sUNC Complete Function Reference

senS' Unified Naming Convention — All Functions with Examples

Source: docs.sunc.su | Compiled 2026

CLOSURES

---

checkcaller

Returns true if the current function was invoked from the executor thread, false if called by game code. Pair with hookfunction / hookmetamethod to filter executor-originated calls from game-originated ones.

Signature

```
function checkcaller(): boolean
```

Example

```
local from_caller
local original; original = hookmetamethod(game, "__namecall", function(...)
    if not from_caller then
        from_caller = checkcaller()
    end
    return original(...)
end)
task.wait(0.1)
hookmetamethod(game, "__namecall", original)
print(from_caller)    -- false (game called it)
print(checkcaller())  -- true  (executor thread)
```

clonefunction

Creates a new closure that shares the same underlying proto as the original but has independent upvalues. Useful for calling the original of a hooked function without going through the hook.

Signature

```
function clonefunction(func: (...any) -> (...any)): (...any) -> (...any)
```

Example

```
local function add(a, b) return a + b end
local cloned = clonefunction(add)
print(cloned(3, 4))  -- 7
```

getfunctionhash

Returns a hash string derived from the bytecode of the given function. Used to verify function integrity — if a function has been replaced or tampered with, its hash will differ.

Signature

```
function getfunctionhash(func: (...any) -> (...any)): string
```

Example

```
local function target() return 42 end
local hash = getfunctionhash(target)
print(hash)  -- "abc123..." (deterministic for same bytecode)
```

hookfunction

Replaces the implementation of target with hook. Returns the original function. Call the original inside hook for passthrough behavior. Works on both Lua closures and C closures.

Signature

```
function hookfunction(target: (...any) -> (...any), hook: (...any) ->
(...any)): (...any) -> (...any)
```

Example

```
local originalPrint = hookfunction(print, function(...)
    -- intercept all print calls
    return originalPrint("[HOOKED]", ...)
end)
print("hello")  -- [HOOKED] hello
```

hookmetamethod

Hooks a metamethod on an object's metatable — including protected metatables that setmetatable cannot touch. Internally uses hookfunction. Returns the original metamethod for passthrough.

Signature

```
function hookmetamethod(object: table | Instance | userdata, metamethodName:
string, hook: (...any) -> (...any)): (...any) -> (...any)
```

Notes

⚠ *Requires hookfunction to be properly implemented in C++. Can be implemented in pure Luau on top of hookfunction.*

Example

```
local original; original = hookmetamethod(game, "__index", function(...)
    local key = select(2, ...)
    print("[__index]", key)
    return original(...)
end)
local _ = game.PlaceId  -- Output: [__index] PlaceId
hookmetamethod(game, "__index", original)  -- restore
```

iscclosure

Returns true if the given function is a C closure (compiled C function), false if it is a Lua closure.

Signature

```
function iscclosure(func: (...any) -> (...any)): boolean
```

Example

```
print(iscclosure(print))            -- true  (C function)
print(iscclosure(function() end))   -- false (Lua closure)
```

isexecutorclosure

Returns true if the function was created by the executor environment, false for Roblox globals or game Lua functions.

Signature

```
function isexecutorclosure(func: (...any) -> (...any)): boolean
```

Example

```
print(isexecutorclosure(isexecutorclosure)) -- true
print(isexecutorclosure(print))             -- false
```

islclosure

Returns true if the given function is a Lua closure. Opposite of iscclosure.

Signature

```
function islclosure(func: (...any) -> (...any)): boolean
```

Example

```
print(islclosure(function() end))  -- true
print(islclosure(print))           -- false
```

loadstring

Compiles a string of Lua source code and returns it as a callable function. Returns nil and an error message on compilation failure. chunkname is optional and sets the debug name.

Signature

```
function loadstring(code: string, chunkname: string?): ((...any) ->
(...any))?, string?
```

Example

```
local fn, err = loadstring("return 1 + 2")
if fn then
    print(fn())  -- 3
else
    print("Error:", err)
end
```

newcclosure

Wraps a Lua function in a C closure wrapper. The resulting function appears as a C closure to iscclosure checks. Used to bypass closure-type detection in games.

