BWAPI-Lua Documentation

Release 0.1.0

Ryan Liptak

Jun 22, 2017

Getting Started

Lua bindings for the Brood War API using the Sol2 Lua bindings library, for the purposes of writing StarCraft AI using the Lua scripting language.

CHAPTER 1

Installation

Starcraft and BWAPI

1. Install and update Starcraft to patch 1.16.1 (not 1.18). The 1.16.1 patch can be downloaded here, and a guide for setting up simultateous 1.16.1 and 1.18 installs can be found here.
2. Download and install BWAPI 4.2.0

BWAPI-Lua

3. Download the BWAPI-Lua .dll for the version of Lua you want to use and move it to `Starcraft/bwapi-data/AI` (e.g. `C:\Program Files (x86)\StarCraft\bwapi-data\AI`).
4. You will also need to make sure that you have a compatible Lua .dll in your Starcraft folder or somewhere in your `PATH` (for Lua 5.1, it should be named `lua51.dll`). If you are unsure if you have this, you should download it from here and put it in your Starcraft folder (e.g. `C:\Program Files (x86)\StarCraft`).
5. Edit `Starcraft/bwapi-data/bwapi.ini` and change the line `ai = bwapi-data/AI/ExampleAIModule.dll` to point to the BWAPI-Lua dll you just installed (e.g. `ai = bwapi-data/AI/Lua51AIModule.dll`)

Next

Your first Lua AI

CHAPTER 2

Your First Lua AI

The entry point to Lua is `init.lua` in the same directory as the loaded BWAPI-Lua AI module .dll (e.g. `C:\Program Files (x86)\StarCraft\bwapi-data\AI`). Once `init.lua` is executed, the *BWAPI callbacks* will be begin to be called (if they are defined in the global `BWAPI` table).

The most basic AI–one that simply sends a string to the chat at the start of a game–would look like this:

Listing 2.1: init.lua

```
function BWAPI.onStart()
  BWAPI.Broodwar:sendText("Hello world!")
end
```

The *`BWAPI.onStart()`* callback gets called immediately after `init.lua` is loaded.

CHAPTER 3

List of Existing Modules

BWEM-Lua

https://github.com/squeek502/BWEM-Lua

Lua bindings for the BWEM map analysis library

SparCraft-Lua

https://github.com/squeek502/SparCraft-Lua

Lua bindings for the SparCraft combat simulation library

BOSS-Lua

https://github.com/squeek502/BOSS-Lua

Lua bindings for the BOSS build order library

CHAPTER 4

Installing and Using Modules

Modules can be loaded and used as in normal Lua. Note that BWAPI-Lua adds the directory of the loaded BWAPI-Lua dll to `package.path` and `package.cpath` (see *package.cpath*/*package.path*).

Note: When loading binary modules using `require`, you should not use any paths in the module name passed to `require`, as binary modules rely on the module name to call the initialization function.

For example, a module like `example.dll` should be loaded by doing `require('example')` rather than `require('sub.dir.example')`, even if the path to `example.dll` is `sub/dir/example.dll`. If you want to put the `.dll` outside the default search paths (for example, in a directory specific to your AI), then you'll need to alter `package.cpath` accordingly (e.g. `package.cpath = "C:/path/to/dir/?.dll;" .. package.cpath`).

CHAPTER 5

Writing Modules

Writing a C/C++ Binary Module

The most simple binary module would be one that is wholly independent of BWAPI-Lua and BWAPI, as it would only need to link to the Lua library. If using the CMake script below, your Lua link library should be named `lua51.lib` and placed in `dependecies/lua/5.1/lib` within the module's project folder; the necessary Lua header files should be placed in `dependecies/lua/5.1/include`.

Listing 5.1: CMakeLists.txt

```
cmake_minimum_required (VERSION 2.6)
project (HelloWorldModule)

# Lua
set(LUA_5_1_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/dependencies/lua/5.1/include)
set(LUA_5_1_LIB_DIR ${PROJECT_SOURCE_DIR}/dependencies/lua/5.1/lib)
find_library(LUA_LIB lua51 ${LUA_5_1_LIB_DIR} NO_DEFAULT_PATH)

# Our Module
add_library( helloworld SHARED helloworld.cpp helloworld.def )
set_target_properties( helloworld PROPERTIES LINK_FLAGS "${CMAKE_LINK_DEF_FILE_FLAG}$
↪{PROJECT_SOURCE_DIR}/helloworld.def /SUBSYSTEM:WINDOWS" )
target_link_libraries ( helloworld ${LUA_LIB} )
```

Listing 5.2: helloworld.cpp

```
#define LUA_LIB
#include "lauxlib.h"

LUALIB_API int luaopen_helloworld(lua_State *L)
{
   // Loading this module will return the string "Hello World!"
   lua_pushstring(L, "Hello World!");
   return 1;
```

```
}
```

Listing 5.3: helloworld.def

```
EXPORTS
luaopen_helloworld
```

Once built and placed next to the BWAPI-Lua dll (in the same directory), the following Lua code would work:

Listing 5.4: init.lua

```lua
local helloWorld = require('helloworld')

assert(helloWorld == "Hello World!")
print(helloWorld)
```

Interacting with BWAPI

To write a binary module that interacts with BWAPI, we will need to link against BWAPILIB and make sure its headers are in our include path. If using the CMake script below:

- Your BWAPILIB release library should be named `BWAPILIB.lib` and placed in `dependencies/bwapi/lib` within the module's project folder
- Your BWAPILIB debug library should be named `BWAPILIBd.lib` and placed in `dependencies/bwapi/lib`
- Your BWAPI header files should be placed in `dependencies/bwapi/include`
- Lua library/headers should be in the same places as in the previous module

---

Important: When linking BWAPILIB, your module will contain a separate instance of `BWAPI::Broodwar` and a NULL `BWAPI::BroodwarPtr`. To use `BWAPI::Broodwar` in your extension, `BWAPI::BroodwarPtr` will need to be set to the value of `BWAPI.Broodwar` from the Lua VM, which is a `BWAPI::Game*` pushed to Lua as userdata by the Sol2 binding library. The code below shows how to retrieve that value using the Lua C API (see also the Sol2 docs on this subject).

Listing 5.5: CMakeLists.txt

```
cmake_minimum_required (VERSION 2.6)
project (HelloWorldModule)

# BWAPILIB
set(BWAPI_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/dependencies/bwapi/include)
set(BWAPI_LIB_DIR ${PROJECT_SOURCE_DIR}/dependencies/bwapi/lib)
find_library(BWAPILIB_RELEASE_LIB BWAPILIB ${BWAPI_LIB_DIR} NO_DEFAULT_PATH)
find_library(BWAPILIB_DEBUG_LIB BWAPILIBd ${BWAPI_LIB_DIR} NO_DEFAULT_PATH)

# Lua
set(LUA_5_1_INCLUDE_DIR ${PROJECT_SOURCE_DIR}/dependencies/lua/5.1/include)
set(LUA_5_1_LIB_DIR ${PROJECT_SOURCE_DIR}/dependencies/lua/5.1/lib)
find_library(LUA_LIB lua51 ${LUA_5_1_LIB_DIR} NO_DEFAULT_PATH)

# Our Module
add_library( helloworld SHARED helloworld.cpp helloworld.def )
```

```
set_target_properties( helloworld PROPERTIES LINK_FLAGS "${CMAKE_LINK_DEF_FILE_FLAG}$
↪{PROJECT_SOURCE_DIR}/helloworld.def /SUBSYSTEM:WINDOWS" )
target_link_libraries ( helloworld ${LUA_LIB} optimized ${BWAPILIB_RELEASE_LIB} debug
↪${BWAPILIB_DEBUG_LIB} )
```

Listing 5.6: helloworld.cpp

```
#define LUA_LIB
#include "lauxlib.h"
#include "BWAPI.h"

static int helloworld_print(lua_State *L)
{
        const char* msg = luaL_checkstring(L, 1);
        BWAPI::Broodwar << msg << std::endl;
        return 0;
}

LUALIB_API int luaopen_helloworld(lua_State *L)
{
        // If BWAPI::Broodwar is used, we need to set it to the same pointer that␣
↪BWAPI-Lua uses,
        // so we need to grab that from BWAPI.Broodwar
        lua_getglobal(L, "BWAPI");
        lua_getfield(L, -1, "Broodwar");
        void* data = lua_touserdata(L, -1);
        BWAPI::Game* bw = *static_cast<BWAPI::Game**>(data);
        BWAPI::BroodwarPtr = bw;
        lua_settop(L, 0); // clear the stack

        // Create a table with a 'print' function that prints to the BW chat
        lua_newtable(L);
        int tableIndex = lua_gettop(L);
        lua_pushcfunction(L, &helloworld_print);
        lua_setfield(L, tableIndex, "print");

        // Return the table since it's still on the stack
        return 1;
}
```

Listing 5.7: helloworld.def

```
EXPORTS
luaopen_helloworld
```

Once built and placed next to the BWAPI-Lua dll (in the same directory), the following Lua code would work:

Listing 5.8: init.lua

```
local helloWorld = require('helloworld')

assert(type(helloWorld) == "table")
assert(type(helloWorld.print) == "function")
helloWorld.print("Hello World!")
```

CHAPTER 6

Differences Between the C++ and Lua API

Most of the API follows the conventions of the C++ API and can be used in exactly the same ways, although there are exceptions in cases where either:

- The Lua programming language does not have the same capabilities as C++, so things must be done a different way
- It makes more sense to do something in a "Lua way" rather than the "C++ way"

This page documents all such differences.

All classes

Every bound class has a static member function `isInstance` which can be used to determine if a `userdata` value (an instance of a C++ class) is an instance of that class.

Listing 6.1: Example

```
function determineType(unknown)
  if BWAPI.Position.isInstance(unknown) then
    print("It's a Position")
  elseif BWAPI.TilePosition.isInstance(unknown) then
    print("It's a TilePosition")
  elseif BWAPI.Unit.isInstance(unknown) then
    print("It's a Unit")
  else
    print("It's something else")
  end
end
```

Interface derived classes (Game, Player, Unit, etc)

registerEvent()

Takes Lua functions for its `action` and `condition` parameters (note: `condition` can be `nil`)

Listing 6.2: Example usage

```
local Broodwar = BWAPI.Broodwar
local pos = u:getPosition()
local lastErr = Broodwar:getLastError()
local action = function()
   Broodwar:drawTextMap(pos, string.format("%c%s", BWAPI.Text.White,␣
↪tostring(lastErr)))
end
Broodwar:registerEvent(action, nil, Broodwar:getLatencyFrames())
```

getClientInfo() and setClientInfo()

`getClientInfo`/`setClientInfo` have been removed in favor of a `clientInfo` property that is a Lua table. This allows for storing arbitrary data in a more user-friendly way.

Listing 6.3: Example usage

```
unit.clientInfo["test"] = 5
print(unit.clientInfo["test"])
```

Important: `setClientInfo` has also been removed from *`Unitset`*. Instead, you should simply iterate the `Unitset` and operate on the clientInfo of each unit as needed.

Listing 6.4: Example

```
local value = 5
for unit in set:iterator() do
  unit.clientInfo["key"] = value
end
```

Functions that take a variable amount of string parameters in C++

All C++ functions that take variable amounts of strings now expect only a single string.

Note: Any formatting must be done in Lua first (`string.format`), and then the formatted string can be passed into the function like normal.

Game.sendTextEx()

A new convenience function has been added to send text to allies: *`sendTextToAllies()`*, which fowards the method to *`sendTextEx()`* with `true` as the first parameter. The following two snippets are exactly equivalent:

- `BWAPI.Broodwar:sendTextEx(true, "your message")`
- `BWAPI.Broodwar:sendTextToAllies("your message")`

Unit

`cancelTrain()` and `getTrainingQueue()`

All methods that deal with training slots have been changed to be one-indexed (like Lua), rather than zero-indexed (like C++). For example, to cancel the first unit being trained, you would now pass a slot of `1`, whereas in C++ you'd pass a slot of `0`. List of affected methods:

- *BWAPI.Unit.canCancelTrainSlot()*
- *BWAPI.Unit.cancelTrain()*
- *BWAPI.Unit.getTrainingQueue()*
- *BWAPI.UnitCommand.getSlot()*
- *BWAPI.UnitCommand.cancelTrain()*
- *BWAPI.Unitset.cancelTrain()*

Similarly, *getTrainingQueue()* returns a Lua array-like table (which is one-indexed) instead of a `std::list` (which is zero-indexed). This allows for the following:

```
-- cancel the first dragoon found in the queue
local queue = building:getTrainingQueue()
for slot, unitType in ipairs(queue) do
  if unitType == BWAPI.UnitTypes.Protoss_Dragoon then
   building:cancelTrain(slot)
   break
  end
end
```

Warning: Iterating a training queue and canceling multiple slots while in the loop will result in unexpected behavior, as the slots will shift as things are canceled. For example, if you cancel slot 1 and then iterate to slot 2 and also cancel it, then you'll actually be canceling what was originally in slot 3.

UnitType

`whatBuilds()`

Returns two values instead of a `std::pair`

Listing 6.5: Example usage

```
local unitType, howMany = ut:whatBuilds()
```

requiredUnits()

Returns a Lua table of the format `{ [<unitTypeID>] = <howMany> }`, where `<unitTypeID>` is the integer ID/Enum of a required UnitType (equal to `UnitType:getID()`) and `<howMany>` is the required number of that unit.

Listing 6.6: Example usage

```
local scv = BWAPI.UnitTypes.SCV
local requiredUnits = scv:requiredUnits()
for unitTypeID, howMany in pairs(requiredUnits) do
  local requiredUnitType = BWAPI.UnitType(unitTypeID)
  local str = string.format("%s requires %d %s",
    tostring(scv),
    howMany,
    tostring(requiredUnitType)
  )
  print(str)
end
```

SetContainer implementations (Unitset, Playerset, etc)

The set can be iterated one of two ways:

- `for x in set:iterator() do`
- `for i, x in ipairs(set:asTable()) do`

Also, any SetContainer types of the format `ClassName::set` are bound as `ClassNameset`, to match the naming convention of the other SetContainer types (`Playerset`, `Unitset`, etc). For example, `UnitType::set` is bound as *`BWAPI.UnitTypeset`*.

An additional convenience method, `filter` (e.g. *`BWAPI.Unitset.filter()`*), has been added to easily filter a set using a predicate function, as well as some aliases for `filter` and `erase_if` (see *`eraseIf()`*, *`keep_if()`*, *`keepIf()`*).

All lists (std::list, Position::list, UnitType::list, etc)

All C++ functions that return lists now return array-like Lua tables.

Listing 6.7: Example

```
local nukeDots = BWAPI.Broodwar:getNukeDots()
for i, pos in ipairs(nukeDots) do
   print(string.format("There's a nuke at %s", tostring(pos)))
end
```

UnitFilter

All functions that take a UnitFilter parameter now expect a Lua function that takes a unit and returns a boolean.

Note: *BWAPI.Filter* functions can be used by calling them with a `Unit` as the parameter (e.g. `BWAPI.Filter.CanAttack(unit)`)

These filters can also be combined by using the normal Lua boolean operators and wrapping/returning the result in a function. The function can then be passed as an argument to functions that would normally take a `UnitFilter` in C++, like so:

```
local myFilter = function(unit)
  return BWAPI.Filter.CanAttack(unit) and not BWAPI.Filter.IsOrganic(unit)
end
local closest = unit:getClosestUnit(myFilter)
```

BestFilter

All functions that take a `BestUnitFilter` parameter now expect a Lua function that takes two parameters: the current best unit, and the unit to compare to, and returns the best unit out of the two.

Listing 6.8: Example

```
local bestFilter = function(a, b)
  if b:getHitPoints() > a:getHitPoints() then
    return b
  end
  return a
end
local best = BWAPI.Broodwar:getBestUnit(bestFilter, BWAPI.Filter.IsOrganic)
```

See also:

BWAPI.Game.getBestUnit()

Event and Game::getEvents

For now, the `getEvents` function of *BWAPI.Game* has been removed, and there are no bindings for the `BWAPI.Event` class. This is subject to change if it's shown to be necessary.

CHAPTER 7

Changes to the Global Environment

print (...)

print() has been overriden to act similarly to the normal Lua print, but redirect its output to the client chat window via *BWAPI.print()*.

It will take all arguments, call `tostring` on them, concatenate them together using `\t` as the separator, append a newline to the end of the string, and then pass that to *BWAPI.print()*.

`package`**.path**

The directory of the loaded LuaAIModule .dll gets appended to the front of *package.path* (using both the `/?.lua` and `/?/init.lua` variants).

`package`**.cpath**

The directory of the loaded LuaAIModule .dll gets appended to the front of *package.cpath* (using the `/?.dll` variant)

BWAPI

See *BWAPI*

CHAPTER 8

The BWAPI Module

The BWAPI bindings are all loaded into the *BWAPI* module, which is loaded as `BWAPI` into the global environment and can also be found in the `"BWAPI"` key in `package.loaded` (retrieved by doing `require('BWAPI')`)

Constants

`BWAPI.`**`Broodwar`**
: The currently running *Game* instance

Callbacks

Note: Replays are considered games and call all of the same callbacks as a standard game would.

`BWAPI.`**`onStart`**`()`
: Called only once at the beginning of a game.

  It is intended that the AI module do any data initialization in this function.

  Note: `init.lua` is loaded with the same environment that *BWAPI.onStart()* is called with, so the use of this callback is optional

`BWAPI.`**`onFrame`**`()`
: Called once for every execution of a logical frame in Broodwar.

  Users will generally put most of their code in this function.

`BWAPI.`**`onEnd`**`(`*isWinner*`)`
: Called once at the end of a game.

Parameters **isWinner** (*boolean*) – A boolean value to determine if the current player has won the match. This value will be true if the current player has won, and false if either the player has lost or the game is actually a replay.

BWAPI.**onError**(*msg*)

If defined, then all Lua error messages will be redirected to this function (instead of the default functionality of the error message being printed to the screen). Can be used for custom error handling/logging errors to file/etc.

Parameters **msg** (*string*) – The error message.

Listing 8.1: examples/LoggingErrorsToFile/init.lua

```lua
-- cwd is the StarCraft .exe directory, so this is relative to that
-- make sure that the target directory exists!
local logFilePath = "bwapi-data/lua-ai-example-error.log"

function BWAPI.onError(msg)
  -- print to the screen
  print(msg)

  -- RFC1123 date
  local fmt = "!%a, %d %b %Y %T GMT"
  local date = os.date(fmt)

  -- log to file
  local log = io.open(logFilePath, "a")
  local str = string.format("[%s] %s\n", date, msg)
  log:write(str)
  log:close()
end

function BWAPI.onStart()
  -- lets trigger an error!
  local missing = nil
  print(missing.index)
end
```

BWAPI.**onNukeDetect**(*target*)

Called when a *Nuke* has been launched somewhere on the map.

Parameters **target** (BWAPI.Position) – A *Position* containing the target location of the *Nuke*. If the target location is not visible and *BWAPI.Flag.CompleteMapInformation* is disabled, then target will be *BWAPI.Positions.Unknown*.

BWAPI.**onPlayerLeft**(*player*)

Called when a Player leaves the game.

All of their units are automatically given to the neutral player with their colour and alliance parameters preserved.

Parameters **player** (BWAPI.Player) – The *Player* that has left the game.

BWAPI.**onReceiveText**(*player*, *text*)

Called when the client receives a message from another Player.

This function can be used to retrieve information from allies in team games, or just to respond to other players.

Parameters

- **player** (BWAPI.Player) – The *Player* that sent the message.
- **text** (*string*) – The text message that the player sent.

BWAPI.**onSaveGame**(*gameName*)
Called when the state of the Broodwar game is saved to file.

Parameters **gameName** (*string*) – The file name that the game was saved as

BWAPI.**onSendText**(*text*)
Called when the user attempts to send a text message.

This function can be used to make the bot execute text commands entered by the user for debugging purposes.

Parameters **text** (*string*) – The exact text message that was sent by the user.

Note: If *BWAPI.Flag.UserInput* is disabled, then this function is not called.

Listing 8.2: examples/SendTextREPL/init.lua

```
-- very basic REPL using the text entered by the client
-- any syntax/execution errors are handled by the default error handler,

function BWAPI.onStart()
  BWAPI.Broodwar:enableFlag(BWAPI.Flag.UserInput)
  print("Anything you type will be executed as Lua!")
end

function BWAPI.onSendText(text)
  -- execute the text as Lua and print the result
  local fn = assert(loadstring(text))
  local ret = {fn()}
  if #ret > 0 then
    print(unpack(ret))
  end
end
```

BWAPI.**onUnitComplete**(*unit*)
Called when the state of a unit changes from incomplete to complete.

Parameters **unit** (BWAPI.Unit) – The *Unit* that has been completed.

BWAPI.**onUnitCreate**(*unit*)
Called when any unit is created.

Parameters **unit** (BWAPI.Unit) – The *Unit* that has just been created.

Note: Due to the internal workings of Broodwar, this function excludes Zerg morphing and the construction of structures over a *Vespene Geyser*.

See also:

BWAPI.onUnitMorph()

BWAPI.**onUnitDestroy**(*unit*)
Called when a unit is removed from the game either through death or other means.

Parameters **unit** (BWAPI.Unit) – The *Unit* that has just been destroyed or otherwise completely removed from the game.

Note: When a *Drone* morphs into an *Extractor*, the *Drone* is removed from the game and the *Vespene*

Geyser morphs into an *Extractor*.

Note: If a unit is visible and destroyed, then *BWAPI.onUnitHide()* is called just before this.

BWAPI.**onUnitDiscover**(*unit*)
Called when a Unit becomes accessible.

Parameters **unit** (BWAPI.Unit) – The *Unit* that has just become accessible.

Note: This function INCLUDES the state of *BWAPI.Flag.CompleteMapInformation*.

See also:

BWAPI.onUnitShow()

BWAPI.**onUnitEvade**(*unit*)
Called when a Unit becomes inaccessible.

Parameters **unit** (BWAPI.Unit) – The *Unit* that has just become inaccessible.

Note: This function INCLUDES the state of *BWAPI.Flag.CompleteMapInformation*.

See also:

BWAPI.onUnitHide()

BWAPI.**onUnitHide**(*unit*)
Called just as a visible unit is becoming invisible.

Parameters **unit** (BWAPI.Unit) – The *Unit* that is about to go out of scope.

Note: This function EXCLUDES the state of *BWAPI.Flag.CompleteMapInformation*.

See also:

BWAPI.onUnitEvade()

BWAPI.**onUnitMorph**(*unit*)
Called when a unit changes its *UnitType*.

For example, when a *Drone* transforms into a *Hatchery*, a Siege Tank uses *Siege Mode*, or a *Vespene Geyser* receives a *Refinery*.

Parameters **unit** (BWAPI.Unit) – The *Unit* that just had its *UnitType* change.

Note: This is NOT called if the unit type changes to or from *Unknown*.

Functions

static BWAPI.**print**(*text*)
Prints the text to the client chat window, with no owner

Parameters **text** (*string*) – The text to be printed

Note: This function does not automatically append a newline to the output

static BWAPI.**getRevision**()
Retrieves the revision of the BWAPILIB module currently being used.

Returns An integer representing the revision number of the library.

Return type number

static BWAPI.**isDebug**()
Checks if the BWAPILIB module was compiled in DEBUG mode.

Returns `true` if this is a DEBUG build, or `false` if this is a RELEASE build

Return type boolean

static BWAPI.**Lowest**(*filter*) → bestFilter
Takes a *comparison filter* (or similar function) and converts it into a 'best' function intended for use with *Game.getBestUnit()*.

Parameters **filter** (*function*) – A function that takes a *Unit* and returns a value that will be used for comparison. The returned value must be of a type that can be compared using the less than operator.

Returns A function that takes two *Unit* objects, and returns the object with the lowest value returned when passing it into `filter`.

Return type function

See also:

The differences between the C++ and Lua implementations of BestFilter, *Game.getBestUnit()*

static BWAPI.**Highest**(*filter*) → bestFilter
Takes a *comparison filter* (or similar function) and converts it into a 'best' function intended for use with *Game.getBestUnit()*.

Parameters **filter** (*function*) – A function that takes a *Unit* and returns a value that will be used for comparison. The returned value must be of a type that can be compared using the greater than operator.

Returns A function that takes two *Unit* objects, and returns the object with the highest value returned when passing it into `filter`.

Return type function

See also:

The differences between the C++ and Lua implementations of BestFilter, *Game.getBestUnit()*

Classes

Bullet

class BWAPI.**Bullet**
An object representing a bullet or missile spawned from an attack.

Bullet allows you to detect bullets, missiles, and other types of non-melee attacks or special abilities that would normally be visible through human eyes (A lurker spike or a Queen's flying parasite), allowing quicker reaction to unavoidable consequences.

For example, ordering medics to restore units that are about to receive a lockdown to compensate for latency and minimize its effects. You can't know entirely which unit will be receiving a lockdown unless you can detect the lockdown missile using the Bullet class.

Bullet objects are re-used after they are destroyed, however their ID is updated when it represents a new Bullet.

If *BWAPI.Flag.CompleteMapInformation* is disabled, then a Bullet is accessible if and only if it is visible. Otherwise if *BWAPI.Flag.CompleteMapInformation* is enabled, then all Bullets in the game are accessible.

See also:

Game.getBullets(), *Bullet.exists()*

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo
A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.3: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

exists() → boolean
Checks if the Bullet exists in the view of the *BWAPI* player.

If *BWAPI.Flag.CompleteMapInformation* is disabled, and a Bullet is not visible, then the return value will be false regardless of the Bullet's true existence. This is because absolutely no state information on invisible enemy bullets is made available to the AI.

Returns Returns `true` if the bullet exists or is visible, or `false` if the bullet was destroyed or has gone out of scope

Return type boolean

See also:

isVisible, *Unit.exists()*

getAngle() → double
Retrieve's the direction the Bullet is facing.

If the angle is 0, then the Bullet is facing right.

Returns A double representing the direction the Bullet is facing. Returns `0.0` if the bullet is inaccessible.

Return type double

getID() → int
Retrieves a unique identifier for the current Bullet.

Returns An integer value containing the identifier.

Return type int

getPlayer() → Player
Retrieves the Player interface that owns the Bullet.

Returns The owning Player interface object. Returns `nil` if the Player object for this Bullet is inaccessible.

Return type *BWAPI.Player*

getPosition() → Position
Retrieves the Bullet's current position.

Returns A Position containing the Bullet's current coordinates. Returns `Positions.Unknown` if the Bullet is inaccessible.

Return type *BWAPI.Position*

See also:

getTargetPosition()

getRemoveTimer() → int
Retrieves the timer that indicates the Bullet's life span.

Bullets are not permanent objects, so they will often have a limited life span. This life span is measured in frames. Normally a Bullet will reach its target before being removed.

Returns An integer representing the remaining number of frames until the Bullet self-destructs. Returns `0` if the Bullet is inaccessible.

Return type int

getSource() → Unit
Retrieves the Unit interface that the Bullet spawned from.

Returns The owning Unit interface object. Returns `nil` if the source can not be identified or is inaccessible.

Return type *BWAPI.Unit*

See also:

getTarget()

getTarget() → Unit
Retrieves the Unit interface that the Bullet is heading to.

Returns The target Unit interface object, if one exists. Returns `nil` if the Bullet's target Unit is inaccessible, the Bullet is targetting the ground, or if the Bullet itself is inaccessible.

Return type *BWAPI.Unit*

See also:

getTargetPosition(), *getSource()*

getTargetPosition() → Position
Retrieves the target position that the Bullet is heading to.

Returns A Position indicating where the Bullet is headed. Returns `Positions.Unknown` if the bullet is inaccessible.

Return type *BWAPI.Position*

See also:

getTarget(), *getPosition()*

getType() → BulletType
Retrieves the type of this Bullet.

Returns A *BulletType* representing the Bullet's type. Returns `BulletTypes.Unknown` if the Bullet is inaccessible.

Return type *BWAPI.BulletType*

getVelocityX() → double
Retrieves the X component of the Bullet's velocity, measured in pixels per frame.

Returns A double representing the number of pixels moved on the X axis per frame. Returns `0.0` if the Bullet is inaccessible.

Return type double

See also:

getVelocityY(), *getAngle()*

getVelocityY() → double
Retrieves the Y component of the Bullet's velocity, measured in pixels per frame.

Returns A double representing the number of pixels moved on the Y axis per frame. Returns `0.0` if the Bullet is inaccessible.

Return type double

See also:

getVelocityX(), *getAngle()*

isVisible([*player* = *nil*]) → boolean
Retrieves the visibility state of the Bullet.

Parameters **player** (`BWAPI.Player`) – (optional) If this parameter is specified, then the Bullet's visibility to the given player is checked. If this parameter is omitted, then a default value of nil is used, which will check if the *BWAPI* player has vision of the Bullet.

Returns Returns `true` if the Bullet is visible to the specified player, or `false` if the Bullet is not visible to the specified player

Return type boolean

Note: If `player` is nil and Broodwar->self() is also nil, then the visibility of the Bullet is determined by checking if at least one other player has vision of the Bullet.

---

registerEvent (*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.
- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

BulletType

class BWAPI.**BulletType**
This class represents a type of bullet.

Note: Internally, these are the same IDs as flingy types in Broodwar.

See also:

BulletTypes

Constructors

BulletType ([*id = BulletTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes *BulletTypes.Unknown*. A type is invalid if its value is less than 0 or greater than *BulletTypes.Unknown*.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes *BulletTypes.Unknown*.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

BulletTypeset

class BWAPI.**BulletTypeset**
A container for a set of *BulletType* objects.

Constructors

BulletTypeset()
Default constructor.

BulletTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.BulletTypeset) – The BulletTypeset to copy.

BulletTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *BulletType* are added to the set.

Parameters **tbl** (*table*) – A table containing *BulletType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Bulletset

class `BWAPI.Bulletset`
A container for a set of *Bullet* objects.

Constructors

Bulletset ()
Default constructor.

Bulletset (*set*)
Copy constructor.

Parameters **set** (`BWAPI.Bulletset`) – The Bulletset to copy.

Bulletset (*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *Bullet* are added to the set.

Parameters **tbl** (*table*) – A table containing *Bullet* objects.

Member Functions

iterator () → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable () → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count (*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets

do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

`contains`(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

`size`() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

`empty`() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

`insert`(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

`erase`(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

`clear`()
Removes all elements from the set, leaving it with a size of 0.

`eraseIf`(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **`pred`** (*`function`*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

`erase_if`(*pred*)
Alias of *`eraseIf()`*

`filter`(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **`pred`** (*`function`*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

`keepIf`(*pred*)
Alias of *`filter()`*/*`keep_if()`*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Color

class BWAPI.**Color**
The *Color* object is used in drawing routines to specify the color to use.

Note: Starcraft uses a 256 color palette for rendering. Thus, the colors available are limited to this palette.

See also:

Colors

Constructors

Color ([*id = 0*])
A constructor that uses the color at the specified palette index.

Parameters **id** (*int*) – The index of the color in the 256-color palette.

Color (*red*, *green*, *blue*)
A constructor that uses the color index in the palette that is closest to the given rgb values.

On its first call, the colors in the palette will be sorted for fast indexing.

Parameters

- **red** (*int*) – The amount of red.
- **green** (*int*) – The amount of green.
- **blue** (*int*) – The amount of blue.

Note: This function computes the distance of the RGB values and may not be accurate.

Member Functions

blue () → int
Retrieves the blue component of the color.

Returns integer containing the value of the blue component.

Return type int

green () → int
Retrieves the green component of the color.

Returns integer containing the value of the green component.

Return type int

red () → int
Retrieves the red component of the color.

Returns integer containing the value of the red component.

Return type int

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

DamageType

class BWAPI.**DamageType**
Damage types are used in Broodwar to determine the amount of damage that will be done to a unit.

This corresponds with *UnitSizeType* to determine the damage done to a unit.

- View on Liquipedia
- View on Starcraft Compendium (Official Website)
- View on Starcraft Wikia

See also:

WeaponType, *DamageTypes*, *UnitSizeType*

Constructors

DamageType([*id = DamageTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

DamageTypeset

class BWAPI.**DamageTypeset**
A container for a set of *DamageType* objects.

Constructors

DamageTypeset()
Default constructor.

DamageTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.DamageTypeset) – The DamageTypeset to copy.

DamageTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *DamageType* are added to the set.

Parameters **tbl** (*table*) – A table containing *DamageType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Error

class BWAPI.**Error**
The *Error* object is generally used to determine why certain functions in *BWAPI* have failed.

For example, you may not have enough resources to construct a unit.

See also:

Game.getLastError(), *Game.setLastError()*, *Errors*

Constructors

Error ([*id = Errors.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName () → string

Returns The variable name of the type.

Return type string

Errorset

class `BWAPI.Errorset`
A container for a set of *`Error`* objects.

Constructors

`Errorset``()`
Default constructor.

`Errorset``(`*set*`)`
Copy constructor.

Parameters **`set`** (`BWAPI.Errorset`) – The Errorset to copy.

`Errorset``(`*tbl*`)`
Constructor to convert a Lua table to a set. Any values in the table that are of type *`Error`* are added to the set.

Parameters **`tbl`** (*`table`*) – A table containing *`Error`* objects.

Member Functions

`iterator``()` → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

`asTable``()` → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

`count``(`*val*`)` → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

`contains``(`*val*`)` → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

ExplosionType

class BWAPI.**ExplosionType**
A representation of a weapon's explosion type.

This indicates how the weapon behaves, such as if it deals splash damage or causes an effect to occur.

See also:

ExplosionTypes

Constructors

ExplosionType([*id = ExplosionTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

ExplosionTypeset

class `BWAPI.`**ExplosionTypeset**
A container for a set of *ExplosionType* objects.

Constructors

ExplosionTypeset()
Default constructor.

ExplosionTypeset(*set*)
Copy constructor.

Parameters **set** (`BWAPI.ExplosionTypeset`) – The ExplosionTypeset to copy.

ExplosionTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *ExplosionType* are added to the set.

Parameters **tbl** (*table*) – A table containing *ExplosionType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert (*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase (*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear ()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Force

class `BWAPI.Force`
The Force class is used to get information about each force in a match.

Normally this is considered a team.

Note: It is not called a team because players on the same force do not necessarily need to be allied at the beginning of a match.

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo
A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.4: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

getID() → int
Retrieves the unique ID that represents this Force.

Returns An integer containing the ID for the Force.

Return type int

getName() → string
Retrieves the name of the Force.

Returns A string containing the name of the force.

Return type string

Listing 8.5: Example usage

```
local myForce = BWAPI.Broodwar:self():getForce()
if myForce:getName() == "Observers" then
  print("Looks like we're observing a match")
end
```

getPlayers() → set
Retrieves the set of players that belong to this Force.

Returns A *Playerset* object containing the players that are part of this Force.

Return type *BWAPI.Playerset*

Listing 8.6: Example usage

```
-- Get the enemy force, but make sure we have an enemy
if BWAPI.Broodwar:enemy() then
  local myEnemyForce = BWAPI.Broodwar:enemy():getForce()
  print("The allies of my enemy are...")
  for ally in myEnemyForce:iterator() do
    print("  - " .. ally:getName())
  end
end
```

registerEvent(*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.
- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

Forceset

class BWAPI.**Forceset**

A container that holds a group of Forces.

See also:

Force

Constructors

Forceset()

Default constructor.

Forceset(*set*)

Copy constructor.

Parameters **set** (BWAPI.Forceset) – The Forceset to copy.

Forceset(*tbl*)

Constructor to convert a Lua table to a set. Any values in the table that are of type *Force* are added to the set.

Parameters **tbl** (*table*) – A table containing *Force* objects.

Member Functions

getPlayers() → set

Retrieves the set of players that belong to this set of forces.

Returns A *Playerset* containing the players that are part of this Forceset.

Return type *BWAPI.Playerset*

iterator() → iteratorFunction

Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Game

class `BWAPI.`**Game**
The *Game* class is implemented by *BWAPI* and is the primary means of obtaining all game state information from Starcraft Broodwar.

Game state information includes all units, resources, players, forces, bullets, terrain, fog of war, regions, etc.

See also:

BWAPI.Broodwar

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo
A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.7: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

registerEvent (*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.
- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

allies() → Playerset

Retrieves a set of all the current player's remaining allies.

Returns *Playerset* containing all allied players.

Return type *BWAPI.Playerset*

canBuildHere(*position*, *type*[, *builder* = *nil*][, *checkExplored* = *false*]) → boolean

Checks if the given unit type can be built at the given build tile position.

This function checks for creep, power, and resource distance requirements in addition to the tiles' buildability and possible units obstructing the build location.

Parameters

- **position** (BWAPI.TilePosition) – Indicates the tile position that the top left corner of the structure is intended to go.
- **type** (BWAPI.UnitType) – The *UnitType* to check for.
- **builder** (BWAPI.Unit) – (optional) The intended unit that will build the structure. If specified, then this function will also check if there is a path to the build site and exclude the builder from the set of units that may be blocking the build site.
- **checkExplored** (*boolean*) – (optional) If this parameter is true, it will also check if the target position has been explored by the current player. This value is false by default, ignoring the explored state of the build site.

Returns true indicating that the structure can be placed at the given tile position, and false if something may be obstructing the build location.

Return type boolean

Note: If the type is an addon and a builer is provided, then the location of the addon will be placed 4 tiles to the right and 1 tile down from the given `position`. If the builder is not given, then the check for the addon will be conducted at position.

Note: If the type is *UnitTypes.Special_Start_Location*, then the are for a resource depot (*Command Center*, *Hatchery*, *Nexus*) is checked as normal, but any potential obstructions (existing structures, creep, units, etc.) are ignored.

---

canMake (*type*[, *builder* = *nil*]) → boolean
Checks all the requirements in order to make a given unit type for the current player.

These include resources, supply, technology tree, availability, and required units.

Parameters

- **type** (BWAPI.UnitType) – The *UnitType* to check.
- **builder** (BWAPI.Unit) – (optional) The Unit that will be used to build/train the provided unit `type`. If this value is nil or excluded, then the builder will be excluded in the check.

Returns true indicating that the type can be made. If `builder` is provided, then it is only true if `builder` can make the `type`. Otherwise it will return false, indicating that the unit type can not be made.

Return type boolean

canResearch (*type*[, *unit* = *nil*][, *checkCanIssueCommandType* = *true*]) → boolean
Checks all the requirements in order to research a given technology type for the current player.

These include resources, technology tree, availability, and required units.

Parameters

- **type** (BWAPI.TechType) – The *TechType* to check.
- **unit** (BWAPI.Unit) – (optional) The Unit that will be used to research the provided technology `type`. If this value is nil or excluded, then the unit will be excluded in the check.
- **checkCanIssueCommandType** (*boolean*) – (optional) TODO fill this in

Returns true indicating that the type can be researched. If `unit` is provided, then it is only true if `unit` can research the `type`. Otherwise it will return false, indicating that the technology can not be researched.

Return type boolean

canUpgrade (*type*[, *unit* = *nil*][, *checkCanIssueCommandType* = *true*]) → boolean
Checks all the requirements in order to upgrade a given upgrade type for the current player.

These include resources, technology tree, availability, and required units.

Parameters

- **type** (BWAPI.UpgradeType) – The *UpgradeType* to check.
- **unit** (BWAPI.Unit) – (optional) The Unit that will be used to upgrade the provided upgrade `type`. If this value is nil or excluded, then the unit will be excluded in the check.
- **checkCanIssueCommandType** (*boolean*) – (optional) TODO fill this in

Returns true indicating that the type can be upgraded. If `unit` is provided, then it is only true if `unit` can upgrade the `type`. Otherwise it will return false, indicating that the upgrade can not be upgraded.

Return type boolean

countdownTimer () → int
Returns the remaining countdown time.

The countdown timer is used in *Capture The Flag* and *Use Map Settings* game types.

Returns Integer containing the time (in game seconds) on the countdown timer.

Return type int

Listing 8.8: Example usage

```
local Broodwar = BWAPI.Broodwar

function BWAPI.onStart()
  -- Register a callback that only occurs once when the countdown timer␣
↪reaches 0
  if Broodwar:getGameType() == BWAPI.GameTypes.Capture_The_Flag or
     Broodwar:getGameType() == BWAPI.GameTypes.Team_Capture_The_Flag
  then
    Broodwar:registerEvent(
      function() Broodwar:sendText("Try to find my flag!") end,
      function() return Broodwar:countdownTimer() == 0 end,
      1
    )
  end
end
```

elapsedTime() → int
Retrieves current amount of time in seconds that the game has elapsed.

Returns Time, in seconds, that the game has elapsed as an integer.

Return type int

enableFlag(*flag*)
Enables the state of a given flag.

Parameters **flag** (*int*) – The *Flag* entry describing the flag's effects on *BWAPI*.

Note: Flags may only be enabled at the start of the match during the *BWAPI.onStart()* callback.

See also:

Flag

enemies() → Playerset
Retrieves a set of all the current player's remaining enemies.

Returns *Playerset* containing all enemy players.

Return type *BWAPI.Playerset*

enemy() → Player
Retrieves the Player interface that represents the enemy player.

If there is more than one enemy, and that enemy is destroyed, then this function will still retrieve the same, defeated enemy. If you wish to handle multiple opponents, see the *Game.enemies()* function.

Returns Player interface representing an enemy player. Returns `nil` if there is no enemy or the current game is a replay.

Return type *BWAPI.Player*

See also:

enemies()

getAllRegions() → Regionset
Retrieves the set of all regions on the map.

Returns *Regionset* containing all map regions.

Return type *BWAPI.Regionset*

getAllUnits() → Unitset
Retrieves the set of all accessible units.

If *BWAPI.Flag.CompleteMapInformation* is enabled, then the set also includes units that are not visible to the player.

Returns *Unitset* containing all known units in the game.

Return type *BWAPI.Unitset*

Note: Units that are inside refineries are not included in this set.

---

getAPM([*includeSelects* = *false*]) → int
Retrieves the Actions Per Minute (APM) that the bot is producing.

Parameters **includeSelects** (*boolean*) – (optional) If true, the return value will include selections as individual commands, otherwise it will exclude selections. This value is false by default.

Returns The number of actions that the bot has executed per minute, on average.

Return type int

getAverageFPS() → double
Retrieves the average logical frame rate of the game in frames per second (FPS).

Returns Average logical frames per second that the game is currently running at as a double.

Return type double

See also:

getFPS()

getBestUnit(*best*, *pred*[, *center* = *Positions.Origin*][, *radius* = *999999*]) → Unit
Compares all units that match `pred` using `best` to determine which of them is the best.

All units are checked. If center and radius are specified, then it will check all units that are within the radius of the position.

Parameters

- **best** (*function*) – A function that takes two *Unit* arguments and returns the best out of the two.
- **pred** (*function*) – A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*).
- **center** (`BWAPI.Position`) – (optional) The position to use in the search. If omitted, then the entire map is searched.
- **radius** (*int*) – (optional) The distance from `center` to search for units. If omitted, then the entire map is searched.

Returns The desired unit that best matches the given criteria. Returns `nil` if a suitable unit was not found.

Return type *BWAPI.Unit*

See also:

getClosestUnit(), *Filter*, *BWAPI.Lowest()*, *BWAPI.Highest()*

getBuildLocation (*type*, *desiredPosition*[, *maxRange = 64*][, *creep = false*]) → TilePosition
Retrieves a basic build position just as the default Computer AI would.

This allows users to find simple build locations without relying on external libraries.

Parameters

- **type** (BWAPI.UnitType) – A valid *UnitType* representing the unit type to accomodate space for.
- **desiredPosition** (BWAPI.TilePosition) – A valid TilePosition containing the desired placement position.
- **maxRange** (*int*) – (optional) The maximum distance (in tiles) to build from desiredPosition.
- **creep** (*boolean*) – (optional) A special boolean value that changes the behaviour of Zerg_Creep_Colony> placement.

Returns A TilePosition containing the location that the structure should be constructed at. Returns TilePositions.Invalid if a build location could not be found within maxRange.

Return type *BWAPI.TilePosition*

getBullets () → Bulletset
Retrieves the set of all accessible bullets.

Returns *Bulletset* containing all accessible Bullet objects.

