Cheatography

# Extending Ruby with C - Part 1 Cheat Sheet

by Ryan Johnson (CITguy) via cheatography.com/138/cs/248/

## Ruby C - Define Objects

V **rb_define_class**(char *name, V superclass)

> Defines new class at top level with given *name* and *superclass*.

V **rb_define_module**(char *name)

> Defines new module at top level with given *name*.

V **rb_define_class_under**(V under, char *name, V superclass)

> Defines nested class under class or module *under*.

V **rb_define_module_under**(V parent, V module)

> Defines nested module under class or module *under*.

void **rb_include_module**(V parent, V module)

> Includes given *module* into class or module *parent*.

void **rb_extend_object** (V obj, V module)

> Extends *obj* with *module*.

V **rb_require**(const char *name)

> Equiv. to "require *name." Returns Qtrue or Qfalse.

V = VALUE

## Ruby C - Calling Methods

V **rb_funcall**(V recv, ID id, int argc, ...)

> Invokes method given by *id* in object *recv* with given number of args *argc* and args themselves.

V **rb_funcall2**(V recv, ID id, int argc, V *args)

> Invokes method given by *id* in object *recv* with given number of args *argc* and args themselves given in C array *args*.

V **rb_funcall3**(V recv, ID id, int argc, V *args)

> Same as *rb_funcall2*, but will not call private methods.

V **rb_apply**(V recv, ID name, int argc, V args)

> Invokes method given by *id* in object *recv* with given number of args *argc* and the args themselves given in Ruby Array *args*.

## Ruby C - Calling Methods (cont)

ID **rb_intern**(char *name)

> Returns ID for given *name*. If name does not exist, a symbol table entry will be created for it.

char * **rb_id2name**(ID id)

> Returns a name for the given *id*.

V **rb_call_super**(int argc, V *args)

> Calls current method in superclass of current object.

V = VALUE

## Ruby C - Object Status

**OBJ_TAINT**(VALUE obj)

**OBJ_FREEZE**(VALUE obj)

int **OBJ_TAINTED**(VALUE obj) => 0|nonzero

int **OBJ_FROZEN**(VALUE obj) => 0|nonzero

## Ruby C - Defining Variables and Constants

void **rb_define_const**(VALUE classmod, char *name, VALUE value)

> Defines constant in class or module *classmod*, with given *name* and *value*.

void **rb_define_global_const**(char *name, VALUE value)

> Defines global constant with given *name* and *value*.

void **rb_define_variable**(const char *name, VALUE *object)

> Exports address of given *object* that was created in C, to the Ruby namespace as *name*. To Ruby, this will be a global variable, so *name* should have "$" prefix. Be sure to honor Ruby's rules for allowed variable names.

void **rb_define_class_variable**(VALUE class, const char *name, VALUE val)

> Defines class variable *name* (must specify "@@" prefix) in given *class*, initialized to *value*.

## Ruby C - Defining Variables and Constants (cont)

void **rb_define_virtual_variable**(const char *name, VALUE (*getter)(), void (*setter)())

> Exports virtual variable to Ruby namespace as global *$name*. No actual storage exists for variable; attempts to get/set value will call the appropriate functions.

void **rb_define_hooked_variable**(const cahr *name, VALUE *variable, VALUE (*getter)(), void (*setter)())

> Defines functions to be called when reading/writing to *variable*. (See also **rb_define_virtual_variable**.)

void **rb_define_readonly_variable**(const char *name, VALUE *value)

> Same as rb_define_variable, but read-only from Ruby.

void **rb_define_attr**(VALUE variable, const char *name, int read, int write)

> Creates accessor methods for given *variable*, with given *name*. If *read* is nonzero, crate read method; if *write* is nonzero, create write method.

void **rb_global_variable**(VALUE *obj)

> Registers given address with garbage collector.

## Ruby C - Security Status

**Check_SafeStr**(VALUE str)

> Raises SecurityError if current safe level > 0 and *str* is tainted, or a TypeError if *str* is not a T_STRING.

int **rb_safe_level**()

void **rb_secure**(int level)

> Raises SecurityError if *level* <= current safe level.

void **rb_set_safe_level**(int newlevel)

By **Ryan Johnson** (CITguy)
cheatography.com/citguy/

Published 15th February, 2012.
Last updated 13th May, 2016.

Cheatography

# Extending Ruby with C - Part 1 Cheat Sheet

by Ryan Johnson (CITguy) via cheatography.com/138/cs/248/

## Ruby C - Defining Methods

void **rb_define_method**(V classmod, char *name, V (*func)(), int argc);

> Defines instance method in class or module *classmod* with given *name*, implemented by C function *func* and taking *argc* args. ***(See Ruby C - Function Prototype)***

void **rb_define_module_function**(V classmod, char *name, V (*func)(), int argc);

> Defines method in class *classmod* with given *name*, implemented by C function *func* taking *argc* args.

void **rb_define_global_function**(char *name, V (*func)(), int argc);

> Defines global function (private Kernel method) with given *name*, implemented by C function *func* and taking *argc* args.

void **rb_define_singleton_method**(V classmod, char *name, V (*func)(), int argc);

> Defines singleton method in class *classmod* with given *name*, implemented by C function *func* taking *argc* args.

int **rb_scan_args**(int argc, V *argv, char *fmt, ...)

> Scans argument list and assigns to variables similar to scanf: *fmt* is string containing zero, one, or two digits followed by optional flag chars. First char indicates count of mandatory args; second is count of optional args. A "*" means to pack remaining args into Ruby array. A "&" means attached code block will be taken and assigned to given variable (Qnil will be assigned if no code block given). After *fmt* string, pointers to VALUE are given to which args are assigned.

## Ruby C - Defining Methods (cont)

void **rb_undef_method**(V classmod, const char *name);

> Undefines method *name* in class or module *classmod*.

void **rb_define_alias**(V classmod, const char *newname, const char *oldname);

> Defines alias for *oldname* in class or module *classmod*.

V = VALUE

## Ruby C - Function Prototype

(argc) **0..17** VALUE func(*VALUE self, VALUE arg...*)

> C function will be called with this many arguments.

(argc) **-1** VALUE func(*int argc, VALUE *argv, VALUE self*)

> C function will be given a variable number of arguments passed as a C array.

(argc) **-2** VALUE func(*VALUE self, VALUE args*)

> C function will be given a variable number of arguments passed as a Ruby array.


By **Ryan Johnson** (CITguy)
cheatography.com/citguy/



Published 15th February, 2012.
Last updated 13th May, 2016.