Signature

```
function newcclosure(func: (...any) -> (...any)): (...any) -> (...any)
```

Example

```
local function myFunc() return "hello" end
print(iscclosure(myFunc))               -- false
local wrapped = newcclosure(myFunc)
print(iscclosure(wrapped))              -- true
print(wrapped())                        -- hello
```

restorefunction

Restores a previously hooked function to its original implementation. Reverses the effect of hookfunction.

Signature

```
function restorefunction(func: (...any) -> (...any)): ()
```

Example

```
local original = hookfunction(print, function(...)
    return original("[hooked]", ...)
end)
print("test")       -- [hooked] test
restorefunction(print)
print("test")       -- test
```

DEBUG

---

debug.getconstant

Returns the constant at the given index in the function's constant table. Constants are strings, numbers, and booleans baked into the bytecode.

Signature

```
function debug.getconstant(func: (...any) -> (...any), index: number): any
```

Example

```
local function greet() print("hello") end
print(debug.getconstant(greet, 1))  -- "print" or "hello" depending on index
```

debug.getconstants

Returns all constants from a function's bytecode as an array.

Signature

```
function debug.getconstants(func: (...any) -> (...any)): { any }
```

Example

```
local function test() return "world", 42 end
local consts = debug.getconstants(test)
for i, v in ipairs(consts) do
    print(i, v)
end
```

debug.getproto

Returns the nested function (proto) at the given index inside func. If activated is true, returns an active closure rather than just the proto.

Signature

```
function debug.getproto(func: (...any) -> (...any), index: number,
activated: boolean?): (...any) -> (...any)
```

Example

```
local function outer()
    local function inner() return "inner" end
end
local innerProto = debug.getproto(outer, 1, true)
print(innerProto())  -- inner
```

debug.getprotos

Returns all nested function protos inside func as an array.

Signature

```
function debug.getprotos(func: (...any) -> (...any)): { (...any) -> (...any)
}
```

Example

```
local function outer()
    local function a() end
    local function b() end
end
local protos = debug.getprotos(outer)
print(#protos)  -- 2
```

debug.getstack

Returns stack values at the given level. If index is provided, returns only that slot. level 1 is the calling function.

Signature

```
function debug.getstack(level: number, index: number?): any
```

Example

```
local function test()
    local x = 99
    print(debug.getstack(1, 1))  -- 99
```

```
end
test()
```

debug.getupvalue

Returns the name and value of the upvalue at the given index in func.

Signature

```
function debug.getupvalue(func: (...any) -> (...any), index: number):
string, any
```

Example

```
local count = 0
local function increment() count = count + 1 end
local name, val = debug.getupvalue(increment, 1)
print(name, val)  -- count  0
```

debug.getupvalues

Returns all upvalues of func as an array of values.

Signature

```
function debug.getupvalues(func: (...any) -> (...any)): { any }
```

Example

```
local x, y = 10, 20
local function fn() return x + y end
local ups = debug.getupvalues(fn)
print(ups[1], ups[2])  -- 10  20
```

debug.setconstant

Replaces the constant at the given index in the function's bytecode constant table.

Signature

```
function debug.setconstant(func: (...any) -> (...any), index: number, value:
any): ()
```

Example

```
local function greet() print("hello") end
debug.setconstant(greet, 2, "world")
greet()  -- world
```

debug.setstack

Sets the value of a stack slot at the given level and index.

Signature

```
function debug.setstack(level: number, index: number, value: any): ()
```

Example

```
local function test()
```

```
    local x = 1
    debug.setstack(1, 1, 99)
    print(x)  -- 99
end
test()
```

debug.setupvalue

Sets the upvalue at the given index in func to value.

