How to convert ArrayBuffer to and from String

By Renato Mangini

Renato is a contributor to Web**Fundamentals**

Note: Update, August 2014: The Encoding API specification has matured, and a number of browsers now support it natively. The information in this article still applies for browsers that don't yet support the Encoding API, but the recommended approach is to use the official API wherever possible. See Easier ArrayBuffer <-> String conversion with the Encoding API for more details.

ArrayBuffers are used to transport raw data and several new APIs rely on them, including WebSockets, Web Intents, XMLHttpRequest version 2 and WebWorkers. However, because they recently landed in the JavaScript world, sometimes they are misinterpreted or misused.

Semantically, an ArrayBuffer is simply an array of bytes viewed through a specific mask. This mask, an instance of ArrayBufferView, defines how bytes are aligned to match the expected structure of the content. For example, if you know that the bytes in an ArrayBuffer represent an array of 16-bit unsigned integers, you just wrap the ArrayBuffer in a **`Uint16Array`** view and you can manipulate its elements using the brackets syntax as if the **`Uint16Array`** was an integer array:

```
// suppose buf contains the bytes [0x02, 0x01, 0x03, 0x07]
// notice the multibyte values respect the hardware endianess, which is little-en
var bufView = new Uint16Array(buf);
if (bufView[0]===258) {   // 258 === 0x0102
  console.log("ok");
}
bufView[0] = 255;    // buf now contains the bytes [0xFF, 0x00, 0x03, 0x07]
bufView[0] = 0xff05; // buf now contains the bytes [0x05, 0xFF, 0x03, 0x07]
bufView[1] = 0x0210; // buf now contains the bytes [0x05, 0xFF, 0x10, 0x02]
```

One common practical question about ArrayBuffer is how to convert a **`String`** to an **`ArrayBuffer`** and vice-versa. Since an ArrayBuffer is, in fact, a byte array, this conversion requires that both ends agree on how to represent the characters in the String as bytes. You probably have seen this "agreement" before: it is the String's character encoding (and the usual "agreement terms" are, for example, Unicode UTF-16 and iso8859-1). Thus, supposing

you and the other party have agreed on the UTF-16 encoding, the conversion code could be something like:

```
function ab2str(buf) {
  return String.fromCharCode.apply(null, new Uint16Array(buf));
}
function str2ab(str) {
  var buf = new ArrayBuffer(str.length*2); // 2 bytes for each char
  var bufView = new Uint16Array(buf);
  for (var i=0, strLen=str.length; i < strLen; i++) {
    bufView[i] = str.charCodeAt(i);
  }
  return buf;
}
```

Note the use of **`Uint16Array`**. This is an ArrayBuffer view that aligns bytes of the ArrayBuffers as 16-bit elements. It doesn't handle the character encoding itself, which is handled as Unicode by **`String.fromCharCode`** and **`str.charCodeAt`**.

Note: A robust implementation of the String to ArrayBuffer conversion capable of handling more encodings is provided by the stringencoding library. But, for simple usage where you control both sides of the communication pipe, the code above is probably enough. A standardized API specification for String encoding is being drafted by the WHATWG working group.

A popular StackOverflow question about this has a highly voted answer with a somewhat convoluted solution to the conversion: create a **`FileReader`** to act as a converter and feed a **`Blob`** containing the String into it. Although this method works, it has poor readability and I suspect it is slow. Since unfounded suspicions have driven many mistakes in the history of humanity, let's take a more scientific approach here. I have jsperf'ed the two methods and the result confirms my suspicion and you check out the demo here.

In Chrome 20, it is almost 27 times faster to use the direct **`ArrayBuffer`** manipulation code on this article than it is to use the **`FileReader/Blob`** method.

Last updated July 2, 2018.