JavaScript File Object

Finnbarr P. Murphy
(fpm@fpmurphy.com)

As you are probably aware `JavaScript` engines such as *`SpiderMonkey`* typically do not allow access to the local filesystem for reasons of security. To enable developers to test the scripts from a command line, *`js`* includes the *`load()`* function which enables you to load one or more `JavaScript` scripts into the *`SpiderMonkey`* engine. However this is not sufficient for our purposes as no means is provided to write to the filesystem. Looking more closely at the source code, I noticed support for *`File`* objects. This support is not enabled by default however. It is not sufficient to simply recompile *`SpiderMonkey`* with this option enabled; you must also download and build the *`Netscape Portable Runtime (NSPR)`* library. This library provides a platform-neutral `API` for system level and *`libc`*-like functions, and is used by a number of Mozilla projects and other third party software developers. The current release is 4.7.3 and you can download it here.

There are some *`gotchas`* to building *`Spidermonkey`* with *`NSPR`*. First of all, you need to successfully build *`NSPR`*. The source code tarball for *`NSPR`* comes with the standard GNU autoconfigure tools. If you are on a 64-bit system, you need to execute *`configure`* with the *`-enable-64bit`* option; otherwise the build will quickly fail. You should then test the build by going to the test subdirectory, building the testsuite and executing it. You also need to modify *`SpiderMonkey's Makefile.ref`* (I am assuming you are building *`SpiderMonkey 1.7`* and not an earlier release) to include *`libnspr`* and the *`NSPR`* headers. Two compile time *`defines`* are needed. You can either define *`JS_HAS_FILE_OBJECT`* and *`JS_THREADSAFE`* in *`Makefile.ref`* or as command line arguments to *`make`*. After than you, should be able to successfully build *`SpiderMonkey`* with native *`File`* object support.

Now that we have *`js`* build with support for *File* objects, what can we do with it. Well, I guess we should start with the expected Hello World script.

```
js> File.output.writeln("Hello World");
Hello World
true
js> File.output.writeln("Hello, world"); "OK"
Hello, world
OK
js> File.output.writeln("Hello, world"); ""
Hello, world

js>
```

Notice that *`true`* is outputted unless you append something else as shown above. Here is another short example which demonstrates how to list the properties of the instance *`File`* object for the current directory.

```
js> dir = new File('.');
/home/fpm/js/.
js> for ( i in dir ) print(i);
length
parent
path
name
isDirectory
isFile
exists
```


canRead
canWrite
canAppend
canReplace
isOpen
type
mode
creationTime
lastModified
size
hasRandomAccess
hasAutoFlush
position
isNative
js>print(dir.path);
/home/fpm/js/.
js>print(dir.size)
4096
js>

The next example shows how to list some information about files in the current directory.

```
js> dir = new File('.');
/home/fpm/js/.
js> files = dir.list(); 'OK'
OK
js> for (i in files ) print (files[i].name + '  ' + files[i].size + '  ' + files[i].creati
onTime);
music.xml  1081  Tue Jan 06 2009 17:37:14 GMT-0500 (EST)
xml.js  259  Tue Dec 30 2008 18:23:22 GMT-0500 (EST)
xml1.js  699  Tue Jan 06 2009 23:33:26 GMT-0500 (EST)
2.xml  96  Tue Jan 06 2009 22:41:37 GMT-0500 (EST)
3.xml  127  Wed Jan 07 2009 00:02:18 GMT-0500 (EST)
multiply.js  249  Tue Dec 30 2008 17:49:02 GMT-0500 (EST)
helloworld.js  88  Tue Dec 30 2008 17:12:50 GMT-0500 (EST)
hw.js  124  Thu Jan 01 2009 00:24:38 GMT-0500 (EST)
xml2.js  502  Wed Jan 07 2009 00:02:17 GMT-0500 (EST)
regex.js  143  Tue Dec 30 2008 18:10:55 GMT-0500 (EST)
1.xml  15  Tue Jan 06 2009 20:35:27 GMT-0500 (EST)
js>
```

In the above example, `list` is an instance method of the `File` object. Using other `File` object instance methods, you can read data from a file and write data to a file.

In the following example, the script reads lines in from `file.in` and write the lines out to another file `file.out`, prepending each line with the corresponding line number.

```
#!/bin/js

var filein = new File("file.in");
filein.open("read", "text");

var fileout = new File('file.out');
fileout.open("write,create", "text");

var n=1;
while (data = filein.readln())
   fileout.writeln(n++ + ' ' + data);

filein.close();
```



```
fileout.close();
```

The *File* object provides two ways to access data inside a file: a text oriented access, based on characters, and a binary oriented access, based on bytes. In text mode, the maximum line length is 256 and the following encodings are supported: *ASCII* (text), *UTF-8* and *UCS-2*. The *File* object has a number of instance methods including *read*, *readLn*, *readAll*, *write*, *writeLn* and *writeAll*. If you just want to copy the file in it's entirety, you can use the *copyTo* method.

```
var file = new File("file.in");
file.open("read", "text");

file.copyTo("file.out");

file.close();
```

Similar instance methods which work on files include *remove* to delete a file or a directory and *removeTo* to rename a file.

Now that I have shown you how to enable js to access the local filesystem, I would be amiss if I did not point out to you the following warning which is in the Javascript File Object Proposal.

> Flaming disclaimer: Do not play with the File object if you are not prepared to have your hard drive erased, smashed, and broken into little bits! It mostly works right now, but no guarantees.

So far I have had no problem using the *File* object on my Fedora 10 64-bit platform but one never knows! Proceed with caution.

