B.12

The buffer object will act as if it were referencing a region in memory which is

1.) **of unchanging size (in bytes, as will be all sizes throughout this section)**

2.) **contiguous, and remains so across any garbage collection in which the object itself is not removed.**

The buffer's size is accessible via its public, read-only, property length.

The buffer is contiguous in the sense that given any set of operations that, for a positive integer "n", have set buffer elements b_n and b_{n+1} of a byte buffer B to a_n and a_{n+1} respectively, a two-byte subregion C created at from B at offset n (for which see below) will have

$c_0 = a_n$

$c_1 = a_{n+1}$

for its first two elements.

Associated with it will be facilities for accessing subregions of a given object as binary objects. The subregion will be specified by an integer offset within the whole, and a size for the subregion, both in bytes. Subregions can either be shared with the original buffer (so that changes to the subregion will affect the original buffer) or be copies of the subregion as of the time of creation.

Any attempt to create a "subregion" which is not completely within the domain of the original region (indices within [0, len) for a buffer of length "len") will generate a runtime error.

Binary objects of appropriate size will be retrievable as and settable by (with specifiable byte ordering, size, and sign convention, where appropriate) ECMAScript numbers. The default endness of the bit-representation of a given integer or float value will be that of the platform. Binary objects of any size may be accessed or set as ECMAScript strings or arrays of numbers. When attempting to retrieve from or alter an extant Buffer using a get or set method, and a byte size greater than the one appropriate to the returned object is specified (e.g., 4 in the case of integers, 2 for a UNICODE character code) is specified for the relevant Buffer domain or range, only the first or last "appropriate" bytes (depending on default or specified endness) of the region specified will be altered or retrieved. Zero-padding as an option?

A runtime error is triggered by any method on a Buffer B which attempts to retrieve or set a byte or bytes from an invalid offset, defined as one which is negative or exceeds B.length .

B.12.1.1 new Buffer(size)

Creates a new Buffer object of the specified size.

B.12.1.2 new Buffer(aBuffer, offset[, size] [, copy])

The default **size** value, here and in subsequent methods, will be

defaultSize = aBuffer.length - offset ,

so the new buffer will be a copy of the old buffer from the specified offset to its end. The default value for **copy** is **false**, meaning that the new **Buffer** object refers to some of the same data as the original.

B.12.1.3 new Buffer(String)

B.12.1.4 new Buffer(integer[] [, signed])

An Array of integer values is used to create the Buffer object, each value being truncated to a [signed] byte. The default value for **signed** is **false**.

B.12.1.5 Buffer subBuffer(offset[, size][, copy])

This method is equivalent to **new Buffer(this, offset, size, copy)** .

B.12.2.1 int getByte(offset[, signed])

Returns value of byte stored at the given offset as a [signed] byte. The default value for **signed** will be **false**, as it will be for all subsequent methods.

B.12.2.2 int getUNICODEValue(offset)

Returns integer value corresponding to the two-byte UNICODE character starting from the specified offset.

B.12.2.3 int getInt16(offset[, endness])

B.12.2.4 int getUInt16(offset[, endness])

B.12.2.5 int getInt32(offset[, endness])

B.12.2.6 int getUInt32(offset[, endness])

B.12.2.7 float getFloat(offset[,endness])

As stated before, all the get methods trigger runtime errors if there aren't enough bytes to create the value requested in the region specified, e.g. **aBuffer.getInt32(aBuffer.length - 1)**. Here and throughout the section, the parameter **endness** is a boolean for which **true** corresponds to a "big-endian" byte-ordering, and **false** to "little-endian" ordering.

B.12.2.8 int[] getByteArray(offset[, size][,signed])

Creates an array of ECMAScript integers from the [signed] byte-values in the buffer.

B.12.2.12 String getString(offset[, size])

Creates a UNICODE string from the buffer, triggering a runtime error if there is an extra byte left over at the end.

B.12.2.9 Buffer[] getBufferArray(offset, size, elementSize)

Creates an array of Buffer objects of size **elementSize**, triggering a runtime error if there aren't an integral number of sub-buffers of size **elementSize** in the region specified. Used rather than defining separate **getUInt16Array**, **getUInt32Array**,**…** methods.

B.12.3.1 void setByte(offset, value[, signed])

Sets a byte at a given offset to an integer value truncated to one [signed] byte.

B.12.3.2 void setUnicodeValue(offset, value)

Sets two bytes, starting at a given offset, to the UNICODE value. In this and all subsequent methods, all values to set are appropriately truncated if they exceed or underflow maximal or minimal values for a given integral type.

B.12.3.3 void setInt16(offset, value, [,endness])

B.12.3.4 void setInt32(offset, value, [,endness])

B.12.3.5 void setUInt16(offset, value, [,endness])

B.12.3.6 void setUInt32(offset, value, [,endness])

The **Int** and **Uint** methods differ only in their treatment of negative values (treated as 0 by the Uint methods) and truncation-points.

B.12.3.7 void setFloat(offset [,endness])

B.12.3.8 void setFromBuffer(offset[, size], srcBuffer[, srcOffset])

Sets **size** bytes in the caller from **offset** onward using the bytes in **srcBuffer** from **srcOffset** (default 0) onward.

B.12.3.9 void setFromString(offset, aString[,nChar])

Set the caller's bytes from **offset** onward using the values of the first **nChar** characters of **aString**. Equivalent to

src = new Buffer(aString) ;

this.setFromBuffer(offset, src, src.length) ; .

B.12.3.10 void setFromBufferArray(offset, aBufferArray[, [arrayOffset[, size]])

Maps an array of buffers into one buffer. Sets the buffer instance's elements from **offset** onward, using the successive elements of an array of buffer objects. Bytes are read from **aBufferArray** from **arrayOffset** (default value 0) onward, and continuing for **size**. A runtime error is triggered if there are fewer than **size** available after **arrayOffset**.