Buffers

Buffer objects are similar to **BLOB** (binary large objects). Wakanda proposes two specific methods allowing you to convert easily objects from one type to the other: **toBlob( )** and **toBuffer( )**.

Buffer objects allow binary data to be read or written directly from JavaScript, and can be used to exchange data. In particular, buffer objects are used to read or write (potentially binary) data from or to TCP sockets (and are used by **net.Socket( )** objects). Buffer objects are global to the application.

A buffer object is made up of two elements:

- an array of bytes and
- the length of the array.

Buffers can be created directly using the **Buffer( )** constructor or through other methods returning binary data, such as the *data* argument of the callback for the 'data' event from **net.Socket( )**. You can also create a Buffer "reference" using the **slice( )** method.

Once created, the array of bytes cannot be resized. However, buffer contents can be modified without limitation. You can read or write buffer contents using various accessors, such as **readUInt16LE( )** or **writeUInt8( )**. It is also possible to access buffer contents using the JavaScript standard array indexing notation. For example:

```
var buf = new Buffer(4);
buf[0] = 0x00;
buf[1] = 0x00;
buf[2] = 0x80;
buf[3] = 0x3f;
```

The **length** attribute is the length in bytes of the buffer array.

You will use different methods depending on the byte order of data in buffers. "LE" means Little Endian and "BE" means Big Endian. Endianness is important, even for floating-point numbers. Intel processors use Little Endian, whereas ARM processors use Big Endian.

Buffer Class Constructor

Buffer( )

void **Buffer**(Number | Array | String *definition* [, String *encoding*])

Parameter	Type	Description
definition	Number, Array, String	Size (number or array) of the buffer or string to set to the buffer
encoding	String	Encoding method (if a string is passed in definition)

Description

The **Buffer( )** method is the constructor of the class objects of the *Buffer* type. It allows you to create new *Buffer* objects on the server.

You can create a new *Buffer* with one of the following constructors:

- This constructor creates a new buffer of a certain *number* of bytes. The buffer contents are undefined.

```
var myBuffer = new Buffer( number );
```

- This constructor creates a new buffer from an array object. The buffer contents are the elements of the given *byteArray*.

```
var myBuffer = new Buffer( byteArray );
```

- This constructor creates a new buffer from a string. The buffer content is the string. In this case, you can provide an *encoding* method for the string. By default, 'utf8' is used.

```
var myBuffer = new Buffer( myString, encoding );
```

Available values for *encoding* are:
  - 'ascii' - for 7-bit ASCII data only. This encoding method is very fast, and will strip the high bit if set.
  - 'utf8' (default value) - Multi-byte encoded Unicode characters.
  - 'ucs2' - 2-bytes, Little Endian encoded Unicode characters. It can encode only BMP (Basic Multilingual Plane, U+0000 - U+FFFF).
  - 'hex' - Encode each byte as two hexadecimal characters.
  - 'base64' - Base64 string encoding.
    Note: A specific processing is applied when you pass 'base64' in encoding: with this statement, you indicate that the string is already encoded in 'base64'. Thus, the constructor will only create a buffer containing the base64-encoded string binary data. No encoding is done actually. Of course, the string must be a valid base-64 encoded string.

Example

This example creates a buffer of 16KB filled with 0xFF:

```
var vData =new Buffer(16*1024);
vData.fill(0xFF);
```

byteLength( )

Number **byteLength**(String *string* [, String *encoding*])

Parameter	Type	Description
string	String	String to evaluate
encoding	String	Encoding method
Returns	Number	Byte length of the string

Description

The **byteLength( )** constructor method can be used with the **Buffer( )** constructor to return the size in bytes of the given *string* when encoded using a specific *encoding*.

In *encoding*, pass the method to use for evaluating the byte size. Available values for *encoding* are:

- 'ascii' - for 7 bit ASCII data only. This encoding method is very fast, and will strip the high bit if set.
- 'utf8' (default value if omitted) - Multi-byte encoded Unicode characters.

- 'ucs2' - 2-bytes, Little Endian encoded Unicode characters. It can encode only BMP (Basic Multilingual Plane, U+0000 - U+FFFF).
- 'hex' - Encode each byte as two hexadecimal characters.
- 'base64' - Base64 string encoding.
  Note: A specific processing is applied when you pass 'base64' in encoding: with this statement, you indicate that the string is encoded in 'base64'. Thus, byteLength( ) will return the byte length of the base64-encoded string decoded in binary.

isBuffer( )

Boolean **isBuffer**(Object *obj*)

Parameter	Type	Description
obj	Object	Object to test
Returns	Boolean	True if obj is a buffer, False otherwise

Description

The **isBuffer( )** constructor method can be used with the **Buffer( )** constructor to check if the given *obj* is a buffer.
The method returns True if *obj* is a buffer; otherwise it returns False.