Return type *BWAPI.Bulletset*

getClosestUnit (*center*[, *pred = nil*][, *radius = 999999*]) → Unit
Retrieves the closest unit to center that matches the criteria of the callback pred within an optional radius.

Parameters

- **center** (BWAPI.Position) – The position to start searching for the closest unit.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns true for units that satisfy the intended filter and false otherwise (can be a *BWAPI.Filter unary filter*). Defaults to nil, which means no filter.
- **radius** (*int*) – (optional) The radius to search in. If omitted, the entire map will be searched.

Returns The desired unit that is closest to center. Returns nil if a suitable unit was not found.

Return type *BWAPI.Unit*

See also:

getBestUnit(), *Filter*

getClosestUnitInRectangle (*center*[, *pred = nil*][, *left = 0*][, *top = 0*][, *right = 999999*][, *bottom = 999999*]) → Unit
Retrieves the closest unit to center that matches the criteria of the callback pred within an optional rectangle.

Parameters

- **center** (BWAPI.Position) – The position to start searching for the closest unit.

- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.
- **left** (*int*) – (optional) The left position of the rectangle. This value is 0 by default.
- **top** (*int*) – (optional) The top position of the rectangle. This value is 0 by default.
- **right** (*int*) – (optional) The right position of the rectangle. This value includes the entire map width by default.
- **bottom** (*int*) – (optional) The bottom position of the rectangle. This value includes the entire map height by default.

Returns

Return type *BWAPI.Unit*

See also:

Filter

getDamageFrom (*fromType*, *toType*[, *fromPlayer = nil*][, *toPlayer = nil*]) → int

Calculates the damage received for a given player.

It can be understood as the damage from `fromType` to `toType`. Does not include shields in calculation. Includes upgrades if players are provided.

Parameters

- **fromType** (`BWAPI.UnitType`) – The unit type that will be dealing the damage.
- **toType** (`BWAPI.UnitType`) – The unit type that will be receiving the damage.
- **fromPlayer** (`BWAPI.Player`) – (optional) The player owner of the given type that will be dealing the damage. If omitted, then no player will be used to calculate the upgrades for `fromType`.
- **toPlayer** (`BWAPI.Player`) – (optional) The player owner of the type that will be receiving the damage. If omitted, then this parameter will default to `Broodwar:self()`.

Returns The amount of damage that fromType would deal to toType.

Return type int

See also:

getDamageTo()

getDamageTo (*toType*, *fromType*[, *toPlayer = nil*][, *fromPlayer = nil*]) → int

Calculates the damage dealt for a given player.

It can be understood as the damage to `toType` from `fromType`. Does not include shields in calculation. Includes upgrades if players are provided.

Parameters

- **toType** (`BWAPI.UnitType`) – The unit type that will be receiving the damage.
- **fromType** (`BWAPI.UnitType`) – The unit type that will be dealing the damage.
- **toPlayer** (`BWAPI.Player`) – (optional) The player owner of the type that will be receiving the damage. If omitted, then no player will be used to calculate the upgrades for `toType`.

- **fromPlayer** (BWAPI.Player) – (optional) The player owner of the given type that will be dealing the damage. If omitted, then this parameter will default to `Broodwar:self()`.

Returns The amount of damage that fromType would deal to toType.

Return type int

Note: This function is nearly the same as getDamageFrom. The only difference is that the last parameter is intended to default to `Broodwar:self()`.

See also:

getDamageFrom()

getRandomSeed() → number
Retrieves the initial random seed that was used in this game's creation.

This is used to identify the seed that started this game, in case an error occurred, so that developers can deterministically reproduce the error. Works in both games and replays.

Returns This game's random seed.

Return type number

getForce(*forceID*) → Force
Retrieves the Force interface object associated with a given identifier.

Parameters **forceID** (*int*) – The identifier for the Force object.

Returns Force interface object mapped to the given `forceID`. Returns `nil` if the given identifier is invalid.

Return type *BWAPI.Force*

getForces() → Forceset
Retrieves the set of all teams/forces.

Forces are commonly seen in *Use Map Settings* game types and some others such as *Top vs Bottom* and the team versions of game types.

Returns *Forceset* containing all forces in the game.

Return type *BWAPI.Forceset*

getFPS() → int
Retrieves the logical frame rate of the game in frames per second (FPS).

Returns Logical frames per second that the game is currently running at as an integer.

Return type int

Listing 8.9: Example:

```
BWAPI.Broodwar:setLocalSpeed(0)

-- Log and display the best logical FPS seen in the game
local bestFPS = 0
function BWAPI.onFrame()
  bestFPS = math.max(bestFPS, BWAPI.Broodwar:getFPS())
```

```
  BWAPI.Broodwar:drawTextScreen(BWAPI.Positions.Origin, string.format("
↪%cBest: %d GFPS\nCurrent: %d GFPS", BWAPI.Text.White, bestFPS, BWAPI.
↪Broodwar:getFPS()))
end
```

See also:

getAverageFPS()

getFrameCount () → int
Retrieves the number of logical frames since the beginning of the match.

If the game is paused, then getFrameCount will not increase.

Returns Number of logical frames that have elapsed since the game started as an integer.

Return type int

getGameType () → GameType
Retrieves the *GameType* of the current game.

Returns *GameType* indicating the rules of the match.

Return type *BWAPI.GameType*

See also:

GameType

getGeysers () → Unitset
Retrieves the set of all accessible *Vespene Geysers* in the game.

Returns *Unitset* containing *Vespene Geysers*

Return type *BWAPI.Unitset*

getGroundHeight (*tileX*, *tileY*) → int
Returns the ground height at the given tile position.

Parameters

- **tileX** (*int*) – X position to query, in tiles
- **tileY** (*int*) – Y position to query, in tiles

Returns The tile height as an integer. Possible values are: 0: Low ground; 1: Low ground doodad; 2: High ground; 3: High ground doodad; 4: Very high ground; 5: Very high ground doodad

Return type int

getGroundHeight (*position*) → int
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.TilePosition) –

Returns The tile height as an integer. Possible values are: 0: Low ground; 1: Low ground doodad; 2: High ground; 3: High ground doodad; 4: Very high ground; 5: Very high ground doodad

Return type int

getInstanceNumber () → int
Retrieves the Starcraft instance number recorded by *BWAPI* to identify which Starcraft instance an AI module belongs to.

The very first instance should return 0.

Returns An integer value representing the instance number.

Return type int

Note: This function is considered thread-safe.

---

getKeyState(*key*) → boolean
Retrieves the state of the given keyboard key.

Parameters **key** (BWAPI.Key) – A Key enum member indicating which key on the keyboard to check.

Returns A bool indicating the state of the given `key`. Returns `true` if the key was pressed and `false` if it was not. Returns `false` always if *Flag.UserInput* is disabled.

Return type boolean

See also:

Key

getLastError() → Error
Returns the last error that was set using setLastError.

If a function call in *BWAPI* has failed, you can use this function to retrieve the reason it failed.

Returns *Error* type containing the reason for failure.

Return type *BWAPI.Error*

See also:

setLastError(), *Errors*

getLastEventTime() → int
Retrieves the amount of time (in milliseconds) that has elapsed when running the last AI module callback.

This is used by tournament modules to penalize AI modules that use too much processing time.

Returns Time in milliseconds spent in last AI module call. Returns `0` when called from an AI module.

Return type int

getLatency() → int
Retrieves the current latency setting that the game is set to.

Latency indicates the delay between issuing a command and having it processed.

Returns The latency setting of the game, which is of *Latency*.

Return type int

See also:

Latency

getLatencyFrames() → int
Retrieves the maximum delay, in number of frames, between a command being issued and the command being executed by Broodwar.

Returns Difference in frames between commands being sent and executed.

Return type int

Note: In Broodwar, latency is used to keep the game synchronized between players without introducing lag.

See also:

getLatencyTime(), *getRemainingLatencyFrames()*

getLatencyTime() → int
Retrieves the maximum delay, in milliseconds, between a command being issued and the command being executed by Broodwar.

Returns Difference in milliseconds between commands being sent and executed.

Return type int

See also:

getLatencyFrames(), *getRemainingLatencyTime()*

getMinerals() → Unitset
Retrieves the set of all accessible *Mineral Fields* in the game.

Returns *Unitset* containing *Mineral Fields*

Return type *BWAPI.Unitset*

getMousePosition() → Position
Retrieves the position of the user's mouse on the screen, in Position coordinates.

Returns Position indicating the location of the mouse. Returns `Positions.Unknown` if `Flag.UserInput` is disabled.

Return type *BWAPI.Position*

getMouseState(*button*) → boolean
Retrieves the state of the given mouse button.

Parameters **button** (`BWAPI.MouseButton`) – A MouseButton enum member indicating which button on the mouse to check.

Returns A bool indicating the state of the given `button`. true if the button was pressed and false if it was not. Returns `false` always if Flag::UserInput is disabled.

Return type boolean

See also:

MouseButton

getNeutralUnits() → Unitset
Retrieves the set of all accessible neutral units in the game.

This includes *Mineral Fields*, *Vespene Geysers*, and Critters.

Returns *Unitset* containing all neutral units.

Return type *BWAPI.Unitset*

getNukeDots() → positions
Retrieves the set of all accessible *Nuke* dots.

Returns Table of *Positions* giving the coordinates of nuke locations.

Return type table

Note: Nuke dots are the red dots painted by a *Ghost* when using the nuclear strike ability.

getPlayer(*playerID*) → Player
Retrieves the *Player* associated with a given identifier.

Parameters **playerID** (*int*) – The identifier for the Player object.

Returns Player interface object mapped to the given playerID. Returns nil if the given identifier is invalid.

Return type *BWAPI.Player*

getPlayers() → Playerset
Retrieves the set of all players in the match.

This includes the neutral player, which owns all the resources and critters by default.

Returns *Playerset* containing all players in the game.

Return type *BWAPI.Playerset*

getRegion(*regionID*) → Region
Retrieves the Region associated with a given identifier.

Parameters **regionID** (*int*) – The identifier for the Region object.

Returns Region object mapped to the given regionID. Returns nil if the given ID is invalid.

Return type *BWAPI.Region*

getRegionAt(*x*, *y*) → Region
Retrieves the region at a given position.

Parameters

- **x** (*int*) – The x coordinate, in pixels.
- **y** (*int*) – The y coordinate, in pixels.

Returns The Region at the given position. Returns nil if the provided position is not valid (i.e. not within the map bounds).

Return type *BWAPI.Region*

Note: If the provided position is invalid, the error *Errors.Invalid_Parameter* is set.

See also:

getAllRegions(), *getRegion()*

getRegionAt(*position*) → Region
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.Position) –

Returns

Return type *BWAPI.Region*

getRemainingLatencyFrames() → int
Retrieves the number of frames it will take before a command sent in the current frame will be executed by the game.

Returns Number of frames until a command is executed if it were sent in the current frame.

Return type int

See also:

getRemainingLatencyTime(), *getLatencyFrames()*

getRemainingLatencyTime() → int
Retrieves the number of milliseconds it will take before a command sent in the current frame will be executed by Broodwar.

Returns Amount of time, in milliseconds, until a command is executed if it were sent in the current frame.

Return type int

See also:

getRemainingLatencyFrames(), *getLatencyTime()*

getReplayFrameCount() → int
Retrieves the maximum number of logical frames that have been recorded in a replay.

If the game is not a replay, then the value returned is undefined.

Returns The number of logical frames that the replay contains.

Return type int

getRevision() → int
Retrieves the current revision of *BWAPI*.

Returns The revision number of the current *BWAPI* interface.

Return type int

Note: This function is considered thread-safe.

---

getScreenPosition() → Position
Retrieves the top left position of the viewport from the top left corner of the map, in pixels.

Returns Position containing the coordinates of the top left corner of the game's viewport. Returns `Positions.Unknown` always if `Flag.UserInput` is disabled.

Return type *BWAPI.Position*

See also:

setScreenPosition()

getSelectedUnits() → Unitset
Retrieves the set of units that are currently selected by the user outside of *BWAPI*.

This function requires that *BWAPI.Flag.UserInput* be enabled.

Returns A *Unitset* containing the user's selected units. If *BWAPI.Flag.UserInput* is disabled, then this set is always empty.

Return type *BWAPI.Unitset*

See also:

enableFlag

getStartLocations () → tilePositions
Retrieves the set of all starting locations for the current map.

A starting location is essentially a candidate for a player's spawn point.

Returns An array-like table containing all the TilePosition objects that indicate a start location.

Return type table

See also:

Player.getStartLocation()

getStaticGeysers () → Unitset
Retrieves the set of all *Vespene Geysers* that were available at the beginning of the game.

Returns *Unitset* containing static *Vespene Geysers*

Return type *BWAPI.Unitset*

Note: This set includes resources that are inaccessible.

---

getStaticMinerals () → Unitset
Retrieves the set of all *Mineral Fields* that were available at the beginning of the game.

Returns *Unitset* containing static *Mineral Fields*

Return type *BWAPI.Unitset*

Note: This set includes resources that have been mined out or are inaccessible.

---

getStaticNeutralUnits () → Unitset
Retrieves the set of all units owned by the neutral player (resources, critters, etc.) that were available at the beginning of the game.

Returns *Unitset* containing static neutral units

Return type *BWAPI.Unitset*

Note: This set includes units that are inaccessible.

---

getUnit (*unitID*) → Unit
Retrieves the Unit interface object associated with a given identifier.

Parameters **unitID** (*int*) – The identifier for the Unit object.

Returns Unit interface object mapped to the given `unitID`. Returns `nil` if the given identifier is invalid.

Return type *BWAPI.Unit*

getUnitsInRadius (*x*, *y*, *radius*[, *pred* = *nil*]) → Unitset
Retrieves the set of accessible units that are within a given radius of a position.

Parameters

- **x** (*int*) – The x coordinate of the center, in pixels.
- **y** (*int*) – The y coordinate of the center, in pixels.
- **radius** (*int*) – The radius from the center, in pixels, to include units.

- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them within the given radius from the center position.

Return type *BWAPI.Unitset*

getUnitsInRadius (*center*, *radius*[, *pred* = *nil*]) → Unitset
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **center** (BWAPI.Position) – The coordinate of the center.
- **radius** (*int*) – The radius from the center, in pixels, to include units.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them within the given radius from the center position.

Return type *BWAPI.Unitset*

getUnitsInRectangle (*left*, *top*, *right*, *bottom*[, *pred* = *nil*]) → Unitset
Retrieves the set of accessible units that are in a given rectangle.

Parameters

- **left** (*int*) – The X coordinate of the left position of the bounding box, in pixels.
- **top** (*int*) – The Y coordinate of the top position of the bounding box, in pixels.
- **right** (*int*) – The X coordinate of the right position of the bounding box, in pixels.
- **bottom** (*int*) – The Y coordinate of the bottom position of the bounding box, in pixels.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them within the given rectangle bounds.

Return type *BWAPI.Unitset*

getUnitsInRectangle (*topLeft*, *bottomRight*[, *pred* = *nil*]) → Unitset
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **topLeft** (BWAPI.Position) –
- **bottomRight** (BWAPI.Position) –
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them within the given rectangle bounds.

Return type *BWAPI.Unitset*

getUnitsOnTile (*tileX*, *tileY*[, *pred* = *nil*]) → Unitset
Retrieves the set of accessible units that are on a given build tile.

Parameters

- **tileX** (*int*) – The X position, in tiles.
- **tileY** (*int*) – The Y position, in tiles.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them on the given build tile.

Return type *BWAPI.Unitset*

getUnitsOnTile (*tile*[, *pred* = *nil*]) → Unitset
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **tile** (BWAPI.TilePosition) –
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* object consisting of all the units that have any part of them on the given build tile.

Return type *BWAPI.Unitset*

hasCreep (*tileX*, *tileY*) → boolean
Checks if the given tile position has *Zerg* creep on it.

Parameters

- **tileX** (*int*) – The x tile coordinate to check.
- **tileY** (*int*) – The y tile coordinate to check.

Returns Returns `true` if the given tile has creep on it, or `false` if the given tile does not have creep, or if it is concealed by the fog of war

Return type boolean

hasCreep (*position*) → boolean
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.TilePosition) – The coordinate to check

Returns Returns `true` if the given tile has creep on it, or `false` if the given tile does not have creep, or if it is concealed by the fog of war

Return type boolean

hasPath (*source*, *destination*) → boolean
Checks if there is a path from source to destination.

This only checks if the source position is connected to the destination position. This function does not check if all units can actually travel from source to destination. Because of this limitation, it has an O(1) complexity, and cases where this limitation hinders gameplay is uncommon at best.

Note: If making queries on a unit, it's better to call *Unit.hasPath()*, since it is a more lenient version of this function that accounts for some edge cases.

Parameters

- **source** (BWAPI.Position) – The source position.
- **destination** (BWAPI.Position) – The destination position.

Returns Returns `true` if there is a path between the two positions, or `false` if there is no path

Return type boolean

See also:

Unit.hasPath()

hasPower (*tileX*, *tileY*[, *unitType* = *UnitTypes.None*]) → boolean

Checks if the given tile position if powered by an owned *Protoss_Pylon* for an optional unit type.

Parameters

- **tileX** (*int*) – The x tile coordinate to check.
- **tileY** (*int*) – The y tile coordinate to check.
- **unitType** (BWAPI.UnitType) – (optional) Checks if the given *UnitType* will be powered if placed at the given tile position. If omitted, then only the immediate tile position is checked for power, and the function will assume that the location requires power for any unit type.

Returns Returns `true` if the type at the given tile position will receive power, or `false` if the type will be unpowered at the given tile position

Return type boolean

hasPower (*position*[, *unitType* = *UnitTypes.None*]) → boolean

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **position** (BWAPI.TilePosition) – The tile coordinate to check
- **unitType** (BWAPI.UnitType) – (optional) Checks if the given *UnitType* will be powered if placed at the given tile position. If omitted, then only the immediate tile position is checked for power, and the function will assume that the location requires power for any unit type.

Returns Returns `true` if the type at the given tile position will receive power, or `false` if the type will be unpowered at the given tile position

Return type boolean

hasPower (*tileX*, *tileY*, *tileWidth*, *tileHeight*[, *unitType* = *UnitTypes.None*]) → boolean

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **tileX** (*int*) –
- **tileY** (*int*) –
- **tileWidth** (*int*) –
- **tileHeight** (*int*) –
- **unitType** (BWAPI.UnitType) –

Returns

Return type boolean

hasPower (*position*, *tileWidth*, *tileHeight*[, *unitType* = *UnitTypes.None*]) → boolean

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **position** (BWAPI.TilePosition) –
- **tileWidth** (*int*) –
- **tileHeight** (*int*) –
- **unitType** (BWAPI.UnitType) –

Returns

Return type boolean

hasPowerPrecise (*x*, *y*[, *unitType* = *UnitTypes.None*]) → boolean

Checks if the given pixel position is powered by an owned *Protoss_Pylon* for an optional unit type.

Parameters

- **x** (*int*) – The x pixel coordinate to check.
- **y** (*int*) – The y pixel coordinate to check.
- **unitType** (BWAPI.UnitType) – (optional) Checks if the given *UnitType* requires power or not. If ommitted, then it will assume that the position requires power for any unit type.

Returns Returns `true` if the type at the given position will have power, or `false` if the type at the given position will be unpowered

Return type boolean

hasPowerPrecise (*position*[, *unitType* = *UnitTypes.None*]) → boolean

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **position** (BWAPI.Position) –
- **unitType** (BWAPI.UnitType) –

Returns

Return type boolean

indexToUnit (*unitIndex*) → Unit
Retrieves a Unit interface object from a given unit index.

The value given as an index maps directly to Broodwar's unit array index and matches the index found in replay files. In order to use this function, *CompleteMapInformation* must be enabled.

Parameters **unitIndex** (*int*) – The unitIndex to identify the Unit with. A valid index is 0 <= unitIndex & 0x7FF < 1700.

Returns Unit interface object that matches the given `unitIndex`. Returns `nil` if the given index is invalid.

Return type *BWAPI.Unit*

isBattleNet () → boolean
Checks if the client is in a game that was created through the Battle.net multiplayer gaming service.

Returns true if the client is in a multiplayer Battle.net game and false if it is not.

Return type boolean

isBuildable (*tileX*, *tileY*[, *includeBuildings* = *false*]) → boolean
Checks if a given tile position is buildable.

This means that, if all other requirements are met, a structure can be placed on this tile. This function uses static map data.

Parameters

- **tileX** (*int*) – The x value of the tile to check.
- **tileY** (*int*) – The y value of the tile to check.
- **includeBuildings** (*boolean*) – (optional) If this is true, then this function will also check if any visible structures are occupying the space. If this value is false, then it only checks the static map data for tile buildability. This value is false by default.

Returns boolean identifying if the given tile position is buildable (true) or not (false). If `includeBuildings` was provided, then it will return false if a structure is currently occupying the tile.

Return type boolean

isBuildable (*position*[, *includeBuildings* = *false*]) → boolean
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **position** (`BWAPI.TilePosition`) –
- **includeBuildings** (*boolean*) –

Returns

Return type boolean

isDebug () → boolean
Retrieves the debug state of the *BWAPI* build.

Returns true if the *BWAPI* module is a DEBUG build, and false if it is a RELEASE build.

Return type boolean

Note: This function is considered thread-safe.

isExplored(*tileX*, *tileY*) → boolean
Checks if a given tile position has been explored by the player.

An explored tile position indicates that the player has seen the location at some point in the match, partially revealing the fog of war for the remainder of the match.

Parameters

- **tileX** (*int*) – The x tile coordinate to check.
- **tileY** (*int*) – The y tile coordinate to check.

Returns Returns `true` if the player has explored the given tile position (partially revealed fog), or `false` if the tile position was never explored (completely black fog)

Return type boolean

See also:

isVisible

isExplored(*position*) → boolean
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.TilePosition) –

Returns

Return type boolean

isFlagEnabled(*flag*) → boolean
Checks if the state of the given flag is enabled or not.

Parameters **flag** (*int*) – The *BWAPI.Flag* entry describing the flag's effects on *BWAPI*.

Returns true if the given `flag` is enabled, false if the flag is disabled.

Return type boolean

Note: Flags may only be enabled at the start of the match during the AIModule::onStart callback.

See also:

BWAPI.Flag

isGUIEnabled() → boolean
Checks if the GUI is enabled.

The GUI includes all drawing functions of *BWAPI*, as well as screen updates from Broodwar.

Returns Returns `true` if the GUI is enabled, and everything is visible, or `false` if the GUI is disabled and drawing functions are rejected

Return type boolean

See also:

setGUIEnabled()

isInGame() → boolean
Checks if the current client is inside a game.

Returns true if the client is in a game, and false if it is not.

Return type boolean

isLatComEnabled() → boolean
Checks the state of latency compensation.

Returns true if latency compensation is enabled, false if it is disabled.

Return type boolean

See also:

setLatCom()

isMultiplayer() → boolean
Checks if the current client is inside a multiplayer game.

Returns true if the client is in a multiplayer game, and false if it is a single player game, a replay, or some other state.

Return type boolean

isPaused() → boolean
Checks if the current game is paused.

While paused, *BWAPI.onFrame()* will still be called.

Returns true if the game is paused and false otherwise

Return type boolean

See also:

pauseGame(), *resumeGame()*

isReplay() → boolean
Checks if the client is watching a replay.

Returns true if the client is watching a replay and false otherwise

Return type boolean

issueCommand(*units*, *command*) → boolean
Issues a given command to a set of units.

This function automatically splits the set into groups of 12 and issues the same command to each of them. If a unit is not capable of executing the command, then it is simply ignored.

Parameters

- **units** (BWAPI.Unitset) – A *Unitset* containing all the units to issue the command for.
- **command** (BWAPI.UnitCommand) – A *UnitCommand* object containing relevant information about the command to be issued. The *Unit* associated with the command will be ignored.

Returns true if any one of the units in the *Unitset* were capable of executing the command, and false if none of the units were capable of executing the command.

Return type boolean

isVisible (*tileX*, *tileY*) → boolean
Checks if a given tile position is visible to the current player.

Parameters

- **tileX** (*int*) – The x value of the tile to check.
- **tileY** (*int*) – The y value of the tile to check.

Returns boolean identifying the visibility of the tile. If the given tile is visible, then the value is `true`. If the given tile is concealed by the fog of war, then this value will be `false`.

Return type boolean

isVisible (*position*) → boolean
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.TilePosition) –

Returns

Return type boolean

isWalkable (*walkX*, *walkY*) → boolean
Checks if the given mini-tile position is walkable.

Parameters

- **walkX** (*int*) – The x coordinate of the mini-tile, in mini-tile units (8 pixels).
- **walkY** (*int*) – The y coordinate of the mini-tile, in mini-tile units (8 pixels).

Returns true if the mini-tile is walkable and false if it is impassable for ground units.

Return type boolean

Note: This function only checks if the static terrain is walkable. Its current occupied state is excluded from this check. To see if the space is currently occupied or not, then see *getUnitsInRectangle()*.

---

isWalkable (*position*) → boolean
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **position** (BWAPI.WalkPosition) –

Returns

Return type boolean

leaveGame ()
Leaves the current game by surrendering and enters the post-game statistics/score screen.

mapFileName () → string
Retrieves the file name of the currently loaded map.

Returns Map file name.

Return type string

See also:

mapPathName(), *mapName()*

mapHash() → string
Calculates the SHA-1 hash of the currently loaded map file.

Returns string containing SHA-1 hash.

Return type string

Note: Campaign maps will return a hash of their internal map chunk components (.chk), while standard maps will return a hash of their entire map archive (.scm, .scx).

---

mapHeight() → int
Retrieves the height of the map in build tile units.

Returns Height of the map in tiles.

Return type int

See also:

mapWidth()

mapName() → string
Retrieves the title of the currently loaded map.

Returns Map title.

Return type string

See also:

mapFileName(), *mapPathName()*

mapPathName() → string
Retrieves the full path name of the currently loaded map.

Returns Map file name.

Return type string

See also:

mapFileName(), *mapName()*

mapWidth() → int
Retrieves the width of the map in build tile units.

Returns Width of the map in tiles.

Return type int

See also:

mapHeight()

neutral() → Player
Retrieves the Player interface object representing the neutral player.

The neutral player owns all the resources and critters on the map by default.

Returns *Player* indicating the neutral player.

Return type *BWAPI.Player*

observers() → Playerset
Retrieves a set of all players currently observing the game.

An observer is defined typically in a *Use Map Settings* game type as not having any impact on the game. This means an observer cannot start with any units, and cannot have any active trigger actions that create units for it.

Returns *Playerset* containing all currently active observer players

Return type *BWAPI.Playerset*

pauseGame()
Pauses the game.

While paused, *BWAPI.onFrame()* will still be called.

See also:

resumeGame()

pingMinimap(*x*, *y*)
Pings the minimap at the given position.

Minimap pings are visible to allied players.

Parameters

- **x** (*int*) – The x coordinate to ping at, in pixels, from the map's origin (left).
- **y** (*int*) – The y coordinate to ping at, in pixels, from the map's origin (top).

pingMinimap(*pos*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **pos** (BWAPI.Position) – The coordinate to ping at, from the map's origin (top left).

printf(*msg*)
Prints text to the screen as a notification.

Important: Any formatting needs to be done before the string is passed to the function (e.g. using `string.format`). See *the differences between the C++ and Lua implementations of this function*.

Note: This function allows text formatting using *BWAPI.Text* members. The best way to do this is to use the `%c` conversion specifier like so: `string.format("%cYour string", BWAPI.Text.White)`

Parameters **msg** (*string*) – The string to be printed.

Note: That text printed through this function is not seen by other players or in replays.

---

restartGame()
Restarts the match.

Works the same as if the match was restarted from the in-game menu (F10). This option is only available in single player games.

resumeGame()
Resumes the game from a paused state.

See also:

pauseGame()

self() → Player
Retrieves the player object that *BWAPI* is controlling.

Returns *Player* object representing the current player. Returns `nil` if the current game is a replay.

Return type *BWAPI.Player*

Listing 8.10: Example usage

```
function BWAPI.onStart()
  if BWAPI.Broodwar:self() then
    BWAPI.Broodwar:sendText("Hello, my name is %s.", BWAPI.
↪Broodwar:self():getName())
  end
end
```

sendText(*msg*)
Sends a text message to all other players in the game.

Important: Any formatting needs to be done before the string is passed to the function (e.g. using `string.format`). See *the differences between the C++ and Lua implementations of this function*.

Parameters **msg** (*string*) – The string to be sent.

Note: In a single player game this function can be used to execute cheat codes.

See also:

sendTextEx(), *sendTextToAllies()*

sendTextEx(*toAllies*, *msg*)
An extended version of *Game.sendText()* which allows messages to be forwarded to allies.

Parameters

- **toAllies** (*boolean*) – If this parameter is set to true, then the message is only sent to allied players, otherwise it will be sent to all players.
- **msg** (*string*) – The string to be sent.

See also:

Differences Between the C++ and Lua API: Game.sendTextEx, *sendTextToAllies()*, *sendText()*

sendTextToAllies(*msg*)
An extended version of *Game.sendText()* which always sends messages to allies.

Parameters **msg** (*string*) – The string to be sent.

See also:

Differences Between the C++ and Lua API: Game.sendTextEx, *sendTextEx()*, *sendText()*

setAlliance (*player*[, *allied* = *true*][, *alliedVictory* = *true*]) → boolean
Sets the alliance state of the current player with the target player.

Parameters

- **player** (BWAPI.Player) – The target player to set alliance with.
- **allied** (*boolean*) – (optional) If true, the current player will ally the target player. If false, the current player will make the target player an enemy. This value is true by default.
- **alliedVictory** (*boolean*) – (optional) Sets the state of "allied victory". If true, the game will end in a victory if all allied players have eliminated their opponents. Otherwise, the game will only end if no other players are remaining in the game. This value is true by default.

Returns Returns `true` if the command was successfully sent to BW, or `false` otherwise.

Return type boolean

setCommandOptimizationLevel (*level*)
Sets the command optimization level.

Command optimization is a feature in *BWAPI* that tries to reduce the APM of the bot by grouping or eliminating unnecessary game actions. For example, suppose the bot told 24 *Zerglings* to *Burrow*. At command optimization level 0, *BWAPI* is designed to select each Zergling to burrow individually, which costs 48 actions. With command optimization level 1, it can perform the same behaviour using only 4 actions. The command optimizer also reduces the amount of bytes used for each action if it can express the same action using a different command. For example, Right_Click uses less bytes than Move.

Parameters **level** (*int*) – An integer representation of the aggressiveness for which commands are optimized. A lower level means less optimization, and a higher level means more optimization.

setFrameSkip (*frameSkip*)
Sets the number of graphical frames for every logical frame.

This allows the game to run more logical frames per graphical frame, increasing the speed at which the game runs.

Parameters **frameSkip** (*int*) – Number of graphical frames per logical frame. If this value is 0 or less, then it will default to 1.

See also:

setLocalSpeed()

setGUI (*enabled*)
Alias for *setGUIEnabled()*.

setGUIEnabled (*enabled*)
Sets the rendering state of the Starcraft GUI.

This typically gives Starcraft a very low graphical frame rate and disables all drawing functionality in *BWAPI*.

Parameters **enabled** (*boolean*) – A boolean value that determines the state of the GUI. Passing false to this function will disable the GUI, and true will enable it.

Listing 8.11: Example Usage

```
-- Make our bot run thousands of games as fast as possible!
function BWAPI.onStart()
  BWAPI.Broodwar:setLocalSpeed(0)
```

```
  BWAPI.Broodwar:setGUIEnabled(false)
end
```

See also:

isGUIEnabled()

setLastError ([*e* = *Errors.None*]) → boolean
Sets the last error so that future calls to getLastError will return the value that was set.

Parameters **e** (BWAPI.Error) – (optional) The error code to set. If omitted, then the last error will be cleared.

Returns Returns `true` if the type passed was *BWAPI.Errors.None*, clearing the last error, or `false` if any other error type was passed

Return type boolean

See also:

getLastError(), *Errors*

setLatCom (*isEnabled*)
Changes the state of latency compensation.

Latency compensation modifies the state of *BWAPI*'s representation of units to reflect the implications of issuing a command immediately after the command was performed, instead of waiting consecutive frames for the results. *Latency* compensation is enabled by default.

Parameters **isEnabled** (*boolean*) – Set whether the latency compensation feature will be enabled (true) or disabled (false).

See also:

isLatComEnabled()

setLocalSpeed (*speed*)
Sets the number of milliseconds Broodwar spends in each frame.

The default values are as follows:

- Fastest: 42ms/frame
- Faster: 48ms/frame
- Fast: 56ms/frame
- Normal: 67ms/frame
- Slow: 83ms/frame
- Slower: 111ms/frame
- Slowest: 167ms/frame

Parameters **speed** (*int*) – The time spent per frame, in milliseconds. A value of 0 indicates that frames are executed immediately with no delay. Negative values will restore the default value as listed above.

Note: Specifying a value of 0 will not guarantee that logical frames are executed as fast as possible. If that is the intention, use this in combination with setFrameSkip.

See also:

setFrameSkip(), *getFPS()*

setMap (*mapFileName*) → boolean
Changes the map to the one specified.

Once restarted, the game will load the map that was provided. Changes do not take effect unless the game is restarted.

Parameters **mapFileName** (*string*) – A string containing the path and file name to the desired map.

Returns Returns `true` if the function succeeded and has changed the map, or `false` if the function failed, does not have permission from the tournament module, failed to find the map specified, or received an invalid parameter

Return type boolean

setRevealAll ([*reveal* = *true*]) → boolean
Sets the state of the fog of war when watching a replay.

Parameters **reveal** (*boolean*) – (optional) The state of the reveal all flag. If false, all fog of war will be enabled. If true, then the fog of war will be revealed. It is true by default.

Returns Returns `true` when successful, or `false` otherwise.

Return type boolean

setScreenPosition (*x*, *y*)
Moves the top left corner of the viewport to the provided position relative to the map's origin (top left (0,0)).

Parameters

- **x** (*int*) – The x coordinate to move the screen to, in pixels.
- **y** (*int*) – The y coordinate to move the screen to, in pixels.

See also:

getScreenPosition()

setScreenPosition (*pos*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **pos** (`BWAPI.Position`) – The coordinate to move the screen to.

setVision (*player*[, *enabled* = *true*]) → boolean
In a game, this function shares the vision of the current *BWAPI* player with the target player.

In a replay, this function toggles the visibility of the target player.

Parameters

- **player** (`BWAPI.Player`) – The target player to toggle vision.
- **enabled** (*boolean*) – (optional) The vision state. If true, and in a game, the current player will enable shared vision with the target player, otherwise it will unshare vision. If in a replay, the vision of the target player will be shown, otherwise the target player will be hidden. This value is true by default.

Returns Returns `true` when successful, or `false` otherwise.

Return type boolean

setTextSize ([*size = Text.Size.Default*])
Sets the size of the text for all calls to drawText following this one.

Parameters **size** (BWAPI.Text.Size) – (optional) The size of the text. This value is one of *BWAPI.Text.Size*. If this value is omitted, then a default value of *Text.Size.Default* is used.

Listing 8.12: Example usage

```
function BWAPI.onFrame()
  -- Centers the name of the player in the upper middle of the screen
  BWAPI.Broodwar:setTextSize(BWAPI.Text.Size.Large)
  BWAPI.Broodwar:drawTextScreen(
    BWAPI.Positions.Origin,
    string.format("%c%c%s",
      BWAPI.Text.Align_Center,
      BWAPI.Text.Green,
      BWAPI.Broodwar:self():getName()
    )
  )
  -- Set text size back to default
  BWAPI.Broodwar:setTextSize()
end
```

See also:

BWAPI.Text.Size

drawText (*ctype*, *x*, *y*, *text*)
Draws text on the screen at the given coordinates.

Text can be drawn in different colors or formatted using the *BWAPI.Text* members.

Important: Any formatting needs to be done before the string is passed to the function (e.g. using `string.format`). See *the differences between the C++ and Lua implementations of this function*.

Note: This function allows text formatting using *BWAPI.Text* members. The best way to do this is to use the `%c` conversion specifier like so: `string.format("%cYour string", BWAPI.Text.White)`

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **x** (*int*) – The x coordinate, in pixels, relative to ctype.
- **y** (*int*) – The y coordinate, in pixels, relative to ctype.
- **text** (*string*) – The text to draw.

drawTextMap (*x*, *y*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) – The x coordinate, in pixels, relative to the top left of the map.

- **y** (*int*) – The y coordinate, in pixels, relative to the top left of the map.
- **text** (*string*) – The text to draw.

drawTextMap (*pos*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **pos** (BWAPI.Position) – The coordinate, relative to the top left of the map.
- **text** (*string*) – The text to draw.

drawTextMouse (*x*, *y*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) – The x coordinate, in pixels, relative to the current mouse position.
- **y** (*int*) – The x coordinate, in pixels, relative to the current mouse position.
- **text** (*string*) – The text to draw.

drawTextMouse (*pos*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **pos** (BWAPI.Position) – The coordinate, relative to the current mouse position.
- **text** (*string*) – The text to draw.

drawTextScreen (*x*, *y*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) – The x coordinate, in pixels, relative to the top left of the screen.
- **y** (*int*) – The y coordinate, in pixels, relative to the top left of the screen.
- **text** (*string*) – The text to draw.

drawTextScreen (*pos*, *text*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **pos** (BWAPI.Position) – The coordinate, relative to the top left of the screen.
- **text** (*string*) – The text to draw.

drawBox (*ctype*, *left*, *top*, *right*, *bottom*, *color*[, *isSolid = false*])
Draws a rectangle on the screen with the given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.

- **left** (*int*) – The x coordinate, in pixels, relative to ctype, of the left edge of the rectangle.
- **top** (*int*) – The y coordinate, in pixels, relative to ctype, of the top edge of the rectangle.
- **right** (*int*) – The x coordinate, in pixels, relative to ctype, of the right edge of the rectangle.
- **bottom** (*int*) – The y coordinate, in pixels, relative to ctype, of the bottom edge of the rectangle.
- **color** (BWAPI.Color) – The color of the rectangle.
- **isSolid** (*boolean*) – (optional) If true, then the shape will be filled and drawn as a solid, otherwise it will be drawn as an outline. If omitted, this value will default to false.

drawBoxMap (*left*, *top*, *right*, *bottom*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **left** (*int*) –
- **top** (*int*) –
- **right** (*int*) –
- **bottom** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawBoxMap (*leftTop*, *rightBottom*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **leftTop** (BWAPI.Position) –
- **rightBottom** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawBoxMouse (*left*, *top*, *right*, *bottom*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **left** (*int*) –
- **top** (*int*) –
- **right** (*int*) –
- **bottom** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawBoxMouse (*leftTop*, *rightBottom*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **leftTop** (BWAPI.Position) –
- **rightBottom** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawBoxScreen (*left*, *top*, *right*, *bottom*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **left** (*int*) –
- **top** (*int*) –
- **right** (*int*) –
- **bottom** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawBoxScreen (*leftTop*, *rightBottom*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **leftTop** (BWAPI.Position) –
- **rightBottom** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangle (*ctype*, *ax*, *ay*, *bx*, *by*, *cx*, *cy*, *color*[, *isSolid = false*])
Draws a triangle on the screen with the given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **ax** (*int*) – The x coordinate, in pixels, relative to ctype, of the first point.
- **ay** (*int*) – The y coordinate, in pixels, relative to ctype, of the first point.
- **bx** (*int*) – The x coordinate, in pixels, relative to ctype, of the second point.
- **by** (*int*) – The y coordinate, in pixels, relative to ctype, of the second point.
- **cx** (*int*) – The x coordinate, in pixels, relative to ctype, of the third point.
- **cy** (*int*) – The y coordinate, in pixels, relative to ctype, of the third point.
- **color** (BWAPI.Color) – The color of the triangle.

- **isSolid** (*boolean*) – (optional) If true, then the shape will be filled and drawn as a solid, otherwise it will be drawn as an outline. If omitted, this value will default to false.

drawTriangleMap (*ax*, *ay*, *bx*, *by*, *cx*, *cy*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **ax** (*int*) –
- **ay** (*int*) –
- **bx** (*int*) –
- **by** (*int*) –
- **cx** (*int*) –
- **cy** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangleMap (*a*, *b*, *c*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **c** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangleMouse (*ax*, *ay*, *bx*, *by*, *cx*, *cy*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **ax** (*int*) –
- **ay** (*int*) –
- **bx** (*int*) –
- **by** (*int*) –
- **cx** (*int*) –
- **cy** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangleMouse (*a*, *b*, *c*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **c** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangleScreen (*ax*, *ay*, *bx*, *by*, *cx*, *cy*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **ax** (*int*) –
- **ay** (*int*) –
- **bx** (*int*) –
- **by** (*int*) –
- **cx** (*int*) –
- **cy** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawTriangleScreen (*a*, *b*, *c*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **c** (BWAPI.Position) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircle (*ctype*, *x*, *y*, *radius*, *color*[, *isSolid = false*])

Draws a circle on the screen with the given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **x** (*int*) – The x coordinate, in pixels, relative to ctype.
- **y** (*int*) – The y coordinate, in pixels, relative to ctype.
- **radius** (*int*) – The radius of the circle, in pixels.
- **color** (BWAPI.Color) – The color of the circle.
- **isSolid** (*boolean*) – (optional) If true, then the shape will be filled and drawn as a solid, otherwise it will be drawn as an outline. If omitted, this value will default to false.

drawCircleMap (*x*, *y*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircleMap (*p*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircleMouse (*x*, *y*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircleMouse (*p*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircleScreen (*x*, *y*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –

- **y** (*int*) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawCircleScreen (*p*, *radius*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **radius** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipse (*ctype*, *x*, *y*, *xrad*, *yrad*, *color*[, *isSolid = false*])
Draws an ellipse on the screen with the given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **x** (*int*) – The x coordinate, in pixels, relative to ctype.
- **y** (*int*) – The y coordinate, in pixels, relative to ctype.
- **xrad** (*int*) – The x radius of the ellipse, in pixels.
- **yrad** (*int*) – The y radius of the ellipse, in pixels.
- **color** (BWAPI.Color) – The color of the ellipse.
- **isSolid** (*boolean*) – (optional) If true, then the shape will be filled and drawn as a solid, otherwise it will be drawn as an outline. If omitted, this value will default to false.

drawEllipseMap (*x*, *y*, *xrad*, *yrad*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **xrad** (*int*) –
- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipseMap (*p*, *xrad*, *yrad*, *color*[, *isSolid = false*])
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –

- **xrad** (*int*) –
- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipseMouse (*x*, *y*, *xrad*, *yrad*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **xrad** (*int*) –
- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipseMouse (*p*, *xrad*, *yrad*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **xrad** (*int*) –
- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipseScreen (*x*, *y*, *xrad*, *yrad*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **xrad** (*int*) –
- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawEllipseScreen (*p*, *xrad*, *yrad*, *color*[, *isSolid = false*])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **xrad** (*int*) –

- **yrad** (*int*) –
- **color** (BWAPI.Color) –
- **isSolid** (*boolean*) –

drawDot (*ctype*, *x*, *y*, *color*)
Draws a dot on the map or screen with a given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **x** (*int*) – The x coordinate, in pixels, relative to ctype.
- **y** (*int*) – The y coordinate, in pixels, relative to ctype.
- **color** (BWAPI.Color) – The color of the dot.

drawDotMap (*x*, *y*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **color** (BWAPI.Color) –

drawDotMap (*p*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **color** (BWAPI.Color) –

drawDotMouse (*x*, *y*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **color** (BWAPI.Color) –

drawDotMouse (*p*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **color** (BWAPI.Color) –

drawDotScreen (*x*, *y*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x** (*int*) –
- **y** (*int*) –
- **color** (BWAPI.Color) –

drawDotScreen (*p*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **p** (BWAPI.Position) –
- **color** (BWAPI.Color) –

drawLine (*ctype*, *x1*, *y1*, *x2*, *y2*, *color*)
Draws a line on the map or screen with a given color.