Signature

```
function debug.setupvalue(func: (...any) -> (...any), index: number, value: any): ()
```

Example

```
local count = 0
local function increment() count = count + 1 end
debug.setupvalue(increment, 1, 100)
increment()
print(count)  -- 101
```

DRAWING

---

cleardrawcache

Clears all cached drawing objects. Frees memory used by Drawing library objects that have been created but not explicitly destroyed.

Signature

```
function cleardrawcache(): ()
```

Example

```
local circle = Drawing.new("Circle")
circle.Radius = 50
circle.Visible = true
-- later:
cleardrawcache()  -- wipes all drawing objects
```

getrenderproperty

Gets a property of a Drawing object by name string. Alternative to direct property access.

Signature

```
function getrenderproperty(object: DrawingObject, property: string): any
```

Example

```
local line = Drawing.new("Line")
line.Thickness = 2
```

```
print(getrenderproperty(line, "Thickness"))  -- 2
```

isrenderobj

Returns true if the given value is a Drawing library object.

Signature

```
function isrenderobj(value: any): boolean
```

Example

```
local square = Drawing.new("Square")
print(isrenderobj(square))   -- true
print(isrenderobj("hello"))  -- false
```

setrenderproperty

Sets a property of a Drawing object by name string.

Signature

```
function setrenderproperty(object: DrawingObject, property: string, value:
any): ()
```

Example

```
local text = Drawing.new("Text")
setrenderproperty(text, "Text", "ESP Label")
setrenderproperty(text, "Size", 18)
setrenderproperty(text, "Visible", true)
```

ENCODING

---

base64decode

Decodes a Base64-encoded string and returns the raw binary/string result.

Signature

```
function base64decode(encoded: string): string
```

Example

```
local decoded = base64decode("aGVsbG8=")
print(decoded)  -- hello
```

base64encode

Encodes a string to Base64 format.

Signature

```
function base64encode(data: string): string
```

Example

```
local encoded = base64encode("hello")
```

```
print(encoded)  -- aGVsbG8=
```

lz4compress

Compresses a string using the LZ4 algorithm. Returns the compressed binary blob.

Signature

```
function lz4compress(data: string): string
```

Example

```
local data = string.rep("aaaa", 100)
local compressed = lz4compress(data)
print(#data, #compressed)  -- 400  much smaller
```

lz4decompress

Decompresses an LZ4-compressed blob. originalSize must be the exact byte length of the original data.

Signature

```
function lz4decompress(compressed: string, originalSize: number): string
```

Example

```
local original = "hello world"
local comp = lz4compress(original)
local restored = lz4decompress(comp, #original)
print(restored)  -- hello world
```

ENVIRONMENT

---

getgc

Returns all objects currently tracked by the Lua garbage collector. With includeTables = false (default), returns only functions and closures. With includeTables = true, also returns live tables. Use for scanning closures to find internal game functions.

Signature

```
function getgc(includeTables: boolean?): { any }
```

Notes

⚠ *getgc(true) is expensive on large games. Always type-check with type() before accessing fields.*

Example

```
-- scan for a function by name
for _, v in ipairs(getgc()) do
    if type(v) == "function" then
        local info = debug.getinfo(v)
        if info and info.name == "TargetFunction" then
            print("found:", v)
        end
```

```
    end
end

-- scan tables for known key structure
for _, v in ipairs(getgc(true)) do
    if type(v) == "table" and v.Health and v.MaxHealth then
        print("found player data table:", v)
    end
end
```

getgenv

Returns the executor's global environment table — shared across all executor scripts in the session. Distinct from _G (game global environment). Write here to share state between scripts without re-declaring.

Signature

```
function getgenv(): table
```

Example

```
-- Script A: write a shared config
getgenv().MyConfig = { enabled = true, speed = 1.5 }

-- Script B (separate execution, same session): read it
local cfg = getgenv().MyConfig
if cfg and cfg.enabled then
    print("speed:", cfg.speed)  -- speed: 1.5
end

-- persistent loop toggle pattern
if getgenv().FeatureRunning then
    getgenv().FeatureRunning = false  -- kill existing loop
end
getgenv().FeatureRunning = true
task.spawn(function()
    while getgenv().FeatureRunning do
        task.wait(0.1)
    end
end)
```

getreg

Returns the Lua registry table — the internal table Lua uses to store references to all active objects. Contains threads, loaded modules, and other engine-level references.

