Carpentry with Duktape

Tyler Brock, Hustle Inc.
JS Culture
MORE

JAVASCRIPT!
What is Duktape?

• Embeddable ECMAScript 5 Engine
  • Extend functionality of C/C++ code through flexible scripting
  • Leverage speedy C functions from within JavaScript program
Features

• ES5 and 5.1 support
• Tiny amount of ES6 (proxy object subset, setPrototypeOf, …)
• Some browser enhancements (like print + alert)
• Built in debugger
• CommonJS, regex, and unicode support (cross platform)
Project Goals

• Compliance
• Portability
• Easy C Interface
• Small footprint
• Reasonable Performance (ASCII String Performance)
What do I need?

- duktape.h (provides API)
- duktape.c (provides implementation)
- duk_config.h (configuration -- included by duktape.h)
How do I use it?

```
#include "duktape.h"

int main() {
    duk_context *ctx = duk_create_heap_default();
    duk_eval_string(ctx, "print('Hello world!');");
    duk_destroy_heap(ctx);
}
```
Leverage JS flexibility from C/C++

```javascript
function cool(str) {
  print('working magic...');
  var coolerStr = str + " is cool";
  return coolerStr;
}
```
duk_eval_file()

Prototype

```c
void duk_eval_file(duk_context *ctx, const char *path);
```

Stack

Like `duk_eval()`, but the eval input is given as a filename. The filename associated with the temporary function created for the eval code is path as is.

NOTE: The path argument is given directly to fopen(). The path encoding cannot be specified and there is no support for a wide character string.

Example

```c
duk_eval_file(ctx, "test.js");
printf("result is: \%s\n", duk_safe_to_string(ctx, -1));
duk_pop(ctx);
```

See also

- `duk_eval_file_noresult`
duk_eval_file()

Prototype

```c
void duk_eval_file(duk_context *ctx, const char *path);
```

Stack

```plaintext
... → ... result
```

Summary

Like `duk_eval()`, but the eval input is given as a filename. The filename associated with the temporary function created for the eval code is `path` as is.
**duk_push_string()**

**Prototype**

```c
const char *duk_push_string(duk_context *ctx, const char *str);
```

**Stack**

- `(str != NULL)`
  - `...` → `... str`

- `(str == NULL)`
  - `...` → `... null`

**Summary**

Push a C string into the stack. String length is automatically detected with a `strlen()` equivalent (i.e. looking for the first NULL character). A pointer to the interned string data is returned. If the operation fails, throws an error.
duk_call()

Prototype:

```c
void duk_call(duk_context *ctx, duk_idx_t nargs);
```

Stack:

```
... func arg1 ... argN    →    ... retval
```

Summary:

Call target function `func` with `nargs` arguments (not counting the function itself). The function and its arguments are replaced by a single return value. An error thrown during the function call is not automatically caught.
Run the function

duk_eval_file(ctx, "cool.js");
duk_push_string(ctx, "duktape");

duk_call(ctx, 1);
printf("%s\n", duk_safe_to_string(ctx, -1));

working magic...
duktape is cool
Stack Attack

eval_file()  

push_string()  

cool()

'duktape'

cool()

'duktape is cool'

call()  
to_string()
int adder(duk_context *ctx) {
    int i;
    int n = duk_get_top(ctx); // # of arguments
    double res = 0.0;

    for (i = 0; i < n; i++) {
        res += duk_to_number(ctx, i); // add em up
    }

    duk_push_number(ctx, res);
    return 1; // single return value
}
Register and Run the C Function

```javascript
duk_push_global_object();
duk_push_c_function(ctx, adder, DUK_VARARGS);
duk_put_prop_string(ctx, 0, "adder");
duk_pop(ctx); /* pop global */

duk_eval_string(ctx, "print('2+3=' + adder(2, 3));");
```
Return Values

• Return 1 indicates that there is a single return value on top of the value stack
• Return 0 indicates that there is no return value (undefined)
• Return negative number causes an error to be thrown
Stack Attack

push_global()

push_c_func()

dadder()

put_prop()

pop()
Endless possibilities...

- Dukluv by Creationix (libuv bindings)
- Dukcurl by Creationix (cURL bindings)