Parameters

- **ctype** (BWAPI.CoordinateType) – The coordinate type. Indicates the relative position to draw the shape.
- **x1** (*int*) – The starting x coordinate, in pixels, relative to ctype.
- **y1** (*int*) – The starting y coordinate, in pixels, relative to ctype.
- **x2** (*int*) – The ending x coordinate, in pixels, relative to ctype.
- **y2** (*int*) – The ending y coordinate, in pixels, relative to ctype.
- **color** (BWAPI.Color) – The color of the line.

drawLineMap (*x1*, *y1*, *x2*, *y2*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x1** (*int*) –
- **y1** (*int*) –
- **x2** (*int*) –
- **y2** (*int*) –
- **color** (BWAPI.Color) –

drawLineMap (*a*, *b*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **color** (BWAPI.Color) –

drawLineMouse (*x1*, *y1*, *x2*, *y2*, *color*)
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x1** (*int*) –
- **y1** (*int*) –
- **x2** (*int*) –
- **y2** (*int*) –
- **color** (BWAPI.Color) –

drawLineMouse(*a*, *b*, *color*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **color** (BWAPI.Color) –

drawLineScreen(*x1*, *y1*, *x2*, *y2*, *color*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **x1** (*int*) –
- **y1** (*int*) –
- **x2** (*int*) –
- **y2** (*int*) –
- **color** (BWAPI.Color) –

drawLineScreen(*a*, *b*, *color*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **a** (BWAPI.Position) –
- **b** (BWAPI.Position) –
- **color** (BWAPI.Color) –

GameType

class BWAPI.**GameType**

A class that represents game types in Broodwar.

A game type is selected when creating a game.

See also:

GameTypes

Constructors

GameType ([*id = GameTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName () → string

Returns The variable name of the type.

Return type string

GameTypeset

class `BWAPI.GameTypeset`
A container for a set of *GameType* objects.

Constructors

GameTypeset ()
Default constructor.

GameTypeset (*set*)
Copy constructor.

Parameters **set** (`BWAPI.GameTypeset`) – The GameTypeset to copy.

GameTypeset (*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *GameType* are added to the set.

Parameters **tbl** (*table*) – A table containing *GameType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase (*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear ()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Order

class BWAPI.**Order**
An *Order* represents a *Unit*'s current action and can be retrieved with *Unit.getOrder()*.

It can also be used to identify the current state of the unit during command execution (gathering minerals can consist of *Orders.MoveToMinerals*, *Orders.WaitForMinerals*, *Orders.MiningMinerals*, etc.).

See also:

Unit.getOrder(), *BWAPI.Orders*

Constructors

Order ([*id = Orders.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

Orderset

class BWAPI.**Orderset**
A container for a set of *Order* objects.

Constructors

Orderset()
Default constructor.

Orderset(*set*)
Copy constructor.

Parameters **set** (BWAPI.Orderset) – The Orderset to copy.

Orderset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *Order* are added to the set.

Parameters **tbl** (*table*) – A table containing *Order* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Player

class BWAPI.**Player**
The Player represents a unique controller in the game.

Each player in a match will have his or her own player instance. There is also a neutral player which owns all the neutral units (such as mineral patches and vespene geysers).

See also:

Playerset, *PlayerType*, *Race*

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo
A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.13: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

allUnitCount ([*unitType* = *UnitTypes.AllUnits*]) → int
Retrieves the total number of units that the player has.

If the information about the player is limited, then this function will only return the number of visible units.

Parameters **unitType** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The total number of units of the given type that the player owns.

Return type int

Note: While in-progress Protoss and Terran units will be counted, in-progress Zerg units (i.e. inside of an egg) do not.

See also:

visibleUnitCount(), *completedUnitCount()*, *incompleteUnitCount()*

armor (*unitType*) → int

Calculates the armor that a given unit type will have, including upgrades.

Parameters **unitType** (BWAPI.UnitType) – The unit type to calculate armor for, using the current player's upgrades.

Returns The amount of armor that the unit will have with the player's upgrades.

Return type int

completedUnitCount ([*unitType* = *UnitTypes.AllUnits*]) → int

Retrieves the number of completed units that the player has.

If the information about the player is limited, then this function will only return the number of visible completed units.

Parameters **unitType** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The number of completed units of the given type that the player owns.

Return type int

See also:

allUnitCount(), *visibleUnitCount()*, *incompleteUnitCount()*

damage (*weaponType*) → int

Calculates the damage that a given weapon type can deal, including upgrades.

Parameters **weaponType** (BWAPI.WeaponType) – The weapon type to calculate for.

Returns The amount of damage that the weapon deals with this player's upgrades.

Return type int

deadUnitCount ([*unitType* = *UnitTypes.AllUnits*]) → int

Retrieves the number units that have died for this player.

Parameters **unitType** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The total number of units that have died throughout the game.

Return type int

gas() → int
Retrieves the current amount of vespene gas that this player has.

Returns Amount of gas that the player currently has for spending.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

gatheredGas() → int
Retrieves the cumulative amount of vespene gas that this player has gathered since the beginning of the game, including the amount that the player starts the game with (if any).

Returns Cumulative amount of gas that the player has gathered.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

gatheredMinerals() → int
Retrieves the cumulative amount of minerals/ore that this player has gathered since the beginning of the game, including the amount that the player starts the game with (if any).

Returns Cumulative amount of minerals that the player has gathered.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

getBuildingScore() → int
Retrieves the total building score, as seen in the end-game score screen.

Returns The player's building score.

Return type int

getColor() → BWAPI.Color
Retrieves the color value of the current player.

Returns *Color* object that represents the color of the current player.

Return type *BWAPI.Color*

getCustomScore() → int
Retrieves the player's custom score.

This score is used in *Use Map Settings* game types.

Returns The player's custom score.

Return type int

getForce() → Force
Retrieves the player's force.

A force is the team that the player is playing on.

Returns The Force object that the player is part of.

Return type *BWAPI.Force*

getID() → int
Retrieves a unique ID that represents the player.

Returns An integer representing the ID of the player.

Return type int

getKillScore() → int
Retrieves the total kill score, as seen in the end-game score screen.

Returns The player's kill score.

Return type int

getMaxUpgradeLevel(*upgrade*) → int
Retrieves the maximum upgrades available specific to the player.

This value is only different from *UpgradeType.maxRepeats()* in *Use Map Settings* games.

Parameters **upgrade** (BWAPI.UpgradeType) – The *UpgradeType* to retrieve the maximum upgrade level for.

Returns Maximum upgrade level of the given `upgrade` type.

Return type int

getName() → string
Retrieves the name of the player.

Returns A string containing the player's name.

Return type string

Listing 8.14: Example usage

```
local myEnemy = BWAPI.Broodwar:enemy()
if myEnemy then    -- Make sure there is an enemy!
  BWAPI.Broodwar:sendText(string.format("Prepare to be crushed, %s!",␣
↪myEnemy:getName()))
end
```

getRace() → Race
Retrieves the race of the player.

This allows you to change strategies against different races, or generalize some commands for yourself.

Returns The *Race* that the player is using. Returns `Races.Unknown` if the player chose `Races.Random` when the game started and they have not been seen.

Return type *BWAPI.Race*

Listing 8.15: Example usage

```
if BWAPI.Broodwar:enemy() then
  local enemyRace = BWAPI.Broodwar:enemy():getRace()
  if enemyRace == BWAPI.Races.Zerg then
    BWAPI.Broodwar:sendText("Do you really think you can beat me with a␣
↪zergling rush?")
  end
end
```

getRazingScore() → int
Retrieves the total razing score, as seen in the end-game score screen.

Returns The player's razing score.

Return type int

getStartLocation() → TilePosition
Retrieve's the player's starting location.

Returns A TilePosition containing the position of the start location. Returns `TilePositions.None` if the player does not have a start location, or `TilePositions.Unknown` if an error occured while trying to retrieve the start location.

Return type *BWAPI.TilePosition*

See also:

Game.getStartLocations(), *Game.getLastError()*

getTextColor() → string
Retrieves the control code character that changes the color of text messages to represent this player.

Returns character code to use for text in Broodwar, encoded as a string.

Return type string

getType() → PlayerType
Retrieves the player's controller type.

This allows you to distinguish betweeen computer and human players.

Returns The *PlayerType* that identifies who is controlling a player.

Return type *BWAPI.PlayerType*

Listing 8.16: Example usage

```
if BWAPI.Broodwar:enemy() then
  if BWAPI.Broodwar:enemy():getType() == BWAPI.PlayerTypes.Computer then
    print("Looks like something I can abuse!")
  end
end
```

Note: Other players using *BWAPI* will be treated as a human player and return *PlayerTypes.Player*.

getUnits() → Unitset
Retrieves the set of all units that the player owns.

This also includes incomplete units.

Returns Reference to a *Unitset* containing the units.

Return type *BWAPI.Unitset*

Listing 8.17: Example usage

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
```

```
  -- Do something with your units
end
```

Note: This does not include units that are loaded into transports, *Bunkers*, *Refineries*, *Assimilators*, or *Extractors*.

getUnitScore() → int
Retrieves the total unit score, as seen in the end-game score screen.

Returns The player's unit score.

Return type int

getUpgradeLevel(*upgrade*) → int
Retrieves the current upgrade level that the player has attained for a given upgrade type.

Parameters **upgrade** (BWAPI.UpgradeType) – The *UpgradeType* to query.

Returns The number of levels that the upgrade has been upgraded for this player.

Return type int

See also:

Unit.upgrade(), *getMaxUpgradeLevel()*

hasResearched(*tech*) → boolean
Checks if the player has already researched a given technology.

Parameters **tech** (BWAPI.TechType) – The *TechType* to query.

Returns true if the player has obtained the given tech, or false if they have not

Return type boolean

See also:

isResearching(), *Unit.research()*, *isResearchAvailable()*

hasUnitTypeRequirement(*unitType*[, *amount* = *1*]) → boolean
Verifies that this player satisfies a unit type requirement.

This verifies complex type requirements involving morphable *Zerg* structures. For example, if something requires a *Spire*, but the player has (or is in the process of morphing) a *Greater Spire*, this function will identify the requirement. It is simply a convenience function that performs all of the requirement checks.

Parameters

- **unitType** (BWAPI.UnitType) – The *UnitType* to check.
- **amount** (*int*) – (optional) The amount of units that are required.

Returns true if the unit type requirements are met, and false otherwise.

Return type boolean

incompleteUnitCount([*unitType* = *UnitTypes.AllUnits*]) → int
Retrieves the number of incomplete units that the player has.

If the information about the player is limited, then this function will only return the number of visible incomplete units.

Parameters **unitType** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The number of incomplete units of the given type that the player owns.

Return type int

Note: This function is a macro for `allUnitCount() - completedUnitCount()`.

Note: Incomplete Zerg units inside of eggs are not counted.

See also:

allUnitCount(), *visibleUnitCount()*, *completedUnitCount()*

isAlly (*player*) → boolean
Checks if this player is allied to the specified player.

Parameters **player** (BWAPI.Player) – The player to check alliance with.

Returns Returns `true` if this player is allied with `player` , or `false` if this player is not allied with `player`

Return type boolean

Note: This function will also return false if this player is neutral or an observer, or if `player` is neutral or an observer.

See also:

isEnemy()

isDefeated() → boolean
Checks if the player has been defeated.

Returns true if the player is defeated, otherwise false

Return type boolean

isEnemy (*player*) → boolean
Checks if this player is unallied to the specified player.

Parameters **player** (BWAPI.Player) – The player to check alliance with.

Returns Returns `true` if this player is allied with `player` , or `false` if this player is not allied with `player`

Return type boolean

Note: This function will also return false if this player is neutral or an observer, or if `player` is neutral or an observer.

See also:

isAlly()

isNeutral() → boolean
Checks if this player is the neutral player.

Returns Returns `true` if this player is the neutral player, or `false` if this player is any other player

Return type boolean

isObserver() → boolean
Checks if the player is an observer player, typically in a *Use Map Settings* observer game.

An observer player does not participate in the game.

Returns true if the player is observing, or false if the player is capable of playing in the game.

Return type boolean

isResearchAvailable(*tech*) → boolean
Checks if a technology can be researched by the player.

Certain technologies may be disabled in *Use Map Settings* game types.

Parameters **tech** (`BWAPI.TechType`) – The *TechType* to query.

Returns true if the `tech` type is available to the player for research.

Return type boolean

isResearching(*tech*) → boolean
Checks if the player is researching a given technology type.

Parameters **tech** (`BWAPI.TechType`) – The *TechType* to query.

Returns true if the player is currently researching the `tech`, or false otherwise

Return type boolean

See also:

Unit.research(), *hasResearched()*

isUnitAvailable(*unitType*) → boolean
Checks if a unit type can be created by the player.

Certain unit types may be disabled in *Use Map Settings* game types.

Parameters **unitType** (`BWAPI.UnitType`) – The *UnitType* to check.

Returns true if the `unitType` is available to the player.

Return type boolean

isUpgrading(*upgrade*) → boolean
Checks if the player is upgrading a given upgrade type.

Parameters **upgrade** (`BWAPI.UpgradeType`) – The upgrade type to query.

Returns true if the player is currently upgrading the given `upgrade`, false otherwise

Return type boolean

See also:

Unit.upgrade()

isVictorious() → boolean
Checks if the player has achieved victory.

Returns true if this player has achieved victory, otherwise false

Return type boolean

killedUnitCount([*unit* = *UnitTypes.AllUnits*]) → int
Retrieves the number units that the player has killed.

Parameters **unit** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The total number of units that the player has killed throughout the game.

Return type int

leftGame() → boolean
Checks if the player has left the game.

Returns true if the player has left the game, otherwise false

Return type boolean

maxEnergy(*unitType*) → int
Retrieves the maximum amount of energy that a unit type will have, taking the player's energy upgrades into consideration.

Parameters **unitType** (BWAPI.UnitType) – The *UnitType* to retrieve the maximum energy for.

Returns Maximum amount of energy that the given unit type can have.

Return type int

minerals() → int
Retrieves the current amount of minerals/ore that this player has.

Returns Amount of minerals that the player currently has for spending.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

refundedGas() → int
Retrieves the cumulative amount of vespene gas that this player has gained from refunding (cancelling) units and structures.

Returns Cumulative amount of gas that the player has received from refunds.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

refundedMinerals() → int
Retrieves the cumulative amount of minerals/ore that this player has gained from refunding (cancelling) units and structures.

Returns Cumulative amount of minerals that the player has received from refunds.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

repairedGas() → int
Retrieves the cumulative amount of vespene gas that this player has spent on repairing units since the beginning of the game.

This function only applies to *Terran* players.

Returns Cumulative amount of gas that the player has spent repairing.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

repairedMinerals() → int
Retrieves the cumulative amount of minerals/ore that this player has spent on repairing units since the beginning of the game.

This function only applies to *Terran* players.

Returns Cumulative amount of minerals that the player has spent repairing.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

sightRange(*unitType*) → int
Retrieves the sight range of a unit type, taking the player's sight range upgrades into consideration.

Parameters **unitType** (BWAPI.UnitType) – The *UnitType* to retrieve the sight range for.

Returns Sight range of the provided unit type for this player.

Return type int

spentGas() → int
Retrieves the cumulative amount of vespene gas that this player has spent, excluding repairs.

Returns Cumulative amount of gas that the player has spent.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

spentMinerals() → int
Retrieves the cumulative amount of minerals/ore that this player has spent, excluding repairs.

Returns Cumulative amount of minerals that the player has spent.

Return type int

Note: This function will return 0 if the player is inaccessible.

---

supplyTotal([*race* = *Races.None*]) → int
Retrieves the total amount of supply the player has available for unit control.

Parameters **race** (BWAPI.Race) – (optional) The race to query the total supply for. If this is omitted, then the player's current race will be used.

Returns The total supply available for this player and the given `race`.

Return type int

Listing 8.18: Example usage

```
if BWAPI.Broodwar:self():supplyUsed() + 8 >= BWAPI.
↪Broodwar:self():supplyTotal() then
  -- Construct pylons, supply depots, or overlords
end
```

Note: In Starcraft programming, the managed supply values are double than what they appear in the game. The reason for this is because *Zerglings* use 0.5 visible supply.

Note: In Starcraft, the supply for each race is separate. Having a *Pylon* and an *Overlord* will not give you 32 supply. It will instead give you 16 *Protoss* supply and 16 *Zerg* supply.

See also:

supplyUsed()

supplyUsed([*race* = *Races.None*]) → int

Retrieves the current amount of supply that the player is using for unit control.

Parameters **race** (BWAPI.Race) – (optional) The race to query the used supply for. If this is omitted, then the player's current race will be used.

Returns The supply that is in use for this player and the given `race`.

Return type int

Note: In Starcraft programming, the managed supply values are double than what they appear in the game. The reason for this is because *Zerglings* use 0.5 visible supply.

See also:

supplyTotal()

topSpeed(*unit*) → double

Retrieves the top speed of a unit type, taking the player's speed upgrades into consideration.

Parameters **unit** (BWAPI.UnitType) – The *UnitType* to retrieve the top speed for.

Returns Top speed of the provided unit type for this player.

Return type double

visibleUnitCount([*unitType* = *UnitTypes.AllUnits*]) → int

Retrieves the total number of strictly visible units that the player has, even if information on the player is unrestricted.

Parameters **unitType** (BWAPI.UnitType) – (optional) The unit type to query. *UnitType* macros are accepted. If this parameter is omitted, then it will use *UnitTypes.AllUnits* by default.

Returns The total number of units of the given type that the player owns, and is visible to the *BWAPI* player.

Return type int

See also:

allUnitCount(), *completedUnitCount()*, *incompleteUnitCount()*

weaponDamageCooldown (*unitType*) → int
Retrieves the weapon cooldown of a unit type, taking the player's attack speed upgrades into consideration.

Parameters **unitType** (BWAPI.UnitType) – The *UnitType* to retrieve the damage cooldown for.

Returns Weapon cooldown of the provided unit type for this player.

Return type int

weaponMaxRange (*weapon*) → int
Retrieves the maximum weapon range of a weapon type, taking the player's weapon upgrades into consideration.

Parameters **weapon** (BWAPI.WeaponType) – The *WeaponType* to retrieve the maximum range for.

Returns Maximum range of the given weapon type for units owned by this player.

Return type int

registerEvent (*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.
- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

PlayerType

class BWAPI.**PlayerType**
Represents the type of controller for the player slot (i.e. human, computer).

See also:

BWAPI.PlayerTypes

Constructors

PlayerType([*id = PlayerTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

isGameType() → boolean
Identifies whether or not this type is used in-game.

A type such as *PlayerTypes.Closed* would not be a valid in-game type.

Returns true if the type can appear in-game, false otherwise.

Return type boolean

See also:

isLobbyType()

isLobbyType() → boolean
Identifies whether or not this type is used for the pre-game lobby.

A type such as *PlayerTypes.ComputerLeft* would only appear in-game when a computer player is defeated.

Returns true if this type can appear in the pre-game lobby, false otherwise.

Return type boolean

PlayerTypeset

class `BWAPI.PlayerTypeset`
A container for a set of *`PlayerType`* objects.

Constructors

`PlayerTypeset``()`
Default constructor.

`PlayerTypeset`(*set*)
Copy constructor.

Parameters **`set`** (`BWAPI.PlayerTypeset`) – The PlayerTypeset to copy.

`PlayerTypeset`(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *`PlayerType`* are added to the set.

Parameters **`tbl`** (*`table`*) – A table containing *`PlayerType`* objects.

Member Functions

`iterator``()` → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

`asTable``()` → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

`count`(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

`contains`(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

Playerset

class BWAPI.**Playerset**
A set containing *Player* objects.

Constructors

Playerset()
Default constructor.

Playerset(*set*)
Copy constructor.

Parameters **set** (BWAPI.Playerset) – The Playerset to copy.

Playerset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *Player* are added to the set.

Parameters **tbl** (*table*) – A table containing *Player* objects.

Member Functions

getRaces() → Raceset
Returns the list of races that each player in the set is.

Returns *Raceset* containing *Playerset*'s races

Return type *BWAPI.Raceset*

See also:

Player.getRace()

getUnits() → Unitset
Returns the set of all units that every player in this set owns.

Returns *Unitset* containing *Playerset*'s units

Return type *BWAPI.Unitset*

See also:

Player.getUnits()

setAlliance(*allies* = *true* [, *alliedVictory* = *true*])
Sets the alliance status with all players contained in the *Playerset*.

Parameters

- **allies** (*boolean*) – Set to true to set the player to allied, or false for enemy.
- **alliedVictory** (*boolean*) – Set to true to turn on allied victory, or false to disable it.

See also:

Game.setAlliance()

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Position

class BWAPI.**Position**
Indicates a position that is 1x1 pixel in size. This is the most precise position type.

See also:

BWAPI.Positions, BWAPI.POSITION_SCALE

Constructors

Position ()
Default constructor.

Position (*x*, *y*)

Parameters

- **x** (*int*) – The x coordinate.
- **y** (*int*) – The y coordinate.

Position (*walkPos*)
A constructor to convert a *WalkPosition* to a *Position*.

Parameters **walkPos** (BWAPI.WalkPosition) – The position to be converted.

Position (*tilePos*)
A constructor to convert a *TilePosition* to a *Position*.

Parameters **tilePos** (BWAPI.TilePosition) – The position to be converted.

Member Variables

x
(integer) The x coordinate.

y
(integer) The y coordinate.

Member Functions

isValid() → boolean
Checks if this point is within the game's map bounds.

Returns true If it is a valid position and on the map/playing field, or false If this is not a valid position.

Return type boolean

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

---

makeValid() → Position
Checks if this point is within the game's map bounds, if not, then it will set the x and y values to be within map bounds. For example, if x is less than 0, then x is set to 0.

Returns Itself

Return type BWAPI.Position

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

See also:

isValid()

getDistance(*pos*) → double
Gets an accurate distance measurement from this point to the given position.

Note: This is a direct distance calculation that ignores all collision.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Parameters **pos** (BWAPI.Position) – The target position to get the distance to.

Returns A double representing the distance between this point and position.

Return type double

See also:

getApproxDistance()

getLength() → double
Gets the length of this point from the top left corner of the map.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Returns A double representing the length of this point from (0,0).

Return type double

See also:

getApproxDistance()

getApproxDistance (*pos*) → int
Retrieves the approximate distance using an algorithm from Starcraft: Broodwar.

Note: This is a direct distance calculation that ignores all collision.

Note: This function is desired because it uses the same "imperfect" algorithm used in Broodwar, so that calculations will be consistent with the game. It is also optimized for performance.

Parameters **pos** (BWAPI.Position) – The target position to get the distance to.

Returns A integer representing the distance between this point and `position`.

Return type int

See also:

getDistance()

setMax (*max_x*, *max_y*) → Position
Sets the maximum x and y values.

If the current x or y values exceed the given maximum, then values are set to the maximum.

Parameters

- **max_x** (*int*) – Maximum x value.
- **max_y** (*int*) – Maximum y value.

Returns Itself.

Return type BWAPI.Position

See also:

setMin()

setMax (*max*) → Position
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.Position) – Point containing the maximum x and y values.

Returns Itself.

Return type BWAPI.Position

setMin (*max_x*, *max_y*) → Position
Sets the minimum x and y values.

If the current x or y values are below the given minimum, then values are set to the minimum.

Parameters

- **max_x** (*int*) – Minimum x value.
- **max_y** (*int*) – Minimum y value.

Returns Itself.

Return type BWAPI.Position

See also:

setMax()

setMin (*max*) → Position
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.Position) – Point containing the minimum x and y values.

Returns Itself.

Return type BWAPI.Position

Race

class BWAPI.**Race**
The *Race* object is used to get information about a particular race.

For example, the default worker and supply provider *UnitType*.

As you should already know, Starcraft has three races: *Terran* , *Protoss* , and *Zerg* .

See also:

UnitType.getRace(), *Player.getRace()*, *BWAPI.Races*

Constructors

Race ([*id = Races.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes Unknown. A type is invalid if its value is less than 0 or greater than Unknown.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes Unknown.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns true if this type is valid and false otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

getResourceDepot() → UnitType

Retrieves the default resource depot *UnitType* that workers of this *Race* can construct and return resources to.

Returns *UnitType* of the center that this race uses.

Return type *BWAPI.UnitType*

Note: In Starcraft, the center is the very first structure of the *Race*'s technology tree. Also known as its base of operations or resource depot.

---

getCenter() → UnitType

Deprecated since version BWAPI: 4.2.0 Use *getResourceDepot()* instead.

getRefinery() → UnitType

Retrieves the default structure *UnitType* for this *Race* that is used to harvest gas from *Vespene Geysers*.

Returns *UnitType* of the structure used to harvest gas.

Return type *BWAPI.UnitType*

Note: In Starcraft, you must first construct a structure over a *Vespene Geyser* in order to begin harvesting Vespene Gas.

---

getSupplyProvider() → UnitType

Retrieves the default supply provider *UnitType* for this race that is used to construct units.

Returns *UnitType* that provides the player with supply.

Return type *BWAPI.UnitType*

Note: In Starcraft, training, morphing, or warping in units requires that the player has sufficient supply available for their *Race*.

---

getTransport() → UnitType

Retrieves the default transport *UnitType* for this race that is used to transport ground units across the map.

Returns *UnitType* for transportation.

Return type *BWAPI.UnitType*

Note: In Starcraft, transports will allow you to carry ground units over unpassable terrain.

---

getWorker() → UnitType

Retrieves the default worker type for this *Race*.

Returns *UnitType* of the worker that this race uses.

Return type *BWAPI.UnitType*

Note: In Starcraft, workers are the units that are used to construct structures.

Raceset

class `BWAPI.Raceset`
A container for a set of *`Race`* objects.

Constructors

`Raceset``()`
Default constructor.

`Raceset``(`*set*`)`
Copy constructor.

Parameters **`set`** (*`BWAPI.Raceset`*) – The Raceset to copy.

`Raceset``(`*tbl*`)`
Constructor to convert a Lua table to a set. Any values in the table that are of type *`Race`* are added to the set.

Parameters **`tbl`** (*`table`*) – A table containing *`Race`* objects.

Member Functions

`iterator``()` → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

`asTable``()` → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

`count``(`*val*`)` → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

Region

class BWAPI.**Region**

Region objects are created by Starcraft: Broodwar to contain several tiles with the same properties, and create a node in pathfinding and other algorithms.

Regions may not contain detailed information, but have a sufficient amount of data to identify general choke-points, accessibility to neighboring terrain, be used in general pathing algorithms, and used as nodes to rally units to.

Most parameters that are available are explicitly assigned by Broodwar itself.

See also:

Game.getAllRegions(), *Game.getRegionAt()*, *Unit.getRegion()*

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo

A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.19: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

getBoundsBottom() → int

Retrieves the approximate bottom boundary of the region.

Returns The y coordinate, in pixels, of the approximate bottom boundary of the region.

Return type int

getBoundsLeft() → int

Retrieves the approximate left boundary of the region.

Returns The x coordinate, in pixels, of the approximate left boundary of the region.

Return type int

getBoundsRight() → int

Retrieves the approximate right boundary of the region.

Returns The x coordinate, in pixels, of the approximate right boundary of the region.

Return type int

getBoundsTop() → int

Retrieves the approximate top boundary of the region.

Returns The y coordinate, in pixels, of the approximate top boundary of the region.

Return type int

getCenter() → Position
Retrieves the center of the region.

This position is used as the node of the region.

Returns A *Position* indicating the center location of the *Region*, in pixels.

Return type *BWAPI.Position*

getClosestAccessibleRegion() → Region
Retrieves the closest accessible neighbor region.

Returns The closest *Region* that is accessible.

Return type *BWAPI.Region*

getClosestInaccessibleRegion() → Region
Retrieves the closest inaccessible neighbor region.

Returns The closest *Region* that is inaccessible.

Return type *BWAPI.Region*

getDefensePriority() → int
Retrieves a value that represents the strategic advantage of this region relative to other regions.

A value of 2 may indicate a possible choke point, and a value of 3 indicates a signficant strategic position.

Returns An integer indicating this region's strategic potential.

Return type int

Note: This value is explicitly assigned by Broodwar.

---

getDistance(*other*) → int
Retrieves the center-to-center distance between two regions.

Parameters **other** (BWAPI.Region) – The target *Region* to calculate distance to.

Returns The integer distance from this Region to `other`.

Return type int

Note: Ignores all collisions.

---

getID() → int
Retrieves a unique identifier for this region.

Returns An integer that represents this region.

Return type int

Note: This identifier is explicitly assigned by Broodwar.

See also:

Game.getRegion()

getNeighbors() → Regionset
Retrieves the set of neighbor Regions that this one is connected to.

Returns A reference to a *Regionset* containing the neighboring Regions.

Return type *BWAPI.Regionset*

getRegionGroupID() → int
Retrieves a unique identifier for a group of regions that are all connected and accessible by each other.

That is, all accessible regions will have the same group ID. This function is generally used to check if a path is available between two points in constant time.

Returns An integer that represents the group of regions that this one is attached to.

Return type int

Note: This identifier is explicitly assigned by Broodwar.

---

getUnits([*pred = nil*]) → Unitset
Retrieves a *Unitset* containing all the units that are in this region.

Also has the ability to filter the units before the creation of the *Unitset*.

Parameters **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* containing all units in this region that have met the requirements of `pred`.

Return type *BWAPI.Unitset*

isAccessible() → boolean
Retrieves the state of accessibility of the region.

The region is considered accessible if it can be accessed by ground units.

Returns true if ground units can traverse this region, and false if the tiles in this region are inaccessible or unwalkable.

Return type boolean

isHigherGround() → boolean
Checks if this region is part of higher ground.

Higher ground may be used in strategic placement of units and structures.

Returns true if this region is part of strategic higher ground, and false otherwise.

Return type boolean

registerEvent(*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.

- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

Regionset

class BWAPI.**Regionset**

A container that holds a set of *Region* objects.

Constructors

Regionset()

Default constructor.

Regionset(*set*)

Copy constructor.

Parameters **set** (BWAPI.Regionset) – The Regionset to copy.

Regionset(*tbl*)

Constructor to convert a Lua table to a set. Any values in the table that are of type *Region* are added to the set.

Parameters **tbl** (*table*) – A table containing *Region* objects.

Member Functions

getCenter() → Position

Retrieves the center of the region.

This position is used as the node of the region.

Returns A Position indicating the center location of the *Regionset*, in pixels.

Return type *BWAPI.Position*

getUnits([*pred = nil*]) → Unitset

Retrieves a *Unitset* containing all the units that are in this region.

Also has the ability to filter the units before the creation of the *Unitset*.

Parameters **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* containing all units in this region that have met the requirements of `pred`.

Return type *BWAPI.Unitset*

iterator() → iteratorFunction

Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

TechType

class BWAPI.**TechType**
The *TechType* (or Technology Type, also referred to as an Ability) represents a *Unit*'s ability which can be researched with *Unit.research()* or used with *Unit.useTech()*.

In order for a Unit to use its own specialized ability, it must first be available and researched.

See also:

BWAPI.TechTypes

Constructors

TechType([*id = TechTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

energyCost() → int
Retrieves the amount of energy needed to use this *TechType* as an ability.

Returns Energy cost of the ability.

Return type int

See also:

Unit.getEnergy()

gasPrice() → int
Retrieves the vespene gas cost of researching this technology.

Returns Amount of vespene gas needed in order to research this technology.

Return type int

getOrder() → Order
Retrieves the *Order* that a *Unit* uses when using this ability.

Returns *Order* representing the action a Unit uses to perform this ability

Return type *BWAPI.Order*

getRace() → Race
Retrieves the race that is required to research or use the *TechType*.

Returns *Race* object indicating which race is designed to use this technology type.

Return type *BWAPI.Race*

Note: There is an exception where *Infested Kerrigan* can use *Psionic Storm*. This does not apply to the behavior of this function.

---

getWeapon() → WeaponType
Retrieves the Weapon that is attached to this tech type.

A technology's *WeaponType* is used to indicate the range and behaviour of the ability when used by a Unit.

Returns *WeaponType* containing information about the ability's behavior. Returns *WeaponTypes.None* if there is no corresponding *WeaponType*.

Return type *BWAPI.WeaponType*

mineralPrice() → int
Retrieves the mineral cost of researching this technology.

Returns Amount of minerals needed in order to research this technology.

Return type int

requiredUnit() → UnitType
Retrieves the *UnitType* required to research this technology.

The required unit type must be a completed unit owned by the player researching the technology.

Returns *UnitType* that is needed to research this tech type. Returns *UnitTypes.None* if no unit is required to research this tech type.

Return type *BWAPI.UnitType*

See also:

Player.completedUnitCount()

researchTime() → int
Retrieves the number of frames needed to research the tech type.

Returns The time, in frames, it will take for the research to complete.

Return type int

See also:

Unit.getRemainingResearchTime()

targetsPosition() → boolean
Checks if this ability can be used on the terrain (ground).

Returns true if the ability can be used on the terrain.

Return type boolean

targetsUnit() → boolean
Checks if this ability can be used on other units.

Returns true if the ability can be used on other units, and false if it can not.

Return type boolean

whatResearches() → UnitType
Retrieves the *UnitType* that can research this technology.

Returns *UnitType* that is able to research the technology in the game. Returns *UnitTypes.None* if the technology/ability is either provided for free or never available.

Return type *BWAPI.UnitType*

whatUses() → UnitTypeset
Retrieves the set of all *UnitTypes* that are capable of using this ability.

Returns Set of *UnitTypes* that can use this ability when researched.

Return type *BWAPI.UnitTypeset*

TechTypeset

class BWAPI.**TechTypeset**
A container for a set of *TechType* objects.

Constructors

TechTypeset()
Default constructor.

TechTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.TechTypeset) – The TechTypeset to copy.

TechTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *TechType* are added to the set.

Parameters **tbl** (*table*) – A table containing *TechType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

TilePosition

class BWAPI.**TilePosition**
Indicates a position that is 32x32 pixels in size. Typically used for building placement.

See also:

BWAPI.TilePositions, BWAPI.TILEPOSITION_SCALE

Constructors

TilePosition()
Default constructor.

TilePosition(*x*, *y*)

Parameters

- **x** (*int*) – The x coordinate.
- **y** (*int*) – The y coordinate.

TilePosition(*walkPos*)
A constructor to convert a *WalkPosition* to a *TilePosition*.

Parameters **walkPos** (BWAPI.WalkPosition) – The position to be converted.

TilePosition(*pos*)
A constructor to convert a *Position* to a *TilePosition*.

Parameters **pos** (BWAPI.Position) – The position to be converted.

Member Variables

x
(integer) The x coordinate.

y
(integer) The y coordinate.

Member Functions

isValid() → boolean
Checks if this point is within the game's map bounds.

Returns true If it is a valid position and on the map/playing field, or false If this is not a valid position.

Return type boolean

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

---

makeValid() → TilePosition
Checks if this point is within the game's map bounds, if not, then it will set the x and y values to be within map bounds. For example, if x is less than 0, then x is set to 0.

Returns Itself

Return type BWAPI.TilePosition

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

See also:

isValid()

getDistance(*pos*) → double
Gets an accurate distance measurement from this point to the given position.

Note: This is a direct distance calculation that ignores all collision.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Parameters **pos** (BWAPI.TilePosition) – The target position to get the distance to.

Returns A double representing the distance between this point and `position`.

Return type double

See also:

getApproxDistance()

getLength() → double
Gets the length of this point from the top left corner of the map.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Returns A double representing the length of this point from (0,0).

Return type double

See also:

getApproxDistance()

getApproxDistance(*pos*) → int
Retrieves the approximate distance using an algorithm from Starcraft: Broodwar.

Note: This is a direct distance calculation that ignores all collision.

Note: This function is desired because it uses the same "imperfect" algorithm used in Broodwar, so that calculations will be consistent with the game. It is also optimized for performance.

Parameters **pos** (BWAPI.TilePosition) – The target position to get the distance to.

Returns A integer representing the distance between this point and `position`.

Return type int

See also:

getDistance()

setMax (*max_x*, *max_y*) → TilePosition
Sets the maximum x and y values.

If the current x or y values exceed the given maximum, then values are set to the maximum.

Parameters

- **max_x** (*int*) – Maximum x value.
- **max_y** (*int*) – Maximum y value.

Returns Itself.

Return type BWAPI.TilePosition

See also:

setMin()

setMax (*max*) → TilePosition
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.TilePosition) – Point containing the maximum x and y values.

Returns Itself.

Return type BWAPI.TilePosition

setMin (*max_x*, *max_y*) → TilePosition
Sets the minimum x and y values.

If the current x or y values are below the given minimum, then values are set to the minimum.

Parameters

- **max_x** (*int*) – Minimum x value.
- **max_y** (*int*) – Minimum y value.

Returns Itself.

Return type BWAPI.TilePosition

See also:

setMax()

setMin (*max*) → TilePosition
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.TilePosition) – Point containing the minimum x and y values.

Returns Itself.

Return type BWAPI.TilePosition

Unit

class BWAPI.**Unit**
The Unit class is used to get information about individual units as well as issue orders to units.

Every Unit in the game is either accessible or inaccessible. To determine if an AI can access a particular unit, BWAPI checks to see if *BWAPI.Flag.CompleteMapInformation* is enabled. So there are two cases to consider - either the flag is enabled, or it is disabled:

If *BWAPI.Flag.CompleteMapInformation* is disabled, then a unit is accessible if and only if it is visible. If *BWAPI.Flag.CompleteMapInformation* is enabled, then all units that exist in the game are accessible, and *Unit.exists()* is accurate for all units. Similarly *BWAPI.onUnitDestroy()* messages are generated for all units that get destroyed, not just visible ones.

Note: Some properties of visible enemy units will not be made available to the AI (such as the contents of visible enemy dropships). If a unit is not visible, *Unit.exists()* will return `false`, regardless of whether or not the unit exists. This is because absolutely no state information on invisible enemy units is made available to the AI. To determine if an enemy unit has been destroyed, the AI must watch for *BWAPI.onUnitDestroy()* messages from BWAPI, which is only called for visible units which get destroyed.

If a Unit is not accessible, then only the `getInitial*` functions will be available to the AI. However for units that were owned by the player, *Unit.getPlayer()* and *Unit.getType()* will continue to work for units that have been destroyed.

Constructors

This class is not constructable through Lua.

Member Variables

clientInfo
A Lua table that can be used to store arbitrary data associated with the current object.

Listing 8.20: Example usage

```
obj.clientInfo["test"] = 5
print(obj.clientInfo["test"]) -- prints "5"
```

See also:

The differences between the C++ and Lua implementations

Member Functions

getID() → id
Retrieves a unique identifier for this unit.

Returns An integer containing the unit's identifier.

Return type number

See also:

getReplayID()

exists() → exists
Checks if the Unit exists in the view of the BWAPI player.

This is used primarily to check if BWAPI has access to a specific unit, or if the unit is alive. This function is more general and would be synonymous to an isAlive function if such a function were necessary.

In the event that this function returns `false`, there are two cases to consider:

1. You own the unit. This means the unit is dead.

2. Another player owns the unit. This could either mean that you don't have access to the unit or that the unit has died. You can specifically identify dead units by polling onUnitDestroy.

Returns `true` if the unit exists on the map and is visible according to BWAPI or `false` if the unit is not accessible or the unit is dead.

Return type boolean

See also:

isVisible(), *isCompleted()*

getReplayID() → id
Retrieves the unit identifier for this unit as seen in replay data.

Note: This is only available if *BWAPI.Flag.CompleteMapInformation* is enabled.

Returns An integer containing the replay unit identifier.

See also:

getID()

getPlayer() → player
Retrieves the player that owns this unit.

Returns The owning *Player* or *Game.neutral()* if the unit is a neutral unit or inaccessible.

Return type *BWAPI.Player*

getType() → type
Retrieves the unit's type.

Returns A *UnitType* representing the unit's type or *Unknown* if this unit is inaccessible or cannot be determined.

Return type *BWAPI.UnitType*

See also:

getInitialType()

getPosition() → position
Retrieves the unit's position from the upper left corner of the map in pixels.

The position returned is roughly the center if the unit.

Returns Position object representing the unit's current position or *Unknown* if this unit is inaccessible.

Return type *BWAPI.Position*

Note: The unit bounds are defined as this value plus/minus the values of *UnitType.dimensionLeft()*, *UnitType.dimensionUp()*, *UnitType.dimensionRight()*, and *UnitType.dimensionDown()*, which is conveniently expressed in *getLeft()*, *getTop()*, *getRight()*, and *getBottom()* respectively.

See also:

getTilePosition(), *getInitialPosition()*, *getLeft()*, *getTop()*

getTilePosition() → tilePosition
Retrieves the unit's build position from the upper left corner of the map in tiles.

Returns TilePosition object representing the unit's current tile position or *Unknown* if this unit is inaccessible.

Return type *BWAPI.TilePosition*

Note: This tile position is the tile that is at the top left corner of the structure.

See also:

getPosition(), *getInitialTilePosition()*

getAngle() → angle
Retrieves the unit's facing direction in radians.

Note: A value of 0.0 means the unit is facing east.

Returns A double with the angle measure in radians.

Return type number

getVelocityX() → velocityX
Retrieves the x component of the unit's velocity, measured in pixels per frame.

Returns A double that represents the velocity's x component.

Return type number

See also:

getVelocityY()

getVelocityY() → velocityY
Retrieves the y component of the unit's velocity, measured in pixels per frame.

Returns A double that represents the velocity's y component.

Return type number

See also:

getVelocityX()

getRegion() → region
Retrieves the Region that the center of the unit is in.

Returns The *Region* object that contains this unit or `nil` if the unit is inaccessible.

Return type *Region*

Listing 8.21: Example

```
local Broodwar = BWAPI.Broodwar
local myUnits = Broodwar:self():getUnits()
for u in myUnits:iterator() do
  if u:isFlying() and u:isUnderAttack() then -- implies exists and isCompleted
    local r = u:getRegion()
```