Signature

```
function getreg(): { any }
```

Example

```
local reg = getreg()
for k, v in pairs(reg) do
    if type(v) == "thread" then
        print("thread in registry:", k)
    end
end
```

getrenv

Returns the Roblox game environment table — the global environment used by game scripts, as opposed to the executor environment (getgenv).

Signature

```
function getrenv(): table
```

Example

```
local renv = getrenv()
print(type(renv.workspace))  -- userdata (the real Workspace instance)
```

filtergc

Filtered version of getgc. Accepts a type ("function" or "table") and an options table to narrow results without scanning everything manually. For functions: keys include Name, Hash, Upvalues. For tables: keys include Keys, Values, Metatable.

Signature

```
function filtergc(type: "function" | "table", options: table): { any }
```

Example

```
-- find all functions named "takeDamage"
local results = filtergc("function", { Name = "takeDamage" })
for _, fn in ipairs(results) do
    print(fn)
end

-- find tables containing a known key
local results2 = filtergc("table", { Keys = { "Health", "MaxHealth" } })
for _, tbl in ipairs(results2) do
    print(tbl.Health, tbl.MaxHealth)
end
```

FILESYSTEM

---

appendfile

Appends content to the end of an existing file. Creates the file if it does not exist.

Signature

```
function appendfile(path: string, content: string): ()
```

Example

```
appendfile("logs/output.txt", "new log entry
")
```

delfile

Deletes the file at the given path.

Signature

```
function delfile(path: string): ()
```

Example

```
writefile("temp.txt", "data")
delfile("temp.txt")
```

delfolder

Deletes the folder at the given path. Folder must be empty.

Signature

```
function delfolder(path: string): ()
```

Example

```
makefolder("myfolder")
delfolder("myfolder")
```

getcustomasset

Returns an rbxasset:// URI pointing to a file in the executor's workspace. Lets you load custom textures, sounds, and images into Roblox instances.

Signature

```
function getcustomasset(path: string): string
```

Example

```
writefile("icon.png", readfile("icon.png"))
local uri = getcustomasset("icon.png")
local img = Instance.new("ImageLabel")
img.Image = uri
```

isfile

Returns true if the given path points to an existing file.

Signature

```
function isfile(path: string): boolean
```

Example

```
print(isfile("config.json"))  -- true or false
```

isfolder

Returns true if the given path points to an existing folder.

Signature

```
function isfolder(path: string): boolean
```

Example

```
print(isfolder("mydata"))  -- true or false
```

listfiles

Returns an array of file and folder names inside the given directory path.

Signature

```
function listfiles(path: string): { string }
```

Example

```
local files = listfiles("mydata")
for _, name in ipairs(files) do
    print(name)
end
```

loadfile

Reads a Lua source file from disk and returns it as a callable function. Returns nil and an error string on failure.

Signature

```
function loadfile(path: string): ((...any) -> (...any))?, string?
```

Example

```
local fn, err = loadfile("scripts/loader.lua")
if fn then fn() else print(err) end
```

makefolder

Creates a folder at the given path. Intermediate directories are created as needed.

Signature

```
function makefolder(path: string): ()
```

Example

```
makefolder("myapp/config")
writefile("myapp/config/settings.json", "{}")
```

readfile

Reads and returns the entire contents of a file as a string.

Signature

```
function readfile(path: string): string
```

Example

```
local content = readfile("config.json")
local data = game:GetService("HttpService"):JSONDecode(content)
print(data.version)
```

writefile

Writes content to a file, overwriting if it already exists. Creates the file if it does not exist.

Signature

```
function writefile(path: string, content: string): ()
```

Example

```
local data = { version = "1.0", enabled = true }
local json = game:GetService("HttpService"):JSONEncode(data)
writefile("config.json", json)
```

INSTANCES

---

cloneref

Returns a cloned reference to the given Instance. The clone points to the same underlying object but is a separate Lua reference — useful for bypassing Instance identity checks in game code.

