DataView

The `DataView` view provides a low-level interface for reading and writing multiple number types in a binary `ArrayBuffer`, without having to care about the platform's endianness.

Description

Endianness

Multi-byte number formats are represented in memory differently depending on machine architecture — see Endianness for an explanation. `DataView` accessors provide explicit control of how data is accessed, regardless of the executing computer's endianness.

```
const littleEndian = (() => {
  const buffer = new ArrayBuffer(2);
  new DataView(buffer).setInt16(0, 256, true /* littleEndian */);
  // Int16Array uses the platform's endianness.
  return new Int16Array(buffer)[0] === 256;
})();
console.log(littleEndian); // true or false
```

64-bit Integer Values

Some browsers don't have support for `DataView.prototype.setBigUint64()` and `DataView.prototype.setBigInt64()`. So to enable 64-bit operations in your code that will work across browsers, you could implement your own `getUint64()` function, to obtain values with precision up to `Number.MAX_SAFE_INTEGER` — which could suffice for certain cases.

```
function getUint64(dataview, byteOffset, littleEndian) {
  // split 64-bit number into two 32-bit (4-byte) parts
  const left = dataview.getUint32(byteOffset, littleEndian);
  const right = dataview.getUint32(byteOffset + 4, littleEndian);

  // combine the two 32-bit values
  const combined = littleEndian
    ? left + 2 ** 32 * right
    : 2 ** 32 * left + right;

  if (!Number.isSafeInteger(combined))
    console.warn(combined, 'exceeds MAX_SAFE_INTEGER. Precision may be lost');

  return combined;
}
```

Alternatively, if you need full 64-bit range, you can create a `BigInt`. Further, although native BigInts are much faster than user-land library equivalents, BigInts will always be much slower than 32-bit integers in JavaScript due to the nature of their variable size.

```
const BigInt = window.BigInt;
  bigThirtyTwo = BigInt(32),
  bigMask = BigInt(0xffffffff);
function getUint64BigInt(dataview, byteOffset, littleEndian) {
  // split 64-bit number into two 32-bit (4-byte) parts
  const left = BigInt(dataview.getUint32(byteOffset, littleEndian)),
    right = BigInt(
      dataview.getUint32(byteOffset + 4, littleEndian)
    );

  // combine the two 32-bit values and return
  return littleEndian
    ? (right << bigThirtyTwo) | left
    : (left << bigThirtyTwo) | right;
}
```

Constructor

`DataView()`
Creates a new `DataView` object.

Instance properties

These properties are defined on `DataView.prototype` and shared by all `DataView` instances.

`DataView.prototype.buffer`
The `ArrayBuffer` referenced by this view. Fixed at construction time and thus **read only**.

`DataView.prototype.byteLength`
The length (in bytes) of this view. Fixed at construction time and thus **read only**.

`DataView.prototype.byteOffset`
The offset (in bytes) of this view from the start of its `ArrayBuffer`. Fixed at construction time and thus **read only**.

`DataView.prototype.constructor`
The constructor function that created the instance object. For `DataView` instances, the initial value is the `DataView` constructor.

`DataView.prototype[@@toStringTag]`
The initial value of the `@@toStringTag` property is the string `"DataView"`. This property is used in `Object.prototype.toString()`.

Instance methods

`DataView.prototype.getBigInt64()`
Reads 8 bytes at the specified byte offset of this `DataView` and interprets them as a 64-bit signed integer.

`DataView.prototype.getBigUint64()`
Reads 8 bytes starting at the specified byte offset of this `DataView` and interprets them as a 64-bit unsigned integer.

`DataView.prototype.getFloat32()`
Reads 4 bytes starting at the specified byte offset of this `DataView` and interprets them as a 32-bit floating point number.

`DataView.prototype.getFloat64()`
Reads 8 bytes starting at the specified byte offset of this `DataView` and interprets them as a 64-bit floating point number.

`DataView.prototype.getInt16()`
Reads 2 bytes starting at the specified byte offset of this `DataView` and interprets them as a 16-bit signed integer.

`DataView.prototype.getInt32()`
Reads 4 bytes starting at the specified byte offset of this `DataView` and interprets them as a 32-bit signed integer.

`DataView.prototype.getInt8()`
Reads 1 byte at the specified byte offset of this `DataView` and interprets it as an 8-bit signed integer.

`DataView.prototype.getUint16()`
Reads 2 bytes starting at the specified byte offset of this `DataView` and interprets them as a 16-bit unsigned integer.

`DataView.prototype.getUint32()`
Reads 4 bytes starting at the specified byte offset of this `DataView` and interprets them as a 32-bit unsigned integer.

`DataView.prototype.getUint8()`
Reads 1 byte at the specified byte offset of this `DataView` and interprets it as an 8-bit unsigned integer.

`DataView.prototype.setBigInt64()`
Takes a BigInt and stores it as a 64-bit signed integer in the 8 bytes starting at the specified byte offset of this `DataView`.

`DataView.prototype.setBigUint64()`
Takes a BigInt and stores it as a 64-bit unsigned integer in the 8 bytes starting at the specified byte offset of this `DataView`.