```
    if r then
      u:move(r:getClosestInaccessibleRegion()) -- Retreat to inaccessible␣
↪region
    end
  end
end
```

Note: If this function returns a successful state, then the following functions will also return a successful state: *exists()*

getLeft() → left
Retrieves the X coordinate of the unit's left boundary, measured in pixels from the left side of the map.

Returns An integer representing the position of the left side of the unit.

Return type number

See also:

getTop(), *getRight()*, *getBottom()*

getTop() → top
Retrieves the Y coordinate of the unit's top boundary, measured in pixels from the top of the map.

Returns An integer representing the position of the top side of the unit.

Return type number

See also:

getLeft(), *getRight()*, *getBottom()*

getRight() → right
Retrieves the X coordinate of the unit's right boundary, measured in pixels from the left side of the map.

Returns An integer representing the position of the right side of the unit.

Return type number

See also:

getLeft(), *getTop()*, *getBottom()*

getBottom() → bottom
Retrieves the Y coordinate of the unit's bottom boundary, measured in pixels from the top of the map.

Returns An integer representing the position of the bottom side of the unit.

Return type number

See also:

getLeft(), *getTop()*, *getRight()*

getHitPoints() → hp
Retrieves the unit's current Hit Points (HP) as seen in the game.

Returns An integer representing the amount of hit points a unit currently has.

Return type number

Note: In Starcraft, a unit usually dies when its HP reaches 0. It is possible however, to have abnormal HP values in the Use Map Settings game type and as the result of a hack over Battle.net. Such values include units that have 0 HP (can't be killed conventionally) or even negative HP (death in one hit).

See also:

UnitType.maxHitPoints(), *getShields()*, *getInitialHitPoints()*

getShields() → shields
Retrieves the unit's current Shield Points (Shields) as seen in the game.

Returns An integer representing the amount of shield points a unit currently has.

Return type number

See also:

UnitType.maxShields(), *getHitPoints()*

getEnergy() → energy
Retrieves the unit's current Energy Points (Energy) as seen in the game.

Returns An integer representing the amount of energy points a unit currently has.

Return type number

Note: Energy is required in order for units to use abilities.

See also:

UnitType.maxEnergy()

getResources() → resources
Retrieves the resource amount from a resource container, such as a Mineral Field and Vespene Geyser.

If the unit is inaccessible, then the last known resource amount is returned.

Returns An integer representing the last known amount of resources remaining in this resource.

Return type number

See also:

getInitialResources()

getResourceGroup() → int
Retrieves a grouping index from a resource container.

Other resource containers of the same value are considered part of one expansion location (group of resources that are close together).

Note: This grouping method is explicitly determined by Starcraft itself and is used only by the internal AI.

Returns An integer with an identifier between 0 and 250 that determine which resources are grouped together to form an expansion.

getDistance(*target*) → distance
Retrieves the distance between this unit and a target.

Note: Distance is calculated from the edge of this unit, using Starcraft's own distance algorithm. Ignores collisions.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. If it is a unit, then it will calculate the distance to the edge of the target unit.

Returns An integer representation of the number of pixels between this unit and the `target`.

Return type number

hasPath(*target*) → hasPath
Using data provided by Starcraft, checks if there is a path available from this unit to the given target.

Note: This function only takes into account the terrain data, and does not include buildings when determining if a path is available. However, the complexity of this function is constant (O(1)), and no extensive calculations are necessary.

Note: If the current unit is an air unit, then this function will always return true.

Note: If the unit somehow gets stuck in unwalkable terrain, then this function may still return true if one of the unit's corners is on walkable terrain (i.e. if the unit is expected to return to the walkable terrain).

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. A Position or a Unit that is used to determine if this unit has a path to the target.

Returns `true` if there is a path between this unit and the target or `false` if the target is on a different piece of land than this one (such as an island).

Return type boolean

getLastCommandFrame() → frameNum
Retrieves the frame number that sent the last successful command.

Note: This value is comparable to *Game.getFrameCount()*.

Returns The frame number that sent the last successfully processed command to BWAPI.

Return type number

See also:

Game.getFrameCount(), *getLastCommand()*

getLastCommand() → command
Retrieves the last successful command that was sent to BWAPI.

Returns The last command that was processed.

Return type *UnitCommand*

See also:

getLastCommandFrame()

getLastAttackingPlayer() → player
Retrieves the Player that last attacked this unit.

Returns Player that last attacked this unit or `nil` if this unit was not attacked.

Return type *Player*

Note: If this function returns a successful state, then the following function calls will also return a successful state: exists()

---

getInitialType() → type
Retrieves the initial type of the unit.

This is the type that the unit starts as in the beginning of the game. This is used to access the types of static neutral units such as mineral fields when they are not visible.

Returns *UnitType* of this unit as it was when it was created or *Unknown* if this unit was not a static neutral unit in the beginning of the game.

Return type *UnitType*

getInitialPosition() → position
Retrieves the initial position of this unit.

This is the position that the unit starts at in the beginning of the game. This is used to access the positions of static neutral units such as mineral fields when they are not visible.

Returns Position indicating the unit's initial position when it was created or *Unknown* if this unit was not a static neutral unit in the beginning of the game.

Return type *Position*

getInitialTilePosition() → tilePosition
Retrieves the initial build tile position of this unit.

This is the tile position that the unit starts at in the beginning of the game. This is used to access the tile positions of static neutral units such as mineral fields when they are not visible. The build tile position corresponds to the upper left corner of the unit.

Returns TilePosition indicating the unit's initial tile position when it was created or *Unknown* if this unit was not a static neutral unit in the beginning of the game.

Return type *TilePosition*

getInitialHitPoints() → hp
Retrieves the amount of hit points that this unit started off with at the beginning of the game.

The unit must be neutral.

Returns Number of hit points that this unit started with or `0` if this unit was not a neutral unit at the beginning of the game.

Return type number

Note: It is possible for the unit's initial hit points to differ from the maximum hit points.

See also:

Game.getStaticNeutralUnits()

getInitialResources() → resources
Retrieves the amount of resources contained in the unit at the beginning of the game.

The unit must be a neutral resource container.

Returns Amount of resources that this unit started with or 0 if this unit was not a neutral unit at the beginning of the game, or if this unit does not contain resources. It is possible that the unit simply contains 0 resources.

See also:

Game.getStaticNeutralUnits()

getKillCount() → kills
Retrieves the number of units that this unit has killed in total.

Note: The maximum amount of recorded kills per unit is 255.

Returns integer indicating this unit's kill count.

Return type number

getAcidSporeCount() → count
Retrieves the number of acid spores that this unit is inflicted with.

Returns Number of acid spores on this unit.

Return type number

getInterceptorCount() → count
Retrieves the number of interceptors that this unit manages.

This function is only for the *Carrier* and its hero.

Returns Number of interceptors in this unit.

Return type number

Note: This number may differ from the number of units returned from *getInterceptors()*. This occurs for cases in which you can see the number of enemy interceptors in the Carrier HUD, but don't actually have access to the individual interceptors.

See also:

getInterceptors()

getScarabCount() → count
Retrieves the number of scarabs that this unit has for use.

This function is only for the *Reaver*.

Returns Number of scarabs this unit has ready.

Return type number

getSpiderMineCount() → count
Retrieves the amount of *Spider Mines* this unit has available.

This function is only for the `Vulture <BWAPI.UnitTypes.Terran_Vulture`.

Returns Number of spider mines available for placement.

Return type number

getGroundWeaponCooldown() → framesLeft
Retrieves the unit's ground weapon cooldown.

This value decreases every frame, until it reaches 0. When the value is 0, this indicates that the unit is capable of using its ground weapon, otherwise it must wait until it reaches 0.

Note: This value will vary, because Starcraft adds an additional random value between (-1) and (+2) to the unit's weapon cooldown.

Returns Number of frames needed for the unit's ground weapon to become available again.

Return type number

getAirWeaponCooldown() → framesLeft
Retrieves the unit's air weapon cooldown.

This value decreases every frame, until it reaches 0. When the value is 0, this indicates that the unit is capable of using its air weapon, otherwise it must wait until it reaches 0.

Note: This value will vary, because Starcraft adds an additional random value between (-1) and (+2) to the unit's weapon cooldown.

Returns Number of frames needed for the unit's air weapon to become available again.

Return type number

getSpellCooldown() → framesLeft
Retrieves the unit's ability cooldown.

This value decreases every frame, until it reaches 0. When the value is 0, this indicates that the unit is capable of using one of its special abilities, otherwise it must wait until it reaches 0.

Note: This value will vary, because Starcraft adds an additional random value between (-1) and (+2) to the unit's ability cooldown.

Returns Number of frames needed for the unit's abilities to become available again.

Return type number

getDefenseMatrixPoints() → hp
Retrieves the amount of hit points remaining on the *Defensive Matrix* created by a *Science Vessel*.

The *Defensive Matrix* ability starts with 250 hit points when it is used.

Returns Number of hit points remaining on this unit's *Defensive Matrix*.

Return type number

See also:

getDefenseMatrixTimer(), *isDefenseMatrixed()*

getDefenseMatrixTimer() → framesLeft
Retrieves the time, in frames, that the *Defensive Matrix* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

getDefenseMatrixPoints(), *isDefenseMatrixed()*

getEnsnareTimer() → framesLeft
Retrieves the time, in frames, that *Ensnare* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isEnsnared()

getIrradiateTimer() → framesLeft
Retrieves the time, in frames, that *Irradiate* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isIrradiated()

getLockdownTimer() → framesLeft
Retrieves the time, in frames, that *Lockdown* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isLockdowned()

getMaelstromTimer() → framesLeft
Retrieves the time, in frames, that *Maelstrom* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isMaelstrommed()

getOrderTimer() → framesLeft
Retrieves an internal timer used for the primary order.

Its use is specific to the order type that is currently assigned to the unit.

Returns A value used as a timer for the primary order.

Return type number

See also:

getOrder()

getPlagueTimer() → framesLeft
Retrieves the time, in frames, that *Plague* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isPlagued()

getRemoveTimer() → framesLeft
Retrieves the time, in frames, until this temporary unit is destroyed or removed. Once this value reaches 0, the unit is destroyed.

This is used to determine the remaining time for the following units that were created by abilities:

- *Hallucination*
- *Broodling*
- *Dark Swarm*
- *Disruption Web*
- *Scanner Sweep*

Returns Number of frames remaining until the unit is destroyed or removed.

Return type number

getStasisTimer() → framesLeft
Retrieves the time, in frames, that *Stasis Field* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isStasised()

getStimTimer() → framesLeft
Retrieves the time, in frames, that *Stim Packs* will remain active on the current unit.

Returns Number of frames remaining until the effect is removed.

Return type number

See also:

isStimmed()

getBuildType() → type
Retrieves the building type that a worker (*SCV*, *Probe*, *Drone*) is about to construct.

If the unit is morphing or is an incomplete structure, then this returns the *UnitType* that it will become when it has completed morphing/constructing.

Returns *UnitType* indicating the type that a worker (*SCV*, *Probe*, *Drone*) is about to construct, or the type that an incomplete unit will be when completed.

Return type *UnitType*

getTrainingQueue() → queue
Retrieves the list of units queued up to be trained.

Important: See *the differences between the C++ and Lua implementations of this function* for more information

---

Returns An array-like table containing all of the *UnitType*'s that are in this building's training queue.

Return type table of the format {1 = *UnitType*, 2 = *UnitType*, ...}

See also:

train(), *cancelTrain()*, *isTraining()*

getTech() → tech
Retrieves the technology that this unit is currently researching.

Returns *TechType* indicating the technology being researched by this unit, or *None* if this unit is not researching anything.

Return type *TechType*

See also:

research(), *cancelResearch()*, *isResearching()*, *getRemainingResearchTime()*

getUpgrade() → upgrade
Retrieves the upgrade that this unit is currently upgrading.

Returns *UpgradeType* indicating the upgrade in progress by this unit, or *None* if this unit is not upgrading anything.

Return type *UpgradeType*

See also:

upgrade(), *cancelUpgrade()*, *isUpgrading()*, *getRemainingUpgradeTime()*

getRemainingBuildTime() → framesLeft
Retrieves the remaining build time for a unit or structure that is being trained or constructed.

Returns Number of frames remaining until the unit's completion.

Return type number

getRemainingResearchTime() → framesLeft
Retrieves the amount of time until the unit is done researching its currently assigned *TechType*.

Returns The remaining research time, in frames, for the current technology being researched by this unit, or `0` if the unit is not researching anything.

Return type number

See also:

research(), *cancelResearch()*, *isResearching()*, *getTech()*

getRemainingTrainTime() → framesLeft
Retrieves the remaining time, in frames, of the unit that is currently being trained.

Note: If the unit is a *Hatchery*, *Lair*, or *Hive*, this retrieves the amount of time until the next larva spawns.

Returns Number of frames remaining until the current training unit becomes completed, or the number of frames remaining until the next larva spawns, or `0` if the unit is not training or has three larvae.

Return type number

See also:

train(), *getTrainingQueue()*

getRemainingUpgradeTime() → framesLeft
Retrieves the amount of time until the unit is done upgrading its current upgrade.

Returns The remaining upgrade time, in frames, for the current upgrade, or `0` if the unit is not upgrading anything.

Return type number

See also:

upgrade(), *cancelUpgrade()*, *isUpgrading()*, *getUpgrade()*

getBuildUnit() → unit
Retrieves the unit currently being trained, or the corresponding paired unit for *SCVs* and *Terran* structures, depending on the context.

For example, if this unit is a *Factory* under construction, this function will return the *SCV* that is constructing it. If this unit is a *SCV*, then it will return the structure it is currently constructing. If this unit is a *Nexus*, and it is training a *Probe*, then the probe will be returned.

Warning: This will return an incorrect unit when called on *Reavers*.

Returns Paired build unit that is either constructing this unit or being constructed by this unit, structure being constructed by this unit, the unit that is being trained by this structure, or `nil` if there is no unit constructing this one or this unit is not constructing another unit.

Return type *Unit*

getTarget() → unit
Generally returns the appropriate target unit after issuing an order that accepts a target unit (i.e. attack, repair, gather, etc.).

To get a target that has been acquired automatically without issuing an order, use *getOrderTarget()*.

Returns Unit that is currently being targeted by this unit.

Return type *Unit*

See also:

getOrderTarget()

getTargetPosition() → position
Retrieves the target position the unit is moving to, provided a valid path to the target position exists.

Returns Target position of a movement action.

Return type *Position*

getOrder() → order
Retrieves the primary *Order* that the unit is assigned.

Primary orders are distinct actions such as *AttackUnit* and *PlayerGuard*.

Returns The primary *Order* that the unit is executing.

Return type *Order*

getSecondaryOrder() → order
Retrieves the secondary *Order* that the unit is assigned.

Secondary orders are run in the background as a sub-order. An example would be *TrainFighter*, because a *Carrier* can move and train fighters at the same time.

Returns The secondary *Order* that the unit is executing.

Return type *Order*

getOrderTarget() → unit
Retrieves the unit's primary order target.

This is usually set when the low level unit AI acquires a new target automatically. For example if an enemy *Probe* comes in range of your *Marine*, the *Marine* will start attacking it, and getOrderTarget will be set in this case, but not getTarget.

Returns The Unit that this unit is currently targetting.

Return type *Unit*

See also:

getTarget(), *getOrder()*

getOrderTargetPosition() → position
Retrieves the target position for the unit's order.

For example, when *Move* is assigned, *getTargetPosition()* returns the end of the unit's path, but this returns the location that the unit is trying to move to.

Returns Position that this unit is currently targetting.

Return type *Position*

See also:

getTargetPosition(), *getOrder()*

getRallyPosition() → position
Retrieves the position the structure is rallying units to once they are completed.

Returns Position that a completed unit coming from this structure will travel to, or *None* if this building does not produce units.

Return type *Position*

Note: If *getRallyUnit()* is valid, then this value is ignored.

See also:

setRallyPoint(), *getRallyUnit()*

getRallyUnit() → unit
Retrieves the unit the structure is rallying units to once they are completed.

Units will then follow the targetted unit.

Returns Unit that a completed unit coming from this structure will travel to, or `nil` if the structure is not rallied to a unit or it does not produce units.

Return type *Unit*

Note: A rallied unit takes precedence over a rallied position. That is if the return value is valid (non-nil), then *getRallyPosition()* is ignored.

See also:

setRallyPoint(), *getRallyPosition()*

getAddon() → unit
Retrieves the add-on that is attached to this unit.

Returns Unit interface that represents the add-on that is attached to this unit, or `nil` if this unit does not have an add-on.

Return type *Unit*

getNydusExit() → unit
Retrieves the *Nydus Canal* that is attached to this one.

Every *Nydus Canal* can place a "Nydus Exit" which, when connected, can be travelled through by *Zerg* units.

Returns Unit interface representing the *Nydus Canal* connected to this one, or `nil` if the unit is not a *Nydus Canal*, is not owned, or has not placed a Nydus Exit.

Return type *Unit*

getPowerUp() → unit
Retrieves the power-up that the worker unit is holding.

Power-ups are special units such as the *Flag* in the *Capture The Flag* game type, which can be picked up by worker units.

Returns The Unit interface object that represents the power-up, or `nil` if the unit is not carrying anything.

Return type *Unit*

Listing 8.22: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  -- If we are carrying a flag
  if u:getPowerUp() u:getPowerUp():getType() == BWAPI.UnitTypes.Powerup_Flag␣
↪then
    local pred = function(x)
      return BWAPI.Filter.IsFlagBeacon(x) and BWAPI.Filter.IsOwned(x)
    end
    -- return it to our flag beacon to score
    u:move( u:getClosestUnit(pred) )
  end
end
```

Note: If your bot is strictly melee/1v1, then this method is not necessary.

Note: If this function returns a successful state, then the following function calls will also return a successful state: `getType():isWorker()`, *isCompleted()*

getTransport() → unit
Retrieves the transport unit (*Dropship*, *Shuttle*, *Overlord*) or *Bunker* unit that has this unit loaded inside of it.

Returns The unit that this unit is loaded inside of, or `nil` if the unit is not loaded inside of a unit.

Return type *Unit*

getLoadedUnits() → units
Retrieves the set of units that are contained within this *Bunker*, *Dropship*, *Shuttle*, or *Overlord*.

Returns A *Unitset* containing all of the units that are loaded inside of the current unit.

Return type *Unitset*

getSpaceRemaining() → spaceRemaining
Retrieves the remaining unit-space available for *Bunkers* and transports (*Dropships*, *Shuttles*, *Overlords*).

Returns The number of spots available to transport a unit.

Return type number

See also:

getLoadedUnits()

getCarrier() → unit
Retrieves the parent *Carrier* that owns this *Interceptor*.

Returns The parent *Carrier* unit that has ownership of this one, or `nil` if the current unit is not an *Interceptor*.

Return type *Unit*

getInterceptors() → units
Retrieves the set of *Interceptors* controlled by this unit.

This is intended for *Carriers* and its hero.

Returns *Unitset* containing *Interceptor* units owned by this carrier.

Return type *Unitset*

See also:

getInterceptorCount()

getHatchery() → unit
Retrieves the parent *Hatchery*, *Lair*, or *Hive* that owns this particular unit.

This is intended for *Larvae*.

Returns Hatchery unit that has ownership of this larva, or `nil` if the current unit is not a *Larva* or has no parent.

Return type *Unit*

See also:

getLarva()

getLarva() → units

Retrieves the set of *Larvae* that were spawned by this unit.

Only *Hatcheries*, *Lairs*, and *Hives* are capable of spawning *Larvae*. This is like clicking the "Select Larva" button and getting the selection of *Larvae*.

Returns *Unitset* containing *Larva* units owned by this unit. The set will be empty if there are none.

Return type *Unitset*

See also:

getHatchery()

getUnitsInRadius(*radius*[, *pred*]) → units

Retrieves the set of all units in a given radius of the current unit.

Takes into account this unit's dimensions. Can optionally specify a filter to include only specific units (such as only ground units, etc.)

Parameters

- **radius** (*number*) – The radius, in pixels, to search for units.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* containing the set of units that match the given criteria.

Return type *Unitset*

Listing 8.23: Example usage

```
-- Get main building closest to start location.
local isResourceDepotPred = function(x) return BWAPI.Filter.
↪IsResourceDepot(x) end
local main = BWAPI.Broodwar:getClosestUnit( BWAPI.
↪Broodwar:self():getStartLocation(), isResourceDepotPred )
if main then -- check if main is valid
  -- Get sets of resources and workers
  local myResources = main:getUnitsInRadius(1024, BWAPI.Filter.
↪IsMineralField);
  if not myResources:empty() then -- check if we have resources nearby
    local workerPred = function(x)
      BWAPI.Filter.IsWorker(x) and BWAPI.Filter.IsIdle(x) and BWAPI.Filter.
↪IsOwned(x)
    end
    local myWorkers = main:getUnitsInRadius(512, workerPred)
    local myResourcesArray = myResources:asTable()
    for worker in myWorkers:iterator() do
      local randomResource = myResourcesArray[math.random(#myResourcesArray)]
      worker:gather(randomResource)
    end
  end
end
```

See also:

UnitFilter differences between C++ and Lua, `getClosestUnit()`, `getUnitsInWeaponRange()`, `Game.getUnitsInRadius()`, `Game.getUnitsInRectangle()`

getUnitsInWeaponRange (*weapon*[, *pred*]) → units
Obtains the set of units within weapon range of this unit.

Parameters

- **weapon** (`BWAPI.WeaponType`) – The weapon type to use as a filter for distance and units that can be hit by it.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). If omitted, no additional filter is used.

Returns The set of units within weapon range of this unit.

Return type *Unitset*

See also:

UnitFilter differences between C++ and Lua, `getUnitsInRadius()`, `getClosestUnit()`, `Game.getUnitsInRadius()`, `Game.getUnitsInRectangle()`

getClosestUnit ([*pred*][, *radius* = *999999*]) → unit
Retrieves the closest unit to this one.

Parameters

- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). If `pred` is omitted or `nil`, then the closest unit owned by any player will be returned.
- **radius** (*number*) – (optional) The maximum radius to check for the closest unit. For performance reasons, a developer can limit the radius that is checked. If omitted, then the entire map is checked.

Returns The closest unit that matches the predicate, or `nil` if no matching unit is found.

Return type *Unit*

See also:

UnitFilter differences between C++ and Lua, `Unit.getUnitsInRadius()`, `Game.getUnitsInRadius()`, `Game.getUnitsInRectangle()`

hasNuke () → bool
Checks if the current unit is housing a *Nuke*.

This is only available for *Nuclear Silos*.

Returns `true` if this unit has a *Nuke* ready, and `false` if there is no *Nuke*.

Return type boolean

isAccelerating () → bool
Checks if the current unit is accelerating.

Returns `true` if this unit is accelerating, and `false` otherwise

Return type boolean

isAttacking() → bool
Checks if this unit is currently attacking something.

Returns `true` if this unit is attacking another unit, and `false` if it is not.

Return type boolean

isAttackFrame() → bool
Checks if this unit is currently playing an attack animation.

Issuing commands while this returns `true` may interrupt the unit's next attack sequence.

Returns `true` if this unit is currently running an attack frame, and `false` if interrupting the unit is feasible.

Return type boolean

Note: This function is only available to some unit types, specifically those that play special animations when they attack.

---

isBeingConstructed() → bool
Checks if the current unit is being constructed.

This is mostly applicable to Terran structures which require an SCV to be constructing a structure.

Returns `true` if this is either a Protoss structure, Zerg structure, or Terran structure being constructed by an attached SCV, and `false` if this is either completed, not a structure, or has no SCV constructing it.

Return type boolean

See also:

build(), *cancelConstruction()*, *haltConstruction()*, *isConstructing()*

isBeingGathered() → bool
Checks this *Mineral Field* or *Refinery* is currently being gathered from.

Returns `true` if this unit is a resource container and being harvested by a worker, and `false` otherwise

Return type boolean

isBeingHealed() → bool
Checks if this unit is currently being healed by a *Medic* or repaired by a *SCV*.

Returns `true` if this unit is being healed, and `false` otherwise.

Return type boolean

isBlind() → bool
Checks if this unit is currently blinded by a *Medic* 's *Optical Flare* ability.

Blinded units have reduced sight range and cannot detect other units.

Returns `true` if this unit is blind, and `false` otherwise

Return type boolean

isBraking() → bool
Checks if the current unit is slowing down to come to a stop.

Returns `true` if this unit is breaking, `false` if it has stopped or is still moving at full speed.

Return type boolean

isBurrowed() → bool
Checks if the current unit is burrowed, either using the *Burrow* ability, or is an armed *Spider Mine*.

Returns `true` if this unit is burrowed, and `false` otherwise

Return type boolean

See also:

burrow(), *unburrow()*

isCarryingGas() → bool
Checks if this worker unit is carrying some vespene gas.

Returns `true` if this is a worker unit carrying vespene gas, and `false` if it is either not a worker, or not carrying gas.

Return type boolean

Listing 8.24: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  if u:isIdle() and (u:isCarryingGas() or u:isCarryingMinerals()) then
    u:returnCargo()
  end
end
```

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*, `getType():isWorker()`

See also:

returnCargo(), *isGatheringGas()*, *isCarryingMinerals()*

isCarryingMinerals() → bool
Checks if this worker unit is carrying some minerals.

Returns `true` if this is a worker unit carrying minerals, and `false` if it is either not a worker, or not carrying minerals.

Return type boolean

Listing 8.25: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  if u:isIdle() and (u:isCarryingGas() or u:isCarryingMinerals()) then
    u:returnCargo()
  end
end
```

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*, `getType():isWorker()`

See also:

returnCargo(), *isGatheringMinerals()*, *isCarryingGas()*

isCloaked() → bool
Checks if this unit is currently *cloaked*.

Returns `true` if this unit is cloaked, and `false` if it is visible.

Return type boolean

See also:

cloak(), *decloak()*

isCompleted() → bool
Checks if this unit has finished being constructed, trained, morphed, or warped in, and can now receive orders.

Returns `true` if this unit is completed, and `false` if it is under construction or inaccessible.

Return type boolean

isConstructing() → bool
Checks if a unit is either constructing something or moving to construct something.

Returns `true` when a unit has been issued an order to build a structure and is moving to the build location, or is currently constructing something.

Return type boolean

See also:

isBeingConstructed(), *build()*, *cancelConstruction()*, *haltConstruction()*

isDefenseMatrixed() → bool
Checks if this unit has the *Defensive Matrix* effect.

Returns `true` if the *Defensive Matrix* ability was used on this unit, and `false` otherwise.

Return type boolean

isDetected() → bool
Checks if this unit is visible or revealed by a detector unit.

If this is `false` and isVisible is `true`, then the unit is only partially visible and requires a detector in order to be targetted.

Returns `true` if this unit is detected, and `false` if it needs a detector unit nearby in order to see it.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isVisible()*

---

isEnsnared() → bool
Checks if the *Queen* ability *Ensnare* has been used on this unit.

Returns `true` if the unit is ensnared, and `false` if it is not

Return type boolean

isFlying() → bool
This macro function checks if this unit is in the air.

That is, the unit is either a flyer or a flying building.

Returns `true` if this unit is in the air, and `false` if it is on the ground

Return type boolean

See also:

UnitType.isFlyer(), *isLifted()*

isFollowing() → bool
Checks if this unit is following another unit.

When a unit is following another unit, it simply moves where the other unit does, and does not attack enemies when it is following.

Returns `true` if this unit is following another unit, and `false` if it is not

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*

See also:

follow(), *getTarget()*

isGatheringGas() → bool
Checks if this unit is currently gathering gas.

That is, the unit is either moving to a refinery, waiting to enter a refinery, harvesting from the refinery, or returning gas to a resource depot.

Returns `true` if this unit is harvesting gas, and `false` if it is not

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*, `getType():isWorker()`

See also:

isCarryingGas()

isGatheringMinerals() → bool
Checks if this unit is currently harvesting minerals.

That is, the unit is either moving to a *Mineral Field*, waiting to mine, mining minerals, or returning minerals to a resource depot.

Returns `true` if this unit is gathering minerals, and `false` if it is not

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*, `getType():isWorker()`

See also:

isCarryingMinerals()

isHallucination() → bool
Checks if this unit is a hallucination.

Hallucinations are created by the *High Templar* using the *Hallucination* ability. Enemy hallucinations are unknown if *BWAPI.Flag.CompleteMapInformation* is disabled. Hallucinations have a time limit until they are destroyed (see *getRemoveTimer()*).

Returns `true` if the unit is a hallucination and `false` otherwise.

Return type boolean

See also:

getRemoveTimer()

isHoldingPosition() → bool
Checks if the unit is currently holding position.

A unit that is holding position will attack other units, but will not chase after them.

Returns `true` if this unit is holding position, and `false` if it is not.

Return type boolean

See also:

holdPosition()

isIdle() → bool
Checks if this unit is running an idle order.

This function is particularly useful when checking for units that aren't doing any tasks that you assigned.

A unit is considered idle if it is **not** doing any of the following:

- Training
- Constructing
- Morphing
- Researching
- Upgrading

In **addition** to running one of the following orders:

- *PlayerGuard*: Player unit idle.
- *Guard*: Generic unit idle.
- *Stop*
- *PickupIdle*
- *Nothing*: Structure/generic idle.
- *Medic*: Medic idle.
- *Carrier*: Carrier idle.
- *Reaver*: Reaver idle.
- *Critter*: Critter idle.
- *Neutral*: Neutral unit idle.
- *TowerGuard*: Turret structure idle.
- *Burrowed*: Burrowed unit idle.

- *NukeTrain*
- *Larva*: Larva idle.

Listing 8.26: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  -- Order idle worker to gather from closest mineral field
  if u:getType():isWorker() and u:isIdle() then
    u:gather( u:getClosestUnit( BWAPI.Filter.IsMineralField ) )
  end
end
```

Returns `true` if this unit is idle, and `false` if this unit is performing any action, such as moving or attacking

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*

See also:

stop()

isInterruptible() → bool
Checks if the unit can be interrupted.

Returns `true` if this unit can be interrupted, or `false` if this unit is uninterruptable

Return type boolean

isInvincible() → bool
Checks the invincibility state for this unit.

Returns `true` if this unit is currently invulnerable, and `false` if it is vulnerable

Return type boolean

isInWeaponRange(*target*) → bool
Checks if the target unit can immediately be attacked by this unit in the current frame.

Parameters **target** (BWAPI.Unit) – The target unit to use in this check.

Returns `true` if `target` is within weapon range of this unit's appropriate weapon, and `false` if `target` is invalid, inaccessible, too close, too far, or this unit does not have a weapon that can attack `target`.

Return type boolean

isIrradiated() → bool
Checks if this unit is irradiated by a *Science Vessel* 's *Irradiate* ability.

Listing 8.27: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  if u:isIrradiated() and u:getIrradiateTimer > 50 and BWAPI.
↪Broodwar:self():hasResearched(BWAPI.TechTypes.Restoration) then
    local medicPred = function(x)
```

```
      return BWAPI.Filter.GetType(x) == BWAPI.UnitTypes.Terran_Medic and
↪BWAPI.Filter.Energy(x) >= BWAPI.TechTypes.Restoration:energyCost()
    end
    local medic = u:getClosestUnit( medicPred )
    if medic then
      medic:useTech(BWAPI.TechTypes.Restoration, u)
    end
  end
end
```

Returns `true` if this unit is irradiated, and `false` otherwise

Return type boolean

See also:

getIrradiateTimer()

isLifted() → bool
Checks if this unit is a *Terran* building and lifted off the ground.

This function generally implies `getType():isBuilding()` and `isCompleted()` both return true.

Returns `true` if this unit is a *Terran* structure lifted off the ground.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*, `getType():isFlyingBuilding()`

See also:

isFlying()

isLoaded() → bool
Checks if this unit is currently loaded into another unit such as a Transport (*Dropship*, *Shuttle*, *Overlord*).

Returns `true` if this unit is loaded in another one, and `false` otherwise

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: *isCompleted()*

See also:

load(), *unload()*, *unloadAll()*

isLockedDown() → bool
Checks if this unit is currently *locked* by a *Ghost*.

Returns `true` if this unit is locked down, and `false` otherwise

Return type boolean

See also:

getLockdownTimer()

isMaelstrommed() → bool
Checks if this unit has been *maelstrommed* by a *Dark Archon*.

Returns `true` if this unit is maelstrommed, and `false` otherwise

Return type boolean

See also:

getMaelstromTimer()

isMorphing() → bool
Finds out if the current unit is morphing or not.

Zerg units and structures often have the ability to morph into different types of units. This function allows you to identify when this process is occurring.

Returns `true` if the unit is currently morphing, and `false` if the unit is not morphing.

Return type boolean

See also:

morph(), *cancelMorph()*, *getBuildType()*, *getRemainingBuildTime()*

isMoving() → bool
Checks if this unit is currently moving.

Returns `true` if this unit is moving, and `false` if it is not

Return type boolean

See also:

stop()

isParasited() → bool
Checks if this unit has been parasited by some other player.

Returns `true` if this unit is inflicted with *Parasite*, and `false` if it is clean

Return type boolean

isPatrolling() → bool
Checks if this unit is patrolling between two positions.

Returns `true` if this unit is patrolling and `false` if it is not

Return type boolean

See also:

patrol()

isPlagued() → bool
Checks if this unit has been been *plagued* by a *Defiler*.

Returns `true` if this unit is inflicted with *Plague* and is taking damage, and `false` if it is clean

Return type boolean

See also:

getPlagueTimer()

isRepairing() → bool
Checks if this unit is repairing or moving to *repair* another unit.

This is only applicable to *SCVs*.

Returns `true` if this unit is currently repairing or moving to *repair* another unit, and `false` if it is not

Return type boolean

isResearching() → bool
Checks if this unit is a structure that is currently researching a technology.

See *TechTypes* for a complete list of technologies in Broodwar.

Returns `true` if this structure is researching a technology, `false` otherwise

Return type boolean

See also:

research(), *cancelResearch()*, *getTech()*, *getRemainingResearchTime()*

Note: If this function returns a successful state, then the following function calls will also return a successful state: *exists()*, *isCompleted()*, `getType():isBuilding()`

---

isSelected() → bool
Checks if this unit has been selected in the user interface.

This function is only available if the flag *BWAPI.Flag.UserInput* is enabled.

Returns `true` if this unit is currently selected, and `false` if this unit is not selected

Return type boolean

See also:

Game.getSelectedUnits()

isSieged() → bool
Checks if this unit is currently *sieged*.

This is only applicable to *Siege Tanks*.

Returns `true` if the unit is in siege mode, and `false` if it is either not in siege mode or not a *Siege Tank*

Return type boolean

See also:

siege(), *unsiege()*

isStartingAttack() → bool
Checks if the unit is starting to attack.

Returns `true` if this unit is starting an attack.

Return type boolean

See also:

attack(), *getGroundWeaponCooldown()*, *getAirWeaponCooldown()*

isStasised() → bool
Checks if this unit is inflicted with *Stasis Field* by an *Arbiter*.

Returns `true` if this unit is locked in a *Stasis Field* and is unable to move, and `false` if it is free.

Return type boolean

Note: This function does not necessarily imply that the unit is invincible, since there is a feature in the *Use Map Settings* game type that allows stasised units to be vulnerable.

See also:

getStasisTimer()

isStimmed() → bool
Checks if this unit is currently under the influence of a *Stim Packs*.

Returns `true` if this unit has used a stim pack, `false` otherwise

Return type boolean

See also:

getStimTimer()

isStuck() → bool
Checks if this unit is currently trying to resolve a collision by randomly moving around.

Returns `true` if this unit is currently stuck and trying to resolve a collision, and `false` if this unit is free

Return type boolean

isTraining() → bool
Checks if this unit is training a new unit.

For example, a *Barracks* training a *Marine*.

Note: It is possible for a unit to remain in the training queue with no progress. In that case, this function will return `false` because of supply or unit count limitations.

Returns `true` if this unit is currently training another unit, and `false` otherwise.

See also:

train(), *getTrainingQueue()*, *cancelTrain()*, *getRemainingTrainTime()*

isUnderAttack() → bool
Checks if the current unit is being attacked.

Has a small delay before this returns `false` again when the unit is no longer being attacked.

Returns `true` if this unit has been attacked within the past few frames, and `false` if it has not

Return type boolean

isUnderDarkSwarm() → bool
Checks if this unit is under the cover of a *Dark Swarm*.

Returns `true` if this unit is protected by a *Dark Swarm*, and `false` if it is not

Return type boolean

isUnderDisruptionWeb() → bool
Checks if this unit is currently being affected by a *Disruption Web*.

Returns `true` if this unit is under the effects of *Disruption Web*.

Return type boolean

isUnderStorm() → bool
Checks if this unit is currently taking damage from a *Psionic Storm*.

Returns `true` if this unit is losing hit points from a *Psionic Storm*, and `false` otherwise.

Return type boolean

isPowered() → bool
Checks if this unit has power.

Most structures are powered by default, but *Protoss* structures require a *Pylon* to be powered and functional.

Returns `true` if this unit has power or is inaccessible, and `false` if this unit is unpowered.

Return type boolean

isUpgrading() → bool
Checks if this unit is a structure that is currently upgrading an upgrade.

See *BWAPI.UpgradeTypes* for a full list of upgrades in Broodwar.

Returns `true` if this structure is upgrading, `false` otherwise

Return type boolean

See also:

upgrade(), *cancelUpgrade()*, *getUpgrade()*, *getRemainingUpgradeTime()*

Note: If this function returns a successful state, then the following function calls will also return a successful state: *exists()*, *isCompleted()*, `getType():isBuilding()`

---

isVisible([*player*]) → bool
Checks if this unit is visible.

Parameters **player** (`BWAPI.Player`) – (optional) The player to check visibility for. If this parameter is omitted, then the BWAPI player obtained from *Game.self()* will be used.

Returns `true` if this unit is visible to the specified `player`, and `false` if it is not.

Return type boolean

Note: If the *BWAPI.Flag.CompleteMapInformation* flag is enabled, existing units hidden by the fog of war will be accessible, but *isVisible()* will still return false.

See also:

exists()

isTargetable() → bool
Performs some cheap checks to attempt to quickly detect whether the unit is unable to be targetted as the target unit of an unspecified command.

Returns `true` if BWAPI was unable to determine whether the unit can be a target, and `false` if an error occurred and the unit can not be a target.

Return type boolean

See also:

Game.getLastError(), *canTargetUnit()*

issueCommand(*command*) → wasIssued

This function issues a command to the unit, however it is used for interfacing only, and is recommended to use one of the more specific command functions when writing an AI.

Parameters **command** (BWAPI.UnitCommand) – The command to be issued

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *Unit.canIssueCommand()*

attack(*target*[, *shiftQueueCommand* = *false*]) → wasIssued

Orders the unit(s) to attack move to the specified position or attack the specified unit.

Parameters

- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. If a Position is used, the unit will perform an Attack Move command.
- **shiftQueueCommand** (*bool*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

A *Medic* will use Heal Move instead of attack.

See also:

Game.getLastError(), *Unit.canAttack()*

build(*type*[, *target*]) → wasIssued

Orders the worker unit(s) to construct a structure at a target position.

Parameters

- **type** (BWAPI.UnitType) – The *UnitType* to build.
- **target** (BWAPI.TilePosition) – A TilePosition to specify the build location, specifically the upper-left corner of the location. If the target is not specified, then the function call will be redirected to the train command.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources and meet the necessary requirements in order to build a structure.

See also:

Game.getLastError(), *train()*, *cancelConstruction()*, *canBuild()*

buildAddon(*type*) → wasIssued

Orders the *Terran* structure(s) to construct an add-on.

Parameters **type** (BWAPI.UnitType) – The add-on *UnitType* to construct.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources and meet the necessary requirements in order to build a structure.

See also:

Game.getLastError(), *build()*, *cancelAddon()*, *canBuildAddon()*

train([*type*]) → wasIssued

Orders the unit(s) to add a *UnitType* to its training queue, or morphs into the *UnitType* if it is *Zerg*.

Parameters **type** (BWAPI.UnitType) – The *UnitType* to train. If the type is not specified, then this function will have no effect unless this unit is a Carrier or Reaver, in which case the type will default to *Interceptors* or *Scarabs*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources, supply, and meet the necessary requirements in order to train a unit.

Note: This command is also used for training *Interceptors* and *Scarabs*.

Note: If you call this using a *Hatchery*, *Lair*, or *Hive*, then it will automatically pass the command to one of its *Larvae*.

See also:

Game.getLastError(), *build()*, *morph()*, *cancelTrain()*, *isTraining()*, *canTrain()*

morph (*type*) → wasIssued
Orders the unit(s) to morph into a different *UnitType*.

Parameters **type** (BWAPI.UnitType) – The *UnitType* to morph into.

Returns true if the command was passed to Broodwar, and false if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

Game.getLastError(), *build()*, *morph()*, *canMorph()*

research (*tech*) → wasIssued
Orders the unit to research the given tech type.

Parameters **tech** (BWAPI.TechType) – The *TechType* to research.

Returns true if the command was passed to Broodwar, and false if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

cancelResearch(), *isResearching()*, *getRemainingResearchTime()*, *getTech()*, *canResearch()*

upgrade (*upgrade*) → wasIssued
Orders the unit to upgrade the given upgrade type.

Parameters **upgrade** (BWAPI.UpgradeType) – The *UpgradeType* to upgrade.

Returns true if the command was passed to Broodwar, and false if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

cancelUpgrade(), *isUpgrading()*, *getRemainingUpgradeTime()*, *getUpgrade()*, *canUpgrade()*

setRallyPoint (*target*) → wasIssued
Orders the unit to set its rally position to the specified position or unit.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. The target position or target unit that this structure will rally to.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

getRallyPosition(), *getRallyUnit()*, *canSetRallyPoint()*, *canSetRallyPosition()*, *canSetRallyUnit()*

move (*target*[, *shiftQueueCommand* = *false*]) → wasIssued
Orders the unit to move from its current position to the specified position.

Parameters

- **target** (BWAPI.Position) – The target position to move to.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is `true`, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isMoving(), *canMove()*

patrol (*target*[, *shiftQueueCommand* = *false*]) → wasIssued
Orders the unit to patrol between its current position and the specified position.

While patrolling, units will attack and chase enemy units that they encounter, and then return to its patrol route. *Medics* will automatically heal units and then return to their patrol route.

Parameters

- **target** (BWAPI.Position) – The target position to patrol to.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is `true`, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isPatrolling(), *canPatrol()*

holdPosition(*[shiftQueueCommand = false]*) → wasIssued
Orders the unit to hold its position.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

canHoldPosition(), *isHoldingPosition()*

stop(*[shiftQueueCommand = false]*) → wasIssued
Orders the unit to stop.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

canStop(), *isIdle()*

follow(*target[, shiftQueueCommand = false]*) → wasIssued
Orders the unit to follow the specified unit.

Units that are following other units will not perform any other actions such as attacking. They will ignore attackers.

Parameters

- **target** (BWAPI.Unit) – The target unit to start following.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isFollowing(), *canFollow()*, *getOrderTarget()*

gather (*target*[, *shiftQueueCommand* = *false*]) → wasIssued
Orders the unit to gather the specified unit (must be mineral or refinery type).

Parameters

- **target** (BWAPI.Unit) – The target unit to gather from.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isGatheringGas(), *isGatheringMinerals()*, *canGather()*

returnCargo ([*shiftQueueCommand* = *false*]) → wasIssued
Orders the unit to return its cargo to a nearby resource depot such as a Command Center.

Only workers that are carrying minerals or gas can be ordered to return cargo.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isCarryingGas(), *isCarryingMinerals()*, *canReturnCargo()*

repair (*target*[, *shiftQueueCommand* = *false*]) → wasIssued
Orders the unit to repair the specified unit.

Only Terran SCVs can be ordered to repair, and the target must be a mechanical *Terran* unit or building.

Parameters

- **target** (BWAPI.Unit) – The target unit to repair.

- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isRepairing(), *canRepair()*

burrow() → wasIssued
Orders the unit to burrow.

Either the unit must be a *Lurker*, or the unit must be a *Zerg* ground unit that is capable of *Burrowing*, and *Burrow* technology must be researched.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

unburrow(), *isBurrowed()*, *canBurrow()*

unburrow() → wasIssued
Orders a burrowed unit to unburrow.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

burrow(), *isBurrowed()*, *canUnburrow()*

cloak() → wasIssued
Orders the unit to cloak.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

decloak(), *isCloaked()*, *canCloak()*

decloak() → wasIssued
Orders a cloaked unit to decloak.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

cloak(), *isCloaked()*, *canDecloak()*

siege() → wasIssued
Orders the unit to siege.

Only works for *Siege Tanks*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

unsiege(), *isSieged()*, *canSiege()*

unsiege() → wasIssued
Orders the unit to unsiege.

Only works for sieged *Siege Tanks*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

siege(), *isSieged()*, *canUnsiege()*

lift() → wasIssued
Orders the unit to lift.

Only works for liftable *Terran* structures.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

land(), *isLifted()*, *canLift()*

land(*target*) → wasIssued
Orders the unit to land.

Only works for *Terran* structures that are currently lifted.

Parameters **target``** (`BWAPI.TilePosition`) – The tile position to land this structure at.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

lift(), *isLifted()*, *canLand()*

load(*target*[, *shiftQueueCommand* = *false*]) → wasIssued

Orders the unit to load the target unit.

Only works if this unit is a Transport (*Dropship*, *Shuttle*, *Overlord*) or *Bunker* type.

Parameters

- **target** (BWAPI.Unit) – The target unit to load into this Transport (*Dropship*, *Shuttle*, *Overlord*) or *Bunker*.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

unload(), *unloadAll()*, *getLoadedUnits()*, *isLoaded()*

unload(*target*) → wasIssued

Orders the unit to unload the target unit.

Only works for Transports (*Dropships*, *Shuttles*, *Overlords*) and *Bunkers*.

Parameters **target** (BWAPI.Unit) – Unloads the target unit from this Transport(*Dropship*, *Shuttle*, *Overlord*) or *Bunker*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

load(), *unloadAll()*, *getLoadedUnits()*, *isLoaded()*, *canUnload()*, *canUnloadAtPosition()*

unloadAll([*shiftQueueCommand* = *false*]) → wasIssued

Orders the unit to unload all loaded units at the unit's current position.

Only works for Transports (*Dropships*, *Shuttles*, *Overlords*) and *Bunkers*.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

load(), *unload()*, *getLoadedUnits()*, *isLoaded()*, *canUnloadAll()*, *canUnloadAtPosition()*

unloadAll (*target*[, *shiftQueueCommand* = *false*]) → wasIssued

Orders the unit to unload all loaded units at the specified location.

Only works for Transports (*Dropships*, *Shuttles*, *Overlords*)

Important: Not applicable to *Bunkers*.