Buffer Instances

length

Description

The **length** property returns the size in bytes of the buffer.

The returned value is the memory size allocated to the buffer; it does not depend on the actual contents of the buffer. Copying data or filling the buffer will not change this **length**. Keep in mind that once a *Buffer* object is created, it cannot be resized.

copy( )

void **copy**(Buffer *targetBuffer* [,Number *targetOffset* [,Number *sourceOffset* [,Number *sourceEnd*]]])

Parameter	Type	Description
targetBuffer	Buffer	Buffer to get the copied data
targetOffset	Number	Destination position for the copy (0 if omitted)
sourceOffset	Number	Source position for the copy (0 if omitted)
sourceEnd	Number	Position of last byte to copy in source (buffer.length if omitted)

Description

The **copy( )** method copies into *targetBuffer* the *Buffer* to which it is applied.

The copy starts at the position defined by *sourceOffset* in the source *Buffer* and takes place at the position defined by *targetOffset* in *targetBuffer*. If *targetOffset* is omitted, the data will be copied at the beginning of the *targetBuffer* (equivalent to 0). If *sourceOffset* is omitted, the data will be taken from the beginning of the source *Buffer* (equivalent to 0).

The position of the last byte to copy from within the source buffer can be specified by *sourceLength*. If this parameter is omitted, the whole source buffer contents will be copied.

If the number of bytes to copy is greater than the *targetBuffer* length, an error is generated.

Example

In the example we create two buffers, copy buffer b1 from byte 8 to 22 and paste it into buffer b2 starting from the 6th byte:

```
b1 = new Buffer(26);
b2 = new Buffer(26);

for (var i = 0 ; i < 26 ; i++) {
  b1[i] = i + 65; // 65 is "A" character in UTF8
  b2[i] = 42; // 42 is "*"
}
b1.copy(b2, 6, 8, 22);
b2.toString('utf8', 0, 26);
// returns "******IJKLMNOPQRSTUV******"
```

fill( )

void **fill**(String *value* [, Number *offset*[, Number *length*]])

Parameter	Type	Description
value	String	Filler character
offset	Number	Offset within the buffer (expressed in bytes)
length	Number	Number of bytes to fill

Description

The **fill( )** method fills the *Buffer* to which it is applied with the character you passed in *value*.

In *offset*, pass the starting position where bytes must be inserted and in *length*, the number of bytes to be inserted. If you omit these parameters, the method fills the entire buffer.

Example

We want to fill a new buffer with zero characters.

```
var b = new Buffer(20);
b.fill("0");
var see = b.toString();
// see contains "00000000000000000000"
```

readDoubleBE( )

Number **readDoubleBE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	64-bit double value with Big Endian format

Description

The **readDoubleBE( )** method returns a 64 bit double value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readDoubleLE( )

Number **readDoubleLE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	64-bit double value with Little Endian format

Description

The **readDoubleLE( )** method returns a 64 bit double value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readFloatBE( )

Number **readFloatBE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	32-bit float value with Big Endian format

Description

The **readFloatBE( )** method returns a 32-bit float value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readFloatLE( )

Number **readFloatLE**(Number *offset*, Boolean *noAssert*)

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	32-bit float value with Little Endian format

Description

The **readFloatLE( )** method returns a 32-bit float value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt16BE( )

Number **readInt16BE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Signed 16-bit integer value with Big Endian ordering format

Description

The **readInt16BE( )** method returns an unsigned 16-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no validation will be performed on the *offset* value, that is, it can be beyond the *Buffer*.

readInt16LE( )

Number **readInt16LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Signed 16-bit integer value with Little Endian ordering format

Description

The **readInt16LE( )** method returns a signed 16-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt24BE( )

Number **readInt24BE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Signed 24-bit integer value with Big Endian ordering format

Description

The **readInt24BE( )** method returns a signed 24-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt24LE( )

Number **readInt24LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Signed 24-bit integer value with Little Endian ordering format

Description

The **readInt24LE( )** method returns a signed 24-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt32BE( )

Number **readInt32BE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation
Returns	Number	Unsigned 32-bit integer value with Big Endian format

Description

The **readInt32BE( )** method returns a signed 32-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt32LE( )

Number **readInt32LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation
Returns	Number	Signed 32-bit integer value with Little Endian format

Description

The **readInt32LE( )** method returns a signed 32-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readInt8( )

Number **readInt8**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Signed 8-bit integer value

Description

The **readInt8( )** method returns a signed 8-bit integer value read from the *Buffer* to which it is applied. The value is read at the *offset* within the buffer. Since the method returns a signed value, buffer contents are treated as two's complement signed values.

