Easier ArrayBuffer to String conversion with the Encoding API

By Jeff Posnick
Web DevRel @ Google

Over two years ago, Renato Mangini described a method for converting between raw ArrayBuffers and the corresponding string representation of that data. At the end of the post, Renato mentioned that an official standardized API to handle the conversion was in the process of being drafted. The specification has now matured, and both Firefox and Google Chrome have added native support for the TextDecoder and TextEncoder interfaces.

As demonstrated by this live sample, excerpted below, the Encoding API makes it simple to translate between raw bytes and native JavaScript strings, regardless of which of the many standard encodings you need to work with.

```
<pre id="results"></pre>

<script>
  if ('TextDecoder' in window) {
    // The local files to be fetched, mapped to the encoding that they're using.
    var filesToEncoding = {
      'utf8.bin': 'utf-8',
      'utf16le.bin': 'utf-16le',
      'macintosh.bin': 'macintosh'
    };

    Object.keys(filesToEncoding).forEach(function(file) {
      fetchAndDecode(file, filesToEncoding[file]);
    });
  } else {
    document.querySelector('#results').textContent = 'Your browser does not suppo
  }

  // Use XHR to fetch `file` and interpret its contents as being encoded with `en
  function fetchAndDecode(file, encoding) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', file);
    // Using 'arraybuffer' as the responseType ensures that the raw data is retur
    // rather than letting XMLHttpRequest decode the data first.
    xhr.responseType = 'arraybuffer';
    xhr.onload = function() {
```

```
      if (this.status == 200) {
        // The decode() method takes a DataView as a parameter, which is a wrappe
        var dataView = new DataView(this.response);
        // The TextDecoder interface is documented at http://encoding.spec.whatwg
        var decoder = new TextDecoder(encoding);
        var decodedString = decoder.decode(dataView);
        // Add the decoded file's text to the <pre> element on the page.
        document.querySelector('#results').textContent += decodedString + '\n';
      } else {
        console.error('Error while requesting', file, this);
      }
    };
    xhr.send();
  }
</script>
```

The sample above uses feature detection to determine whether the required **`TextDecoder`** interface is available in the current browser, and displays an error message if it's not. In a real application, you would normally want to fall back on an alternative implementation if native support isn't available. Fortunately, the text-encoding library that Renato mentioned in his original article is still a good choice. The library uses the native methods on browsers that support them, and offers polyfills for the Encoding API on browsers that haven't yet added support.

Update, September 2014: Modified the sample to illustrate checking whether the Encoding API is available in the current browser.

Last updated July 2, 2018.