Parameters

- **target** (BWAPI.Position) – The target position to unload the units at.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

load(), *unload()*, *getLoadedUnits()*, *isLoaded()*, *canUnloadAllPosition()*, *canUnloadAtPosition()*

rightClick (*target*[, *shiftQueueCommand* = *false*]) → wasIssued

Works like the right click in the GUI.

Parameters

- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. The target position or target unit to right click.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

canRightClick(), *canRightClickPosition()*, *canRightClickUnit()*

haltConstruction() → wasIssued
Orders a *SCV* to stop constructing a structure.

This leaves the structure in an incomplete state until it is either cancelled, razed, or completed by another *SCV*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isConstructing(), *canHaltConstruction()*

cancelConstruction() → wasIssued
Orders this unit to cancel and refund itself from begin constructed.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

isBeingConstructed(), *build()*, *canCancelConstruction()*

cancelAddon() → wasIssued
Orders this unit to cancel and refund an add-on that is being constructed.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

canCancelAddon(), *buildAddon()*

cancelTrain([*slot* = *-2*]) → wasIssued
Orders the unit to remove the specified unit from its training queue.

Important: The first `slot` has an index of `1`. See *the differences between the C++ and Lua implementations of this function* for more information.

Parameters **`slot`** (*`number`*) – (optional) Identifies the slot that will be cancelled. If the specified value is at least 1, then the unit in the corresponding slot from the list provided by *`getTrainingQueue()`* will be cancelled. If the slot is either omitted or `-2`, then the last slot is cancelled.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: Zero and negative values other than `-2` have no effect.

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

`train()`, *`cancelTrain()`*, *`isTraining()`*, *`getTrainingQueue()`*, *`canCancelTrain()`*, *`canCancelTrainSlot()`*

`cancelMorph``()` → wasIssued

Orders this unit to cancel and refund a unit that is morphing.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

`morph()`, *`isMorphing()`*, *`canCancelMorph()`*

`cancelResearch``()` → wasIssued

Orders this unit to cancel and refund a research that is in progress.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

`research()`, *`isResearching()`*, *`getTech()`*, *`canCancelResearch()`*

`cancelUpgrade``()` → wasIssued

Orders this unit to cancel and refund an upgrade that is in progress.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

upgrade(), *isUpgrading()*, *getUpgrade()*, *canCancelUpgrade()*

useTech (*tech*[, *target*]) → wasIssued

Orders the unit to use a technology.

Parameters

- **tech** (BWAPI.TechType) – The technology type to use.
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – (optional) Can be either a *Position* or *Unit*. If specified, indicates the target location or unit to use the tech on. If unspecified, causes the `tech` to be used without a target (i.e. *Stim Packs*).

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

See also:

canUseTechWithOrWithoutTarget(), *canUseTech()*, *canUseTechWithoutTarget()*, *canUseTechUnit()*, *canUseTechPosition()*, *BWAPI.TechTypes*

placeCOP (*target*) → bool

Moves a *Flag Beacon* to a different location.

This is only used for *Capture The Flag* or *Use Map Settings* game types.

Parameters **target** (BWAPI.TilePosition) – The target tile position to place the *Flag Beacon*.

Returns `true` if the command was passed to Broodwar, and `false` if BWAPI determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: This command is only available for the first 10 minutes of the game, as in Broodwar.

See also:

canPlaceCOP

canIssueCommand (*command*[, *checkCanUseTechPositionOnPositions = true*, *checkCanUseTechUnitOnUnits = true*, *checkCanBuildUnitType = true*, *checkCanTargetUnit = true*, *checkCanIssueCommandType = true*, *checkCommandibility = true*]) → bool

Checks whether the unit is able to execute the given command.

If you are calling this function repeatedly (e.g. to generate a collection of valid commands), you can avoid repeating the same kinds of checks by specifying `false` for some of the optional boolean arguments.

Make sure that the state hasn't changed since the check was done though (eg a new frame/event, or a command issued). Also see the more specific functions.

Parameters

- **command** (BWAPI.UnitCommand) – A *UnitCommand* to check.
- **checkCanUseTechPositionOnPositions** (*boolean*) – Only used if the command type is *BWAPI.UnitCommandTypes.Use_Tech_Position*. A boolean for whether to perform cheap checks for whether the unit is unable to target any positions using the command's *TechType* (i.e. regardless of what the other command parameters are). You can set this to `false` if you know this check has already just been performed.
- **checkCanUseTechUnitOnUnits** (*boolean*) – Only used if the command type is *BWAPI.UnitCommandTypes.Use_Tech_Unit*. A boolean for whether to perform cheap checks for whether the unit is unable to target any units using the command's *TechType* (i.e. regardless of what the other command parameters are). You can set this to `false` if you know this check has already just been performed.
- **checkCanBuildUnitType** (*boolean*) – Only used if the command type is *BWAPI.UnitCommandTypes.Build*. A boolean for whether to perform cheap checks for whether the unit is unable to build the specified *UnitType* (i.e. regardless of what the other command parameters are). You can set this to `false` if you know this check has already just been performed.
- **checkCanTargetUnit** (*boolean*) – Only used for command types that can target a unit. A boolean for whether to perform *canTargetUnit()* as a check. You can set this to `false` if you know this check has already just been performed.
- **checkCanIssueCommandType** (*boolean*) – A boolean for whether to perform *Unit.canIssueCommandType()* as a check. You can set this to `false` if you know this check has already just been performed.
- **checkCommandibility** (*boolean*) – A boolean for whether to perform *canCommand()* as a check. You can set this to `false` if you know this check has already just been performed.

Returns `true` if BWAPI determined that the command is valid, or `false` if an error occurred and the command is invalid

Return type boolean

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *canCommand()*, *Unit.canIssueCommandType()*, *canTargetUnit()*

canIssueCommandGrouped(*command*[, *checkCanUseTechPositionOnPositions* = *true*, *checkCanUseTechUnitOnUnits* = *true*, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool

Checks whether the unit is able to execute the given command as part of a *Unitset* (even if none of the units in the *Unitset* are able to execute the command individually).

The reason this function exists is because some commands are valid for an individual unit but not for those individuals as a group (e.g. buildings, critters) and some commands are only valid for a unit if it is commanded as part of a unit group, e.g.:

1. attackMove/attackUnit for a *Unitset*, some of which can't attack, e.g. *High Templar*. This is supported simply for consistency with BW's behaviour - you could issue move command(s) individually instead.

2.attackMove/move/patrol/rightClickPosition for air unit(s) + e.g. *Larva*, as part of the air stacking technique. This is supported simply for consistency with BW's behaviour - you could issue move/patrol/rightClickPosition command(s) for them individually instead.

Note: BWAPI allows the following special cases to command a unit individually (rather than only allowing it to be commanded as part of a *Unitset*). These commands are not available to a user in BW when commanding units individually, but BWAPI allows them for convenience:

- attackMove for *Medic*, which is equivalent to Heal Move.
- holdPosition for burrowed *Lurker*, for ambushes.
- stop for *Larva*, to move it to a different side of the *Hatchery* / *Lair* / *Hive* (e.g. so that *Drones* morphed later morph nearer to minerals/gas).

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *Unit.canIssueCommand()*, *canCommandGrouped()*, *Unit.canIssueCommandTypeGrouped()*, *canTargetUnit()*

canCommand() → bool

Performs some cheap checks to attempt to quickly detect whether the unit is unable to execute any commands (eg the unit is stasised).

Returns `true` if BWAPI was unable to determine whether the unit can be commanded, or `false` if an error occurred and the unit can not be commanded.

Return type boolean

See also:

Game.getLastError(), *Unit.canIssueCommand()*

canCommandGrouped([*checkCommandibility* = *true*]) → bool

Performs some cheap checks to attempt to quickly detect whether the unit is unable to execute any commands as part of a *Unitset* (eg buildings, critters).

Returns `true` if BWAPI was unable to determine whether the unit can be commanded grouped, or `false` if an error occurred and the unit can not be commanded grouped.

Return type boolean

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *Unit.canIssueCommand()*

canIssueCommandType(*commandType*[, *checkCommandibility* = *true*]) → bool

Performs some cheap checks to attempt to quickly detect whether the unit is unable to execute the given command type (i.e. regardless of what other possible command parameters could be).

Parameters

- **commandType** (BWAPI.UnitCommandType) – A *UnitCommandType*.
- **checkCommandibility** (*boolean*) – A boolean for whether to perform *canCommand()* as a check. You can set this to `false` if you know this check has already just been performed.

Returns `true` if BWAPI was unable to determine whether the command type is invalid, or `false` if an error occurred and the command type is invalid.

Return type boolean

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *Unit.canIssueCommand()*

canIssueCommandTypeGrouped (*commandType*[, *checkCommandibilityGrouped = true*, *checkCommandibility = true*]) → bool

Performs some cheap checks to attempt to quickly detect whether the unit is unable to execute the given command type (i.e.

regardless of what other possible command parameters could be) as part of a *Unitset*.

Parameters

- **commandType** (BWAPI.UnitCommandType) – A *UnitCommandType*.
- **checkCommandibilityGrouped** (*boolean*) – A boolean for whether to perform *canCommandGrouped()* as a check. You can set this to `false` if you know this check has already just been performed.
- **checkCommandibility** (*boolean*) – A boolean for whether to perform *canCommand()* as a check. You can set this to `false` if you know this check has already just been performed.

Returns `true` if BWAPI was unable to determine whether the command type is invalid, or `false` if an error occurred and the command type is invalid.

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *Unit.canIssueCommandGrouped()*

canTargetUnit (*targetUnit*[, *checkCommandibility = true*]) → bool

Performs some cheap checks to attempt to quickly detect whether the unit is unable to use the given unit as the target unit of an unspecified command.

Parameters

- **targetUnit** (BWAPI.Unit) – A target unit for an unspecified command.
- **checkCommandibility** (*boolean*) – A boolean for whether to perform *canCommand()* as a check. You can set this to `false` if you know this check has already just been performed.

Returns `true` if BWAPI was unable to determine whether the unit can target the given unit, or `false` if an error occurred and the unit can not target the given unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *isTargetable()*

canAttack ([*checkCommandibility = true*]) → bool

Cheap checks for whether the unit is able to execute an attack command to attack-move or attack a unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *attack()*, *canAttackMove()*, *canAttackUnit()*

canAttack (*target*[, *checkCanTargetUnit = true*, *checkCanIssueCommandType = true*, *checkCommandibility = true*]) → bool

Checks whether the unit is able to execute an attack command to attack-move or attack a (non-null) unit.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *attack()*, *canAttackMove()*, *canAttackUnit()*

canAttackGrouped([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an attack command to attack-move or attack a unit, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canAttack()*

canAttackGrouped(*target*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an attack command to attack-move or attack a (non-null) unit, as part of a *Unitset*.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canAttack()*

canAttackMove([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an attack command to attack-move.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *attack()*

canAttackMoveGrouped([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an attack command to attack-move, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canAttackMove()*

canAttackUnit([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an attack command to attack a unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *attack()*

canAttackUnit(*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an attack command to attack a unit.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *attack()*

canAttackUnitGrouped([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an attack command to attack a unit, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canAttackUnit()*

canAttackUnitGrouped(*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an attack command to attack a unit, as part of a *Unitset*.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canAttackUnit()*

canBuild([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a build command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *build()*

canBuild(*unitType*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a build command for the given *UnitType*.

Parameters **unitType** (BWAPI.UnitType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *build()*

canBuild(*unitType*, *tilePos*[, *checkTargetUnitType* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a build command.

Parameters

- **unitType** (BWAPI.UnitType) –
- **tilePos** (BWAPI.TilePosition) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *build()*

canBuildAddon([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a buildAddon command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *buildAddon()*

canBuildAddon(*unitType*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a buildAddon command.

Parameters **unitType** (BWAPI.UnitType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *buildAddon()*

canTrain([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a train command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *train()*

canTrain(*unitType*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a train command.

Parameters **unitType** (BWAPI.UnitType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *train()*

canMorph ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a morph command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *morph()*

canMorph (*unitType* [, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a morph command.

Parameters **unitType** (BWAPI.UnitType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *morph()*

canResearch ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a research command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *research()*

canResearch (*type* [, *checkCanIssueCommandType* = *true*]) → bool
Checks whether the unit is able to execute a research command.

Parameters **type** (BWAPI.TechType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *research()*

canUpgrade ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an upgrade command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *upgrade()*

canUpgrade (*type* [, *checkCanIssueCommandType* = *true*]) → bool
Checks whether the unit is able to execute an upgrade command.

Parameters **type** (BWAPI.UpgradeType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *upgrade()*

canSetRallyPoint ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a setRallyPoint command to a position or unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *setRallyPoint()*, *canSetRallyPosition()*, *canSetRallyUnit()*.

canSetRallyPoint (*target* [, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a setRallyPoint command to a position or (non-null) unit.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *setRallyPoint()*, *canSetRallyPosition()*, *canSetRallyUnit()*.

canSetRallyPosition ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a setRallyPoint command to a position.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *setRallyPoint()*

canSetRallyUnit ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a setRallyPoint command to a unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *setRallyPoint()*

canSetRallyUnit (*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a setRallyPoint command to a unit.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *setRallyPoint()*

canMove ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a move command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *move()*

canMoveGrouped ([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a move command, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canMove()*

canPatrol ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a patrol command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *patrol()*

canPatrolGrouped ([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a patrol command, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canPatrol()*

canFollow ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a follow command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *follow()*

canFollow (*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a follow command.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *follow()*

canGather ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a gather command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *gather()*

canGather (*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a gather command.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *gather()*

canReturnCargo ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a returnCargo command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *returnCargo()*

canHoldPosition ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a holdPosition command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *holdPosition()*

canStop ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a stop command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *stop()*

canRepair ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a repair command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *repair()*

canRepair (*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a repair command.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *repair()*

canBurrow ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a burrow command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *burrow()*

canUnburrow([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an unburrow command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unburrow()*

canCloak([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a cloak command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cloak()*

canDecloak([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a decloak command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *decloak()*

canSiege([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a siege command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *siege()*

canUnsiege([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an unsiege command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unsiege()*

canLift([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a lift command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *lift()*

canLand([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a land command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *land()*

canLand(*target*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a land command.

Parameters **target** (BWAPI.TilePosition) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *land()*

canLoad(*checkCommandibility* = *true*) → bool
Cheap checks for whether the unit is able to execute a load command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *load()*

canLoad(*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a load command.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *load()*

canUnloadWithOrWithoutTarget ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an unload command or unloadAll at current position command or unloadAll at a different position command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unload()*, *unloadAll()*

canUnloadAtPosition (*target*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an unload command or unloadAll at current position command or unloadAll at a different position command, for a given position.

Parameters **target** (BWAPI.Position) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unload()*, *unloadAll()*

canUnload ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an unload command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unload()*

canUnload (*targetUnit*[, *checkCanTargetUnit* = *true*, *checkPosition* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an unload command.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unload()*, *canUnloadAtPosition()*

canUnloadAll ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an unloadAll command for the current position.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unloadAll()*

canUnloadAllPosition ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute an unloadAll command for a different position.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unloadAll()*

canUnloadAllPosition (*target*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute an unloadAll command for a different position.

Parameters **target** (BWAPI.Position) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *unloadAll()*

canRightClick([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a rightClick command to a position or unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *rightClick()*, *canRightClickPosition()*, *canRightClickUnit()*.

canRightClick(*target*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a rightClick command to a position or (non-null) unit.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *rightClick()*, *canRightClickPosition()*, *canRightClickUnit()*.

canRightClickGrouped([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a rightClick command to a position or unit, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canRightClickUnit()*

canRightClickGrouped(*target*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a rightClick command to a position or (non-null) unit, as part of a *Unitset*.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canRightClickUnit()*

canRightClickPosition([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a rightClick command for a position.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *rightClick()*

canRightClickPositionGrouped([*checkCommandibilityGrouped* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a rightClick command for a position, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canRightClick()*

canRightClickUnit([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a rightClick command to a unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *rightClick()*

canRightClickUnit(*targetUnit*[, *checkCanTargetUnit* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a rightClick command to a unit.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *rightClick()*

canRightClickUnitGrouped (*[checkCommandibilityGrouped = true, checkCommandibility = true]*) → bool

Cheap checks for whether the unit is able to execute a rightClick command to a unit, as part of a *Unitset*.

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canRightClickUnit()*

canRightClickUnitGrouped (*targetUnit[, checkCanTargetUnit = true, checkCanIssueCommandType = true, checkCommandibilityGrouped = true, checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a rightClick command to a unit, as part of a *Unitset*.

Parameters **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommandGrouped()*, *canRightClickUnit()*

canHaltConstruction (*[checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a haltConstruction command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *haltConstruction()*

canCancelConstruction (*[checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a cancelConstruction command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelConstruction()*

canCancelAddon (*[checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a cancelAddon command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelAddon()*

canCancelTrain (*[checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a cancelTrain command for any slot.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelTrain()*

canCancelTrainSlot (*[checkCommandibility = true]*) → bool

Cheap checks for whether the unit is able to execute a cancelTrain command for an unspecified slot.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelTrain()*

canCancelTrainSlot (*slot[, checkCanIssueCommandType = true, checkCommandibility = true]*) → bool

Checks whether the unit is able to execute a cancelTrain command for a specified slot.

Parameters **slot** (*number*) –

Important: The first `slot` has an index of `1`. See *cancelTrain()* and *the differences between the C++ and Lua implementations of this function* for more information.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelTrain()*

canCancelMorph ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a cancelMorph command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelMorph()*

canCancelResearch ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a cancelResearch command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelResearch()*

canCancelUpgrade ([*checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a cancelUpgrade command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *cancelUpgrade()*

canUseTechWithOrWithoutTarget ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a useTech command without a target or or a useTech command with a target position or a useTech command with a target unit.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTechWithOrWithoutTarget (*tech*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a useTech command without a target or or a useTech command with a target position or a useTech command with a target unit, for a given *TechType*.

Parameters **tech** (BWAPI.TechType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTech (*tech*[, *target*, *checkCanTargetUnit* = *true*, *checkTargetsType* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a useTech command for a specified position or unit (only specify nullptr if the *TechType* does not target another position/unit).

Parameters

- **tech** (BWAPI.TechType) –
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*, *canUseTechWithoutTarget()*, *canUseTechUnit()*, *canUseTechPosition()*

canUseTechWithoutTarget (*tech*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a useTech command without a target.

Parameters **tech** (BWAPI.TechType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTechUnit (*tech*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a useTech command with an unspecified target unit.

Parameters **tech** (BWAPI.TechType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTechUnit (*tech*, *targetUnit*[, *checkCanTargetUnit* = *true*, *checkTargetsUnits* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a useTech command with a target unit.

Parameters

- **tech** (BWAPI.TechType) –
- **targetUnit** (BWAPI.Unit) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTechPosition (*tech*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a useTech command with an unspecified target position.

Parameters **tech** (BWAPI.TechType) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canUseTechPosition (*tech*, *target*[, *checkTargetsPositions* = *true*, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a useTech command with a target position.

Parameters

- **tech** (BWAPI.TechType) –
- **target** (BWAPI.Position) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *useTech()*

canPlaceCOP ([*checkCommandibility* = *true*]) → bool
Cheap checks for whether the unit is able to execute a placeCOP command.

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *placeCOP()*

canPlaceCOP (*target*[, *checkCanIssueCommandType* = *true*, *checkCommandibility* = *true*]) → bool
Checks whether the unit is able to execute a placeCOP command.

Parameters **target** (BWAPI.TilePosition) –

See also:

Game.getLastError(), *Unit.canIssueCommand()*, *placeCOP()*

registerEvent (*action*[, *condition = nil*][, *timesToRun = -1*][, *framesToCheck = 0*])
Registers an event and associates it with the current object.

Events can be used to automate tasks (like train X Marines until Y of them have been created by the given Barracks) or to create user-defined callbacks.

Parameters

- **action** (*function*) – The callback to be executed when the event conditions are true.
- **condition** (*function*) – (optional) The condition callback which will return true if the action is intended to be executed. The condition will always be true if omitted.
- **timesToRun** (*int*) – (optional) The number of times to execute the action before the event is removed. If the value is negative, then the event will never be removed. The value will be -1 if omitted, causing the event to execute until the game ends.
- **framesToCheck** (*int*) – (optional) The number of frames to skip between checks. If this value is 0, then a condition check is made once per frame. If this value is 1, then the condition for this event is only checked every other frame. This value is 0 by default, meaning the event's condition is checked every frame.

UnitCommand

class BWAPI.**UnitCommand**

Constructors

UnitCommand()

UnitCommand(*unit*, *commandType*, *target*, *x*, *y*, *extra*)

Parameters

- **unit** (BWAPI.Unit) –
- **commandType** (BWAPI.UnitCommandType) –
- **target** (BWAPI.Unit) –
- **x** (*int*) –
- **y** (*int*) –
- **extra** (*int*) –

Member Functions

getSlot() → int

Important: See *the differences between the C++ and Lua implementations of this function* for more information

Returns

Return type int

getTarget() → Unit

Returns

Return type *BWAPI.Unit*

getTargetPosition() → Position

Returns

Return type *BWAPI.Position*

getTargetTilePosition() → TilePosition

Returns

Return type *BWAPI.TilePosition*

getTechType() → TechType

Returns

Return type *BWAPI.TechType*

getType() → UnitCommandType

Returns

Return type *BWAPI.UnitCommandType*

getUnit() → Unit

Returns

Return type *BWAPI.Unit*

getUnitType() → UnitType

Returns

Return type *BWAPI.UnitType*

getUpgradeType() → UpgradeType

Returns

Return type *BWAPI.UpgradeType*

isQueued() → boolean

Returns

Return type boolean

assignTarget(*target*)

Parameters **target** (BWAPI.Position) – The new target.

Static Functions

static attack(*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static build(*unit*, *target*, *type*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.TilePosition) –
- **type** (BWAPI.UnitType) –

Returns

Return type UnitCommand

static buildAddon(*unit*, *type*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **type** (BWAPI.UnitType) –

Returns

Return type UnitCommand

static burrow(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static cancelAddon(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static cancelConstruction(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static cancelMorph(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static cancelResearch(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static **cancelTrain**(*unit*[, *slot* = *-2*]) → UnitCommand

> **Important:** See *the differences between the C++ and Lua implementations of this function* for more information

Parameters

- **unit** (BWAPI.Unit) –
- **slot** (*int*) – Identifies the slot that will be cancelled. If the specified value is at least 1, then the unit in the corresponding slot will be cancelled. If the value is either omitted or -2, then the last slot is cancelled.

Returns

Return type UnitCommand

static **cancelUpgrade**(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static **cloak**(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static **decloak**(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static **follow**(*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static **gather**(*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –

- **target** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static haltConstruction(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static holdPosition(*unit*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static land(*unit*, *target*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.TilePosition) –

Returns

Return type UnitCommand

static lift(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static load(*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static morph(*unit*, *type*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **type** (BWAPI.UnitType) –

Returns

Return type UnitCommand

static move (*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Position) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static patrol (*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Position) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static placeCOP (*unit*, *target*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.TilePosition) –

Returns

Return type UnitCommand

static repair (*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static research (*unit*, *tech*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **tech** (BWAPI.TechType) –

Returns

Return type UnitCommand

static returnCargo (*unit*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static rightClick(*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static setRallyPoint(*unit*, *target*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

Returns

Return type UnitCommand

static siege(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static stop(*unit*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static train(*unit*, *type*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **type** (BWAPI.UnitType) –

Returns

Return type UnitCommand

static unburrow(*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static unload (*unit*, *target*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Unit) –

Returns

Return type UnitCommand

static unloadAll (*unit*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static unloadAll (*unit*, *target*[, *shiftQueueCommand* = *false*]) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **target** (BWAPI.Position) –
- **shiftQueueCommand** (*boolean*) –

Returns

Return type UnitCommand

static unsiege (*unit*) → UnitCommand

Parameters **unit** (BWAPI.Unit) –

Returns

Return type UnitCommand

static upgrade (*unit*, *upgrade*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **upgrade** (BWAPI.UpgradeType) –

Returns

Return type UnitCommand

static useTech (*unit*, *tech*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **tech** (BWAPI.TechType) –

Returns

Return type UnitCommand

static useTech (*unit*, *tech*, *target*) → UnitCommand

Parameters

- **unit** (BWAPI.Unit) –
- **tech** (BWAPI.TechType) –
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*.

Returns

Return type UnitCommand

UnitCommandType

class BWAPI.**UnitCommandType**

A representation of a unit command in *BWAPI*.

This is used by bots to notify *BWAPI* which commands to use. *BWAPI* filters commands accordingly and then converts them to Broodwar commands, which differ in complexity.

See also:

BWAPI.UnitCommandTypes

Constructors

UnitCommandType ([*id = UnitCommandTypes.Enum.None*])

Expected type constructor.

If the type is an invalid type, then it becomes Unknown. A type is invalid if its value is less than 0 or greater than Unknown.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes Unknown.

Member Functions

getID () → int

Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean

Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns true if this type is valid and false otherwise.

Return type boolean

getName () → string

Returns The variable name of the type.

Return type string

UnitCommandTypeset

class `BWAPI.UnitCommandTypeset`
A container for a set of *`UnitCommandType`* objects.

Constructors

`UnitCommandTypeset``()`
Default constructor.

`UnitCommandTypeset`(*set*)
Copy constructor.

Parameters **`set`** (`BWAPI.UnitCommandTypeset`) – The UnitCommandTypeset to copy.

`UnitCommandTypeset`(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *`UnitCommandType`* are added to the set.

Parameters **`tbl`** (*`table`*) – A table containing *`UnitCommandType`* objects.

Member Functions

`iterator``()` → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

`asTable``()` → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

`count`(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

`contains`(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

UnitSizeType

class BWAPI.**UnitSizeType**
Size types are used by unit types in Broodwar to determine how much damage will be applied.

This corresponds with *DamageType* for several different damage reduction applications.

View on Starcraft Campendium (Official Website)

See also:

DamageType, *UnitType*, *UnitSizeTypes*

Inherits from BWAPI::Type< UnitSizeType, UnitSizeTypes::Enum::Unknown >

Constructors

UnitSizeType ([*id = UnitSizeTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName () → string

Returns The variable name of the type.

Return type string

UnitSizeTypeset

class `BWAPI.`**UnitSizeTypeset**
A container for a set of *UnitSizeType* objects.

Constructors

UnitSizeTypeset ()
Default constructor.

UnitSizeTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.UnitSizeTypeset) – The UnitSizeTypeset to copy.

UnitSizeTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *UnitSizeType* are added to the set.

Parameters **tbl** (*table*) – A table containing *UnitSizeType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert (*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase (*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear ()
Removes all elements from the set, leaving it with a size of 0.

eraseIf (*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

UnitType

class `BWAPI.UnitType`
The *UnitType* is used to get information about a particular type of unit, such as its cost, build time, weapon, hit points, abilities, etc.

See also:

Unit.getType(), *UnitTypes*

Constructors

UnitType ([*id = UnitTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

getID() → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid() → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

getName() → string

Returns The variable name of the type.

Return type string

abilities() → TechTypeset
Retrieves the set of abilities that this unit can use, provided it is available to you in the game.

Returns Set of *TechType*'s containing ability information.

Return type *BWAPI.TechTypeset*

acceleration() → int
Retrieves the unit's acceleration amount.

Returns How fast the unit can accelerate to its top speed.

Return type int

airWeapon() → WeaponType
Retrieves this unit type's weapon type used when attacking targets in the air.

Returns *WeaponType* used as this unit type's air weapon.

Return type *BWAPI.WeaponType*

See also:

maxAirHits(), *groundWeapon()*

armor() → int
Retrieves the default amount of armor that the unit type starts with, excluding upgrades.

Returns The amount of armor the unit type has.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

armorUpgrade() → UpgradeType
Retrieves the upgrade type used to increase the armor of this unit type.

For each upgrade, this unit type gains +1 additional armor.

Returns *UpgradeType* indicating the upgrade that increases this unit type's armor amount.

Return type *BWAPI.UpgradeType*

buildScore() → int
Retrieves the amount of score points awarded for constructing this unit type.

This value is used for calculating scores in the post-game score screen.

Returns Number of points awarded for constructing this unit type.

Return type int

See also:

destroyScore()

buildsWhat() → UnitTypeset
Retrieves the set of units that this unit type is capable of creating.

This includes training, constructing, warping, and morphing.

Returns *UnitTypeset* containing the units it can build.

Return type *BWAPI.UnitTypeset*

Note: Some maps have special parameters that disable construction of units that are otherwise normally available. Use *Player.isUnitAvailable()* to determine if a unit type is actually available in the current game for a specific player.

See also:

Player.isUnitAvailable()

buildTime() → int
Retrieves the default time, in frames, needed to train, morph, or build the unit.

Returns Number of frames needed in order to build the unit.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

See also:

Unit.getRemainingBuildTime()

canAttack() → boolean
Checks if this unit is capable of attacking.

Returns true if this unit type is capable of damaging other units with a standard attack, and false otherwise.

Return type boolean

Note: This function returns false for units that can only inflict damage via special abilities, such as the *High Templar*.

canBuildAddon() → boolean
Checks if this unit type is capable of constructing an add-on.

An add-on is an extension or attachment for *Terran* structures, specifically the *Command Center*, *Factory*, *Starport*, and *Science Facility*.

Returns true if this unit type can construct an add-on, and false if it can not.

Return type boolean

See also:

isAddon()

canMove() → boolean
Checks if this unit type is capable of movement.

Returns true if this unit can use a movement command, and false if they cannot move.

Return type boolean

Note: Buildings will return false, including *Terran* liftable buildings which are capable of moving when lifted.

canProduce() → boolean
Determines if a unit can train other units.

For example, `BWAPI.UnitTypes.Terran_Barracks:canProduce()` will return `true`, while `BWAPI.UnitTypes.Terran_Marine:canProduce()` will return `false`. This is also true for two non-structures: *Protoss_Carrier* (can produce interceptors) and *Protoss_Reaver* (can produce scarabs).

Returns true if this unit type can have a production queue, and false otherwise.

Return type boolean

cloakingTech() → TechType
Retrieves the cloaking technology associated with certain units.

Returns *TechType* referring to the cloaking technology that this unit type uses as an ability. Returns `TechTypes.None` if this unit type does not have an active cloak ability.

Return type *BWAPI.TechType*

destroyScore() → int
Retrieves the amount of score points awarded for killing this unit type.

This value is used for calculating scores in the post-game score screen.

Returns Number of points awarded for killing this unit type.

Return type int

See also:

buildScore()

dimensionDown() → int
Retrieves the distance from the center of the unit type to its bottom edge.

Returns Distance to this unit type's bottom edge from its center, in pixels.

Return type int

dimensionLeft() → int
Retrieves the distance from the center of the unit type to its left edge.

Returns Distance to this unit type's left edge from its center, in pixels.

Return type int

dimensionRight() → int
Retrieves the distance from the center of the unit type to its right edge.

Returns Distance to this unit type's right edge from its center, in pixels.

Return type int

dimensionUp() → int
Retrieves the distance from the center of the unit type to its top edge.

Returns Distance to this unit type's top edge from its center, in pixels.

Return type int

gasPrice() → int
Retrieves the default vespene gas price of purchasing the unit.

Returns Vespene gas cost of the unit.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

---

getRace() → Race
Retrieves the *Race* that the unit type belongs to.

Returns *Race* indicating the race that owns this unit type. Returns `Race.None` indicating that the unit type does not belong to any particular race (a critter for example).

Return type *BWAPI.Race*

groundWeapon() → WeaponType
Retrieves this unit type's weapon type used when attacking targets on the ground.

Returns *WeaponType* used as this unit type's ground weapon.

Return type *BWAPI.WeaponType*

See also:

maxGroundHits(), *airWeapon()*

haltDistance() → int
Retrieves the unit's halting distance.

This determines how fast a unit can stop moving.

Returns A halting distance value.

Return type int

hasPermanentCloak() → boolean
Checks if this unit type is permanently cloaked.

This means the unit type is always cloaked and requires a detector in order to see it.

Returns true if this unit type is permanently cloaked, and false otherwise.

Return type boolean

height() → int
A macro for retrieving the height of the unit type, which is calculated using dimensionUp + dimensionDown + 1.

Returns Height of the unit, in pixels.

Return type int

isAddon() → boolean
Checks if this unit is an add-on.

Add-ons are attachments used by some *Terran* structures such as the *Comsat Station*.

Returns true if this unit is an add-on, and false otherwise.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `getRace() == BWAPI.Races.Terran`, `isBuilding()`

---

isBeacon() → boolean
Checks if this unit type is a beacon.

Each race has exactly one beacon each. They are *UnitTypes.Special_Zerg_Beacon*, *UnitTypes.Special_Terran_Beacon*, and *UnitTypes.Special_Protoss_Beacon*.

Returns true if this unit type is one of the three race beacons, and false otherwise.

Return type boolean

See also:

isFlagBeacon()

isBuilding() → boolean
Checks if this unit is a structure.

This includes *Mineral Fields* and *Vespene Geysers*.

Returns true if this unit is a building, and false otherwise.

Return type boolean

isBurrowable() → boolean
Checks if this unit type has the capability to use the *Burrow* technology when it is researched.

Returns true if this unit can use the *Burrow* ability, and false otherwise.

Return type boolean

Note: The *~BWAPI.UnitTypes.Zerg_Lurker* can burrow even without researching the ability.

See also:

BWAPI.TechTypes.Burrowing

Note: If this function returns a successful state, then the following function calls will also return a successful state: `getRace() == BWAPI.Races.Zerg`, `not isBuilding()`, `canMove()`

isCloakable() → boolean
Checks if this unit type has the capability to use a cloaking ability when it is researched.

This applies only to *Wraiths* and *Ghosts*, and does not include units which are permanently cloaked.

Returns true if this unit has a cloaking ability, false otherwise.

Return type boolean

See also:

hasPermanentCloak(), *TechTypes.Cloaking_Field*, *TechTypes.Personnel_Cloaking*

isCritter() → boolean
Checks if this unit type is a neutral critter.

Returns true if this unit type is a critter, and false otherwise.

Return type boolean

Listing 8.28: Example usage

```
local myBasePosition = BWAPI.Position( BWAPI.
↪Broodwar:self():getStartLocation() )
local pred = function(unit) return not BWAPI.Filter.IsOwned(unit) and not␣
↪BWAPI.Filter.IsParasited(unit) end
local unitsAroundTheBase = BWAPI.Broodwar:getUnitsInRadius(myBasePosition,␣
↪1024, pred)
for u in unitsAroundTheBase:iterator() do
  if u:getType():isCritter() and not u:isInvincible() then
    local myQueenPred = function(unit) return unit:getType() == BWAPI.
↪UnitTypes.Zerg_Queen and BWAPI.Filter.IsOwned(unit) end
    local myQueen = u:getClosestUnit(myQueenPred)
    if myQueen then
      myQueen:useTech(BWAPI.TechTypes.Parasite, u)
    end
  end
end
```

isDetector() → boolean
Checks if this unit type is capable of detecting units that are cloaked or burrowed.

Returns true if this unit type is a detector by default, false if it does not have this property

Return type boolean

isFlagBeacon() → boolean
Checks if this unit type is a flag beacon.

Each race has exactly one flag beacon each. They are *UnitTypes.Special_Zerg_Flag_Beacon*, *UnitTypes.Special_Terran_Flag_Beacon*, and *UnitTypes.Special_Protoss_Flag_Beacon*. Flag beacons spawn a flag after some ARBITRARY I FORGOT AMOUNT OF FRAMES.

Returns true if this unit type is one of the three race flag beacons, and false otherwise.

Return type boolean

See also:

isBeacon()

isFlyer() → boolean
Checks if this unit type is a flying unit.

Flying units ignore ground pathing and collisions.

Returns true if this unit type is in the air by default, and false otherwise.

Return type boolean

isFlyingBuilding() → boolean
Checks if this structure has the capability to use the lift-off command.

Returns true if this unit type is a flyable building, false otherwise.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `isBuilding()`

---

isHero() → boolean
Checks if this unit type is a hero.

Heroes are types that the player cannot obtain normally, and are identified by the white border around their icon when selected with a group.

Returns true if this unit type is a hero type, and false otherwise.

Return type boolean

Note: There are two non-hero units included in this set, the *Civilian <BWAPI.UnitTypes.Terran_Civilian>* and *Dark Templar Hero <BWAPI.UnitTypes.Hero_Dark_Templar>*.

---

isInvincible() → boolean
Checks if this unit type is invincible by default.

Invincible units cannot take damage.

Returns true if this unit type is invincible, and false if it is vulnerable to attacks.

Return type boolean

isMechanical() → boolean
Checks if this unit is mechanical.

The mechanical property is required for some actions such as repair.

Returns true if this unit type has the mechanical property, and false otherwise.

Return type boolean

isMineralField() → boolean
Checks if this unit type is a mineral field and contains a resource amount.

This indicates that the unit type is either *UnitTypes.Resource_Mineral_Field*, *UnitTypes.Resource_Mineral_Field_Type_2*, or *UnitTypes.Resource_Mineral_Field_Type_3*.

Returns true if this unit type is a mineral field resource.

Return type boolean

isNeutral() → boolean
Checks if this unit type is a neutral type, such as critters and resources.

Returns true if this unit is intended to be neutral, and false otherwise.

Return type boolean

isOrganic() → boolean
Checks if this unit is an organic unit.

The organic property is required for some abilities such as *~BWAPI.TechTypes.Heal*.

Returns true if this unit type has the organic property, and false otherwise.

Return type boolean

isPowerup() → boolean
Checks if this unit type is a powerup.

Powerups can be picked up and carried by workers. They are usually only seen in campaign maps and *Capture The Flag*.

Returns true if this unit type is a powerup type, and false otherwise.

Return type boolean

isRefinery() → boolean
Checks if this unit type is a refinery.

A refinery is a structure that is placed on top of a *Vespene Geyser* . Refinery types are *BWAPI.UnitTypes.Terran_Refinery* , *BWAPI.UnitTypes.Zerg_Extractor* , and *BWAPI.UnitTypes.Protoss_Assimilator*.

Returns true if this unit type is a refinery, and false if it is not.

Return type boolean

Listing 8.29: Example

```
local myUnits = BWAPI.Broodwar:self():getUnits()
for u in myUnits:iterator() do
  if u:getType():isRefinery() then
    local workersAssigned = u.clientInfo["work"]
    if not workersAssigned or workersAssigned < 3 then
      local idleWorkerFilter = function(unit) return BWAPI.Filter.
↪IsWorker(unit) and BWAPI.Filter.IsIdle(unit) end
      local closestIdleWorker = u:getClosestUnit(idleWorkerFilter)
      if closestIdleWorker then
        -- gather from the refinery (and check if successful)
        if closestIdleWorker:gather(u) then
          -- set a back reference for when the unit is killed or re-assigned␣
↪(code not provided)
          closestIdleWorker.clientInfo['ref'] = u

          -- Increment the number of workers assigned and associate it with␣
↪the refinery
          workersAssigned = workersAssigned + 1
          u.clientInfo['work'] = workersAssigned
        end
      end
```

```
    end
  end
end
```

isResourceContainer() → boolean
Checks if this unit type is capable of storing resources such as *Mineral Fields*.

Resources are harvested from resource containers.

Returns true if this unit type may contain resources that can be harvested, false otherwise.

Return type boolean

isResourceDepot() → boolean
Checks if this unit type is a resource depot.

Resource depots must be placed a certain distance from resources. Resource depots are typically the main building for any particular race. Workers will return resources to the nearest resource depot.

Returns true if the unit type is a resource depot, false if it is not.

Return type boolean

Listing 8.30: Example

```
if BWAPI.Broodwar:self() then
  local myUnits = BWAPI.Broodwar:self():getUnits()
  for u in myUnits:iterator() do
    if u:isIdle() and u:getType():isResourceDepot() then
      u:train( u:getType():getRace():getWorker() )
   end
  end
end
```

isRobotic() → boolean
Checks if this unit is robotic.

The robotic property is applied to robotic units such as the *Protoss_Probe* which prevents them from taking damage from *BWAPI.TechTypes.Irradiate*.

Returns true if this unit type has the robotic property, and false otherwise.

Return type boolean

isSpecialBuilding() → boolean
Checks if this structure is special and cannot be obtained normally within the game.

Returns true if this structure is a special building, and false otherwise.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `isBuilding()`

---

isSpell() → boolean
Identifies if this unit type is used to complement some *abilities*.

These include *UnitTypes.Spell_Dark_Swarm*, *UnitTypes.Spell_Disruption_Web*, and *UnitTypes.Spell_Scanner_Sweep*, which correspond to *TechTypes.Dark_Swarm*, *TechTypes.Disruption_Web*, and *TechTypes.Scanner_Sweep* respectively.

Returns true if this unit type is used for an ability, and false otherwise.

Return type boolean

isSpellcaster() → boolean
Checks if this unit type has the capacity to store energy and use it for special abilities.

Returns true if this unit type generates energy, and false if it does not have an energy pool.

Return type boolean

isSuccessorOf(*unitType*) → boolean
Checks if the current type is equal to the provided type, or a successor of the provided type.

For example, a Hive is a successor of a Hatchery, since it can still research the *Burrowing* technology.

Parameters **unitType** (BWAPI.UnitType) – The unit type to check.

Returns true if this unit type is a successor of the given unit type, and false if it is not.

Return type boolean

isTwoUnitsInOneEgg() → boolean
Checks if this unit type spawns two units when being hatched from an *Zerg_Egg*.

This is only applicable to *Zerglings*.

Returns true if morphing this unit type will spawn two of them, and false if only one is spawned.

Return type boolean

isWorker() → boolean
Checks if this unit type is a worker unit.

Worker units can harvest resources and build structures. Worker unit types include the *Terran_SCV* , *Protoss_Probe*, and *Zerg_Drone*.

Returns true if this unit type is a worker, and false if it is not.

Return type boolean

maxAirHits() → int
Retrieves the maximum number of hits this unit can deal to a flying target using its air weapon.

This value is multiplied by the air weapon's damage to calculate the unit type's damage potential.

Returns Maximum number of hits given to air targets.

Return type int

See also:

airWeapon(), *maxGroundHits()*

maxEnergy() → int
Retrieves the maximum amount of energy this unit type can have by default.

Returns Integer indicating the maximum amount of energy for this unit type. Returns 0 if this unit does not gain energy for abilities.

Return type int

maxGroundHits() → int
Retrieves the maximum number of hits this unit can deal to a ground target using its ground weapon.

This value is multiplied by the ground weapon's damage to calculate the unit type's damage potential.

Returns Maximum number of hits given to ground targets.

Return type int

See also:

groundWeapon(), *maxAirHits()*

maxHitPoints() → int
Retrieves the default maximum amount of hit points that this unit type can have.

Returns Integer indicating the maximum amount of hit points for this unit type.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

---

maxShields() → int
Retrieves the default maximum amount of shield points that this unit type can have.

Returns Integer indicating the maximum amount of shield points for this unit type. Returns `0` if this unit type does not have shields.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

---

mineralPrice() → int
Retrieves the default mineral price of purchasing the unit.

Returns Mineral cost of the unit.

Return type int

Note: This value may not necessarily match the value seen in the *Use Map Settings* game type.

---

producesCreep() → boolean
Checks if this structure type produces creep.

That is, the unit type spreads creep over a wide area so that *Zerg* structures can be placed on it.

Returns true if this unit type spreads creep.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `getRace() == BWAPI.Races.Zerg`, `isBuilding()`

---

producesLarva() → boolean
Checks if this unit type produces larva.

This is essentially used to check if the unit type is a *Zerg_Hatchery*, *Zerg_Lair*, or *Zerg_Hive*.

Returns true if this unit type produces larva.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `getRace() == BWAPI.Races.Zerg`, `isBuilding()`

regeneratesHP() → boolean
Checks if this unit type can regenerate hit points.

This generally applies to *Zerg* units.

Returns true if this unit type regenerates its hit points, and false otherwise.

Return type boolean

requiredTech() → TechType
Identifies the required *TechType* in order to create certain units.

Returns *TechType* indicating the technology that must be researched in order to create this unit type. Returns `TechTypes.None` if creating this unit type does not require a technology to be researched.