Signature

```
function cloneref(instance: Instance): Instance
```

Example

```
local ws = cloneref(workspace)
print(ws == workspace)          -- false (different reference)
print(compareinstances(ws, workspace))  -- true (same object)
```

compareinstances

Returns true if both references point to the same underlying Roblox Instance, even if one or both are cloneref copies.

Signature

```
function compareinstances(a: Instance, b: Instance): boolean
```

Example

```
local ref = cloneref(workspace)
print(compareinstances(ref, workspace))  -- true
```

fireclickdetector

Fires a ClickDetector as if the local player clicked it. distance defaults to 0. Triggers server-side click events.

Signature

```
function fireclickdetector(detector: ClickDetector, distance: number?,
inputType: string?): ()
```

Example

```
local part = workspace:FindFirstChild("Button")
local detector = part and part:FindFirstChildOfClass("ClickDetector")
if detector then
    fireclickdetector(detector)
end
```

fireproximityprompt

Triggers a ProximityPrompt as if the local player interacted with it. Fires the server-side Triggered event.

Signature

```
function fireproximityprompt(prompt: ProximityPrompt): ()
```

Example

```
local prompt = workspace:FindFirstChildOfClass("ProximityPrompt", true)
if prompt then
    fireproximityprompt(prompt)
end
```

firetouchinterest

Simulates a touch event between two parts. toggle: 0 = TouchEnded, 1 = Touched.

Signature

```
function firetouchinterest(part: BasePart, toTouch: BasePart, toggle:
number): ()
```

Example

```
local trigger = workspace.TriggerZone
local hrp = game.Players.LocalPlayer.Character.HumanoidRootPart
firetouchinterest(trigger, hrp, 1)  -- simulate entering the zone
```

getcallbackvalue

Returns the current callback function assigned to a write-only callback property (e.g. BindableFunction.OnInvoke). These properties cannot be read with normal indexing.

Signature

```
function getcallbackvalue(instance: Instance, property: string): ((...any)
-> (...any))?
```

Example

```
local bf = Instance.new("BindableFunction")
bf.OnInvoke = function(x) return x * 2 end
```

```
local cb = getcallbackvalue(bf, "OnInvoke")
print(cb(5))  -- 10
```

gethui

Returns the executor's hidden UI container — a ScreenGui not visible to game scripts or the developer console. Use it to parent executor UI elements safely.

Signature

```
function gethui(): Instance
```

Example

```
local label = Instance.new("TextLabel")
label.Text = "ESP"
label.Parent = gethui()
```

getinstances

Returns every Instance currently in the game's DataModel, including ones parented to nil or hidden containers.

Signature

```
function getinstances(): { Instance }
```

Example

```
local all = getinstances()
print(#all, "total instances")
```

getnilinstances

Returns all Instances currently parented to nil — objects that exist in memory but are not part of the game tree.

Signature

```
function getnilinstances(): { Instance }
```

Example

```
local nils = getnilinstances()
for _, v in ipairs(nils) do
    print(v.ClassName, v.Name)
end
```

METATABLE

---

getnamecallmethod

Returns the method name of the currently executing __namecall. Only valid inside a __namecall hook. Use to identify which method is being called.

Signature

```
function getnamecallmethod(): string
```

Example

```
local original; original = hookmetamethod(game, "__namecall", function(self,
...)
    local method = getnamecallmethod()
    if method == "FindFirstChild" then
        print("[namecall] FindFirstChild on", self.Name)
    end
    return original(self, ...)
end)
```

getrawmetatable

Returns the raw metatable of an object, bypassing __metatable protection. Works on Roblox Instances and locked tables.

Signature

```
function getrawmetatable(object: any): table?
```

Example

```
local mt = getrawmetatable(game)
print(type(mt))  -- table
```

isreadonly

Returns true if the given table is read-only (setreadonly was called on it).

