CSS Assets Archive

CSS stylesheets reference assests by URL, e.g. bitmaps for background-image, list-style-image, border-image and border-corner-image. For fully skinnable applications this can result in big CSS files referencing hundreds of assets. If CSS is applied at runtime, this would result in hundreds of GET requests to the server. Even if the files are cached, depending on browser settings, it is going to take significantly long to load all those files.

To give the author the option to minimize server requests, DENG allows the use of a proprietary 'CSS Assets Archive' (CAA) that bundles assets referenced by the CSS.

A CAA is a ZIP archive containing one, some or all of the external assets referenced by the CSS stylesheet, accompanied by a catalog.xml file describing URL-to-filename/id mappings and additional attributes.

Consider the following CSS stylesheet:

```
.window {
    background-image: url(“img/wnd_background.png");
    border-top-image: url("img/wnd_border_top.png");
    border-right-image: url("img/wnd_border_right.png");
    border-bottom-image: url("img/wnd_border_bottom.png");
    border-left-image: url("img/wnd_border_left.png");
    border-top-left-image: url("img/wnd_border_top_left.png");
    border-top-right-image: url("img/wnd_border_top_right.png");
    border-bottom-left-image: url("img/wnd_border_bottom_left.png");
    border-bottom-right-image: url("img/wnd_border_bottom_right.png");
}
```

A CAA could contain the files:

- wnd_border_top.png
- wnd_border_right.png
- wnd_border_bottom.png
- wnd_border_left.png
- wnd_border_top_left.png
- wnd_border_top_right.png
- wnd_border_bottom_left.png
- wnd_border_bottom_right.png
- catalog.xml

The catalog.xml could look like this:

```
<assets baseUrl=”img/”>
    <asset id=”wnd_border_top.png” src=”wnd_border_top.png” />
    <asset id=”wnd_border_right.png” src=”wnd_border_right.png” />
    <asset id=”wnd_border_bottom.png” src=”wnd_border_bottom.png” />
    <asset id=”wnd_border_left.png” src=”wnd_border_left.png” />
    <asset id=”wnd_border_top_left.png” src=”wnd_border_top_left.png” />
    <asset id=”wnd_border_top_right.png” src=”wnd_border_top_right.png” />
    <asset id=”wnd_border_bottom_left.png” src=”wnd_border_bottom_left.png” />
    <asset id=”wnd_border_bottom_right.png” src=”wnd_border_bottom_right.png” />
</assets>
```

For DENG to recognize, load and use this CAA, the author/developer needs to reference it in the respective CSS using an “assets” at-rule at the top of the stylesheet:

```
@assets url(“wnd.caa”)
```

Note that such an at-rule is silently ignored by compliant CSS parsers. It's a non-intrusive extension only relevant for DENG's CSS parser implementation.

If DENG finds such an at-rule in a stylesheet, it loads the CAA, parses the contained catalog.xml and stores the contained assets along with the info from catalog.xml in an internal assets library.

While rendering, each time it encounters a property value referencing an external asset by URL, it

first checks with the assets library if that assets URL is already registered. If that's the case, the library's asset is used directly, if possible. Otherwise it tries to load the external asset normally from the server, and caches it in the library for potential repeated use.

This enables designers, artists and even laymen to skin every aspect of an application in an extremely easy fashion by using their prefered tools, while significantly reducing server requests at runtime and still being standards compliant.

Win-Win-Win.

Technical problem:

Flash Player 9 is natively capable of loading binary files, as well as compressing and uncompressing binary streams at runtime. However, the ZIP file format uses CRC-32 checksums over the uncompressed files, while the Flash Player uses ADLER32 (see RFC 1950: zlib file format). The Flash Player thus can't uncompress files contained in ZIP archives produced by commonly used 3rd party tools.

A possible workaround would be to write a small script that unobtrusively injects ADLER32 checksums into the ZIP, or in case most of the assets are already compressed (like PNG, JPG, GIF, SWF) to simply create a stored ZIP and not (re)compress the files at all.