Return type *BWAPI.TechType*

Note: The only unit that requires a technology is the *Zerg_Lurker*, which needs *Lurker_Aspect*.

requiredUnits() → table
Retrieves the immediate technology tree requirements to make this unit type.

Important: See *the differences between the C++ and Lua implementations of this function* for more information

Returns Table containing the number of required units of each type that are necessary to make this unit type.

Return type `table` of the format `{ [<unitTypeID>] = <howMany> }`, where `<unitTypeID>` is the integer ID/Enum of a required UnitType (equal to `UnitType:getID()`) and `<howMany>` is the required number of that unit.

Listing 8.31: Example usage

```
local scv = BWAPI.UnitTypes.SCV
local requiredUnits = scv:requiredUnits()
for unitTypeID, howMany in pairs(requiredUnits) do
  local requiredUnitType = BWAPI.UnitType(unitTypeID)
  local str = string.format("%s requires %d %s",
    tostring(scv),
    howMany,
    tostring(requiredUnitType)
  )
  print(str)
end
```

requiresCreep() → boolean
Checks if this structure must be placed on *Zerg* creep.

Returns true if this unit type requires creep, false otherwise.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `isBuilding()`, `getRace() == BWAPI.Races.Zerg`

`requiresPsi``()` → boolean
Checks if this structure is powered by a psi field.

Structures powered by psi can only be placed near a *`Protoss_Pylon`*. If the *`Protoss_Pylon`* is destroyed, then this unit will lose power.

Returns true if this unit type can only be placed in a psi field, false otherwise.

Return type boolean

Note: If this function returns a successful state, then the following function calls will also return a successful state: `isBuilding()`, `getRace() == BWAPI.Races.Protoss`

`researchesWhat``()` → TechTypeset
Retrieves the set of technologies that this unit type is capable of researching.

Returns *`TechTypeset`* containing the technology types that can be researched.

Return type *`BWAPI.TechTypeset`*

Note: Some maps have special parameters that disable certain technologies. Use *`Player.isResearchAvailable()`* to determine if a technology is actually available in the current game for a specific player.

See also:

`Player.isResearchAvailable()`

`seekRange``()` → int
Retrieves the range at which this unit type will start targeting enemy units.

Returns Distance at which this unit type begins to seek out enemy units, in pixels.

Return type int

`sightRange``()` → int
Retrieves the sight range of this unit type.

Returns Sight range of this unit type, measured in pixels.

Return type int

`size``()` → UnitSizeType
Retrieves the *`UnitSizeType`* of this unit, which is used in calculations along with weapon damage types to determine the amount of damage that will be dealt to this type.

Returns *`UnitSizeType`* indicating the conceptual size of the unit type.

Return type *`BWAPI.UnitSizeType`*

See also:

`WeaponType.damageType()`

spaceProvided() → int
Retrieves the amount of space provided by this *Terran_Bunker* or Transport(*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*) for unit transportation.

Returns The number of slots provided by this unit type.

Return type int

See also:

spaceRequired()

spaceRequired() → int
Retrieves the amount of space required by this unit type to fit inside a *Terran_Bunker* or Transport(*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*).

Returns Amount of space required by this unit type for transport. Returns `255` if this unit type can not be transported.

Return type int

See also:

spaceProvided()

supplyProvided() → int
Retrieves the amount of supply that this unit type produces for its appropriate *Race*'s supply pool.

Returns

Return type int

Note: In Starcraft programming, the managed supply values are double than what they appear in the game. The reason for this is because *Zerglings* use 0.5 visible supply.

See also:

supplyRequired(), *Player.supplyTotal()*, *Player.supplyUsed()*

supplyRequired() → int
Retrieves the amount of supply that this unit type will use when created.

It will use the supply pool that is appropriate for its *Race*.

Returns Integer containing the supply required to build this unit.

Return type int

Note: In Starcraft programming, the managed supply values are double than what they appear in the game. The reason for this is because *Zerglings* use 0.5 visible supply.

See also:

supplyProvided(), *Player.supplyTotal()*, *Player.supplyUsed()*

tileHeight() → int
Retrieves the height of this unit type, in tiles.

Used for determining the tile size of structures.

Returns Height of this unit type, in tiles.

Return type int

tileSize() → TilePosition
Retrieves the tile size of this unit type.

Used for determining the tile size of structures.

Returns TilePosition containing the width (x) and height (y) of the unit type, in tiles.

Return type *BWAPI.TilePosition*

tileWidth() → int
Retrieves the width of this unit type, in tiles.

Used for determining the tile size of structures.

Returns Width of this unit type, in tiles.

Return type int

topSpeed() → double
Retrieves this unit type's top movement speed with no upgrades.

Returns The approximate top speed, in pixels per frame, as a double. For liftable *Terran* structures, this function returns their movement speed while lifted.

Return type double

Note: That some units have inconsistent movement and this value is sometimes an approximation.

---

turnRadius() → int
Retrieves a unit's turning radius.

This determines how fast a unit can turn.

Returns A turn radius value.

Return type int

upgrades() → UpgradeTypeset
Retrieves the set of upgrades that this unit can use to enhance its fighting ability.

Returns Set of *UpgradeType*'s containing upgrade types that will impact this unit type.

Return type *BWAPI.UpgradeTypeset*

upgradesWhat() → UpgradeTypeset
Retrieves the set of upgrades that this unit type is capable of upgrading.

Returns *UpgradeTypeset* containing the upgrade types that can be upgraded.

Return type *BWAPI.UpgradeTypeset*

Note: Some maps have special upgrade limitations. Use *Player.getMaxUpgradeLevel()* to check if an upgrade is available.

See also:

Player.getMaxUpgradeLevel()

whatBuilds() → unitType, int
Obtains the source unit type that is used to build or train this unit type, as well as the amount of them that are required.

Important: See *the differences between the C++ and Lua implementations of this function* for more information

Returns Two values, where the first value is the *UnitType* that builds this unit type, and the second value is the number of those types that are required (this value is 2 for *Archons <BWAPI.UnitType.Protoss_Archon>*, and 1 for all other types). Returns `nil` if this unit type cannot be made by the player.

Return type *UnitType*, int

width() → int
A macro for retrieving the width of the unit type, which is calculated using dimensionLeft + dimensionRight + 1.

Returns Width of the unit, in pixels.

Return type int

UnitTypeset

class BWAPI.**UnitTypeset**
A container for a set of *UnitType* objects.

Constructors

UnitTypeset()
Default constructor.

UnitTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.UnitTypeset) – The UnitTypeset to copy.

UnitTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *UnitType* are added to the set.

Parameters **tbl** (*table*) – A table containing *UnitType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

Unitset

class BWAPI.**Unitset**
The *Unitset* is a container for a set of *Unit* objects.

It is typically used for groups of units instead of having to manage each Unit individually.

See also:

Unit

Constructors

Unitset ()
Default constructor.

Unitset (*set*)
Copy constructor.

Parameters **set** (BWAPI.Unitset) – The Unitset to copy.

Unitset (*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *Unit* are added to the set.

Parameters **tbl** (*table*) – A table containing *Unit* objects.

Member Functions

issueCommand (*command*) → boolean
This function issues a command to the unit(s), however it is used for interfacing only, and is recommended to use one of the more specific command functions when writing an AI.

Parameters **command** (BWAPI.UnitCommand) – A *UnitCommand* containing command parameters such as the type, position, target, etc.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

BWAPI.UnitCommandTypes, *Game.getLastError()*, *Unit.canIssueCommand()*

attack (*target*[, *shiftQueueCommand* = *false*]) → boolean

Orders the unit(s) to attack move to the specified position or attack the specified unit.

Parameters

- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. A Position or a Unit to designate as the target. If a Position is used, the unit will perform an Attack Move command.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: A *Medic* will use Heal Move instead of attack.

See also:

Game.getLastError(), *Unit.canAttack()*

build (*type*[, *target* = *TilePositions.None*]) → boolean

Orders the worker unit(s) to construct a structure at a target position.

Parameters

- **type** (BWAPI.UnitType) – The *UnitType* to build.
- **target** (BWAPI.TilePosition) – A *TilePosition* to specify the build location, specifically the upper-left corner of the location. If the target is not specified, then the function call will be redirected to the train command.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources and meet the necessary requirements in order to build a structure.

See also:

Game.getLastError(), *Unit.train()*, *Unit.cancelConstruction()*, *Unit.canBuild()*

buildAddon (*type*) → boolean

Orders the *Terran* structure(s) to construct an add-on.

Parameters **type** (BWAPI.UnitType) – The add-on *UnitType* to construct.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources and meet the necessary requirements in order to build a structure.

See also:

Game.getLastError(), *Unit.build()*, *Unit.cancelAddon()*, *Unit.canBuildAddon()*

train (*type*) → boolean

Orders the unit(s) to add a *UnitType* to its training queue, or morphs into the *UnitType* if it is *Zerg*.

Parameters **type** (BWAPI.UnitType) – The *UnitType* to train.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

Note: You must have sufficient resources, supply, and meet the necessary requirements in order to train a unit.

Note: This command is also used for training *Interceptors* and *Scarabs*.

Note: If you call this using a *Zerg_Hatchery*, *Zerg_Lair*, or *Zerg_Hive*, then it will automatically pass the command to one of its *Zerg_Larva*.

See also:

Game.getLastError(), *Unit.build()*, *Unit.morph()*, *Unit.cancelTrain()*, *Unit.isTraining()*, *Unit.canTrain()*

morph (*type*) → boolean

Orders the unit(s) to morph into a different *UnitType*.

Parameters **type** (BWAPI.UnitType) – The *UnitType* to morph into.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

Game.getLastError(), *Unit.build()*, *Unit.morph()*, *Unit.canMorph()*

setRallyPoint (*target*) → boolean

Orders the unit to set its rally position to the specified position or unit.

Parameters **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. The target position or target unit that this structure will rally to.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

getRallyPosition(), getRallyUnit(), canSetRallyPoint(), canSetRallyPosition(), canSetRallyUnit()

move (*target*[, *shiftQueueCommand* = *false*]) → boolean

Orders the unit to move from its current position to the specified position.

Parameters

- **target** (BWAPI.Position) – The target position to move to.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

patrol (*target*[, *shiftQueueCommand* = *false*]) → boolean

Orders the unit to patrol between its current position and the specified position.

While patrolling, units will attack and chase enemy units that they encounter, and then return to its patrol route. *Medics* will automatically heal units and then return to their patrol route.

Parameters

- **target** (BWAPI.Position) – The position to patrol to.

- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

holdPosition ([*shiftQueueCommand* = *false*]) → boolean

Orders the unit to hold its position.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

stop ([*shiftQueueCommand* = *false*]) → boolean

Orders the unit to stop.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

follow (*target* [, *shiftQueueCommand* = *false*]) → boolean

Orders the unit to follow the specified unit.

Units that are following other units will not perform any other actions such as attacking. They will ignore attackers.

Parameters

- **target** (BWAPI.Unit) – The target unit to start following.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

gather (*target*[, *shiftQueueCommand* = *false*]) → boolean
Orders the unit to gather the specified unit (must be mineral or refinery type).

Parameters

- **target** (BWAPI.Unit) – The target unit to gather from.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

returnCargo ([*shiftQueueCommand* = *false*]) → boolean
Orders the unit to return its cargo to a nearby resource depot such as a Command Center.

Only workers that are carrying minerals or gas can be ordered to return cargo.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

repair (*target*[, *shiftQueueCommand* = *false*]) → boolean
Orders the unit to repair the specified unit.

Only Terran SCVs can be ordered to repair, and the target must be a mechanical *Terran* unit or building.

Parameters

- **target** (BWAPI.Unit) – The unit to repair.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

burrow() → boolean
Orders the unit to burrow.

Either the unit must be a *Zerg_Lurker*, or the unit must be a *Zerg* ground unit that is capable of *Burrowing*, and *Burrow* technology must be researched.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

unburrow()

unburrow() → boolean
Orders a burrowed unit to unburrow.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

burrow()

cloak() → boolean
Orders the unit to cloak.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

decloak()

decloak() → boolean
Orders a cloaked unit to decloak.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

cloak()

siege() → boolean
Orders the unit to siege.

Only works for *Siege Tanks*.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

unsiege()

unsiege() → boolean
Orders the unit to unsiege.

Only works for sieged *Siege Tanks*.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

siege()

lift() → boolean
Orders the unit to lift.

Only works for liftable *Terran* structures.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

Unit.land()

load(*target*[, *shiftQueueCommand* = *false*]) → boolean
Orders the unit to load the target unit.

Only works if this unit is a Transport (*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*) or *Terran_Bunker* type.

Parameters

- **target** (BWAPI.Unit) – The target unit to load into this Transport(*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*) or *Terran_Bunker*.

- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

`unload()`, *unloadAll()*, *getLoadedUnits()*

unloadAll ([*shiftQueueCommand* = *false*]) → boolean

Orders the unit to unload all loaded units at the unit's current position.

Only works for Transports (*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*) and *Terran_Bunker*.

Parameters **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

load(), *getLoadedUnits()*

unloadAll (*target* [, *shiftQueueCommand* = *false*]) → boolean

Orders the unit to unload all loaded units at the specified location.

Only works for Transports (*Terran_Dropship*, *Protoss_Shuttle*, *Zerg_Overlord*). Not applicable to *Bunkers*.

Parameters

- **target** (`BWAPI.Position`) – The target position to unload the units at.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

load(), *getLoadedUnits()*

rightClick (*target*[, *shiftQueueCommand* = *false*]) → boolean
Works like the right click in the GUI.

Parameters

- **target** (*BWAPI.Position* or *BWAPI.Unit*) – Can be either a *Position* or *Unit*. The target position or target unit to right click.
- **shiftQueueCommand** (*boolean*) – (optional) If this value is true, then the order will be queued instead of immediately executed. If this value is omitted, then the order will be executed immediately by default.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

haltConstruction () → boolean
Orders a *Terran_SCV* to stop constructing a structure.

This leaves the structure in an incomplete state until it is either cancelled, razed, or completed by another *Terran_SCV*.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

cancelConstruction () → boolean
Orders this unit to cancel and refund itself from begin constructed.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

cancelAddon () → boolean
Orders this unit to cancel and refund an add-on that is being constructed.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

---

cancelTrain ([*slot* = *-2*]) → boolean
Orders the unit to remove the specified unit from its training queue.

Important: See *the differences between the C++ and Lua implementations of this function* for more information

Parameters **slot** (*int*) – (optional) Identifies the slot that will be cancelled. If the specified value is at least 1, then the unit in the corresponding slot will be cancelled. If the value is either omitted or -2, then the last slot is cancelled.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: The value of slot is passed directly to Broodwar. Other negative values have no effect.

See also:

train()

cancelMorph() → boolean
Orders this unit to cancel and refund a unit that is morphing.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

See also:

morph()

cancelResearch() → boolean
Orders this unit to cancel and refund a research that is in progress.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

cancelUpgrade() → boolean
Orders this unit to cancel and refund an upgrade that is in progress.

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

Note: There is a small chance for a command to fail after it has been passed to Broodwar.

useTech (*tech*[, *target* = *nil*]) → boolean
Orders the unit to use a technology.

Parameters

- **tech** (BWAPI.TechType) – The technology type to use.
- **target** (*BWAPI.Position* or *BWAPI.Unit*) – (optional) Can be either a *Position* or *Unit*. If specified, indicates the target location or unit to use the tech on. If unspecified, causes the tech to be used without a target (i.e. *Stim_Packs*).

Returns true if the command was passed to Broodwar, and false if *BWAPI* determined that the command would fail.

Return type boolean

See also:

TechTypes

getClosestUnit ([*pred* = *nil*][, *radius* = *999999*]) → Unit
Retrieves the closest unit relative to the value of *getPosition()*.

Parameters

- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns true for units that satisfy the intended filter and false otherwise (can be a *BWAPI.Filter unary filter*). Defaults to nil, which means no filter.
- **radius** (*int*) – (optional) The maximum radius to check for the closest unit. For performance reasons, a developer can limit the radius that is checked. If omitted, then the entire map is checked.

Returns The closest unit that matches the predicate, or nil if no matching unit is found.

Return type *BWAPI.Unit*

See also:

getUnitsInRadius(), *Game.getUnitsInRadius()*, *Game.getUnitsInRectangle()*

getInterceptors () → Unitset
Creates a single set containing all the *Interceptors* of all *Carriers* in this set.

Returns The set of all *Interceptors*.

Return type *BWAPI.Unitset*

See also:

Unit.getInterceptors()

getLarva () → Unitset
Creates a single set containing all the *Zerg_Larva* of all *Hatcheries*, *Lairs*, and *Hives* in this set.

Returns The set of all *Zerg_Larva*.

Return type *BWAPI.Unitset*

See also:

Unit.getLarva()

getLoadedUnits () → Unitset
Creates a single set containing all units that are loaded into units of this set.

Returns The set of all loaded units.

Return type *BWAPI.Unitset*

See also:

Unit.getLoadedUnits()

getPosition() → Position
Calculates the average of all valid Unit positions in this set.

Returns Average Position of all units in the set.

Return type *BWAPI.Position*

See also:

Unit.getPosition()

getUnitsInRadius(*radius*[, *pred* = *nil*]) → Unitset
Retrieves the set of all units in a given radius relative to the value of *getPosition()*.

Takes into account this unit's dimensions. Can optionally specify a filter to include only specific units (such as only ground units, etc.)

Parameters

- **radius** (*int*) – The radius, in pixels, to search for units.
- **pred** (*function*) – (optional) A predicate function that takes a *Unit* and returns `true` for units that satisfy the intended filter and `false` otherwise (can be a *BWAPI.Filter unary filter*). Defaults to `nil`, which means no filter.

Returns A *Unitset* containing the set of units that match the given criteria.

Return type *BWAPI.Unitset*

See also:

getClosestUnit(), *Unit.getUnitsInRadius()*, *Game.getUnitsInRectangle()*

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *contains()* instead.

See also:

contains(), std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

UpgradeType

class `BWAPI.`**`UpgradeType`**
The upgrade type represents a passive upgrade that can be obtained with *`Unit.upgrade()`*.

See also:

`UpgradeTypes`

Constructors

`UpgradeType``(`[*id = UpgradeTypes.Enum.None*]`)`
Expected type constructor.

If the type is an invalid type, then it becomes `Unknown`. A type is invalid if its value is less than 0 or greater than `Unknown`.

Parameters **`id`** (*`int`*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes `Unknown`.

Member Functions

`getID``()` → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

`isValid``()` → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns `true` if this type is valid and `false` otherwise.

Return type boolean

`getName``()` → string

Returns The variable name of the type.

Return type string

`gasPrice``(`[*level = 1*]`)` → int
Returns the vespene gas price for the first upgrade.

Parameters **`level`** (*`int`*) – (optional) The next upgrade level.

Returns The gas cost of the upgrade for the given `level`.

Return type int

Note: Upgrades start at level 0.

`gasPriceFactor``()` → int
Returns the amount that the vespene gas price increases for each additional upgrade.

Returns The gas cost added to the upgrade after each level.

Return type int

getRace() → Race
Retrieves the race the upgrade is for.

For example, `BWAPI.UpgradeTypes.Terran_Infantry_Armor:getRace()` will return *Races.Terran*.

Returns *Race* that this upgrade belongs to.

Return type *BWAPI.Race*

maxRepeats() → int
Returns the maximum number of times the upgrade can be researched.

Returns Maximum number of times this upgrade can be upgraded.

Return type int

mineralPrice([*level = 1*]) → int
Returns the mineral price for the upgrade.

Parameters **level** (*int*) – (optional) The next upgrade level.

Returns The mineral cost of the upgrade for the given `level`.

Return type int

Note: Upgrades start at level 0.

---

mineralPriceFactor() → int
The amount that the mineral price increases for each additional upgrade.

Returns The mineral cost added to the upgrade after each level.

Return type int

upgradeTime([*level = 1*]) → int
Returns the number of frames needed to research the first upgrade.

Parameters **level** (*int*) – (optional) The next upgrade level.

Returns The time cost of the upgrade for the given `level`.

Return type int

Note: Upgrades start at level 0.

---

upgradeTimeFactor() → int
Returns the number of frames that the upgrade time increases for each additional upgrade.

Returns The time cost added to the upgrade after each level.

Return type int

whatsRequired([*level = 1*]) → UnitType
Returns the type of unit that is required for the upgrade.

The player must have at least one of these units completed in order to start upgrading this upgrade.

Parameters **level** (*int*) – (optional) The next upgrade level.

Returns *UnitType* required to obtain this upgrade.

Return type *BWAPI.UnitType*

Note: Upgrades start at level 0.

whatUpgrades() → UnitType
Returns the type of unit that researches the upgrade.

Returns The *UnitType* that is used to upgrade this type.

Return type *BWAPI.UnitType*

whatUses() → UnitTypeset
Returns the set of units that are affected by this upgrade.

Returns Set of unit types that passively use this upgrade type.

Return type *BWAPI.UnitTypeset*

UpgradeTypeset

class BWAPI.**UpgradeTypeset**
A container for a set of *UpgradeType* objects.

Constructors

UpgradeTypeset()
Default constructor.

UpgradeTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.UpgradeTypeset) – The UpgradeTypeset to copy.

UpgradeTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *UpgradeType* are added to the set.

Parameters **tbl** (*table*) – A table containing *UpgradeType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if (*pred*)
Alias of *eraseIf()*

filter (*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf (*pred*)
Alias of *filter()*/*keep_if()*

keep_if (*pred*)
Alias of *filter()*/*keepIf()*

WalkPosition

class BWAPI.**WalkPosition**
Indicates a position that is 8x8 pixels in size. Sometimes referred to as "mini-tiles".

See also:

BWAPI.WalkPositions, BWAPI.WALKPOSITION_SCALE, *Game.isWalkable()*

Constructors

WalkPosition ()
Default constructor.

WalkPosition (*x*, *y*)

Parameters

- **x** (*int*) – The x coordinate.
- **y** (*int*) – The y coordinate.

WalkPosition (*pos*)
A constructor to convert a *Position* to a *WalkPosition*.

Parameters **pos** (BWAPI.Position) – The position to be converted.

WalkPosition (*tilePos*)
A constructor to convert a *TilePosition* to a *WalkPosition*.

Parameters **tilePos** (BWAPI.TilePosition) – The position to be converted.

Member Variables

x
(integer) The x coordinate.

y
(integer) The y coordinate.

Member Functions

isValid() → boolean
Checks if this point is within the game's map bounds.

Returns true If it is a valid position and on the map/playing field, or false If this is not a valid position.

Return type boolean

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

---

makeValid() → WalkPosition
Checks if this point is within the game's map bounds, if not, then it will set the x and y values to be within map bounds. For example, if x is less than 0, then x is set to 0.

Returns Itself

Return type BWAPI.WalkPosition

Note: If the Broodwar pointer is not initialized, this function will check validity against the largest (256x256) map size.

See also:

isValid()

getDistance(*pos*) → double
Gets an accurate distance measurement from this point to the given position.

Note: This is a direct distance calculation that ignores all collision.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Parameters **pos** (BWAPI.WalkPosition) – The target position to get the distance to.

Returns A double representing the distance between this point and position.

Return type double

See also:

getApproxDistance()

getLength() → double
Gets the length of this point from the top left corner of the map.

Note: This function impedes performance. In most cases you should use *getApproxDistance()*.

Returns A double representing the length of this point from (0,0).

Return type double

See also:

getApproxDistance()

getApproxDistance (*pos*) → int
Retrieves the approximate distance using an algorithm from Starcraft: Broodwar.

Note: This is a direct distance calculation that ignores all collision.

Note: This function is desired because it uses the same "imperfect" algorithm used in Broodwar, so that calculations will be consistent with the game. It is also optimized for performance.

Parameters **pos** (BWAPI.WalkPosition) – The target position to get the distance to.

Returns A integer representing the distance between this point and `position`.

Return type int

See also:

getDistance()

setMax (*max_x*, *max_y*) → WalkPosition
Sets the maximum x and y values.

If the current x or y values exceed the given maximum, then values are set to the maximum.

Parameters

- **max_x** (*int*) – Maximum x value.
- **max_y** (*int*) – Maximum y value.

Returns Itself.

Return type BWAPI.WalkPosition

See also:

setMin()

setMax (*max*) → WalkPosition
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.WalkPosition) – Point containing the maximum x and y values.

Returns Itself.

Return type BWAPI.WalkPosition

setMin (*max_x*, *max_y*) → WalkPosition
Sets the minimum x and y values.

If the current x or y values are below the given minimum, then values are set to the minimum.

Parameters

- **max_x** (*int*) – Minimum x value.
- **max_y** (*int*) – Minimum y value.

Returns Itself.

Return type BWAPI.WalkPosition

See also:

setMax()

setMin (*max*) → WalkPosition
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters **max** (BWAPI.WalkPosition) – Point containing the minimum x and y values.

Returns Itself.

Return type BWAPI.WalkPosition

WeaponType

class BWAPI.**WeaponType**
This object identifies a weapon type used by a unit to attack and deal damage.

Some weapon types can be upgraded while others are used for special abilities.

See also:

WeaponTypes

Constructors

WeaponType ([*id = WeaponTypes.Enum.None*])
Expected type constructor.

If the type is an invalid type, then it becomes Unknown. A type is invalid if its value is less than 0 or greater than Unknown.

Parameters **id** (*int*) – The id that corresponds to this type. It is typically an integer value that corresponds to an internal Broodwar type. If the given id is invalid, then it becomes Unknown.

Member Functions

getID () → int
Retrieves this type's identifier as an integer.

Returns An integer representation of this type.

Return type int

isValid () → boolean
Checks if the current type has a valid identifier. The purpose of this function is to prevent buffer overflows if a type has been handled improperly.

A type is valid if it is between 0 and Unknown (inclusive).

Returns true if this type is valid and false otherwise.

Return type boolean

getName () → string

Returns The variable name of the type.

Return type string

damageAmount() → int
Retrieves the base amount of damage that this weapon can deal per attack.

Returns Amount of base damage that this weapon deals.

Return type int

Note: That this damage amount must go through a *DamageType* and *UnitSizeType* filter before it is applied to a unit.

---

damageBonus() → int
Determines the bonus amount of damage that this weapon type increases by for every upgrade to this type.

Returns Amount of damage added for every weapon upgrade.

Return type int

See also:

upgradeType()

damageCooldown() → int
Retrieves the base amount of cooldown time between each attack, in frames.

Returns The amount of base cooldown applied to the unit after an attack.

Return type int

See also:

Unit.getGroundWeaponCooldown(), *Unit.getAirWeaponCooldown()*

damageFactor() → int
Obtains the intended number of missiles/attacks that are used.

This is used to multiply with the damage amount to obtain the full amount of damage for an attack.

Returns The damage factor multiplied by the amount to obtain the total damage.

Return type int

See also:

damageAmount()

damageType() → DamageType
Retrieves the damage type that this weapon applies to a unit type.

Returns *DamageType* used for damage calculation.

Return type *BWAPI.DamageType*

See also:

DamageType, *UnitSizeType*

explosionType() → ExplosionType
Retrieves the explosion type that indicates how the weapon deals damage.

Returns *ExplosionType* identifying how damage is applied to a target location.

Return type *BWAPI.ExplosionType*

getTech() → TechType
Retrieves the technology type that must be researched before this weapon can be used.

Returns *TechType* required by this weapon. Returns `TechTypes.None` if no tech type is required to use this weapon.

Return type *BWAPI.TechType*

See also:

TechType.getWeapon()

innerSplashRadius() → int
Retrieves the inner radius used for splash damage calculations, in pixels.

Returns Radius of the inner splash area, in pixels.

Return type int

maxRange() → int
Retrieves the maximum attack range of the weapon, measured in pixels.

Returns Maximum attack range, in pixels.

Return type int

medianSplashRadius() → int
Retrieves the middle radius used for splash damage calculations, in pixels.

Returns Radius of the middle splash area, in pixels.

Return type int

minRange() → int
Retrieves the minimum attack range of the weapon, measured in pixels.

This value is 0 for almost all weapon types, except for *WeaponTypes.Arclite_Shock_Cannon* and *WeaponTypes.Arclite_Shock_Cannon_Edmund_Duke*.

Returns Minimum attack range, in pixels.

Return type int

outerSplashRadius() → int
Retrieves the outer radius used for splash damage calculations, in pixels.

Returns Radius of the outer splash area, in pixels.

Return type int

targetsAir() → boolean
Checks if this weapon type can target air units.

Returns true if this weapon type can target air units, and false otherwise.

Return type boolean

See also:

Unit.isFlying(), *UnitType.isFlyer()*

targetsGround() → boolean
Checks if this weapon type can target ground units.

Returns true if this weapon type can target ground units, and false otherwise.

Return type boolean

See also:

Unit.isFlying(), *UnitType.isFlyer()*

targetsMechanical() → boolean
Checks if this weapon type can only target mechanical units.

Returns true if this weapon type can only target mechanical units, and false otherwise.

Return type boolean

See also:

targetsOrgOrMech(), *UnitType.isMechanical()*

targetsNonBuilding() → boolean
Checks if this weapon type cannot target structures.

Returns true if this weapon type cannot target buildings, and false if it can.

Return type boolean

See also:

UnitType.isBuilding()

targetsNonRobotic() → boolean
Checks if this weapon type cannot target robotic units.

Returns true if this weapon type cannot target robotic units, and false if it can.

Return type boolean

See also:

UnitType.isRobotic()

targetsOrganic() → boolean
Checks if this weapon type can only target organic units.

Returns true if this weapon type can only target organic units, and false otherwise.

Return type boolean

See also:

targetsOrgOrMech(), *UnitType.isOrganic()*

targetsOrgOrMech() → boolean
Checks if this weapon type can only target organic or mechanical units.

Returns true if this weapon type can only target organic or mechanical units, and false otherwise.

Return type boolean

See also:

targetsOrganic(), *targetsMechanical()*, *UnitType.isOrganic()*, *UnitType.isMechanical()*

targetsOwn() → boolean
Checks if this weapon type can only target units owned by the same player.

This is used for *WeaponTypes.Consume*.

Returns true if this weapon type can only target your own units, and false otherwise.

Return type boolean

See also:

Unit.getPlayer()

targetsTerrain() → boolean
Checks if this weapon type can target the ground.

Returns true if this weapon type can target a location, and false otherwise.

Return type boolean

Note: This is more for attacks like *Psionic Storm <BWAPI.TechTypes.Psionic_Storm>* which can target a location, not to be confused with attack move.

---

upgradeType() → UpgradeType
Retrieves the upgrade type that increases this weapon's damage output.

Returns The *UpgradeType* used to upgrade this weapon's damage.

Return type *BWAPI.UpgradeType*

See also:

damageBonus()

whatUses() → UnitType
Retrieves the unit type that is intended to use this weapon type.

Returns The *UnitType* that uses this weapon.

Return type *BWAPI.UnitType*

Note: There is a rare case where some hero unit types use the same weapon.

See also:

UnitType.groundWeapon(), *UnitType.airWeapon()*

WeaponTypeset

class BWAPI.**WeaponTypeset**
A container for a set of *WeaponType* objects.

Constructors

WeaponTypeset()
Default constructor.

WeaponTypeset(*set*)
Copy constructor.

Parameters **set** (BWAPI.WeaponTypeset) – The WeaponTypeset to copy.

WeaponTypeset(*tbl*)
Constructor to convert a Lua table to a set. Any values in the table that are of type *WeaponType* are added to the set.

Parameters **tbl** (*table*) – A table containing *WeaponType* objects.

Member Functions

iterator() → iteratorFunction
Returns an iterator function intended to be used in `for` loops (e.g. `for item in set:iterator() do`).

Returns

An iterator function that will return the next value in the set with each successive call.

Return type function

asTable() → table
Returns the values of the set as an array-like Lua table.

Note: The ordering of the returned table is arbitrary (due to sets being unordered in the C++ implementation).

Returns An array-like Lua table containing each value in the set.

Return type table

count(*val*) → int
Searches the set for elements with a value of `val` and returns the number of elements found. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`. Because of this, it's recommended to use *`contains()`* instead.

See also:

`contains()`, std::unordered_set::count

contains(*val*) → boolean
Checks if this set contains a specific value.

Returns `true` if the set contains the specified value, or `false` otherwise.

Return type boolean

size() → int

Returns The number of values in the set.

Return type int

Note: `set:size()` is exactly equivalent to `#set`

---

empty() → boolean

Returns `true` if the set is empty (`size() == 0`), or `false` otherwise.

Return type boolean

insert(*val*)
Inserts the value into the set.

Note: Sets cannot contain duplicate values. If the value already exists in the set, the set will not be modified.

---

erase(*val*) → numElementsErased
Removes `val` from the set if it exists.

Returns The number of elements removed. Because sets do not allow for duplicate values, this means that the function will return either `1` or `0`.

Return type int

clear()
Removes all elements from the set, leaving it with a size of 0.

eraseIf(*pred*)
Iterates the set and erases each element x where pred(x) returns true. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be erased and `false` otherwise.

erase_if(*pred*)
Alias of *eraseIf()*

filter(*pred*)
Iterates the set and erases each element x where pred(x) returns false. The set is modified in place.

Parameters **pred** (*function*) – A predicate function that takes a value and returns `true` for values that should be kept and `false` for elements that should be erased.

keepIf(*pred*)
Alias of *filter()*/*keep_if()*

keep_if(*pred*)
Alias of *filter()*/*keepIf()*

Data

BulletTypes

Module containing bullet types.

See also:

BWAPI.BulletType

Functions

static BWAPI.BulletTypes.**allBulletTypes**() → set
Retrieves a set of all BulletTypes.

Returns Set of all BulletTypes.

Return type *BulletTypeset*

Constants

All constants are instances of the *BulletType* class

BWAPI.BulletTypes.**Melee**

BWAPI.BulletTypes.**Fusion_Cutter_Hit**

BWAPI.BulletTypes.**Gauss_Rifle_Hit**

BWAPI.BulletTypes.**C_10_Canister_Rifle_Hit**

BWAPI.BulletTypes.**Gemini_Missiles**

BWAPI.BulletTypes.**Fragmentation_Grenade**

BWAPI.BulletTypes.**Longbolt_Missile**

BWAPI.BulletTypes.**ATS_ATA_Laser_Battery**

BWAPI.BulletTypes.**Burst_Lasers**

BWAPI.BulletTypes.**Arclite_Shock_Cannon_Hit**

BWAPI.BulletTypes.**EMP_Missile**

BWAPI.BulletTypes.**Dual_Photon_Blasters_Hit**

BWAPI.BulletTypes.**Particle_Beam_Hit**

BWAPI.BulletTypes.**Anti_Matter_Missile**

BWAPI.BulletTypes.**Pulse_Cannon**

BWAPI.BulletTypes.**Psionic_Shockwave_Hit**

BWAPI.BulletTypes.**Psionic_Storm**

BWAPI.BulletTypes.**Yamato_Gun**

BWAPI.BulletTypes.**Phase_Disruptor**

BWAPI.BulletTypes.**STA_STS_Cannon_Overlay**

BWAPI.BulletTypes.**Sunken_Colony_Tentacle**

BWAPI.BulletTypes.**Acid_Spore**

BWAPI.BulletTypes.**Glave_Wurm**

BWAPI.BulletTypes.**Seeker_Spores**

BWAPI.BulletTypes.**Queen_Spell_Carrier**

BWAPI.BulletTypes.**Plague_Cloud**

BWAPI.BulletTypes.**Consume**

BWAPI.BulletTypes.**Ensnare**

BWAPI.BulletTypes.**Needle_Spine_Hit**

BWAPI.BulletTypes.**Invisible**

BWAPI.BulletTypes.**Optical_Flare_Grenade**

BWAPI.BulletTypes.**Halo_Rockets**

BWAPI.BulletTypes.**Subterranean_Spines**

BWAPI.BulletTypes.**Corrosive_Acid_Shot**

BWAPI.BulletTypes.**Neutron_Flare**

BWAPI.BulletTypes.**None**

BWAPI.BulletTypes.**Unknown**

Enum

Enumeration of bullet types.

Important: Enum values rarely need to be used in Lua.

BWAPI.BulletTypes.Enum.**Melee**
Value of 0 [0x0].

BWAPI.BulletTypes.Enum.**Fusion_Cutter_Hit**
Value of 141 [0x8d].

BWAPI.BulletTypes.Enum.**Gauss_Rifle_Hit**
Value of 142 [0x8e].

BWAPI.BulletTypes.Enum.**C_10_Canister_Rifle_Hit**
Value of 143 [0x8f].

BWAPI.BulletTypes.Enum.**Gemini_Missiles**
Value of 144 [0x90].

BWAPI.BulletTypes.Enum.**Fragmentation_Grenade**
Value of 145 [0x91].

BWAPI.BulletTypes.Enum.**Longbolt_Missile**
Value of 146 [0x92].

BWAPI.BulletTypes.Enum.**Unused_Lockdown**

BWAPI.BulletTypes.Enum.**ATS_ATA_Laser_Battery**
Value of 148 [0x94].

BWAPI.BulletTypes.Enum.**Burst_Lasers**
Value of 149 [0x95].

BWAPI.BulletTypes.Enum.**Arclite_Shock_Cannon_Hit**
Value of 150 [0x96].

BWAPI.BulletTypes.Enum.**EMP_Missile**
Value of 151 [0x97].

BWAPI.BulletTypes.Enum.**Dual_Photon_Blasters_Hit**
Value of 152 [0x98].

BWAPI.BulletTypes.Enum.**Particle_Beam_Hit**
Value of 153 [0x99].

BWAPI.BulletTypes.Enum.**Anti_Matter_Missile**
Value of 154 [0x9a].

BWAPI.BulletTypes.Enum.**Pulse_Cannon**
Value of 155 [0x9b].

BWAPI.BulletTypes.Enum.**Psionic_Shockwave_Hit**
Value of 156 [0x9c].

BWAPI.BulletTypes.Enum.**Psionic_Storm**
Value of 157 [0x9d].

BWAPI.BulletTypes.Enum.**Yamato_Gun**
Value of 158 [0x9e].

BWAPI.BulletTypes.Enum.**Phase_Disruptor**
Value of 159 [0x9f].

BWAPI.BulletTypes.Enum.**STA_STS_Cannon_Overlay**
Value of 160 [0xa0].

BWAPI.BulletTypes.Enum.**Sunken_Colony_Tentacle**
Value of 161 [0xa1].

BWAPI.BulletTypes.Enum.**Venom_Unused**

BWAPI.BulletTypes.Enum.**Acid_Spore**
Value of 163 [0xa3].

BWAPI.BulletTypes.Enum.**Plasma_Drip_Unused**

BWAPI.BulletTypes.Enum.**Glave_Wurm**
Value of 165 [0xa5].

BWAPI.BulletTypes.Enum.**Seeker_Spores**
Value of 166 [0xa6].

BWAPI.BulletTypes.Enum.**Queen_Spell_Carrier**
Value of 167 [0xa7].

BWAPI.BulletTypes.Enum.**Plague_Cloud**
Value of 168 [0xa8].

BWAPI.BulletTypes.Enum.**Consume**
Value of 169 [0xa9].

BWAPI.BulletTypes.Enum.**Ensnare**
Value of 170 [0xaa].

BWAPI.BulletTypes.Enum.**Needle_Spine_Hit**
Value of 171 [0xab].

BWAPI.BulletTypes.Enum.**Invisible**
Value of 172 [0xac].

BWAPI.BulletTypes.Enum.**Optical_Flare_Grenade**
Value of 201 [0xc9].

BWAPI.BulletTypes.Enum.**Halo_Rockets**
Value of 202 [0xca].

BWAPI.BulletTypes.Enum.**Subterranean_Spines**
Value of 203 [0xcb].

BWAPI.BulletTypes.Enum.**Corrosive_Acid_Shot**
Value of 204 [0xcc].

BWAPI.BulletTypes.Enum.**Corrosive_Acid_Hit**

BWAPI.BulletTypes.Enum.**Neutron_Flare**
Value of 206 [0xce].

BWAPI.BulletTypes.Enum.**None**

BWAPI.BulletTypes.Enum.**Unknown**

BWAPI.BulletTypes.Enum.**MAX**

Colors

Module containing colors.

See also:

BWAPI.Color

Constants

All constants are instances of the *Color* class

BWAPI.Colors.**Red**
: The default color for Player 1.

BWAPI.Colors.**Blue**
: The default color for Player 2.

BWAPI.Colors.**Teal**
: The default color for Player 3.

BWAPI.Colors.**Purple**
: The default color for Player 4.

BWAPI.Colors.**Orange**
: The default color for Player 5.

BWAPI.Colors.**Brown**
: The default color for Player 6.

BWAPI.Colors.**White**
: A bright white. Note that this is lighter than Player 7's white.

BWAPI.Colors.**Yellow**
: The default color for Player 8.

BWAPI.Colors.**Green**
: The alternate color for Player 7 on Ice tilesets.

BWAPI.Colors.**Cyan**
: The default color for Neutral (Player 12).

BWAPI.Colors.**Black**
: The color black

BWAPI.Colors.**Grey**
: The color grey

CoordinateType

Module containing coordinate types.

Constants

BWAPI.CoordinateType.**None**
: A default value for uninitialized coordinate types.

  Equal to *BWAPI.CoordinateType.Enum.None*

BWAPI.CoordinateType.**Screen**
Origin (0,0) corresponds to the top left corner of the screen.

Equal to *BWAPI.CoordinateType.Enum.Screen*

BWAPI.CoordinateType.**Map**
Origin (0,0) corresponds to the top left corner of the map.

Equal to *BWAPI.CoordinateType.Enum.Map*

BWAPI.CoordinateType.**Mouse**
Origin (0,0) corresponds to the location of the mouse cursor.

Equal to *BWAPI.CoordinateType.Enum.Mouse*

Enum

Enumeration of coordinate types.

BWAPI.CoordinateType.Enum.**None**
Value of 0.

BWAPI.CoordinateType.Enum.**Screen**
Value of 1.

BWAPI.CoordinateType.Enum.**Map**
Value of 2.

BWAPI.CoordinateType.Enum.**Mouse**
Value of 3.

DamageTypes

Module containing damage types.

- View on Liquipedia
- View on Starcraft Campendium (Official Website)
- View on Starcraft Wikia

See also:

BWAPI.DamageType

Functions

static BWAPI.DamageTypes.**allDamageTypes**() → set
Retrieves a set of all DamageTypes.

Returns Set of all DamageTypes.

Return type *DamageTypeset*

Constants

All constants are instances of the *DamageType* class

BWAPI.DamageTypes.**Independent**

BWAPI.DamageTypes.**Explosive**

BWAPI.DamageTypes.**Concussive**

BWAPI.DamageTypes.**Normal**

BWAPI.DamageTypes.**Ignore_Armor**

BWAPI.DamageTypes.**None**

BWAPI.DamageTypes.**Unknown**

Enum

Enumeration of damage types.

Important: Enum values rarely need to be used in Lua.

BWAPI.DamageTypes.Enum.**Independent**
Value of 0.

BWAPI.DamageTypes.Enum.**Explosive**
Value of 1.

BWAPI.DamageTypes.Enum.**Concussive**
Value of 2.

BWAPI.DamageTypes.Enum.**Normal**
Value of 3.

BWAPI.DamageTypes.Enum.**Ignore_Armor**
Value of 4.

BWAPI.DamageTypes.Enum.**None**
Value of 5.

BWAPI.DamageTypes.Enum.**Unknown**
Value of 6.

BWAPI.DamageTypes.Enum.**MAX**
Value of 7.

Errors

Module containing error codes.

See also:

BWAPI.Error

Functions

static BWAPI.Errors.**allErrors**() → set
Retrieves a set of all Errors.

Returns Set of all Errors.