Signature

```
function isreadonly(table: table): boolean
```

Example

```
local t = {}
setreadonly(t, true)
print(isreadonly(t))  -- true
```

setrawmetatable

Sets the metatable of an object, bypassing __metatable protection.

Signature

```
function setrawmetatable(object: any, metatable: table?): ()
```

Example

```
local mt = getrawmetatable(game)
setrawmetatable(game, mt)  -- replace with modified version
```

setreadonly

Sets whether a table is read-only. When true, any attempt to write to the table raises an error.

Signature

```
function setreadonly(table: table, readonly: boolean): ()
```

Example

```
local cfg = { speed = 16 }
setreadonly(cfg, true)
cfg.speed = 32  -- error: attempt to modify read-only table
```

MISCELLANEOUS

---

identifyexecutor

Returns the name and version of the current executor as two strings.

Signature

```
function identifyexecutor(): string, string
```

Example

```
local name, version = identifyexecutor()
print(name, version)  -- e.g. "MyExecutor"  "1.0.0"
```

request

Sends an HTTP request from the executor context, bypassing Roblox's HttpService restrictions. Supports GET, POST, PUT, DELETE etc. Returns a response table.

Signature

```
function request(options: {    Url: string,    Method: string?,    Headers:
{ [string]: string }?,    Body: string?,    Cookies: { [string]: string
}?,}): {    StatusCode: number,    StatusMessage: string,    Success:
boolean,    Headers: { [string]: string },    Body: string,}
```

Example

```
local response = request({
    Url = "https://httpbin.org/json",
    Method = "GET",
    Headers = { ["User-Agent"] = "executor" }
})
print(response.StatusCode)  -- 200
print(response.Body)
```

REFLECTION

---

gethiddenproperty

Returns the value of a hidden (non-scriptable) property on an Instance, plus a boolean indicating whether the property is hidden.

Signature

```
function gethiddenproperty(instance: Instance, property: string): any, boolean
```

Example

```
local val, isHidden = gethiddenproperty(workspace.Terrain, "SomeHiddenProp")
print(val, isHidden)
```

getthreadidentity

Returns the thread identity level of the current execution context. Higher levels have more permissions. Executor threads typically run at level 6 or 8.

Signature

```
function getthreadidentity(): number
```

Example

```
print(getthreadidentity())  -- e.g. 8
```

isscriptable

Returns true if the given property is scriptable (accessible via normal Lua property access).

Signature

```
function isscriptable(instance: Instance, property: string): boolean
```

Example

```
print(isscriptable(workspace, "FilteringEnabled"))  -- true or false
```

sethiddenproperty

Sets the value of a hidden (non-scriptable) property on an Instance. Returns true on success.

Signature

```
function sethiddenproperty(instance: Instance, property: string, value: any): boolean
```

Example

```
local success = sethiddenproperty(workspace.Terrain, "HiddenProp", 42)
print(success)  -- true
```

setscriptable

Changes the scriptability of a property. When set to true, a previously hidden property can be read and written normally. Returns the previous scriptability state.

Signature

```
function setscriptable(instance: Instance, property: string, scriptable: boolean): boolean
```

Example

```
local prev = setscriptable(workspace.Terrain, "SomeProp", true)
print(workspace.Terrain.SomeProp)  -- now readable
```

setthreadidentity

Sets the thread identity level of the current execution context. Used to gain higher permission levels for certain operations.

Signature

```
function setthreadidentity(level: number): ()
```

Example

```
print(getthreadidentity())  -- 3
setthreadidentity(8)
print(getthreadidentity())  -- 8
```

SCRIPTS

---

getcallingscript

Returns the script instance that is currently calling this function, or nil if called from the executor thread.

Signature

```
function getcallingscript(): BaseScript?
```

Example

```
print(getcallingscript())  -- nil (called from executor)

-- inside a hooked function called by game code:
local original; original = hookfunction(someFunc, function(...)
    print(getcallingscript())  -- the game script instance
    return original(...)
end)
```

getloadedmodules

Returns all ModuleScript instances that have been required and are currently loaded in memory.

