XS chunk

Draft 1.0
October 7, 2004

1 Introduction

The chunk element, constructor and prototype provide XML and ECMAScript interfaces to binary data in XS. A chunk object is a host object whose host data is a memory block managed by the C functions *malloc, realloc* and *free* (or the platform dependent corresponding functions).

In XML, a chunk is parsed from and serialized to base 64 text. In ECMAScript, the chunk constructor and prototype allows scripts to create chunks, to convert chunks from and to base 64 strings, and so on… The XS in C programming interface allows the use of the memory block in C.

2 Element

Like the boolean, date, number, regexp and string elements, the chunk element is a literal property element.

The value attribute of the chunk element contains a base 64 value that becomes the data of the chunk property that the element prototypes. The default value is "".

The parser decodes a base 64 text node into the data of the chunk property. The serializer encodes a base 64 text node from the data of the chunk property.

Grammar:
```
<object name="user" pattern="/user"/>
      <chunk name="signature" value="" pattern="."/>
</object>
```

Prototype:
```
user = {
      /* signature: chunk without data */
}
```

Document:
```
<user>
      VHJpcGVsIFdlc3RtYWxsZQ==
</user>
```

Instance:
```
{     /* proto: user */
      /* signature: chunk with data */
}
```

3 Constructor

Chunk()
If the *Chunk* constructor is invoked without argument, it returns a new chunk whose length is 0. No memory block is allocated.

Chunk(number)

If the argument is a number, the *Chunk* constructor converts it into an integer and returns a new chunk whose length is *number*. A memory block is allocated with the C function *malloc* but uninitialized. The *number* has to be >= 0 else a *RangeError* exception is thrown.

Chunk(string)
If the argument is a string, the *Chunk* constructor returns a new chunk whose length is the number of bytes necessary to decode the *string* as base 64 text. A memory block is allocated with the C function *malloc* and initialized by decoding the *string* as base 64 text. Invalid characters and invalid sequences of characters are skipped.

Example:
```
chunk = new Chunk()
chunk = new Chunk(1024)
chunk = new Chunk("VHJpcGVsIFdlc3RtYWxsZQ==")
```

A chunk is a host object. When the garbage collector reclaims a chunk, it invokes its destructor. The destructor calls the C function *free* to deallocate the memory block, if any.

4 Prototype

For all functions, *this* has to be an instance of *Chunk.prototype* else a *TypeError* exception is thrown.

Chunk.prototype.length
The *length* property has getter and setter functions. The getter function returns the number of bytes in the data of *this*. The setter function change the number of bytes in the data of *this* with the C function *realloc*.

Chunk.prototype.append(chunk)
The *append* function takes one argument, *chunk* and append its data to the data of *this*. The length of the *chunk* is added to the length of *this*. The *chunk* has to be an instance of *Chunk.prototype* else a *TypeError* exception is thrown.

Chunk.prototype.free()
The *free* function invokes the destructor of *this* and set the length of *this* to 0. Use the *free* function to recover memory space immediately instead of waiting for the garbage collector to invoke the destructor.

Chunk.prototype.peek(offset)
The *peek* function takes one argument, *offset*, converts it into an integer and returns the byte at position *offset* in the data of *this*. The *offset* has to be >= 0 and < *this.length* else a *RangeError* exception is thrown. The result is always >= 0 and < 256.

Chunk.prototype.poke(offset, value)
The *poke* function takes two arguments, *offset* and *value*, converts them into integers and replaces the byte at position *offset* in the data of *this* by the *value*. The *offset* has to be >= 0 and < *this.length* else a *RangeError* exception is thrown. The *value* has to be >= 0 and < 256 else a *RangeError* exception is thrown.

Chunk.prototype.slice(start, end)
The *slice* function takes two arguments, *start* and *end,* converts them into integers and returns a new chunk whose length is *end – start*. A memory block is allocated with the C function *malloc* and initialized by copying bytes from the data of *this*, starting from position *start* and running to, but not including,

position *end.* If *start* is *undefined*, it is treated a 0. If *start* is negative, it is treated as *this.length* + *start*. If *end* is *undefined*, it is treated a *this.length*. If end is negative, it is treated as *this.length* + *end*. Use the *slice* function without arguments to duplicate a chunk.

Chunk.prototype.toString()
The *toString* function encodes the data of *this* into base 64 text and returns it in a string value.

Example:
```
chunk = new Chunk("VHJpcGVsIFdlc3RtYWxsZQ==")
for (i = 0; i < chunk.length; i++)
      chunk.poke(i, 255 - chunk.peek(i))
chunk.append(chunk)
chunk = chunk.slice(-3)
trace(chunk.toString())
chunk.free()
```

5 In C

To handle chunks in C, use the XS in C programming interface to invoke the *Chunk* constructor and call functions of the *Chunk.prototype*. To get a pointer to the memory block, use the *xsGetHostData* macro.

Example:
```
void* data;
xsResult = xsNew1(xsGet(xsGlobal, xsID("Chunk")), xsInteger(1024));
data = xsGetHostData(xsResult);
memset(data, 0, 1024);
```

Use the length getter and setter to get and set the size of the memory block. Changing the length of a chunk can change the pointer to the memory block.

Example:
```
XsIntegerValue length;
void* data;
length = xsToInteger(xsGet(xsArg(0), xsID("length")));
xsSet(xsArg(0), xsID("length"), xsInteger(length + 1024));
data = xsGetHostData(xsArg(0));
```

With the *xsSetHostData* macro, it is possible to build a "fake" chunk, i.e. a chunk whose data is no memory block managed by the C functions *malloc, realloc* and *free.* However it requires to use the *xsSetHostDestructor* macro to change the destructor and then to set the length of the chunk.

Example:
```
xsResult = xsNewInstanceOf(xsChunkPrototype);
xsSetHostData(xsResult, (void*)0x12345678);
xsSetHostDestructor(xsResult, NULL);
xsSet(xsResult, xsID("length"), xsInteger(9));
```