Return type *Errorset*

Constants

All constants are instances of the *Error* class

BWAPI.Errors.**Unit_Does_Not_Exist**

BWAPI.Errors.**Unit_Not_Visible**

BWAPI.Errors.**Unit_Not_Owned**

BWAPI.Errors.**Unit_Busy**

BWAPI.Errors.**Incompatible_UnitType**

BWAPI.Errors.**Incompatible_TechType**

BWAPI.Errors.**Incompatible_State**

BWAPI.Errors.**Already_Researched**

BWAPI.Errors.**Fully_Upgraded**

BWAPI.Errors.**Currently_Researching**

BWAPI.Errors.**Currently_Upgrading**

BWAPI.Errors.**Insufficient_Minerals**

BWAPI.Errors.**Insufficient_Gas**

BWAPI.Errors.**Insufficient_Supply**

BWAPI.Errors.**Insufficient_Energy**

BWAPI.Errors.**Insufficient_Tech**

BWAPI.Errors.**Insufficient_Ammo**

BWAPI.Errors.**Insufficient_Space**

BWAPI.Errors.**Invalid_Tile_Position**

BWAPI.Errors.**Unbuildable_Location**

BWAPI.Errors.**Unreachable_Location**

BWAPI.Errors.**Out_Of_Range**

BWAPI.Errors.**Unable_To_Hit**

BWAPI.Errors.**Access_Denied**

BWAPI.Errors.**File_Not_Found**

BWAPI.Errors.**Invalid_Parameter**

BWAPI.Errors.**None**

BWAPI.Errors.**Unknown**

Enum

Enumeration of error codes.

Important: Enum values rarely need to be used in Lua.

BWAPI.Errors.Enum.**Unit_Does_Not_Exist**
Value of 0.

BWAPI.Errors.Enum.**Unit_Not_Visible**
Value of 1.

BWAPI.Errors.Enum.**Unit_Not_Owned**
Value of 2.

BWAPI.Errors.Enum.**Unit_Busy**
Value of 3.

BWAPI.Errors.Enum.**Incompatible_UnitType**
Value of 4.

BWAPI.Errors.Enum.**Incompatible_TechType**
Value of 5.

BWAPI.Errors.Enum.**Incompatible_State**
Value of 6.

BWAPI.Errors.Enum.**Already_Researched**
Value of 7.

BWAPI.Errors.Enum.**Fully_Upgraded**
Value of 8.

BWAPI.Errors.Enum.**Currently_Researching**
Value of 9.

BWAPI.Errors.Enum.**Currently_Upgrading**
Value of 10.

BWAPI.Errors.Enum.**Insufficient_Minerals**
Value of 11.

BWAPI.Errors.Enum.**Insufficient_Gas**
Value of 12.

BWAPI.Errors.Enum.**Insufficient_Supply**
Value of 13.

BWAPI.Errors.Enum.**Insufficient_Energy**
Value of 14.

BWAPI.Errors.Enum.**Insufficient_Tech**
Value of 15.

BWAPI.Errors.Enum.**Insufficient_Ammo**
Value of 16.

BWAPI.Errors.Enum.**Insufficient_Space**
Value of 17.

BWAPI.Errors.Enum.**Invalid_Tile_Position**
Value of 18.

BWAPI.Errors.Enum.**Unbuildable_Location**
Value of 19.

BWAPI.Errors.Enum.**Unreachable_Location**
Value of 20.

BWAPI.Errors.Enum.**Out_Of_Range**
Value of 21.

BWAPI.Errors.Enum.**Unable_To_Hit**
Value of 22.

BWAPI.Errors.Enum.**Access_Denied**
Value of 23.

BWAPI.Errors.Enum.**File_Not_Found**
Value of 24.

BWAPI.Errors.Enum.**Invalid_Parameter**
Value of 25.

BWAPI.Errors.Enum.**None**
Value of 26.

BWAPI.Errors.Enum.**Unknown**
Value of 27.

BWAPI.Errors.Enum.**MAX**
Value of 28.

ExplosionTypes

Module containing explosion types.

See also:

BWAPI.ExplosionType

Functions

static BWAPI.ExplosionTypes.**allExplosionTypes**() → set
Retrieves a set of all ExplosionTypes.

Returns Set of all ExplosionTypes.

Return type *ExplosionTypeset*

Constants

All constants are instances of the *ExplosionType* class

BWAPI.ExplosionTypes.**None**

BWAPI.ExplosionTypes.**Normal**

BWAPI.ExplosionTypes.**Radial_Splash**

BWAPI.ExplosionTypes.**Enemy_Splash**

BWAPI.ExplosionTypes.**Lockdown**

BWAPI.ExplosionTypes.**Nuclear_Missile**

BWAPI.ExplosionTypes.**Parasite**

BWAPI.ExplosionTypes.**Broodlings**

BWAPI.ExplosionTypes.**EMP_Shockwave**

BWAPI.ExplosionTypes.**Irradiate**

BWAPI.ExplosionTypes.**Ensnare**

BWAPI.ExplosionTypes.**Plague**

BWAPI.ExplosionTypes.**Stasis_Field**

BWAPI.ExplosionTypes.**Dark_Swarm**

BWAPI.ExplosionTypes.**Consume**

BWAPI.ExplosionTypes.**Yamato_Gun**

BWAPI.ExplosionTypes.**Restoration**

BWAPI.ExplosionTypes.**Disruption_Web**

BWAPI.ExplosionTypes.**Corrosive_Acid**

BWAPI.ExplosionTypes.**Mind_Control**

BWAPI.ExplosionTypes.**Feedback**

BWAPI.ExplosionTypes.**Optical_Flare**

BWAPI.ExplosionTypes.**Maelstrom**

BWAPI.ExplosionTypes.**Air_Splash**

BWAPI.ExplosionTypes.**Unknown**

Enum

Enumeration of explosion types.

Important: Enum values rarely need to be used in Lua.

BWAPI.ExplosionTypes.Enum.**None**
: Value of 0.

BWAPI.ExplosionTypes.Enum.**Normal**
: Value of 1.

BWAPI.ExplosionTypes.Enum.**Radial_Splash**
: Value of 2.

BWAPI.ExplosionTypes.Enum.**Enemy_Splash**
: Value of 3.

BWAPI.ExplosionTypes.Enum.**Lockdown**
: Value of 4.

BWAPI.ExplosionTypes.Enum.**Nuclear_Missile**
: Value of 5.

BWAPI.ExplosionTypes.Enum.**Parasite**
: Value of 6.

BWAPI.ExplosionTypes.Enum.**Broodlings**
: Value of 7.

BWAPI.ExplosionTypes.Enum.**EMP_Shockwave**
: Value of 8.

BWAPI.ExplosionTypes.Enum.**Irradiate**
Value of 9.

BWAPI.ExplosionTypes.Enum.**Ensnare**
Value of 10.

BWAPI.ExplosionTypes.Enum.**Plague**
Value of 11.

BWAPI.ExplosionTypes.Enum.**Stasis_Field**
Value of 12.

BWAPI.ExplosionTypes.Enum.**Dark_Swarm**
Value of 13.

BWAPI.ExplosionTypes.Enum.**Consume**
Value of 14.

BWAPI.ExplosionTypes.Enum.**Yamato_Gun**
Value of 15.

BWAPI.ExplosionTypes.Enum.**Restoration**
Value of 16.

BWAPI.ExplosionTypes.Enum.**Disruption_Web**
Value of 17.

BWAPI.ExplosionTypes.Enum.**Corrosive_Acid**
Value of 18.

BWAPI.ExplosionTypes.Enum.**Mind_Control**
Value of 19.

BWAPI.ExplosionTypes.Enum.**Feedback**
Value of 20.

BWAPI.ExplosionTypes.Enum.**Optical_Flare**
Value of 21.

BWAPI.ExplosionTypes.Enum.**Maelstrom**
Value of 22.

BWAPI.ExplosionTypes.Enum.**Air_Splash**
Value of 24.

BWAPI.ExplosionTypes.Enum.**Unknown**
Value of 25.

BWAPI.ExplosionTypes.Enum.**MAX**
Value of 25.

Filter

Unary Filters

Unary filters are predicate functions that return boolean values.

These functions can be used as predicates for functions that take a unit predicate/filter parameter (e.g. *BWAPI.Unit.getClosestUnit()*, *BWAPI.Game.getUnitsInRadius()*). They can also be used as convenience functions to access values that may usually take multiple function calls to access.

See also:

The differences between the C++ and Lua implementations of UnitFilter

UnitType delegate boolean filters

static `BWAPI.Filter.`**`IsTransport`**(*unit*) → bool
Equivalent to `unit:getType():spaceProvided() > 0 and u:getType() ~= BWAPI.UnitTypes.Terran_Bunker`

static `BWAPI.Filter.`**`CanProduce`**(*unit*) → bool
Equivalent to `unit:getType():canProduce()`

static `BWAPI.Filter.`**`CanAttack`**(*unit*) → bool
Equivalent to `unit:getType():canAttack()`

static `BWAPI.Filter.`**`CanMove`**(*unit*) → bool
Equivalent to `unit:getType():canMove()`

static `BWAPI.Filter.`**`IsFlyer`**(*unit*) → bool
Equivalent to `unit:getType():isFlyer()`

static `BWAPI.Filter.`**`RegeneratesHP`**(*unit*) → bool
Equivalent to `unit:getType():regeneratesHP()`

static `BWAPI.Filter.`**`IsSpellcaster`**(*unit*) → bool
Equivalent to `unit:getType():isSpellcaster()`

static `BWAPI.Filter.`**`HasPermanentCloak`**(*unit*) → bool
Equivalent to `unit:getType():hasPermanentCloak()`

static `BWAPI.Filter.`**`IsOrganic`**(*unit*) → bool
Equivalent to `unit:getType():isOrganic()`

static `BWAPI.Filter.`**`IsMechanical`**(*unit*) → bool
Equivalent to `unit:getType():isMechanical()`

static `BWAPI.Filter.`**`IsRobotic`**(*unit*) → bool
Equivalent to `unit:getType():isRobotic()`

static `BWAPI.Filter.`**`IsDetector`**(*unit*) → bool
Equivalent to `unit:getType():isDetector()`

static `BWAPI.Filter.`**`IsResourceContainer`**(*unit*) → bool
Equivalent to `unit:getType():isResourceContainer()`

static `BWAPI.Filter.`**`IsResourceDepot`**(*unit*) → bool
Equivalent to `unit:getType():isResourceDepot()`

static `BWAPI.Filter.`**`IsRefinery`**(*unit*) → bool
Equivalent to `unit:getType():isRefinery()`

static `BWAPI.Filter.`**`IsWorker`**(*unit*) → bool
Equivalent to `unit:getType():isWorker()`

static `BWAPI.Filter.`**`RequiresPsi`**(*unit*) → bool
Equivalent to `unit:getType():requiresPsi()`

static `BWAPI.Filter.`**`RequiresCreep`**(*unit*) → bool
Equivalent to `unit:getType():requiresCreep()`

static `BWAPI.Filter.`**`IsBurrowable`**(*unit*) → bool
Equivalent to `unit:getType():isBurrowable()`

static `BWAPI.Filter.`**`IsCloakable`**(*unit*) → bool
Equivalent to `unit:getType():isCloakable()`

static `BWAPI.Filter.`**`IsBuilding`**(*unit*) → bool
Equivalent to `unit:getType():isBuilding()`

static `BWAPI.Filter.`**`IsAddon`**(*unit*) → bool
Equivalent to `unit:getType():isAddon()`

static `BWAPI.Filter.`**`IsFlyingBuilding`**(*unit*) → bool
Equivalent to `unit:getType():isFlyingBuilding()`

static `BWAPI.Filter.`**`IsNeutral`**(*unit*) → bool
Equivalent to `unit:getType():isNeutral()`

static `BWAPI.Filter.`**`IsHero`**(*unit*) → bool
Equivalent to `unit:getType():isHero()`

static `BWAPI.Filter.`**`IsPowerup`**(*unit*) → bool
Equivalent to `unit:getType():isPowerup()`

static `BWAPI.Filter.`**`IsBeacon`**(*unit*) → bool
Equivalent to `unit:getType():isBeacon()`

static `BWAPI.Filter.`**`IsFlagBeacon`**(*unit*) → bool
Equivalent to `unit:getType():isFlagBeacon()`

static `BWAPI.Filter.`**`IsSpecialBuilding`**(*unit*) → bool
Equivalent to `unit:getType():isSpecialBuilding()`

static `BWAPI.Filter.`**`IsSpell`**(*unit*) → bool
Equivalent to `unit:getType():isSpell()`

static `BWAPI.Filter.`**`ProducesLarva`**(*unit*) → bool
Equivalent to `unit:getType():producesLarva()`

static `BWAPI.Filter.`**`IsMineralField`**(*unit*) → bool
Equivalent to `unit:getType():isMineralField()`

static `BWAPI.Filter.`**`IsCritter`**(*unit*) → bool
Equivalent to `unit:getType():isCritter()`

static `BWAPI.Filter.`**`CanBuildAddon`**(*unit*) → bool
Equivalent to `unit:getType():canBuildAddon()`

Unit boolean filters

static `BWAPI.Filter.`**`IsFlying`**(*unit*) → bool
Equivalent to `unit:isFlying()`

static `BWAPI.Filter.`**`Exists`**(*unit*) → bool
Equivalent to `unit:exists()`

static `BWAPI.Filter.`**`IsAttacking`**(*unit*) → bool
Equivalent to `unit:isAttacking()`

static `BWAPI.Filter.`**`IsBeingConstructed`**(*unit*) → bool
Equivalent to `unit:isBeingConstructed()`

static `BWAPI.Filter.`**`IsBeingGathered`**(*unit*) → bool
Equivalent to `unit:isBeingGathered()`

static `BWAPI.Filter.`**`IsBeingHealed`**(*unit*) → bool
Equivalent to `unit:isBeingHealed()`

static `BWAPI.Filter.`**`IsBlind`**(*unit*) → bool
Equivalent to `unit:isBlind()`

static `BWAPI.Filter.`**`IsBraking`**(*unit*) → bool
Equivalent to `unit:isBraking()`

static `BWAPI.Filter.`**`IsBurrowed`**(*unit*) → bool
Equivalent to `unit:isBurrowed()`

static `BWAPI.Filter.`**`IsCarryingGas`**(*unit*) → bool
Equivalent to `unit:isCarryingGas()`

static `BWAPI.Filter.`**`IsCarryingMinerals`**(*unit*) → bool
Equivalent to `unit:isCarryingMinerals()`

static `BWAPI.Filter.`**`IsCarryingSomething`**(*unit*) → bool
Equivalent to `unit:isCarryingMinerals() or unit:isCarryingGas()`

static `BWAPI.Filter.`**`IsCloaked`**(*unit*) → bool
Equivalent to `unit:isCloaked()`

static `BWAPI.Filter.`**`IsCompleted`**(*unit*) → bool
Equivalent to `unit:isCompleted()`

static `BWAPI.Filter.`**`IsConstructing`**(*unit*) → bool
Equivalent to `unit:isConstructing()`

static `BWAPI.Filter.`**`IsDefenseMatrixed`**(*unit*) → bool
Equivalent to `unit:isDefenseMatrixed()`

static `BWAPI.Filter.`**`IsDetected`**(*unit*) → bool
Equivalent to `unit:isDetected()`

static `BWAPI.Filter.`**`IsEnsnared`**(*unit*) → bool
Equivalent to `unit:isEnsnared()`

static `BWAPI.Filter.`**`IsFollowing`**(*unit*) → bool
Equivalent to `unit:isFollowing()`

static `BWAPI.Filter.`**`IsGatheringGas`**(*unit*) → bool
Equivalent to `unit:isGatheringGas()`

static `BWAPI.Filter.`**`IsGatheringMinerals`**(*unit*) → bool
Equivalent to `unit:isGatheringMinerals()`

static `BWAPI.Filter.`**`IsHallucination`**(*unit*) → bool
Equivalent to `unit:isHallucination()`

static `BWAPI.Filter.`**`IsHoldingPosition`**(*unit*) → bool
Equivalent to `unit:isHoldingPosition()`

static `BWAPI.Filter.`**`IsIdle`**(*unit*) → bool
Equivalent to `unit:isIdle()`

static `BWAPI.Filter.`**`IsInterruptible`**(*unit*) → bool
Equivalent to `unit:isInterruptible()`

static `BWAPI.Filter.`**`IsInvincible`**(*unit*) → bool
Equivalent to `unit:isInvincible()`

static `BWAPI.Filter.`**`IsIrradiated`**`(`*unit*`)` → bool
Equivalent to `unit:isIrradiated()`

static `BWAPI.Filter.`**`IsLifted`**`(`*unit*`)` → bool
Equivalent to `unit:isLifted()`

static `BWAPI.Filter.`**`IsLoaded`**`(`*unit*`)` → bool
Equivalent to `unit:isLoaded()`

static `BWAPI.Filter.`**`IsLockedDown`**`(`*unit*`)` → bool
Equivalent to `unit:isLockedDown()`

static `BWAPI.Filter.`**`IsMaelstrommed`**`(`*unit*`)` → bool
Equivalent to `unit:isMaelstrommed()`

static `BWAPI.Filter.`**`IsMorphing`**`(`*unit*`)` → bool
Equivalent to `unit:isMorphing()`

static `BWAPI.Filter.`**`IsMoving`**`(`*unit*`)` → bool
Equivalent to `unit:isMoving()`

static `BWAPI.Filter.`**`IsParasited`**`(`*unit*`)` → bool
Equivalent to `unit:isParasited()`

static `BWAPI.Filter.`**`IsPatrolling`**`(`*unit*`)` → bool
Equivalent to `unit:isPatrolling()`

static `BWAPI.Filter.`**`IsPlagued`**`(`*unit*`)` → bool
Equivalent to `unit:isPlagued()`

static `BWAPI.Filter.`**`IsRepairing`**`(`*unit*`)` → bool
Equivalent to `unit:isRepairing()`

static `BWAPI.Filter.`**`IsResearching`**`(`*unit*`)` → bool
Equivalent to `unit:isResearching()`

static `BWAPI.Filter.`**`IsSieged`**`(`*unit*`)` → bool
Equivalent to `unit:isSieged()`

static `BWAPI.Filter.`**`IsStartingAttack`**`(`*unit*`)` → bool
Equivalent to `unit:isStartingAttack()`

static `BWAPI.Filter.`**`IsStasised`**`(`*unit*`)` → bool
Equivalent to `unit:isStasised()`

static `BWAPI.Filter.`**`IsStimmed`**`(`*unit*`)` → bool
Equivalent to `unit:isStimmed()`

static `BWAPI.Filter.`**`IsStuck`**`(`*unit*`)` → bool
Equivalent to `unit:isStuck()`

static `BWAPI.Filter.`**`IsTraining`**`(`*unit*`)` → bool
Equivalent to `unit:isTraining()`

static `BWAPI.Filter.`**`IsUnderAttack`**`(`*unit*`)` → bool
Equivalent to `unit:isUnderAttack()`

static `BWAPI.Filter.`**`IsUnderDarkSwarm`**`(`*unit*`)` → bool
Equivalent to `unit:isUnderDarkSwarm()`

static `BWAPI.Filter.`**`IsUnderDisruptionWeb`**`(`*unit*`)` → bool
Equivalent to `unit:isUnderDisruptionWeb()`

static `BWAPI.Filter.`**`IsUnderStorm`**`(`*unit*`)` → bool
Equivalent to `unit:isUnderStorm()`

static `BWAPI.Filter.`**`IsPowered`**`(`*unit*`)` → bool
Equivalent to `unit:isPowered()`

static `BWAPI.Filter.`**`IsVisible`**`(`*unit*`)` → bool
Equivalent to `unit:isVisible()`

Comparison Filters

Comparison filters are functions that return a value.

These can either be used as convenience functions to access values that usually take multiple function calls to get, or can be used with *`BWAPI.Lowest()`*/*`BWAPI.Highest()`* to create a 'best' function to be used with *`BWAPI.Game.getBestUnit()`* (as long as the returned value from the comparison filter can be compared using less than/greater than).

See also:

The differences between the C++ and Lua implementations of BestFilter

static `BWAPI.Filter.`**`HP`**`(`*unit*`)` → int
Equivalent to `unit:getHitPoints()`

static `BWAPI.Filter.`**`MaxHP`**`(`*unit*`)` → int
Equivalent to `unit:getType():maxHitPoints()`

static `BWAPI.Filter.`**`HP_Percent`**`(`*unit*`)` → int
Equivalent to `math.floor(unit:getHitPoints() / unit:getType():maxHitPoints() * 100)` if `unit:getType():maxHitPoints() ~= 0`, or `0` otherwise.

static `BWAPI.Filter.`**`Shields`**`(`*unit*`)` → int
Equivalent to `unit:getShields()`

static `BWAPI.Filter.`**`MaxShields`**`(`*unit*`)` → int
Equivalent to `unit:getType():maxShields()`

static `BWAPI.Filter.`**`Shields_Percent`**`(`*unit*`)` → int
Equivalent to `math.floor(unit:getShields() / unit:getType():maxShields() * 100)` if `unit:getType():maxShields() ~= 0`, or `0` otherwise.

static `BWAPI.Filter.`**`Energy`**`(`*unit*`)` → int
Equivalent to `unit:getEnergy()`

static `BWAPI.Filter.`**`MaxEnergy`**`(`*unit*`)` → int
Equivalent to `unit:getPlayer():maxEnergy(unit:getType())`

static `BWAPI.Filter.`**`Energy_Percent`**`(`*unit*`)` → int
Equivalent to `math.floor(unit:getEnergy() / unit:getPlayer():maxEnergy(unit:getType()) * 100)` if `unit:getPlayer():maxEnergy(unit:getType()) ~= 0`, or `0` otherwise.

static `BWAPI.Filter.`**`Armor`**`(`*unit*`)` → int
Equivalent to `unit:getPlayer():armor(unit:getType())`

static `BWAPI.Filter.`**`ArmorUpgrade`**`(`*unit*`)` → UpgradeType
Equivalent to `unit:getType():armorUpgrade()`

static `BWAPI.Filter.`**`MineralPrice`**`(`*unit*`)` → int
Equivalent to `unit:getType():mineralPrice()`

static `BWAPI.Filter.`**`GasPrice`**(*unit*) → int
Equivalent to `unit:getType():gasPrice()`

static `BWAPI.Filter.`**`BuildTime`**(*unit*) → int
Equivalent to `unit:getType():buildTime()`

static `BWAPI.Filter.`**`SupplyRequired`**(*unit*) → int
Equivalent to `unit:getType():supplyRequired()`

static `BWAPI.Filter.`**`SupplyProvided`**(*unit*) → int
Equivalent to `unit:getType():supplyProvided()`

static `BWAPI.Filter.`**`SpaceRequired`**(*unit*) → int
Equivalent to `unit:getType():spaceRequired()`

static `BWAPI.Filter.`**`SpaceRemaining`**(*unit*) → int
Equivalent to `unit:spaceRemaining()`

static `BWAPI.Filter.`**`SpaceProvided`**(*unit*) → int
Equivalent to `unit:getType():spaceProvided()`

static `BWAPI.Filter.`**`BuildScore`**(*unit*) → int
Equivalent to `unit:getType():buildScore()`

static `BWAPI.Filter.`**`DestroyScore`**(*unit*) → int
Equivalent to `unit:getType():destroyScore()`

static `BWAPI.Filter.`**`TopSpeed`**(*unit*) → double
Equivalent to `unit:getPlayer():topSpeed(unit:getType())`

static `BWAPI.Filter.`**`SightRange`**(*unit*) → int
Equivalent to `unit:getPlayer():sightRange(unit:getType())`

static `BWAPI.Filter.`**`MaxWeaponCooldown`**(*unit*) → int
Equivalent to `unit:getPlayer():weaponDamageCooldown(unit:getType())`

static `BWAPI.Filter.`**`SizeType`**(*unit*) → UnitSizeType
Equivalent to `unit:getType():size()`

static `BWAPI.Filter.`**`GroundWeapon`**(*unit*) → WeaponType
Equivalent to `unit:getType():groundWeapon()`

static `BWAPI.Filter.`**`AirWeapon`**(*unit*) → WeaponType
Equivalent to `unit:getType():airWeapon()`

static `BWAPI.Filter.`**`GetType`**(*unit*) → UnitType
Equivalent to `unit:getType()`

static `BWAPI.Filter.`**`GetRace`**(*unit*) → Race
Equivalent to `unit:getType():getRace()`

static `BWAPI.Filter.`**`GetPlayer`**(*unit*) → Player
Equivalent to `unit:getPlayer()`

static `BWAPI.Filter.`**`Resources`**(*unit*) → int
Equivalent to `unit:getResources()`

static `BWAPI.Filter.`**`ResourceGroup`**(*unit*) → int
Equivalent to `unit:getResourceGroup()`

static `BWAPI.Filter.`**`AcidSporeCount`**(*unit*) → int
Equivalent to `unit:getAcidSporeCount()`

static `BWAPI.Filter.`**`InterceptorCount`**(*unit*) → int
Equivalent to `unit:getInterceptorCount()`

static `BWAPI.Filter.`**`ScarabCount`**(*unit*) → int
Equivalent to `unit:getScarabCount()`

static `BWAPI.Filter.`**`SpiderMineCount`**(*unit*) → int
Equivalent to `unit:getSpiderMineCount()`

static `BWAPI.Filter.`**`WeaponCooldown`**(*unit*) → int
Equivalent to `unit:getGroundWeaponCooldown()`

static `BWAPI.Filter.`**`SpellCooldown`**(*unit*) → int
Equivalent to `unit:getSpellCooldown()`

static `BWAPI.Filter.`**`DefenseMatrixPoints`**(*unit*) → int
Equivalent to `unit:getDefenseMatrixPoints()`

static `BWAPI.Filter.`**`DefenseMatrixTime`**(*unit*) → int
Equivalent to `unit:getDefenseMatrixTimer()`

static `BWAPI.Filter.`**`EnsnareTime`**(*unit*) → int
Equivalent to `unit:getEnsnareTimer()`

static `BWAPI.Filter.`**`IrradiateTime`**(*unit*) → int
Equivalent to `unit:getIrradiateTimer()`

static `BWAPI.Filter.`**`LockdownTime`**(*unit*) → int
Equivalent to `unit:getLockdownTimer()`

static `BWAPI.Filter.`**`MaelstromTime`**(*unit*) → int
Equivalent to `unit:getMaelstromTimer()`

static `BWAPI.Filter.`**`OrderTime`**(*unit*) → int
Equivalent to `unit:getOrderTimer()`

static `BWAPI.Filter.`**`PlagueTimer`**(*unit*) → int
Equivalent to `unit:getPlagueTimer()`

static `BWAPI.Filter.`**`RemoveTime`**(*unit*) → int
Equivalent to `unit:getRemoveTimer()`

static `BWAPI.Filter.`**`StasisTime`**(*unit*) → int
Equivalent to `unit:getStasisTimer()`

static `BWAPI.Filter.`**`StimTime`**(*unit*) → int
Equivalent to `unit:getStimTimer()`

static `BWAPI.Filter.`**`BuildType`**(*unit*) → UnitType
Equivalent to `unit:getBuildType()`

static `BWAPI.Filter.`**`RemainingBuildTime`**(*unit*) → int
Equivalent to `unit:getRemainingBuildTime()`

static `BWAPI.Filter.`**`RemainingTrainTime`**(*unit*) → int
Equivalent to `unit:getRemainingTrainTime()`

static `BWAPI.Filter.`**`Target`**(*unit*) → Unit
Equivalent to `unit:getTarget()`

static `BWAPI.Filter.`**`CurrentOrder`**(*unit*) → Order
Equivalent to `unit:getOrder()`

static `BWAPI.Filter.`**`SecondaryOrder`**`(`*unit*`)` → Order
Equivalent to `unit:getSecondaryOrder()`

static `BWAPI.Filter.`**`OrderTarget`**`(`*unit*`)` → Unit
Equivalent to `unit:getOrderTarget()`

static `BWAPI.Filter.`**`GetLeft`**`(`*unit*`)` → int
Equivalent to `unit:getLeft()`

static `BWAPI.Filter.`**`GetTop`**`(`*unit*`)` → int
Equivalent to `unit:getTop()`

static `BWAPI.Filter.`**`GetRight`**`(`*unit*`)` → int
Equivalent to `unit:getRight()`

static `BWAPI.Filter.`**`GetBottom`**`(`*unit*`)` → int
Equivalent to `unit:getBottom()`

Flag

Contains flag enumerations for BWAPI.

See also:

BWAPI.Game.enableFlag(), *BWAPI.Game.isFlagEnabled()*

Enum

`BWAPI.Flag.`**`CompleteMapInformation`**
Value of 0. Enable to get information about all units on the map, not just the visible units.

`BWAPI.Flag.`**`UserInput`**
Value of 1. Enable to get information from the user (what units are selected, chat messages the user enters, etc)

`BWAPI.Flag.`**`Max`**
Value of 2. The maximum number of different flags available.

GameTypes

Module containing game types.

See also:

BWAPI.GameType

Functions

static `BWAPI.GameTypes.`**`allGameTypes`**`()` → set
Retrieves a set of all GameTypes.

Returns Set of all GameTypes.

Return type *GameTypeset*

Constants

All constants are instances of the *GameType* class

BWAPI.GameTypes.**Melee**

BWAPI.GameTypes.**Free_For_All**

BWAPI.GameTypes.**One_on_One**

BWAPI.GameTypes.**Capture_The_Flag**

BWAPI.GameTypes.**Greed**

BWAPI.GameTypes.**Slaughter**

BWAPI.GameTypes.**Sudden_Death**

BWAPI.GameTypes.**Ladder**

BWAPI.GameTypes.**Use_Map_Settings**

BWAPI.GameTypes.**Team_Melee**

BWAPI.GameTypes.**Team_Free_For_All**

BWAPI.GameTypes.**Team_Capture_The_Flag**

BWAPI.GameTypes.**Top_vs_Bottom**

BWAPI.GameTypes.**None**

BWAPI.GameTypes.**Unknown**

Enum

Enumeration of game types.

Important: Enum values rarely need to be used in Lua.

BWAPI.GameTypes.Enum.**None**
: Value of 0.

BWAPI.GameTypes.Enum.**Custom**
: Value of 1.

BWAPI.GameTypes.Enum.**Melee**
: Value of 2.

BWAPI.GameTypes.Enum.**Free_For_All**
: Value of 3.

BWAPI.GameTypes.Enum.**One_on_One**
: Value of 4.

BWAPI.GameTypes.Enum.**Capture_The_Flag**
: Value of 5.

BWAPI.GameTypes.Enum.**Greed**
: Value of 6.

BWAPI.GameTypes.Enum.**Slaughter**
: Value of 7.

BWAPI.GameTypes.Enum.**Sudden_Death**
Value of 8.

BWAPI.GameTypes.Enum.**Ladder**
Value of 9.

BWAPI.GameTypes.Enum.**Use_Map_Settings**
Value of 10.

BWAPI.GameTypes.Enum.**Team_Melee**
Value of 11.

BWAPI.GameTypes.Enum.**Team_Free_For_All**
Value of 12.

BWAPI.GameTypes.Enum.**Team_Capture_The_Flag**
Value of 13.

BWAPI.GameTypes.Enum.**Unknown_0x0E**
Value of 14.

BWAPI.GameTypes.Enum.**Top_vs_Bottom**
Value of 15.

BWAPI.GameTypes.Enum.**Iron_Man_ladder**
Value of 16.

BWAPI.GameTypes.Enum.**Pro_Gamer_League**
Value of 32.

BWAPI.GameTypes.Enum.**Unknown**
Value of 33.

BWAPI.GameTypes.Enum.**MAX**
Value of 34.

Key

An enumeration of keyboard input values.

See also:

BWAPI.Game.getKeyState()

Enum

BWAPI.Key.**K_LBUTTON**
Value of 1

BWAPI.Key.**K_RBUTTON**
Value of 2

BWAPI.Key.**K_CANCEL**
Value of 3

BWAPI.Key.**K_MBUTTON**
Value of 4

BWAPI.Key.**K_XBUTTON1**
Value of 5

BWAPI.Key.**K_XBUTTON2**
Value of 6

BWAPI.Key.**__UNDEFINED_7**
Value of 7

BWAPI.Key.**K_BACK**
Value of 8

BWAPI.Key.**K_TAB**
Value of 9

BWAPI.Key.**__RESERVED_A**
Value of 10

BWAPI.Key.**__RESERVED_B**
Value of 11

BWAPI.Key.**K_CLEAR**
Value of 12

BWAPI.Key.**K_RETURN**
Value of 13

BWAPI.Key.**__UNDEFINED_E**
Value of 14

BWAPI.Key.**__UNDEFINED_F**
Value of 15

BWAPI.Key.**K_SHIFT**
Value of 16

BWAPI.Key.**K_CONTROL**
Value of 17

BWAPI.Key.**K_MENU**
Value of 18

BWAPI.Key.**K_PAUSE**
Value of 19

BWAPI.Key.**K_CAPITAL**
Value of 20

BWAPI.Key.**K_KANA**
Value of 21

BWAPI.Key.**K_UNDEFINED_16**
Value of 22

BWAPI.Key.**K_JUNJA**
Value of 23

BWAPI.Key.**K_FINAL**
Value of 24

BWAPI.Key.**K_KANJI**
Value of 25

BWAPI.Key.**__UNDEFINED_1A**
Value of 26

BWAPI.Key.**K_ESCAPE**
Value of 27

BWAPI.Key.**K_CONVERT**
Value of 28

BWAPI.Key.**K_NONCONVERT**
Value of 29

BWAPI.Key.**K_ACCEPT**
Value of 30

BWAPI.Key.**K_MODECHANGE**
Value of 31

BWAPI.Key.**K_SPACE**
Value of 32

BWAPI.Key.**K_PRIOR**
Value of 33

BWAPI.Key.**K_NEXT**
Value of 34

BWAPI.Key.**K_END**
Value of 35

BWAPI.Key.**K_HOME**
Value of 36

BWAPI.Key.**K_LEFT**
Value of 37

BWAPI.Key.**K_UP**
Value of 38

BWAPI.Key.**K_RIGHT**
Value of 39

BWAPI.Key.**K_DOWN**
Value of 40

BWAPI.Key.**K_SELECT**
Value of 41

BWAPI.Key.**K_PRINT**
Value of 42

BWAPI.Key.**K_EXECUTE**
Value of 43

BWAPI.Key.**K_SNAPSHOT**
Value of 44

BWAPI.Key.**K_INSERT**
Value of 45

BWAPI.Key.**K_DELETE**
Value of 46

BWAPI.Key.**K_HELP**
Value of 47

BWAPI.Key.**K_0**
Value of 48

BWAPI.Key.**K_1**
Value of 49

BWAPI.Key.**K_2**
Value of 50

BWAPI.Key.**K_3**
Value of 51

BWAPI.Key.**K_4**
Value of 52

BWAPI.Key.**K_5**
Value of 53

BWAPI.Key.**K_6**
Value of 54

BWAPI.Key.**K_7**
Value of 55

BWAPI.Key.**K_8**
Value of 56

BWAPI.Key.**K_9**
Value of 57

BWAPI.Key.**__UNDEFINED_3A**
Value of 58

BWAPI.Key.**__UNDEFINED_3B**
Value of 59

BWAPI.Key.**__UNDEFINED_3C**
Value of 60

BWAPI.Key.**__UNDEFINED_3D**
Value of 61

BWAPI.Key.**__UNDEFINED_3E**
Value of 62

BWAPI.Key.**__UNDEFINED_3F**
Value of 63

BWAPI.Key.**__UNDEFINED_40**
Value of 64

BWAPI.Key.**K_A**
Value of 65

BWAPI.Key.**K_B**
Value of 66

BWAPI.Key.**K_C**
Value of 67

BWAPI.Key.**K_D**
Value of 68

BWAPI.Key.**K_E**
Value of 69

BWAPI.Key.**K_F**
Value of 70

BWAPI.Key.**K_G**
Value of 71

BWAPI.Key.**K_H**
Value of 72

BWAPI.Key.**K_I**
Value of 73

BWAPI.Key.**K_J**
Value of 74

BWAPI.Key.**K_K**
Value of 75

BWAPI.Key.**K_L**
Value of 76

BWAPI.Key.**K_M**
Value of 77

BWAPI.Key.**K_N**
Value of 78

BWAPI.Key.**K_O**
Value of 79

BWAPI.Key.**K_P**
Value of 80

BWAPI.Key.**K_Q**
Value of 81

BWAPI.Key.**K_R**
Value of 82

BWAPI.Key.**K_S**
Value of 83

BWAPI.Key.**K_T**
Value of 84

BWAPI.Key.**K_U**
Value of 85

BWAPI.Key.**K_V**
Value of 86

BWAPI.Key.**K_W**
Value of 87

BWAPI.Key.**K_X**
Value of 88

BWAPI.Key.**K_Y**
Value of 89

BWAPI.Key.**K_Z**
Value of 90

BWAPI.Key.**K_LWIN**
Value of 91

BWAPI.Key.**K_RWIN**
Value of 92

BWAPI.Key.**K_APPS**
Value of 93

BWAPI.Key.**__RESERVED_5E**
Value of 94

BWAPI.Key.**K_SLEEP**
Value of 95

BWAPI.Key.**K_NUMPAD0**
Value of 96

BWAPI.Key.**K_NUMPAD1**
Value of 97

BWAPI.Key.**K_NUMPAD2**
Value of 98

BWAPI.Key.**K_NUMPAD3**
Value of 99

BWAPI.Key.**K_NUMPAD4**
Value of 100

BWAPI.Key.**K_NUMPAD5**
Value of 101

BWAPI.Key.**K_NUMPAD6**
Value of 102

BWAPI.Key.**K_NUMPAD7**
Value of 103

BWAPI.Key.**K_NUMPAD8**
Value of 104

BWAPI.Key.**K_NUMPAD9**
Value of 105

BWAPI.Key.**K_MULTIPLY**
Value of 106

BWAPI.Key.**K_ADD**
Value of 107

BWAPI.Key.**K_SEPARATOR**
Value of 108

BWAPI.Key.**K_SUBTRACT**
Value of 109

BWAPI.Key.**K_DECIMAL**
Value of 110

BWAPI.Key.**K_DIVIDE**
Value of 111

BWAPI.Key.**K_F1**
Value of 112

BWAPI.Key.**K_F2**
Value of 113

BWAPI.Key.**K_F3**
Value of 114

BWAPI.Key.**K_F4**
Value of 115

BWAPI.Key.**K_F5**
Value of 116

BWAPI.Key.**K_F6**
Value of 117

BWAPI.Key.**K_F7**
Value of 118

BWAPI.Key.**K_F8**
Value of 119

BWAPI.Key.**K_F9**
Value of 120

BWAPI.Key.**K_F10**
Value of 121

BWAPI.Key.**K_F11**
Value of 122

BWAPI.Key.**K_F12**
Value of 123

BWAPI.Key.**K_F13**
Value of 124

BWAPI.Key.**K_F14**
Value of 125

BWAPI.Key.**K_F15**
Value of 126

BWAPI.Key.**K_F16**
Value of 127

BWAPI.Key.**K_F17**
Value of 128

BWAPI.Key.**K_F18**
Value of 129

BWAPI.Key.**K_F19**
Value of 130

BWAPI.Key.**K_F20**
Value of 131

BWAPI.Key.**K_F21**
Value of 132

BWAPI.Key.**K_F22**
Value of 133

BWAPI.Key.**K_F23**
Value of 134

BWAPI.Key.**K_F24**
Value of 135

BWAPI.Key.**__UNASSIGNED_88**
Value of 136

BWAPI.Key.**__UNASSIGNED_89**
Value of 137

BWAPI.Key.**__UNASSIGNED_8A**
Value of 138

BWAPI.Key.**__UNASSIGNED_8B**
Value of 139

BWAPI.Key.**__UNASSIGNED_8C**
Value of 140

BWAPI.Key.**__UNASSIGNED_8D**
Value of 141

BWAPI.Key.**__UNASSIGNED_8E**
Value of 142

BWAPI.Key.**__UNASSIGNED_8F**
Value of 143

BWAPI.Key.**K_NUMLOCK**
Value of 144

BWAPI.Key.**K_SCROLL**
Value of 145

BWAPI.Key.**K_OEM_NEC_EQUAL**
Value of 146

BWAPI.Key.**K_OEM_FJ_JISHO**
Value of 147

BWAPI.Key.**K_OEM_FJ_MASSHOU**
Value of 148

BWAPI.Key.**K_OEM_FJ_TOUROKU**
Value of 149

BWAPI.Key.**K_OEM_FJ_LOYA**
Value of 150

BWAPI.Key.**__UNASSIGNED_97**
Value of 151

BWAPI.Key.**__UNASSIGNED_98**
Value of 152

BWAPI.Key.**__UNASSIGNED_99**
Value of 153

BWAPI.Key.**__UNASSIGNED_9A**
Value of 154

BWAPI.Key.**__UNASSIGNED_9B**
Value of 155

BWAPI.Key.**__UNASSIGNED_9C**
Value of 156

BWAPI.Key.**__UNASSIGNED_9D**
Value of 157

BWAPI.Key.**__UNASSIGNED_9E**
Value of 158

BWAPI.Key.**__UNASSIGNED_9F**
Value of 159

BWAPI.Key.**K_LSHIFT**
Value of 160

BWAPI.Key.**K_RSHIFT**
Value of 161

BWAPI.Key.**K_LCONTROL**
Value of 162

BWAPI.Key.**K_RCONTROL**
Value of 163

BWAPI.Key.**K_LMENU**
Value of 164

BWAPI.Key.**K_RMENU**
Value of 165

BWAPI.Key.**K_BROWSER_BACK**
Value of 166

BWAPI.Key.**K_BROWSER_FORWARD**
Value of 167

BWAPI.Key.**K_BROWSER_REFRESH**
Value of 168

BWAPI.Key.**K_BROWSER_STOP**
Value of 169

BWAPI.Key.**K_BROWSER_SEARCH**
Value of 170

BWAPI.Key.**K_BROWSER_FAVORITES**
Value of 171

BWAPI.Key.**K_BROWSER_HOME**
Value of 172

BWAPI.Key.**K_VOLUME_MUTE**
Value of 173

BWAPI.Key.**K_VOLUME_DOWN**
Value of 174

BWAPI.Key.**K_VOLUME_UP**
Value of 175

BWAPI.Key.**K_MEDIA_NEXT_TRACK**
Value of 176

BWAPI.Key.**K_MEDIA_PREV_TRACK**
Value of 177

BWAPI.Key.**K_MEDIA_STOP**
Value of 178

BWAPI.Key.**K_MEDIA_PLAY_PAUSE**
Value of 179

BWAPI.Key.**K_LAUNCH_MAIL**
Value of 180

BWAPI.Key.**K_LAUNCH_MEDIA_SELECT**
Value of 181

BWAPI.Key.**K_LAUNCH_APP1**
Value of 182

BWAPI.Key.**K_LAUNCH_APP2**
Value of 183

BWAPI.Key.**__RESERVED_B8**
Value of 184

BWAPI.Key.**__RESERVED_B9**
Value of 185

BWAPI.Key.**K_OEM_1**
Value of 186

BWAPI.Key.**K_OEM_PLUS**
Value of 187

BWAPI.Key.**K_OEM_COMMA**
Value of 188

BWAPI.Key.**K_OEM_MINUS**
Value of 189

BWAPI.Key.**K_OEM_PERIOD**
Value of 190

BWAPI.Key.**K_OEM_2**
Value of 191

BWAPI.Key.**K_OEM_3**
Value of 192

BWAPI.Key.**K_OEM_4**
Value of 219

BWAPI.Key.**K_OEM_5**
Value of 220

BWAPI.Key.**K_OEM_6**
Value of 221

BWAPI.Key.**K_OEM_7**
Value of 222

BWAPI.Key.**K_OEM_8**
Value of 223

BWAPI.Key.**__RESERVED_E0**
Value of 224

BWAPI.Key.**K_OEM_AX**
Value of 225

BWAPI.Key.**K_OEM_102**
Value of 226

BWAPI.Key.**K_ICO_HELP**
Value of 227

BWAPI.Key.**K_ICO_00**
Value of 228

BWAPI.Key.**K_PROCESSKEY**
Value of 229

BWAPI.Key.**K_ICO_CLEAR**
Value of 230

BWAPI.Key.**K_PACKET**
Value of 231

BWAPI.Key.**__UNASSIGNED_E8**
Value of 232

BWAPI.Key.**K_OEM_RESET**
Value of 233

BWAPI.Key.**K_OEM_JUMP**
Value of 234

BWAPI.Key.**K_OEM_PA1**
Value of 235

BWAPI.Key.**K_OEM_PA2**
Value of 236

BWAPI.Key.**K_OEM_PA3**
Value of 237

BWAPI.Key.**K_OEM_WSCTRL**
Value of 238

BWAPI.Key.**K_OEM_CUSEL**
Value of 239

BWAPI.Key.**K_OEM_ATTN**
Value of 240

BWAPI.Key.**K_OEM_FINISH**
Value of 241

BWAPI.Key.**K_OEM_COPY**
Value of 242

BWAPI.Key.**K_OEM_AUTO**
Value of 243

BWAPI.Key.**K_OEM_ENLW**
Value of 244

BWAPI.Key.**K_OEM_BACKTAB**
Value of 245

BWAPI.Key.**K_ATTN**
Value of 246

BWAPI.Key.**K_CRSEL**
Value of 247

BWAPI.Key.**K_EXSEL**
Value of 248

BWAPI.Key.**K_EREOF**
Value of 249

BWAPI.Key.**K_PLAY**
Value of 250

BWAPI.Key.**K_ZOOM**
Value of 251

BWAPI.Key.**K_NONAME**
Value of 252

BWAPI.Key.**K_PA1**
Value of 253

BWAPI.Key.**K_OEM_CLEAR**
Value of 254

BWAPI.Key.**K_MAX**
Value of 255

Latency

Contains enumeration of known latency values.

