Sifting the Sandbox:

Improving the docs and API of Caja

Kevin Reid

kpreid@switchb.org http://switchb.org/kpreid/
http://code.google.com/p/google-caja/

What is Caja?

“Caja allows websites to safely embed DHTML web applications from third parties, and enables rich interaction between the embedding page and the embedded applications. It uses an object-capability security model to allow for a wide range of flexible security policies, so that the containing page can effectively control the embedded applications' use of user data and to allow gadgets to prevent interference between gadgets' UI elements.”

— http://code.google.com/p/google-caja/

Documentation (mid-June)

Before starting work, I tried to use Caja for a project of mine to get up to speed. I found the documentation to be lacking, and that was my initial motivation.

I promptly began to make improvements to the Caja wiki documentation and examples, **while the new information was still fresh in my mind.**

Notable: `wiki/HostingModules` is a completely new page which provides an overview of the issues in writing a Caja host page (web page which contains gadgets).

HostTools (late June)

From the documentation and examples, I concluded there was much unnecessary boilerplate in Caja host pages, and developed the HostTools library to eliminate it.

Instead of user code referring to various Caja components and wiring them together correctly, HostTools provides an interface where the defaults Just Work and are short, and additional configuration is calls to setters.

This is decoupling/separation of concerns: if we change the relationship between Cajita and Domita, how module loading works, etc., the user need not care.

```
var ht = new HostTools();
var sandbox = new ht.Sandbox();
sandbox.attach(document.getElementById("theGadget"));
sandbox.run("gadget-trivial.html");
```

Iframes (June-July)

The Caja runtime modifies the global objects (Object.prototype, etc.) in ways that are not necessarily compatible with the assumptions of existing code (notably, e.g. jQuery). Therefore we now load Caja and all cajoled modules into an iframe (which gets its own set of globals). This is almost transparent; insofar as it isn't (Caja loads asynchronously) it's good for page load responsiveness.

```
loadCaja(function (caja) {
  var sandbox = new caja.hostTools.Sandbox();
  sandbox.attach(document.getElementById("theGadget"));
  sandbox.run("gadget-trivial.html");
});
```

Canvas (July-August)

I wrote a *taming* for the <canvas> element and its context object, currently under review. Taming is the process of taking existing objects, not designed or implemented in the object-capability style, and filtering/wrapping their interfaces to implement a useful and sound object-capability security policy.

Given the <canvas> taming, sandboxed gadgets can now use the <canvas> element to draw custom graphics and animations.

The 2D context features which allow copying <img> images into the canvas for further manipulation have been omitted for now as further consideration will be needed as to the security implications of allowing access to the contents of images (rather than merely specifying *which* image is displayed).

3D (WebGL) has not been considered yet.

This work is completely unrelated to my major projects, but it was something that needed doing and benefited from my understanding of capability design principles.

Before

```
<html>
  <head>
    <title>Trivial Caja Container</title>

    <link rel="stylesheet" href="caja-gadget.css" type="text/css">

    <script src="http://caja.appspot.com/domita-minified.js"
            type="text/javascript"></script>
    <script src="http://caja.appspot.com/setup-valija.js"
            type="text/javascript"></script>
    <script src="http://caja.appspot.com/valija.out.js"
            type="text/javascript"></script>
  </head>

  <body>

    <h1>Trivial Caja Container</h1>

    <p>This is a Caja container page. Assuming everything works, there is a Caja
    gadget below. The orange border in this example indicates the boundaries of
    the area which is under the control of the gadget.</p>

    <div id="theGadget" style="border: .1em solid orange; padding: .2em;"
                    class="vdoc-body___"></div>
    <script type="text/javascript">
        var root = document.getElementById("theGadget");

        var imports = ___.copy(___.sharedImports);
        imports.outers = imports;

        attachDocumentStub('-theGadget___', {
            rewrite: function() {
                return null;
            }
        }, imports, root);

        imports.htmlEmitter___ = new HtmlEmitter(root, imports.document);

        imports.$v = valijaMaker.CALL___(imports.outers);

        ___.getNewModuleHandler().setImports(imports);
    </script>

    <script src="gadget-trivial.out.js" type="text/javascript"></script>

    <p>See comments in the source of this page for more information.</p>
</body>
</html>
```

After

```
<html>
  <head>
    <title>Trivial Caja Container</title>

    <link rel="stylesheet" href="caja-gadget.css" type="text/css">

    <script src="http://caja.appspot.com/caja.js"
            type="text/javascript"></script>
  </head>

  <body>

    <h1>Trivial Caja Container</h1>

    <p>This is a Caja container page. Assuming everything works, there is a Caja
    gadget below. The orange border in this example indicates the boundaries of
    the area which is under the control of the gadget.</p>

    <div id="theGadget" style="border: .1em solid orange; padding: .2em;"></div>
    <script type="text/javascript">
        loadCaja(function (caja) {
          var sandbox = new caja.hostTools.Sandbox();
          sandbox.attach(document.getElementById("theGadget"));
          sandbox.run("gadget-trivial.html");
        }, {
          cajaServer: "http://localhost:8080/"
        });
    </script>

    <p>See comments in the source of this page for more information.</p>
</body>
</html>
```

(Comparison by FileMerge — Xcode Tools)

Caja Corkboard (July)

`http://caja-corkboard.appspot.com/`

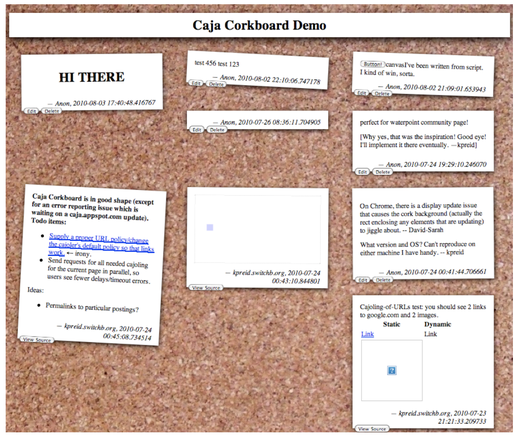
The Corkboard allows anyone on the Web to post HTML, including scripts, on its front page. It is a demo of using Caja as a “better HTML sanitizer”.

Unlike the existing Caja examples, which were all static pages using JavaScript to load modules, the corkboard incorporates cajoled content into a dynamically-generated page, such that the content appears even if the browser does not support scripting. It is also designed to be a straightforward example for the casual web programmer, who is not necessarily writing their server in Java.

`wiki/CorkboardDemo` describes how to add Caja integration to the Google App Engine Python guestbook tutorial in a few simple steps.

The corkboard site is also an example of my opinions on simple good practice in web design:

- does not require JavaScript
- uses CSS3 features for optional visual enhancements and optimized mobile-device view (no separate mobile site)
- scales to any screen width without scrolling or unreasonable line lengths



Results

Hopefully, Caja is now easier to use, both in what needs to be learned to use it and in the coverage of the documentation. However, I have no data on whether my changes are actually helpful to new users; I deliberately chose to focus on doing the work rather than evaluating it, and relied on feedback from my team for guidance.