Signature

```
function getloadedmodules(): { ModuleScript }
```

Example

```
local modules = getloadedmodules()
for _, m in ipairs(modules) do
    print(m.Name, m:GetFullName())
end
```

getrunningscripts

Returns all scripts (LocalScript and Script) currently running in the game.

Signature

```
function getrunningscripts(): { BaseScript }
```

Example

```
local scripts = getrunningscripts()
print(#scripts, "scripts running")
```

getscriptbytecode

Returns the raw Luau bytecode of a script instance.

Signature

```
function getscriptbytecode(script: BaseScript | ModuleScript): string
```

Example

```
local bytecode =
getscriptbytecode(game.Players.LocalPlayer.PlayerScripts.SomeScript)
print(#bytecode, "bytes")
```

getscriptclosure

Returns the main closure (function) of a script instance — the top-level function that runs when the script executes.

Signature

```
function getscriptclosure(script: BaseScript | ModuleScript): (...any) ->
(...any)
```

Example

```
local fn =
getscriptclosure(game.Players.LocalPlayer.PlayerScripts.SomeScript)
local ups = debug.getupvalues(fn)
print(#ups, "upvalues")
```

getscriptfromthread

Returns the script instance associated with the given coroutine thread.

Signature

```
function getscriptfromthread(thread: thread): BaseScript?
```

Example

```
local thread = coroutine.running()
print(getscriptfromthread(thread))  -- the current script
```

getscripthash

Returns a hash of the script's source or bytecode. Used to verify script integrity.

Signature

```
function getscripthash(script: BaseScript | ModuleScript): string
```

Example

```
local hash =
getscripthash(game.Players.LocalPlayer.PlayerScripts.SomeScript)
print(hash)  -- "abc123..."
```

getscripts

Returns all Script and LocalScript instances in the game, whether running or not.

Signature

```
function getscripts(): { BaseScript }
```

Example

```
local all = getscripts()
for _, s in ipairs(all) do
    print(s:GetFullName())
end
```

getsenv

Returns the environment table of the given script — all globals that script has access to.

Signature

```
function getsenv(script: BaseScript | ModuleScript): table
```

Example

```
local env = getsenv(game.Players.LocalPlayer.PlayerScripts.SomeScript)
print(type(env.game))  -- userdata
```

SIGNALS

---

firesignal

Fires an RBXScriptSignal, triggering all connected callbacks with the provided arguments.

Signature

```
function firesignal(signal: RBXScriptSignal, ...: any): ()
```

Example

```
local part = workspace:FindFirstChild("SomePart")
if part then
    firesignal(part.Touched, part)
end
```

getconnections

Returns all Connection objects currently connected to an RBXScriptSignal. Each Connection has fields: Function, Thread, Connected, Enabled, ForeignState, LuaConnection.

Signature

```
function getconnections(signal: RBXScriptSignal): { Connection }
```

Example

```
local conns =
getconnections(game.Players.LocalPlayer.Character.Humanoid.Died)
for _, conn in ipairs(conns) do
    print(conn.Function)  -- the callback function
    conn:Disable()        -- or conn:Enable() / conn:Fire()
end
```

replicatesignal

Fires a signal and replicates the event to the server. Similar to firesignal but triggers server-side handling.

Signature

```
function replicatesignal(signal: RBXScriptSignal, ...: any): ()
```

Example

```
replicatesignal(game.Players.LocalPlayer.Character.Humanoid.Died)
```

WEBSOCKET

WebSocket.connect

Connects to a WebSocket server and returns a WebSocket object with Send, Close, OnMessage, and OnClose members.

Signature

```
function WebSocket.connect(url: string): WebSocketObject
```

Example

```
local ws = WebSocket.connect("ws://localhost:8080")

ws.OnMessage:Connect(function(msg)
    print("received:", msg)
end)

ws.OnClose:Connect(function()
    print("connection closed")
end)

ws:Send("hello from executor")

task.wait(5)
```

```
ws:Close()
```