If you pass **true** in the *noAssert* parameter, no validation will be performed on the *offset* value, that is, it can be beyond the *Buffer*.

readUInt16BE( )

Number **readUInt16BE**([Number *offset* [, Boolean *noAssert*]])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Unsigned 16-bit integer value with Big Endian ordering format

Description

The **readUInt16BE( )** method returns an unsigned 16-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no validation will be performed on the *offset* value, that is, it can be beyond the *Buffer*.

readUInt16LE( )

Number **readUInt16LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Unsigned 16-bit integer value with Little Endian ordering format

Description

The **readUInt16LE( )** method returns an unsigned 16-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readUInt24BE( )

Number **readUInt24BE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Unsigned 24-bit integer value with Big Endian ordering format

Description

The **readUInt24BE( )** method returns an unsigned 24-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readUInt24LE( )

Number **readUInt24LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Unsigned 24-bit integer value with Little Endian ordering format

Description

The **readUInt24LE( )** method returns an unsigned 24-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readUInt32BE( )

Number **readUInt32BE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation
Returns	Number	Unsigned 32-bit integer value with Big Endian format

Description

The **readUInt32BE( )** method returns an unsigned 32-bit integer value read from the *Buffer* with the Big Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readUInt32LE( )

Number **readUInt32LE**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation
Returns	Number	Unsigned 32-bit integer value with Little Endian format

Description

The **readUInt32LE( )** method returns an unsigned 32-bit integer value read from the *Buffer* with the Little Endian format. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *offset* argument. In this mode, if the *offset* is located beyond the limit of the buffer, no exception is generated and the bytes returned by the method are filled with zeros (0).

readUInt8( )

Number **readUInt8**(Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation
Returns	Number	Unsigned 8-bit integer value

Description

The **readUInt8( )** method returns an unsigned 8-bit integer value read from the *Buffer* to which it is applied. The value is read at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no validation will be performed on the *offset* value, that is, it can be beyond the *Buffer*.

slice( )

Buffer **slice**(Number *start* [, Number *end*])

Parameter	Type	Description
start	Number	Starting element in the original byte array
end	Number	Last element to get
Returns	Buffer	New buffer referencing memory

Description

The **slice( )** method creates a new *Buffer* object by referencing the contents of the bytes array of the *Buffer* to which it is applied, from *start* to *end*.

In *start*, pass the element index from which to get contents (starting at 0) and in *end*, the last element index to get. In other words, the length of the new buffer is *end* - *start*. If you omit the *end* parameter, the new buffer references all data until the end of the source buffer.

Keep in mind that, since the new buffer is a reference to a part of the source buffer, any modification of its contents will also affect the source, and vice versa.

Buffer slices are very useful when performance is an issue, since they avoid copying and duplicating data. However, they should be handled with care because modifications in one buffer can affect other ones.

toBlob( )

Blob **toBlob**([String *mimeType*])

Parameter	Type	Description
mimeType	String	Media type of the Blob
Returns	Blob	Blob object containing a copy of the Buffer

Description

The **toBlob( )** method returns a *Blob* object containing a copy of the *Buffer* bytes.

In the optional *mimeType* parameter, you can pass a lower case string representing the media type of the Blob, expressed as a MIME type (see RFC2046). By default if you omit this parameter, the *Blob* media type is "application/octet-stream".

Pay attention to the size of manipulated objects since the method creates in memory a copy of the Buffer. An error is thrown if there is not enough memory available to execute the operation.

Note: The toBlob( ) method is not part of the W3C Buffer Interface specification.

toString( )

String **toString**(String *encoding* [, Number *start*[, Number *end*]])

Parameter	Type	Description
encoding	String	Encoding method
start	Number	Starting byte to return (0 if omitted)
end	Number	Position of the last byte to return (buffer.length if omitted)

Returns	String	Buffer contents expressed as string

Description

The **toString( )** method converts the buffer contents into a string.

Data are decoded using the *encoding* method. Available values for *encoding* are:

- 'ascii' - for 7-bit ASCII data only. This encoding method is very fast, and will strip the high bit if set.
- 'utf8' (default value) - Multi-byte encoded Unicode characters.
- 'ucs2' - 2-bytes, Little Endian encoded Unicode characters. It can encode only BMP (Basic Multilingual Plane, U+0000 - U+FFFF).
- 'hex' - Encode each byte as two hexadecimal characters.
- 'base64' - Base64 string encoding.
  Note: A specific processing is applied when you pass 'base64' in encoding: in this case, toString( ) returns a base-64 encoded string corresponding to the binary data stored in the buffer. This is actually the opposite of the other encoding values, where the binary data in the buffer is decoded in a string using the encoding method.

