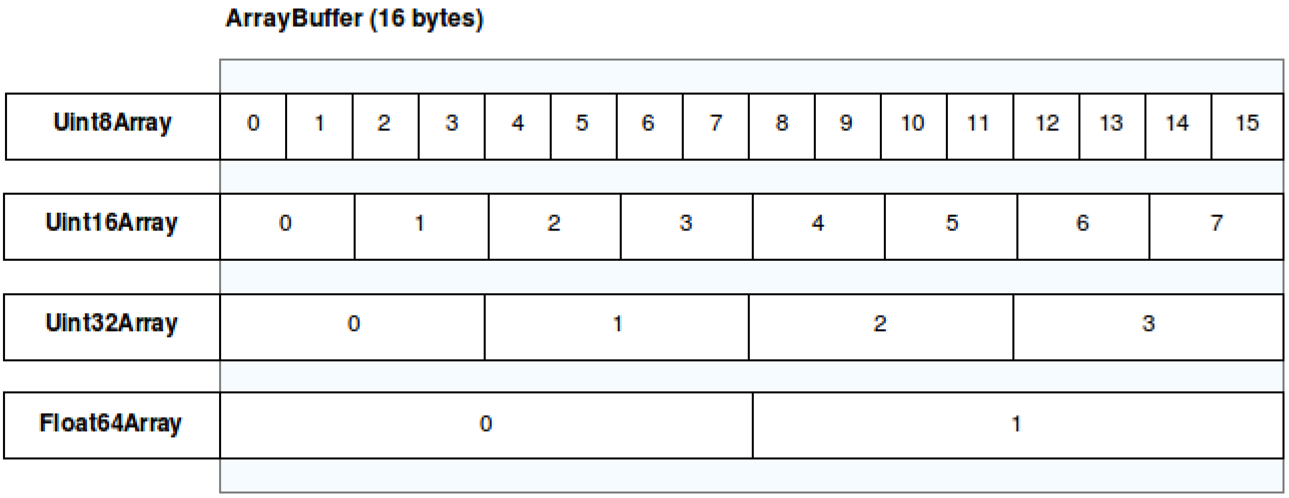
JavaScript typed arrays

JavaScript typed arrays are array-like objects and provide a mechanism for accessing raw binary data. As you may already know, `Array` objects grow and shrink dynamically and can have any JavaScript value. JavaScript engines perform optimizations so that these arrays are fast. However, as web applications become more and more powerful, adding features such as audio and video manipulation, access to raw data using WebSockets, and so forth, it has become clear that there are times when it would be helpful for JavaScript code to be able to quickly and easily manipulate raw binary data in typed arrays.

However, typed arrays are not to be confused with normal arrays, as calling `Array.isArray()` on a typed array returns `false`. Moreover, not all methods available for normal arrays are supported by typed arrays (e.g. push and pop).

Buffers and views: typed array architecture

To achieve maximum flexibility and efficiency, JavaScript typed arrays split the implementation into **buffers** and **views**. A buffer (implemented by the `ArrayBuffer` object) is an object representing a chunk of data; it has no format to speak of and offers no mechanism for accessing its contents. In order to access the memory contained in a buffer, you need to use a view. A view provides a context — that is, a data type, starting offset, and the number of elements — that turns the data into a typed array.



ArrayBuffer

The `ArrayBuffer` is a data type that is used to represent a generic, fixed-length binary data buffer. You can't directly manipulate the contents of an `ArrayBuffer`; instead, you create a typed array view or a `DataView` which represents the buffer in a specific format, and use that to read and write the contents of the buffer.

Typed array views

Typed array views have self-descriptive names and provide views for all the usual numeric types like `Int8`, `Uint32`, `Float64` and so forth. There is one special typed array view, the `Uint8ClampedArray`. It clamps the values between 0 and 255. This is useful for Canvas data processing, for example.

Type	Value Range	Size in bytes	Description	Web IDL type	Equivalent C type
`Int8Array`	-128 to 127	1	8-bit two's complement signed integer	`byte`	`int8_t`
`Uint8Array`	0 to 255	1	8-bit unsigned integer	`octet`	`uint8_t`
`Uint8ClampedArray`	0 to 255	1	8-bit unsigned integer (clamped)	`octet`	`uint8_t`
`Int16Array`	-32768 to 32767	2	16-bit two's complement	`short`	`int16_t`

			signed integer		
`Uint16Array`	0 to 65535	2	16-bit unsigned integer	`unsigned short`	`uint16_t`
`Int32Array`	-2147483648 to 2147483647	4	32-bit two's complement signed integer	`long`	`int32_t`
`Uint32Array`	0 to 4294967295	4	32-bit unsigned integer	`unsigned long`	`uint32_t`
`Float32Array`	1.2x10^{-38} to 3.4x10^{38}	4	32-bit IEEE floating point number (7 significant digits e.g. 1.1234567)	`unrestricted float`	`float`
`Float64Array`	5.0x10^{-324} to 1.8x10^{308}	8	64-bit IEEE floating point number (16 significant digits e.g. 1.123...15)	`unrestricted double`	`double`
`BigInt64Array`	-2^{63} to 2^{63}-1	8	64-bit two's complement signed integer	`bigint`	`int64_t (signed long long)`
`BigUint64Array`	0 to 2^{64}-1	8	64-bit unsigned integer	`bigint`	`uint64_t (unsigned long long)`

DataView

The `DataView` is a low-level interface that provides a getter/setter API to read and write arbitrary data to the buffer. This is useful when dealing with different types of data, for example. Typed array views are in the native byte-order (see Endianness) of your platform. With a

`DataView` you are able to control the byte-order. It is big-endian by default and can be set to little-endian in the getter/setter methods.