`DataView.prototype.setFloat32()`
Takes a number and stores it as a 32-bit float in the 4 bytes starting at the specified byte offset of this `DataView`.

`DataView.prototype.setFloat64()`
Takes a number and stores it as a 64-bit float in the 8 bytes starting at the specified byte offset of this `DataView`.

`DataView.prototype.setInt16()`
Takes a number and stores it as a 16-bit signed integer in the 2 bytes at the specified byte offset of this `DataView`.

`DataView.prototype.setInt32()`
Takes a number and stores it as a 32-bit signed integer in the 4 bytes at the specified byte offset of this `DataView`.

`DataView.prototype.setInt8()`
Takes a number and stores it as an 8-bit signed integer in the byte at the specified byte offset of this `DataView`.

`DataView.prototype.setUint16()`
Takes a number and stores it as a 16-bit unsigned integer in the 2 bytes at the specified byte offset of this `DataView`.

`DataView.prototype.setUint32()`
Takes a number and stores it as a 32-bit unsigned integer in the 4 bytes at the specified byte offset of this `DataView`.

`DataView.prototype.setUint8()`
Takes a number and stores it as an 8-bit unsigned integer in the byte at the specified byte offset of this `DataView`.

Examples

Using DataView

```
const buffer = new ArrayBuffer(16);
const view = new DataView(buffer, 0);

view.setInt16(1, 42);
view.getInt16(1); // 42
```

Specifications

Specification
ECMAScript Language Specification #sec-dataview-objects

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	Deno	Node.js
`DataView`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`DataView()` constructor	✓ 9	✓ 12	✓ 15	✓ 12.1	✓ 5.1	✓ 18	✓ 15	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4	✓ 1.0	✓ 0.10.0
`DataView()` without `new` throws	✓ 10	✓ 12	✓ 43	✓ 15	✓ 6.1	✓ 18	✓ 43	✓ 15	✓ 6.1	✓ 5.0	✓ 37	✓ 1.0	✓ 4.0.0
`this` is `arrayBuffer` accepted as buffer	✓ 44	✓ 12	✓ 78	✓ 55	✓ 13.2	✓ 89	✓ 79	✓ 63	✓ 13.3	✓ 10.0	✓ 89	✓ 1.0	✓ 8.10.0
`buffer`	✓ 9	✓ 12	✓ 15	✓ 12.1	✓ 5.1	✓ 18	✓ 15	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4	✓ 1.0	✓ 0.10.0
`this` is `sharedArrayBuffer` accepted as buffer	✓ 60	✓ 79	✓ 78	✓ 47	✓ 14.1	✓ 60	✓ 79	✓ 44	✓ 14.5	✓ 8.2	✓ 60	✓ 1.0	✓ 8.10.0
`byteLength`	✓ 9	✓ 12	✓ 15	✓ 12.1	✓ 5.1	✓ 18	✓ 15	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4	✓ 1.0	✓ 0.10.0
`byteOffset`	✓ 9	✓ 12	✓ 15	✓ 12.1	✓ 5.1	✓ 18	✓ 15	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4	✓ 1.0	✓ 0.10.0
`getBigInt64()`	✓ 67	✓ 79	✓ 68	✓ 54	✓ 15	✓ 67	✓ 68	✓ 48	✓ 15	✓ 9.0	✓ 67	✓ 1.0	✓ 10.4.0
`getBigUint64()`	✓ 67	✓ 79	✓ 68	✓ 54	✓ 15	✓ 67	✓ 68	✓ 48	✓ 15	✓ 9.0	✓ 67	✓ 1.0	✓ 10.4.0
`getFloat16()`	✓ 127	✓ 127	✓ 127	✓ 113	✓ 18.2	✓ 127	✓ 127	✓ 91	✓ 18.2	✓ 24.0	✓ 127	✓ 2.0	✓ 25.0.0
`getFloat64()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getInt16()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getInt32()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getInt8()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getUint16()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getUint32()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`getUint8()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setBigInt64()`	✓ 67	✓ 79	✓ 68	✓ 54	✓ 15	✓ 67	✓ 68	✓ 48	✓ 15	✓ 9.0	✓ 67	✓ 1.0	✓ 10.4.0
`setBigUint64()`	✓ 67	✓ 79	✓ 68	✓ 54	✓ 15	✓ 67	✓ 68	✓ 48	✓ 15	✓ 9.0	✓ 67	✓ 1.0	✓ 10.4.0
`setFloat16()`	✓ 127	✓ 127	✓ 127	✓ 113	✓ 18.2	✓ 127	✓ 127	✓ 91	✓ 18.2	✓ 24.0	✓ 127	✓ 2.0	✓ 25.0.0
`setFloat64()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setInt16()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setInt32()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setInt8()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setUint16()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setUint32()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0
`setUint8()`	✓ 1	✓ 12	✓ 15	✓ 10.1	✓ 5.1	✓ 18	✓ 14	✓ 12.1	✓ 5.1	✓ 3.0	✓ 4.0	✓ 1.0	✓ 0.12.0

Tip: you can click/tap on a cell for more information.

✓ Full support

See also

- Polyfill of `DataView` in `core-js`
- `ArrayBuffer`
- `SharedArrayBuffer`