In *start*, pass the starting byte to convert and in *end*, the position of the last byte to convert. If you omit these parameters, the method converts the whole buffer.

write( )

Number **write**(String *string* [, Number *offset*] [, String *encoding*])

Parameter	Type	Description
string	String	String to write
offset	Number	Byte position to start writing
encoding	String	Encoding method
Returns	Number	Number of bytes actually written

Description

The **write( )** method writes the *string* parameter to the *Buffer* at the *offset* position and returns the number of bytes written. If the buffer cannot contain the whole *string*, the string is truncated according to the size of the buffer.

In *encoding*, pass the method to use for writing characters. Available values for *encoding* are:

- 'ascii' - for 7-bit ASCII data only. This encoding method is very fast, and will strip the high bit if set.
- 'utf8' (default value if omitted) - Multi byte encoded Unicode characters.
- 'ucs2' - 2-bytes, Little Endian encoded Unicode characters. It can encode only BMP (Basic Multilingual Plane, U+0000 - U+FFFF).
- 'base64' - Base64 string encoding.
- 'hex' - Encode each byte as two hexadecimal characters.

The number of characters actually written (which can be different from the number of bytes written, depending on the encoding) can be read from the *_charsWritten* property. This property is reset on each call of the **write( )** method. The number of bytes to write can be evaluated using the **byteLength( )** method.

writeDoubleBE( )

void **writeDoubleBE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeDoubleBE( )** method writes the 64-bit double *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeDoubleLE( )

void **writeDoubleLE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the Buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeDoubleLE( )** method writes the 64-bit double *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeFloatBE( )

void **writeFloatBE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeFloatBE( )** method writes the 32-bit float *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeFloatLE( )

void **writeFloatLE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeFloatLE( )** method writes the 32-bit float *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt16BE( )

void **writeInt16BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt16BE( )** method writes the 16-bit signed integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt16LE( )

void **writeInt16LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt16LE( )** method writes the 16-bit signed integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt24BE( )

void **writeInt24BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt24BE( )** method writes the 24-bit signed integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt24LE( )

void **writeInt24LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt24LE( )** method writes the 24-bit signed integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no

exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt32BE( )

void **writeInt32BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt32BE( )** method writes the 32-bit signed integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt32LE( )

void **writeInt32LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt32LE( )** method writes the 32-bit signed integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeInt8( )

void **writeInt8**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeInt8( )** method writes the 8-bit signed integer *value* to the *Buffer* to which it is applied. The value is written at the *offset* within the buffer. Since the *value* is signed, it is written out as a two's complement signed value.

If you pass **true** in the *noAssert* parameter, no validation will be performed on both the *value* and *offset* value, which means that the value size can be too large and *offset* can be beyond the *Buffer* boundary. Since this can lead to loss of values, it must be used carefully.

writeUInt16BE( )

void **writeUInt16BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer

offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt16BE( )** method writes the 16-bit unsigned integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no validation will be performed on both the *value* and *offset* value, which means that the value size can be too large and *offset* can be beyond the *Buffer* boundary. Since this can lead to loss of values, it must be used carefully.

writeUInt16LE( )

void **writeUInt16LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt16LE( )** method writes the 16-bit unsigned integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeUInt24BE( )

void **writeUInt24BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip offset validation, false or omitted = offset validation

Description

The **writeUInt24BE( )** method writes the 24-bit unsigned integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeUInt24LE( )

void **writeUInt24LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt24LE( )** method writes the 24-bit unsigned integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since

this option may lead to invalid bytes, it must be used with precaution.

writeUInt32BE( )

void **writeUInt32BE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt32BE( )** method writes the 32-bit unsigned integer *value* to the *Buffer* with the Big Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeUInt32LE( )

void **writeUInt32LE**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt32LE( )** method writes the 32-bit unsigned integer *value* to the *Buffer* with the Little Endian format. The *value* is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no internal validation will be performed on the *value* or *offset* arguments. In this mode, if the size of the *value* exceeds the buffer size or if the *offset* is located beyond the limit of the buffer, no exception is generated. In the first case, the value is partially written, and in the second case, no value is written. Since this option may lead to invalid bytes, it must be used with precaution.

writeUInt8( )

void **writeUInt8**(Number *value* , Number *offset* [, Boolean *noAssert*])

Parameter	Type	Description
value	Number	Value to write into the buffer
offset	Number	Offset within the buffer (expressed in bytes)
noAssert	Boolean	true = skip value and offset validation, false or omitted = do validation

Description

The **writeUInt8( )** method writes the 8-bit unsigned integer *value* to the *Buffer* to which it is applied. The value is written at the *offset* within the buffer.

If you pass **true** in the *noAssert* parameter, no validation will be performed on both the *value* and *offset* value, which means that the value size can be too large and *offset* can be beyond the *Buffer* boundary. Since this can lead to loss of values, it must be used carefully.