Web APIs using typed arrays

`FileReader.prototype.readAsArrayBuffer()`

The `FileReader.prototype.readAsArrayBuffer()` method starts reading the contents of the specified `Blob` or `File`.

`XMLHttpRequest.prototype.send()`

`XMLHttpRequest` instances' `send()` method now supports typed arrays and `ArrayBuffer` objects as argument.

`ImageData.data`

Is a `Uint8ClampedArray` representing a one-dimensional array containing the data in the RGBA order, with integer values between `0` and `255` inclusive.

Examples

Using views with buffers

First of all, we will need to create a buffer, here with a fixed length of 16-bytes:

```
let buffer = new ArrayBuffer(16);
```

At this point, we have a chunk of memory whose bytes are all pre-initialized to 0. There's not a lot we can do with it, though. We can confirm that it is indeed 16 bytes long, and that's about it:

```
if (buffer.byteLength === 16) {
  console.log("Yes, it's 16 bytes.");
} else {
  console.log("Oh no, it's the wrong size!");
}
```

Before we can really work with this buffer, we need to create a view. Let's create a view that treats the data in the buffer as an array of 32-bit signed integers:

```
let int32View = new Int32Array(buffer);
```

Now we can access the fields in the array just like a normal array:

```
for (let i = 0; i < int32View.length; i++) {
  int32View[i] = i * 2;
}
```

This fills out the 4 entries in the array (4 entries at 4 bytes each makes 16 total bytes) with the values 0, 2, 4, and 6.

Multiple views on the same data

Things start to get really interesting when you consider that you can create multiple views onto the same data. For example, given the code above, we can continue like this:

```
let int16View = new Int16Array(buffer);

for (let i = 0; i < int16View.length; i++) {
  console.log('Entry ' + i + ': ' + int16View[i]);
}
```

Here we create a 16-bit integer view that shares the same buffer as the existing 32-bit view and we output all the values in the buffer as 16-bit integers. Now we get the output 0, 0, 2, 0, 4, 0, 6, 0.

You can go a step farther, though. Consider this:

```
int16View[0] = 32;
console.log('Entry 0 in the 32-bit array is now ' + int32View[0]);
```

The output from this is "Entry 0 in the 32-bit array is now 32". In other words, the two arrays are indeed simply viewed on the same data buffer, treating it as different formats. You can do this with any view types.

Working with complex data structures

By combining a single buffer with multiple views of different types, starting at different offsets into the buffer, you can interact with data objects containing multiple data types. This lets you, for example, interact with complex data structures from WebGL, data files, or C structures you need to use while using js-ctypes.

Consider this C structure:

```
struct someStruct {
  unsigned long id;
  char username[16];
  float amountDue;
};
```

You can access a buffer containing data in this format like this:

```
let buffer = new ArrayBuffer(24);

// ... read the data into the buffer ...

let idView = new Uint32Array(buffer, 0, 1);
let usernameView = new Uint8Array(buffer, 4, 16);
let amountDueView = new Float32Array(buffer, 20, 1);
```

Then you can access, for example, the amount due with `amountDueView[0]`.

> **Note:** The data structure alignment in a C structure is platform-dependent. Take precautions and considerations for these padding differences.

Conversion to normal arrays

After processing a typed array, it is sometimes useful to convert it back to a normal array in order to benefit from the `Array` prototype. This can be done using `Array.from`, or using the following code where `Array.from` is unsupported.

```
let typedArray = new Uint8Array([1, 2, 3, 4]),
    normalArray = Array.prototype.slice.call(typedArray);

```

```
normalArray.length === 4;
normalArray.constructor === Array;
```

Specifications

Specification	Status	Comment
Typed Array Specification	Obsolete	Superseded by ECMAScript 2015.
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'TypedArray Objects' in that specification.	ST Standard	Initial definition in an ECMA standard.
ECMAScript Latest Draft (ECMA-262) The definition of 'TypedArray Objects' in that specification.	D Draft	

Browser compatibility

Update compatibility data on GitHub

Int8Array	
Chrome	7
Edge	Yes
Firefox	4
IE	10
Opera	11.6
Safari	5.1
WebView Android	4
Chrome Android	Yes
Firefox Android	4

Opera Android	12
Safari iOS	4.2
Samsung Internet Android	Yes
nodejs	0.10

`Int8Array()` without `new` throws

Chrome	Yes
Edge	Yes
Firefox	44
IE	No
Opera	Yes
Safari	?
WebView Android	Yes
Chrome Android	Yes
Firefox Android	44
Opera Android	?
Safari iOS	?
Samsung Internet Android	Yes
nodejs	0.12

Iterable in constructor

Chrome	?
Edge	?
Firefox	52
IE	?
Opera	?
Safari	?
WebView Android	?
Chrome Android	?
Firefox Android	52

Opera Android	?
Safari iOS	?
Samsung Internet Android	?
nodejs	4.0.0

Constructor without arguments

Chrome	?
Edge	?
Firefox	55
IE	?
Opera	?
Safari	?
WebView Android	?
Chrome Android	?
Firefox Android	55
Opera Android	?
Safari iOS	?
Samsung Internet Android	?
nodejs	?

□ Full support

□ No support

□ Compatibility unknown

See also

- Getting `ArrayBuffers` or typed arrays from *Base64*-encoded strings

- `StringView` – a C-like representation of strings based on typed arrays
- Faster Canvas Pixel Manipulation with Typed Arrays
- Typed Arrays: Binary Data in the Browser
- Endianness