Cycript

Cycript	
Cydia Package	
Developer	saurik
Package ID	cycript
Latest Version	0.9.594

Cycript is a JavaScript interpreter which also understands Objective-C syntax. The `cycript` binary is also a REPL (https://en.wikipedia.org/wiki/Read-eval-print_loop) for this language.

Besides evaluating scripts, Cycript can also hook into a running process (using `cycript -p` *process*) and modify its property.

Official documentation can be found on cycript.org (http://www.cycript.org/manual).

Presentations about Cycript: saurik explaining how to use Cycript on OS X, targeting the iOS simulator (https://www.youtube.com/watch?v=5d1cK0nq4GY); Adam Bell explaining how to use Cycript to prototype a tweak (https://www.youtube.com/watch?v=Ox09rWJTuCA) (example code (https://github.com/b3ll/JailbreakCon2013)).

Contents

JS/ObjC Object Bridging

Some native JavaScript types are bridged to the corresponding Objective-C types for convenient, so you can use

```
[[41,"foo",true,[8,6],{a:12,b:46},36] indexOfObject:"foo"]
```

instead of

```
[[NSArray arrayWithObjects:
  [NSNumber numberWithInt:41],
  "foo",
  [NSNumber numberWithBool:YES],
  [NSArray arrayWithObjects:[NSNumber numberWithInt:8], [NSNumber numberWithInt:6], nil],
  [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:12], "a",
    [NSNumber numberWithInt:46], "b",
  nil],
  [NSNumber numberWithInt:36],
nil] indexOfObject:"foo"]
```

JS type	ObjC type
number	NSNumber (CFNumber)
boolean	NSNumber (CFBoolean)
string	NSString
Array	NSArray
object (Associative array)	NSDictionary

`null` in Cycript is equivalent to `nil` in Objective-C. Additionally, `nil`, `YES` and `NO` are also defined in Cycript.

JavaScript 1.6+ Features

Cycript supports the following JavaScript 1.6+ features extended by Mozilla:

- `for each/in` (https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Statements/for_each...in) (JS1.6)
- Array comprehensions (https://developer.mozilla.org/en/New_in_JavaScript_1.7#Array_comprehensions) (JS1.7)
- E4X (not available on iPhoneOS).

Additional syntax

New Syntax	Meaning	"Desugared" representation
[*obj* *msg*:*var*]	Sends *msg* to *obj* with parameters *var*. This is the Objective-C's message sending syntax.	`objc_msgSend(`*obj*`, @selector(`*msg*`).value, `*var*` ...)`
`@import(`*module*`)`	Import a .cy file. Similar to JavaScript's *require()* function.	
`@selector(`*selname*`)`	Returns the selector named *selname* with Objective-C syntax.	`new Selector("`*selname*`");`
obj`->`*ivar*	Obtain the instant variable *ivar* of an Objective-C object *obj*.	*obj*`.$cyi.`*ivar*
`*`*ptr*	Dereference the pointer, or list all ivars of an object (so that you can access them using `(*`*obj*`).`*ivar*).	*ptr*`.$cyi`
obj`->[`*key*`]`	Equivalent to `(*`*obj*`)[`*key*`]`.	
`&`*var*	Takes the address of a variable. Only instances of ObjC class can have addresses.	*var*`.$cya()`

`@class` *classname* `:` *superclass* `{}`
`+` *methodname* `{` *function body* `}`
`-` *methodname* `{` *function body* `}`
`...`
`@end`
	Declare an Objective-C class. The *classname* can be omitted, where an anonymous class will be declared.	

`@class` *existingclass*
`+` *methodname* `{` *function body* `}`
`-` *methodname* `{` *function body* `}`
`...`
`@end`
	Insert extra methods to an existing class. The *existingclass* itself can be an expression e.g. `@class ([obj class]) ....`	

`new` *classname*	Although not exactly a new syntax, this construction has a new meaning for Objective-C classes. This is similar to `[`*classname* `alloc]`, but the resource will be managed by JavaScriptCore's garbage collector. To fully initialize the class, you need to call `[new` *classname* `initWithFoo:...]`.	
`@"`*str*`"`	Equivalent to `"`*str*`"`.	
`[super ...]`	A local variable representing the superclass.	`objc_msgSend(???, ... )`
`0b`*xxxxxx*	Binary literal.	

