  ".html"         =>      "text/html",
  ".htm"          =>      "text/html",
  ".js"           =>      "text/javascript",
  ".asc"          =>      "text/plain",
  ".c"            =>      "text/plain",
  ".cpp"          =>      "text/plain",
  ".log"          =>      "text/plain",
  ".conf"         =>      "text/plain",
  ".text"         =>      "text/plain",
  ".txt"          =>      "text/plain",
  ".dtd"          =>      "text/xml",
  ".xml"          =>      "text/xml",
  ".mpeg"         =>      "video/mpeg",
  ".mpg"          =>      "video/mpeg",
  ".mov"          =>      "video/quicktime",
  ".qt"           =>      "video/quicktime",
  ".avi"          =>      "video/x-msvideo",
  ".asf"          =>      "video/x-ms-asf",
  ".asx"          =>      "video/x-ms-asf",
  ".wmv"          =>      "video/x-ms-wmv",
  ".bz2"          =>      "application/x-bzip",
```

```
  ".tbz"          =>      "application/x-bzip-compressed-tar",
  ".tar.bz2"      =>      "application/x-bzip-compressed-tar",
  # default mime type
  ""              =>      "application/octet-stream",
 )
```

This will cause lighttpd to listen on port 4243 on localhost.[3] If you put the backdraft and dojo-sie project trees some place other than `~/dev`, then change the `sever.document.root` configuration value accordingly (for example, assuming you installed at `My Documents\dev` in Windows, `~/dev` becomes `My Documents/dev`. The configuration also sets up both error and access logging which can be useful for diagnosing problems.

You don't actually need all the mime types listed above, but sooner or later you'll be requesting a file type that you haven't configured which will result in a wasted hour trying to figure out why the server isn't working; better to just do it now and be done with it.

Assuming you've stored the configuration given above in `~/dev/lighttpd.conf`, start lighttpd by issuing the command...

```
lighttpd -D -f ~/dev/lighttpd.conf
```

On Windows, assuming you're using the `My Documents` directory, the command looks like this:

```
C:\Program Files\LightTPD\LightTPD.exe -D -f "My Documents/dev/lighttpd.conf"
```

[3] So, you probably wondering why 4243? That's because the awesome lisp http server hunchentoot http://weitz.de/hunchentoot/ uses 4242 by default!