REPL-only additions

These are used for debugging.

Line	Usage
`?debug`	Toggles debug output.
`?bypass`	Bypass syntax error pretty-printing.
`?expand`	Toggles whether to display the line break characters, etc. as really a line break or just `\n`.
`?gc`	Force a JavaScript garbage collection.
`?syntax`	Toggle syntax highlighting.

Additional types

Type/Constructor	Usage
`Selector(`*selname*`)`	Declare a selector.
`Functor(`*function body*, *type encoding*`)`	Associate an Objective-C type encoding to a function, e.g. `new Functor(function(x,y){return (x+y).toString(16);}, "*dd");` to declare a (**double**, **double**) → **char*** function.
`Pointer(`*address*, *type encoding*`)`	Treat the input number as a pointer. Like C pointers, the result can be dereferenced using `*` and subscripted using `[`*i*`]`, but pointer arithmetic is not directly supported.
`Type(`*type encoding*`)`	Create a type. The resulting value can be `new`-ed to get a `Pointer`, e.g. `var p = new new Type("d");`. To deallocate the pointer, use the `free()` function.
`Instance(`*address*`)`	Treat the address as an instance of Objective-C object.
`Super(`*self*, *selector*`)`	Returns an object which, when being sent a message, will be forwarded to *self*'s superclass.

Additionally, the identifiers like `int`, `id`, `char`, `double`, etc. are predefined to the corresponding types (`new Type("i")`, etc). Therefore, to allocator a pointer you may simply use `new int` or even `new int[42]`.

Additional variables and methods

Variable	Meaning
`_`	Last evaluated value (REPL only).
`ObjectiveC.images`	An associative array, with keys beings the path of loaded libraries, and value is the classes of this library.
`ObjectiveC.classes`	An associative array of classes. The keys are class names and the values are the classes themselves.
`ObjectiveC.protocols`	An associative array of protocols. The keys are protocol names and the values are the protocols themselves.
obj`.toJSON()`	Convert the object to JSON.
obj`.toCYON()`	Convert the object to CYON (Cycript object notation).
obj`.value`	For some objects, returns the address.
class`.messages`	Contains an associative array of messages in the class. The keys are the selector names and the values are implementations (`function`s).
`system.print(`*string*`)`	Print the string to syslog.
`system.args`	Parameters of the executable
selector`.type(`*class*`)`	Returns the type encoding for the *selector* in *class*. For example, `@selector(copyWithZone:).type(NSString)` returns `@12@0:4^{_NSZone=}8`.

Other uses

The cycript binary can be used to "compile" Cycript into standard JavaScript 1.5 with the `-c` flag, e.g.

```
Your-iPhone:~ mobile$ echo "[x*x for each(x in [1,2,3])]" | cycript -c > x.js
Your-iPhone:~ mobile$ cat x.js
(function($cyv,x){$cyv=[];(function($cys){$cys=[1,2,3];for(x in $cys){x=$cys[x];$cyv.push(x*x)}})();return $cyv})()
```

Considerations

- Every command typed into the console is run in an autorelease pool so variables "declared" in one command might be deallocated by the time the next command that uses it is run.

See also

- Cycript Tricks

External links

- Official website (http://www.cycript.org/)
- Source: http://gitweb.saurik.com/cycript.git (git://git.saurik.com/cycript.git)

Objective-C bridges [hide]

PyObjC • JocStrap • JSCocoa • Cycript

Retrieved from "https://iphonedevwiki.net/index.php?title=Cycript&oldid=4769"

This page was last edited on 30 November 2016, at 22:13.