See also:

BWAPI.Game.getLatency()

Enum

BWAPI.Latency.**SinglePlayer**
Value of 2.

BWAPI.Latency.**LanLow**
Value of 5.

BWAPI.Latency.**LanMedium**
Value of 7.

BWAPI.Latency.**LanHigh**
Value of 9.

BWAPI.Latency.**BattlenetLow**
Value of 14.

BWAPI.Latency.**BattlenetMedium**
Value of 19.

BWAPI.Latency.**BattlenetHigh**
Value of 24.

MouseButton

An enumeration of mouse button inputs.

See also:

BWAPI.Game.getMouseState()

Enum

BWAPI.MouseButton.**M_LEFT**
Value of 0.

BWAPI.MouseButton.**M_RIGHT**
Value of 1.

BWAPI.MouseButton.**M_MIDDLE**
Value of 2.

BWAPI.MouseButton.**M_MAX**
Value of 3.

Orders

Module containing unit orders.

See also:

BWAPI.Order

Functions

static BWAPI.Orders.**allOrders**() → set
Retrieves a set of all Orders.

Returns Set of all Orders.

Return type *Orderset*

Constants

All constants are instances of the *Order* class

BWAPI.Orders.**Die**

BWAPI.Orders.**Stop**

BWAPI.Orders.**Guard**

BWAPI.Orders.**PlayerGuard**

BWAPI.Orders.**TurretGuard**

BWAPI.Orders.**BunkerGuard**

BWAPI.Orders.**Move**

BWAPI.Orders.**AttackUnit**

BWAPI.Orders.**AttackTile**

BWAPI.Orders.**Hover**

BWAPI.Orders.**AttackMove**

BWAPI.Orders.**InfestedCommandCenter**

BWAPI.Orders.**UnusedNothing**

BWAPI.Orders.**UnusedPowerup**

BWAPI.Orders.**TowerGuard**

BWAPI.Orders.**VultureMine**

BWAPI.Orders.**Nothing**

BWAPI.Orders.**CastInfestation**

BWAPI.Orders.**InfestingCommandCenter**

BWAPI.Orders.**PlaceBuilding**

BWAPI.Orders.**CreateProtossBuilding**

BWAPI.Orders.**ConstructingBuilding**

BWAPI.Orders.**Repair**

BWAPI.Orders.**PlaceAddon**

BWAPI.Orders.**BuildAddon**

BWAPI.Orders.**Train**

BWAPI.Orders.**RallyPointUnit**

BWAPI.Orders.**RallyPointTile**

BWAPI.Orders.**ZergBirth**

BWAPI.Orders.**ZergUnitMorph**

BWAPI.Orders.**ZergBuildingMorph**

BWAPI.Orders.**IncompleteBuilding**

BWAPI.Orders.**BuildNydusExit**

BWAPI.Orders.**EnterNydusCanal**

BWAPI.Orders.**Follow**

BWAPI.Orders.**Carrier**

BWAPI.Orders.**ReaverCarrierMove**

BWAPI.Orders.**CarrierIgnore2**

BWAPI.Orders.**Reaver**

BWAPI.Orders.**TrainFighter**

BWAPI.Orders.**InterceptorAttack**

BWAPI.Orders.**ScarabAttack**

BWAPI.Orders.**RechargeShieldsUnit**

BWAPI.Orders.**RechargeShieldsBattery**

BWAPI.Orders.**ShieldBattery**

BWAPI.Orders.**InterceptorReturn**

BWAPI.Orders.**BuildingLand**

BWAPI.Orders.**BuildingLiftOff**

BWAPI.Orders.**DroneLiftOff**

BWAPI.Orders.**LiftingOff**

BWAPI.Orders.**ResearchTech**

BWAPI.Orders.**Upgrade**

BWAPI.Orders.**Larva**

BWAPI.Orders.**SpawningLarva**

BWAPI.Orders.**Harvest1**

BWAPI.Orders.**Harvest2**

BWAPI.Orders.**MoveToGas**

BWAPI.Orders.**WaitForGas**

BWAPI.Orders.**HarvestGas**

BWAPI.Orders.**ReturnGas**

BWAPI.Orders.**MoveToMinerals**

BWAPI.Orders.**WaitForMinerals**

BWAPI.Orders.**MiningMinerals**

BWAPI.Orders.**Harvest3**

BWAPI.Orders.**Harvest4**

BWAPI.Orders.**ReturnMinerals**

BWAPI.Orders.**Interrupted**

BWAPI.Orders.**EnterTransport**

BWAPI.Orders.**PickupIdle**

BWAPI.Orders.**PickupTransport**

BWAPI.Orders.**PickupBunker**

BWAPI.Orders.**Pickup4**

BWAPI.Orders.**PowerupIdle**

BWAPI.Orders.**Sieging**

BWAPI.Orders.**Unsieging**

BWAPI.Orders.**InitCreepGrowth**

BWAPI.Orders.**SpreadCreep**

BWAPI.Orders.**StoppingCreepGrowth**

BWAPI.Orders.**GuardianAspect**

BWAPI.Orders.**ArchonWarp**

BWAPI.Orders.**CompletingArchonSummon**

BWAPI.Orders.**HoldPosition**

BWAPI.Orders.**Cloak**

BWAPI.Orders.**Decloak**

BWAPI.Orders.**Unload**

BWAPI.Orders.**MoveUnload**

BWAPI.Orders.**FireYamatoGun**

BWAPI.Orders.**CastLockdown**

BWAPI.Orders.**Burrowing**

BWAPI.Orders.**Burrowed**

BWAPI.Orders.**Unburrowing**

BWAPI.Orders.**CastDarkSwarm**

BWAPI.Orders.**CastParasite**

BWAPI.Orders.**CastSpawnBroodlings**

BWAPI.Orders.**CastEMPShockwave**

BWAPI.Orders.**NukeWait**

BWAPI.Orders.**NukeTrain**

BWAPI.Orders.**NukeLaunch**

BWAPI.Orders.**NukePaint**

BWAPI.Orders.**NukeUnit**

BWAPI.Orders.**CastNuclearStrike**

BWAPI.Orders.**NukeTrack**

BWAPI.Orders.**CloakNearbyUnits**

BWAPI.Orders.**PlaceMine**

BWAPI.Orders.**RightClickAction**

BWAPI.Orders.**CastRecall**

BWAPI.Orders.**Teleport**

BWAPI.Orders.**CastScannerSweep**

BWAPI.Orders.**Scanner**

BWAPI.Orders.**CastDefensiveMatrix**

BWAPI.Orders.**CastPsionicStorm**

BWAPI.Orders.**CastIrradiate**

BWAPI.Orders.**CastPlague**

BWAPI.Orders.**CastConsume**

BWAPI.Orders.**CastEnsnare**

BWAPI.Orders.**CastStasisField**

BWAPI.Orders.**CastHallucination**

BWAPI.Orders.**Hallucination2**

BWAPI.Orders.**ResetCollision**

BWAPI.Orders.**Patrol**

BWAPI.Orders.**CTFCOPInit**

BWAPI.Orders.**CTFCOPStarted**

BWAPI.Orders.**CTFCOP2**

BWAPI.Orders.**ComputerAI**

BWAPI.Orders.**AtkMoveEP**

BWAPI.Orders.**HarassMove**

BWAPI.Orders.**AIPatrol**

BWAPI.Orders.**GuardPost**

BWAPI.Orders.**RescuePassive**

BWAPI.Orders.**Neutral**

BWAPI.Orders.**ComputerReturn**

BWAPI.Orders.**SelfDestructing**

BWAPI.Orders.**Critter**

BWAPI.Orders.**HiddenGun**

BWAPI.Orders.**OpenDoor**

BWAPI.Orders.**CloseDoor**

BWAPI.Orders.**HideTrap**

BWAPI.Orders.**RevealTrap**

BWAPI.Orders.**EnableDoodad**

BWAPI.Orders.**DisableDoodad**

BWAPI.Orders.**WarpIn**

BWAPI.Orders.**Medic**

BWAPI.Orders.**MedicHeal**

BWAPI.Orders.**HealMove**

BWAPI.Orders.**MedicHealToIdle**

BWAPI.Orders.**CastRestoration**

BWAPI.Orders.**CastDisruptionWeb**

BWAPI.Orders.**CastMindControl**

BWAPI.Orders.**DarkArchonMeld**

BWAPI.Orders.**CastFeedback**

BWAPI.Orders.**CastOpticalFlare**

BWAPI.Orders.**CastMaelstrom**

BWAPI.Orders.**JunkYardDog**

BWAPI.Orders.**Fatal**

BWAPI.Orders.**None**

BWAPI.Orders.**Unknown**

Enum

Enumeration of unit orders.

Important: Enum values rarely need to be used in Lua.

BWAPI.Orders.Enum.**Die**
: Value of 0.

BWAPI.Orders.Enum.**Stop**
: Value of 1.

BWAPI.Orders.Enum.**Guard**
: Value of 2.

BWAPI.Orders.Enum.**PlayerGuard**
: Value of 3.

BWAPI.Orders.Enum.**TurretGuard**
: Value of 4.

BWAPI.Orders.Enum.**BunkerGuard**
: Value of 5.

BWAPI.Orders.Enum.**Move**
: Value of 6.

BWAPI.Orders.Enum.**ReaverStop**
: Value of 7.

BWAPI.Orders.Enum.**Attack1**
: Value of 8.

BWAPI.Orders.Enum.**Attack2**
: Value of 9.

BWAPI.Orders.Enum.**AttackUnit**
: Value of 10.

BWAPI.Orders.Enum.**AttackFixedRange**
: Value of 11.

BWAPI.Orders.Enum.**AttackTile**
Value of 12.

BWAPI.Orders.Enum.**Hover**
Value of 13.

BWAPI.Orders.Enum.**AttackMove**
Value of 14.

BWAPI.Orders.Enum.**InfestedCommandCenter**
Value of 15.

BWAPI.Orders.Enum.**UnusedNothing**
Value of 16.

BWAPI.Orders.Enum.**UnusedPowerup**
Value of 17.

BWAPI.Orders.Enum.**TowerGuard**
Value of 18.

BWAPI.Orders.Enum.**TowerAttack**
Value of 19.

BWAPI.Orders.Enum.**VultureMine**
Value of 20.

BWAPI.Orders.Enum.**StayInRange**
Value of 21.

BWAPI.Orders.Enum.**TurretAttack**
Value of 22.

BWAPI.Orders.Enum.**Nothing**
Value of 23.

BWAPI.Orders.Enum.**Unused_24**
Value of 24.

BWAPI.Orders.Enum.**DroneStartBuild**
Value of 25.

BWAPI.Orders.Enum.**DroneBuild**
Value of 26.

BWAPI.Orders.Enum.**CastInfestation**
Value of 27.

BWAPI.Orders.Enum.**MoveToInfest**
Value of 28.

BWAPI.Orders.Enum.**InfestingCommandCenter**
Value of 29.

BWAPI.Orders.Enum.**PlaceBuilding**
Value of 30.

BWAPI.Orders.Enum.**PlaceProtossBuilding**
Value of 31.

BWAPI.Orders.Enum.**CreateProtossBuilding**
Value of 32.

BWAPI.Orders.Enum.**ConstructingBuilding**
Value of 33.

BWAPI.Orders.Enum.**Repair**
Value of 34.

BWAPI.Orders.Enum.**MoveToRepair**
Value of 35.

BWAPI.Orders.Enum.**PlaceAddon**
Value of 36.

BWAPI.Orders.Enum.**BuildAddon**
Value of 37.

BWAPI.Orders.Enum.**Train**
Value of 38.

BWAPI.Orders.Enum.**RallyPointUnit**
Value of 39.

BWAPI.Orders.Enum.**RallyPointTile**
Value of 40.

BWAPI.Orders.Enum.**ZergBirth**
Value of 41.

BWAPI.Orders.Enum.**ZergUnitMorph**
Value of 42.

BWAPI.Orders.Enum.**ZergBuildingMorph**
Value of 43.

BWAPI.Orders.Enum.**IncompleteBuilding**
Value of 44.

BWAPI.Orders.Enum.**IncompleteMorphing**
Value of 45.

BWAPI.Orders.Enum.**BuildNydusExit**
Value of 46.

BWAPI.Orders.Enum.**EnterNydusCanal**
Value of 47.

BWAPI.Orders.Enum.**IncompleteWarping**
Value of 48.

BWAPI.Orders.Enum.**Follow**
Value of 49.

BWAPI.Orders.Enum.**Carrier**
Value of 50.

BWAPI.Orders.Enum.**ReaverCarrierMove**
Value of 51.

BWAPI.Orders.Enum.**CarrierStop**
Value of 52.

BWAPI.Orders.Enum.**CarrierAttack**
Value of 53.

BWAPI.Orders.Enum.**CarrierMoveToAttack**
Value of 54.

BWAPI.Orders.Enum.**CarrierIgnore2**
Value of 55.

BWAPI.Orders.Enum.**CarrierFight**
Value of 56.

BWAPI.Orders.Enum.**CarrierHoldPosition**
Value of 57.

BWAPI.Orders.Enum.**Reaver**
Value of 58.

BWAPI.Orders.Enum.**ReaverAttack**
Value of 59.

BWAPI.Orders.Enum.**ReaverMoveToAttack**
Value of 60.

BWAPI.Orders.Enum.**ReaverFight**
Value of 61.

BWAPI.Orders.Enum.**ReaverHoldPosition**
Value of 62.

BWAPI.Orders.Enum.**TrainFighter**
Value of 63.

BWAPI.Orders.Enum.**InterceptorAttack**
Value of 64.

BWAPI.Orders.Enum.**ScarabAttack**
Value of 65.

BWAPI.Orders.Enum.**RechargeShieldsUnit**
Value of 66.

BWAPI.Orders.Enum.**RechargeShieldsBattery**
Value of 67.

BWAPI.Orders.Enum.**ShieldBattery**
Value of 68.

BWAPI.Orders.Enum.**InterceptorReturn**
Value of 69.

BWAPI.Orders.Enum.**DroneLand**
Value of 70.

BWAPI.Orders.Enum.**BuildingLand**
Value of 71.

BWAPI.Orders.Enum.**BuildingLiftOff**
Value of 72.

BWAPI.Orders.Enum.**DroneLiftOff**
Value of 73.

BWAPI.Orders.Enum.**LiftingOff**
Value of 74.

BWAPI.Orders.Enum.**ResearchTech**
Value of 75.

BWAPI.Orders.Enum.**Upgrade**
Value of 76.

BWAPI.Orders.Enum.**Larva**
Value of 77.

BWAPI.Orders.Enum.**SpawningLarva**
Value of 78.

BWAPI.Orders.Enum.**Harvest1**
Value of 79.

BWAPI.Orders.Enum.**Harvest2**
Value of 80.

BWAPI.Orders.Enum.**MoveToGas**
Value of 81.

BWAPI.Orders.Enum.**WaitForGas**
Value of 82.

BWAPI.Orders.Enum.**HarvestGas**
Value of 83.

BWAPI.Orders.Enum.**ReturnGas**
Value of 84.

BWAPI.Orders.Enum.**MoveToMinerals**
Value of 85.

BWAPI.Orders.Enum.**WaitForMinerals**
Value of 86.

BWAPI.Orders.Enum.**MiningMinerals**
Value of 87.

BWAPI.Orders.Enum.**Harvest3**
Value of 88.

BWAPI.Orders.Enum.**Harvest4**
Value of 89.

BWAPI.Orders.Enum.**ReturnMinerals**
Value of 90.

BWAPI.Orders.Enum.**Interrupted**
Value of 91.

BWAPI.Orders.Enum.**EnterTransport**
Value of 92.

BWAPI.Orders.Enum.**PickupIdle**
Value of 93.

BWAPI.Orders.Enum.**PickupTransport**
Value of 94.

BWAPI.Orders.Enum.**PickupBunker**
Value of 95.

BWAPI.Orders.Enum.**Pickup4**
Value of 96.

BWAPI.Orders.Enum.**PowerupIdle**
Value of 97.

BWAPI.Orders.Enum.**Sieging**
Value of 98.

BWAPI.Orders.Enum.**Unsieging**
Value of 99.

BWAPI.Orders.Enum.**WatchTarget**
Value of 100.

BWAPI.Orders.Enum.**InitCreepGrowth**
Value of 101.

BWAPI.Orders.Enum.**SpreadCreep**
Value of 102.

BWAPI.Orders.Enum.**StoppingCreepGrowth**
Value of 103.

BWAPI.Orders.Enum.**GuardianAspect**
Value of 104.

BWAPI.Orders.Enum.**ArchonWarp**
Value of 105.

BWAPI.Orders.Enum.**CompletingArchonSummon**
Value of 106.

BWAPI.Orders.Enum.**HoldPosition**
Value of 107.

BWAPI.Orders.Enum.**QueenHoldPosition**
Value of 108.

BWAPI.Orders.Enum.**Cloak**
Value of 109.

BWAPI.Orders.Enum.**Decloak**
Value of 110.

BWAPI.Orders.Enum.**Unload**
Value of 111.

BWAPI.Orders.Enum.**MoveUnload**
Value of 112.

BWAPI.Orders.Enum.**FireYamatoGun**
Value of 113.

BWAPI.Orders.Enum.**MoveToFireYamatoGun**
Value of 114.

BWAPI.Orders.Enum.**CastLockdown**
Value of 115.

BWAPI.Orders.Enum.**Burrowing**
Value of 116.

BWAPI.Orders.Enum.**Burrowed**
Value of 117.

BWAPI.Orders.Enum.**Unburrowing**
Value of 118.

BWAPI.Orders.Enum.**CastDarkSwarm**
Value of 119.

BWAPI.Orders.Enum.**CastParasite**
Value of 120.

BWAPI.Orders.Enum.**CastSpawnBroodlings**
Value of 121.

BWAPI.Orders.Enum.**CastEMPShockwave**
Value of 122.

BWAPI.Orders.Enum.**NukeWait**
Value of 123.

BWAPI.Orders.Enum.**NukeTrain**
Value of 124.

BWAPI.Orders.Enum.**NukeLaunch**
Value of 125.

BWAPI.Orders.Enum.**NukePaint**
Value of 126.

BWAPI.Orders.Enum.**NukeUnit**
Value of 127.

BWAPI.Orders.Enum.**CastNuclearStrike**
Value of 128.

BWAPI.Orders.Enum.**NukeTrack**
Value of 129.

BWAPI.Orders.Enum.**InitializeArbiter**
Value of 130.

BWAPI.Orders.Enum.**CloakNearbyUnits**
Value of 131.

BWAPI.Orders.Enum.**PlaceMine**
Value of 132.

BWAPI.Orders.Enum.**RightClickAction**
Value of 133.

BWAPI.Orders.Enum.**SuicideUnit**
Value of 134.

BWAPI.Orders.Enum.**SuicideLocation**
Value of 135.

BWAPI.Orders.Enum.**SuicideHoldPosition**
Value of 136.

BWAPI.Orders.Enum.**CastRecall**
Value of 137.

BWAPI.Orders.Enum.**Teleport**
Value of 138.

BWAPI.Orders.Enum.**CastScannerSweep**
Value of 139.

BWAPI.Orders.Enum.**Scanner**
Value of 140.

BWAPI.Orders.Enum.**CastDefensiveMatrix**
Value of 141.

BWAPI.Orders.Enum.**CastPsionicStorm**
Value of 142.

BWAPI.Orders.Enum.**CastIrradiate**
Value of 143.

BWAPI.Orders.Enum.**CastPlague**
Value of 144.

BWAPI.Orders.Enum.**CastConsume**
Value of 145.

BWAPI.Orders.Enum.**CastEnsnare**
Value of 146.

BWAPI.Orders.Enum.**CastStasisField**
Value of 147.

BWAPI.Orders.Enum.**CastHallucination**
Value of 148.

BWAPI.Orders.Enum.**Hallucination2**
Value of 149.

BWAPI.Orders.Enum.**ResetCollision**
Value of 150.

BWAPI.Orders.Enum.**ResetHarvestCollision**
Value of 151.

BWAPI.Orders.Enum.**Patrol**
Value of 152.

BWAPI.Orders.Enum.**CTFCOPInit**
Value of 153.

BWAPI.Orders.Enum.**CTFCOPStarted**
Value of 154.

BWAPI.Orders.Enum.**CTFCOP2**
Value of 155.

BWAPI.Orders.Enum.**ComputerAI**
Value of 156.

BWAPI.Orders.Enum.**AtkMoveEP**
Value of 157.

BWAPI.Orders.Enum.**HarassMove**
Value of 158.

BWAPI.Orders.Enum.**AIPatrol**
Value of 159.

BWAPI.Orders.Enum.**GuardPost**
Value of 160.

BWAPI.Orders.Enum.**RescuePassive**
Value of 161.

BWAPI.Orders.Enum.**Neutral**
Value of 162.

BWAPI.Orders.Enum.**ComputerReturn**
Value of 163.

BWAPI.Orders.Enum.**InitializePsiProvider**
Value of 164.

BWAPI.Orders.Enum.**SelfDestructing**
Value of 165.

BWAPI.Orders.Enum.**Critter**
Value of 166.

BWAPI.Orders.Enum.**HiddenGun**
Value of 167.

BWAPI.Orders.Enum.**OpenDoor**
Value of 168.

BWAPI.Orders.Enum.**CloseDoor**
Value of 169.

BWAPI.Orders.Enum.**HideTrap**
Value of 170.

BWAPI.Orders.Enum.**RevealTrap**
Value of 171.

BWAPI.Orders.Enum.**EnableDoodad**
Value of 172.

BWAPI.Orders.Enum.**DisableDoodad**
Value of 173.

BWAPI.Orders.Enum.**WarpIn**
Value of 174.

BWAPI.Orders.Enum.**Medic**
Value of 175.

BWAPI.Orders.Enum.**MedicHeal**
Value of 176.

BWAPI.Orders.Enum.**HealMove**
Value of 177.

BWAPI.Orders.Enum.**MedicHoldPosition**
Value of 178.

BWAPI.Orders.Enum.**MedicHealToIdle**
Value of 179.

BWAPI.Orders.Enum.**CastRestoration**
Value of 180.

BWAPI.Orders.Enum.**CastDisruptionWeb**
Value of 181.

BWAPI.Orders.Enum.**CastMindControl**
Value of 182.

BWAPI.Orders.Enum.**DarkArchonMeld**
Value of 183.

BWAPI.Orders.Enum.**CastFeedback**
Value of 184.

BWAPI.Orders.Enum.**CastOpticalFlare**
Value of 185.

BWAPI.Orders.Enum.**CastMaelstrom**
Value of 186.

BWAPI.Orders.Enum.**JunkYardDog**
Value of 187.

BWAPI.Orders.Enum.**Fatal**
Value of 188.

BWAPI.Orders.Enum.**None**
Value of 189.

BWAPI.Orders.Enum.**Unknown**
Value of 190.

BWAPI.Orders.Enum.**MAX**
Value of 191.

PlayerTypes

Module containing player types.

See also:

BWAPI.PlayerType

Functions

static BWAPI.PlayerTypes.**allPlayerTypes**() → set
Retrieves a set of all PlayerTypes.

Returns Set of all PlayerTypes.

Return type *PlayerTypeset*

Constants

All constants are instances of the *PlayerType* class

BWAPI.PlayerTypes.**None**

BWAPI.PlayerTypes.**Computer**

BWAPI.PlayerTypes.**Player**

BWAPI.PlayerTypes.**RescuePassive**

BWAPI.PlayerTypes.**EitherPreferComputer**

BWAPI.PlayerTypes.**EitherPreferHuman**

BWAPI.PlayerTypes.**Neutral**

BWAPI.PlayerTypes.**Closed**

BWAPI.PlayerTypes.**PlayerLeft**

BWAPI.PlayerTypes.**ComputerLeft**

BWAPI.PlayerTypes.**Unknown**

Enum

Enumeration of player types.

Important: Enum values rarely need to be used in Lua.

BWAPI.PlayerTypes.Enum.**None**
Value of 0.

BWAPI.PlayerTypes.Enum.**Computer**
Value of 1.

BWAPI.PlayerTypes.Enum.**Player**
Value of 2.

BWAPI.PlayerTypes.Enum.**RescuePassive**
Value of 3.

BWAPI.PlayerTypes.Enum.**RescueActive**
Value of 4.

BWAPI.PlayerTypes.Enum.**EitherPreferComputer**
Value of 5.

BWAPI.PlayerTypes.Enum.**EitherPreferHuman**
Value of 6.

BWAPI.PlayerTypes.Enum.**Neutral**
Value of 7.

BWAPI.PlayerTypes.Enum.**Closed**
Value of 8.

BWAPI.PlayerTypes.Enum.**Observer**
Value of 9.

BWAPI.PlayerTypes.Enum.**PlayerLeft**
Value of 10.

BWAPI.PlayerTypes.Enum.**ComputerLeft**
Value of 11.

BWAPI.PlayerTypes.Enum.**Unknown**
Value of 12.

BWAPI.PlayerTypes.Enum.**MAX**
Value of 13.

Positions

Module containing special position constants.

See also:

Position

Constants

All constants are instances of the *Position* class

BWAPI.Positions.**Invalid**

BWAPI.Positions.**None**

BWAPI.Positions.**Unknown**

BWAPI.Positions.**Origin**
(0,0)

WalkPositions

Module containing special position constants.

See also:

WalkPosition

Constants

All constants are instances of the *WalkPosition* class

BWAPI.WalkPositions.**Invalid**

BWAPI.WalkPositions.**None**

BWAPI.WalkPositions.**Unknown**

BWAPI.WalkPositions.**Origin**
(0,0)

TilePositions

Module containing special position constants.

See also:

TilePosition

Constants

All constants are instances of the *TilePosition* class

`BWAPI.TilePositions.`**`Invalid`**

`BWAPI.TilePositions.`**`None`**

`BWAPI.TilePositions.`**`Unknown`**

`BWAPI.TilePositions.`**`Origin`**
(0,0)

Races

Module containing races.

See also:

BWAPI.Race

Functions

static `BWAPI.Races.`**`allRaces`**`()` → set
Retrieves a set of all Races.

Returns Set of all Races.

Return type *Raceset*

Constants

All constants are instances of the *Race* class

`BWAPI.Races.`**`Zerg`**

- View on Liquipedia
- View on Starcraft Campendium (Official Website)
- View on Starcraft Wikia

`BWAPI.Races.`**`Terran`**

- View on Liquipedia
- View on Starcraft Campendium (Official Website)
- View on Starcraft Wikia

`BWAPI.Races.`**`Protoss`**

- View on Liquipedia
- View on Starcraft Campendium (Official Website)
- View on Starcraft Wikia

`BWAPI.Races.`**`Random`**

`BWAPI.Races.`**`None`**

`BWAPI.Races.`**`Unknown`**

Enum

Enumeration of races.

Important: Enum values rarely need to be used in Lua.

`BWAPI.Races.Enum.`**`Zerg`**
Value of 0.

`BWAPI.Races.Enum.`**`Terran`**
Value of 1.

`BWAPI.Races.Enum.`**`Protoss`**
Value of 2.

`BWAPI.Races.Enum.`**`Other`**
Value of 3.

`BWAPI.Races.Enum.`**`Unused`**
Value of 4.

`BWAPI.Races.Enum.`**`Select`**
Value of 5.

`BWAPI.Races.Enum.`**`Random`**
Value of 6.

`BWAPI.Races.Enum.`**`None`**
Value of 7.

`BWAPI.Races.Enum.`**`Unknown`**
Value of 8.

`BWAPI.Races.Enum.`**`MAX`**
Value of 9.

TechTypes

Module containing tech types.

See also:

BWAPI.TechType

Functions

static `BWAPI.TechTypes.`**`allTechTypes`**`()` → set
Retrieves a set of all TechTypes.

Returns Set of all TechTypes.

Return type *TechTypeset*

Constants

All constants are instances of the *TechType* class

Terran Abilities

BWAPI.TechTypes.**Stim_Packs**

BWAPI.TechTypes.**Lockdown**

BWAPI.TechTypes.**EMP_Shockwave**

BWAPI.TechTypes.**Spider_Mines**

BWAPI.TechTypes.**Scanner_Sweep**

BWAPI.TechTypes.**Tank_Siege_Mode**

BWAPI.TechTypes.**Defensive_Matrix**

BWAPI.TechTypes.**Irradiate**

BWAPI.TechTypes.**Yamato_Gun**

BWAPI.TechTypes.**Cloaking_Field**

BWAPI.TechTypes.**Personnel_Cloaking**

BWAPI.TechTypes.**Restoration**

BWAPI.TechTypes.**Optical_Flare**

BWAPI.TechTypes.**Healing**

BWAPI.TechTypes.**Nuclear_Strike**

Zerg Abilities

BWAPI.TechTypes.**Burrowing**

BWAPI.TechTypes.**Infestation**

BWAPI.TechTypes.**Spawn_Broodlings**

BWAPI.TechTypes.**Dark_Swarm**

BWAPI.TechTypes.**Plague**

BWAPI.TechTypes.**Consume**

BWAPI.TechTypes.**Ensnare**

BWAPI.TechTypes.**Parasite**

BWAPI.TechTypes.**Lurker_Aspect**

Protoss Abilities

BWAPI.TechTypes.**Psionic_Storm**

BWAPI.TechTypes.**Hallucination**

BWAPI.TechTypes.**Recall**

BWAPI.TechTypes.**Stasis_Field**

BWAPI.TechTypes.**Archon_Warp**

BWAPI.TechTypes.**Disruption_Web**

BWAPI.TechTypes.**Mind_Control**

BWAPI.TechTypes.**Dark_Archon_Meld**

BWAPI.TechTypes.**Feedback**

BWAPI.TechTypes.**Maelstrom**

Misc

BWAPI.TechTypes.**None**

BWAPI.TechTypes.**Unknown**

Enum

Enumeration of tech types.

Important: Enum values rarely need to be used in Lua.

BWAPI.TechTypes.Enum.**Stim_Packs**
Value of 0.

BWAPI.TechTypes.Enum.**Lockdown**
Value of 1.

BWAPI.TechTypes.Enum.**EMP_Shockwave**
Value of 2.

BWAPI.TechTypes.Enum.**Spider_Mines**
Value of 3.

BWAPI.TechTypes.Enum.**Scanner_Sweep**
Value of 4.

BWAPI.TechTypes.Enum.**Tank_Siege_Mode**
Value of 5.

BWAPI.TechTypes.Enum.**Defensive_Matrix**
Value of 6.

BWAPI.TechTypes.Enum.**Irradiate**
Value of 7.

BWAPI.TechTypes.Enum.**Yamato_Gun**
Value of 8.

BWAPI.TechTypes.Enum.**Cloaking_Field**
Value of 9.

BWAPI.TechTypes.Enum.**Personnel_Cloaking**
Value of 10.

BWAPI.TechTypes.Enum.**Burrowing**
Value of 11.

BWAPI.TechTypes.Enum.**Infestation**
Value of 12.

BWAPI.TechTypes.Enum.**Spawn_Broodlings**
Value of 13.

BWAPI.TechTypes.Enum.**Dark_Swarm**
Value of 14.

BWAPI.TechTypes.Enum.**Plague**
Value of 15.

BWAPI.TechTypes.Enum.**Consume**
Value of 16.

BWAPI.TechTypes.Enum.**Ensnare**
Value of 17.

BWAPI.TechTypes.Enum.**Parasite**
Value of 18.

BWAPI.TechTypes.Enum.**Psionic_Storm**
Value of 19.

BWAPI.TechTypes.Enum.**Hallucination**
Value of 20.

BWAPI.TechTypes.Enum.**Recall**
Value of 21.

BWAPI.TechTypes.Enum.**Stasis_Field**
Value of 22.

BWAPI.TechTypes.Enum.**Archon_Warp**
Value of 23.

BWAPI.TechTypes.Enum.**Restoration**
Value of 24.

BWAPI.TechTypes.Enum.**Disruption_Web**
Value of 25.

BWAPI.TechTypes.Enum.**Unused_26**
Value of 26.

BWAPI.TechTypes.Enum.**Mind_Control**
Value of 27.

BWAPI.TechTypes.Enum.**Dark_Archon_Meld**
Value of 28.

BWAPI.TechTypes.Enum.**Feedback**
Value of 29.

BWAPI.TechTypes.Enum.**Optical_Flare**
Value of 30.

BWAPI.TechTypes.Enum.**Maelstrom**
Value of 31.

BWAPI.TechTypes.Enum.**Lurker_Aspect**
Value of 32.

BWAPI.TechTypes.Enum.**Unused_33**
Value of 33.

BWAPI.TechTypes.Enum.**Healing**
Value of 34.

BWAPI.TechTypes.Enum.**None**
Value of 44.

BWAPI.TechTypes.Enum.**Nuclear_Strike**
Value of 45.

BWAPI.TechTypes.Enum.**Unknown**
Value of 46.

BWAPI.TechTypes.Enum.**MAX**
Value of 47.

Text

Contains enumeration of text formatting codes.

Such codes are used in calls to *BWAPI.Game.drawText()*, *BWAPI.Game.printf()*, and `print`

Important: Although the values are stored as a `number` type, they are most useful when printed as a character (which Lua does not have a type for; it only has `string`). The necessary `int` to `char` conversion can be done using `string.format` or `string.char` like so:

```
local text = "Your string"
-- using string.format
local formatted1 = string.format("%c%s", BWAPI.Text.White, text)
-- using string.char
local formatted2 = string.char(BWAPI.Text.White) .. text
-- these two techniques will result in exactly the same result
assert(formatted1 == formatted2)
```

See also:

BWAPI.Game.drawText()

Functions

static `BWAPI.Text.`**`isColor`**`(`*c*`)` → bool
Checks if the given character is a color-changing control code.

Parameters **c** (*number*) – The `BWAPI.Text` constant to check

Returns `true` if `c` is a regular color (not `Previous`, `Invisible*`, or `Align*`)

Return type boolean

Enum

BWAPI.Text.**Previous**
Value of 1.

Uses the previous color that was specified before the current one.

BWAPI.Text.**Default**
Value of 2.

Uses the default blueish color. This color is used in standard game messages.

BWAPI.Text.**Yellow**
Value of 3.

A solid yellow.

This yellow is used in notifications and is also the default color when printing text to Broodwar.

BWAPI.Text.**White**
Value of 4.

A bright white. This is used for timers.

BWAPI.Text.**Grey**
Value of 5.

A dark grey. This color code will override all color formatting that follows.

BWAPI.Text.**Red**
Value of 6.

A deep red. This color code is used for error messages.

BWAPI.Text.**Green**
Value of 7.

A solid green. This color is used for sent messages and resource counters.

BWAPI.Text.**BrightRed**
Value of 8.

A type of red. This color is used to color the name of the red player.

BWAPI.Text.**Invisible**
Value of 11.

This code hides all text and formatting that follows.

BWAPI.Text.**Blue**
Value of 14.

A deep blue. This color is used to color the name of the blue player.

BWAPI.Text.**Teal**
Value of 15.

A teal color. This color is used to color the name of the teal player.

BWAPI.Text.**Purple**
Value of 16.

A deep purple. This color is used to color the name of the purple player.

BWAPI.Text.**Orange**
Value of 17.

A solid orange. This color is used to color the name of the orange player.

BWAPI.Text.**Align_Right**
Value of 18.

An alignment directive that aligns the text to the right side of the screen.

BWAPI.Text.**Align_Center**
Value of 19.

An alignment directive that aligns the text to the center of the screen.

BWAPI.Text.**Invisible2**
Value of 20.

This code hides all text and formatting that follows.

BWAPI.Text.**Brown**
Value of 21.

A dark brown. This color is used to color the name of the brown player.

BWAPI.Text.**PlayerWhite**
Value of 22.

A dirty white. This color is used to color the name of the white player.

BWAPI.Text.**PlayerYellow**
Value of 23.

A deep yellow. This color is used to color the name of the yellow player.

BWAPI.Text.**DarkGreen**
Value of 24.

A dark green. This color is used to color the name of the green player.

BWAPI.Text.**LightYellow**
Value of 25.

A bright yellow.

BWAPI.Text.**Cyan**
Value of 26.

A cyan color. Similar to Default.

BWAPI.Text.**Tan**
Value of 27.

A tan color.

BWAPI.Text.**GreyBlue**
Value of 28.

A dark blueish color.

BWAPI.Text.**GreyGreen**
Value of 29.

A type of Green.

BWAPI.Text.**GreyCyan**
Value of 30.

A different type of Cyan.

BWAPI.Text.**Turquoise**
Value of 31.

A bright blue color.

Text.Size

Enumeration of available text sizes.

See also:

`Game.setTextSize()`

Enum

`BWAPI.Text.Size.`**`Small`**
Value of 0. The smallest text size in the game.

`BWAPI.Text.Size.`**`Default`**
Value of 1. The standard text size, used for most things in the game such as chat messages.

`BWAPI.Text.Size.`**`Large`**
Value of 2. A larger text size. This size is used for the in-game countdown timer seen in *Capture The Flag* or *Use Map Settings* game types.

`BWAPI.Text.Size.`**`Huge`**
Value of 3. The largest text size in the game.

UnitCommandTypes

Module containing unit command types.

See also:

BWAPI.UnitCommandType

Functions

static `BWAPI.UnitCommandTypes.`**`allUnitCommandTypes`**`()` → set
Retrieves a set of all UnitCommandTypes.

Returns Set of all UnitCommandTypes.

Return type *UnitCommandTypeset*

Constants

All constants are instances of the *UnitCommandType* class

`BWAPI.UnitCommandTypes.`**`Attack_Unit`**

`BWAPI.UnitCommandTypes.`**`Attack_Move`**

`BWAPI.UnitCommandTypes.`**`Build`**

`BWAPI.UnitCommandTypes.`**`Build_Addon`**

`BWAPI.UnitCommandTypes.`**`Train`**

`BWAPI.UnitCommandTypes.`**`Morph`**

`BWAPI.UnitCommandTypes.`**`Research`**

`BWAPI.UnitCommandTypes.`**`Upgrade`**

BWAPI.UnitCommandTypes.**Set_Rally_Position**

BWAPI.UnitCommandTypes.**Set_Rally_Unit**

BWAPI.UnitCommandTypes.**Move**

BWAPI.UnitCommandTypes.**Patrol**

BWAPI.UnitCommandTypes.**Hold_Position**

BWAPI.UnitCommandTypes.**Stop**

BWAPI.UnitCommandTypes.**Follow**

BWAPI.UnitCommandTypes.**Gather**

BWAPI.UnitCommandTypes.**Return_Cargo**

BWAPI.UnitCommandTypes.**Repair**

BWAPI.UnitCommandTypes.**Burrow**

BWAPI.UnitCommandTypes.**Unburrow**

BWAPI.UnitCommandTypes.**Cloak**

BWAPI.UnitCommandTypes.**Decloak**

BWAPI.UnitCommandTypes.**Siege**

BWAPI.UnitCommandTypes.**Unsiege**

BWAPI.UnitCommandTypes.**Lift**

BWAPI.UnitCommandTypes.**Land**

BWAPI.UnitCommandTypes.**Load**

BWAPI.UnitCommandTypes.**Unload**

BWAPI.UnitCommandTypes.**Unload_All**

BWAPI.UnitCommandTypes.**Unload_All_Position**

BWAPI.UnitCommandTypes.**Right_Click_Position**

BWAPI.UnitCommandTypes.**Right_Click_Unit**

BWAPI.UnitCommandTypes.**Halt_Construction**

BWAPI.UnitCommandTypes.**Cancel_Construction**

BWAPI.UnitCommandTypes.**Cancel_Addon**

BWAPI.UnitCommandTypes.**Cancel_Train**

BWAPI.UnitCommandTypes.**Cancel_Train_Slot**

BWAPI.UnitCommandTypes.**Cancel_Morph**

BWAPI.UnitCommandTypes.**Cancel_Research**

BWAPI.UnitCommandTypes.**Cancel_Upgrade**

BWAPI.UnitCommandTypes.**Use_Tech**

BWAPI.UnitCommandTypes.**Use_Tech_Position**

BWAPI.UnitCommandTypes.**Use_Tech_Unit**

BWAPI.UnitCommandTypes.**Place_COP**

BWAPI.UnitCommandTypes.**None**

BWAPI.UnitCommandTypes.**Unknown**

Enum

Enumeration of unit command types.

Important: Enum values rarely need to be used in Lua.

BWAPI.UnitCommandTypes.Enum.**Attack_Move**
Value of 0.

BWAPI.UnitCommandTypes.Enum.**Attack_Unit**
Value of 1.

BWAPI.UnitCommandTypes.Enum.**Build**
Value of 2.

BWAPI.UnitCommandTypes.Enum.**Build_Addon**
Value of 3.

BWAPI.UnitCommandTypes.Enum.**Train**
Value of 4.

BWAPI.UnitCommandTypes.Enum.**Morph**
Value of 5.

BWAPI.UnitCommandTypes.Enum.**Research**
Value of 6.

BWAPI.UnitCommandTypes.Enum.**Upgrade**
Value of 7.

BWAPI.UnitCommandTypes.Enum.**Set_Rally_Position**
Value of 8.

BWAPI.UnitCommandTypes.Enum.**Set_Rally_Unit**
Value of 9.

BWAPI.UnitCommandTypes.Enum.**Move**
Value of 10.

BWAPI.UnitCommandTypes.Enum.**Patrol**
Value of 11.

BWAPI.UnitCommandTypes.Enum.**Hold_Position**
Value of 12.

BWAPI.UnitCommandTypes.Enum.**Stop**
Value of 13.

BWAPI.UnitCommandTypes.Enum.**Follow**
Value of 14.

BWAPI.UnitCommandTypes.Enum.**Gather**
Value of 15.

BWAPI.UnitCommandTypes.Enum.**Return_Cargo**
Value of 16.

BWAPI.UnitCommandTypes.Enum.**Repair**
Value of 17.

BWAPI.UnitCommandTypes.Enum.**Burrow**
Value of 18.

BWAPI.UnitCommandTypes.Enum.**Unburrow**
Value of 19.

BWAPI.UnitCommandTypes.Enum.**Cloak**
Value of 20.

BWAPI.UnitCommandTypes.Enum.**Decloak**
Value of 21.

BWAPI.UnitCommandTypes.Enum.**Siege**
Value of 22.

BWAPI.UnitCommandTypes.Enum.**Unsiege**
Value of 23.

BWAPI.UnitCommandTypes.Enum.**Lift**
Value of 24.

BWAPI.UnitCommandTypes.Enum.**Land**
Value of 25.

BWAPI.UnitCommandTypes.Enum.**Load**
Value of 26.

BWAPI.UnitCommandTypes.Enum.**Unload**
Value of 27.

BWAPI.UnitCommandTypes.Enum.**Unload_All**
Value of 28.

BWAPI.UnitCommandTypes.Enum.**Unload_All_Position**
Value of 29.

BWAPI.UnitCommandTypes.Enum.**Right_Click_Position**
Value of 30.

BWAPI.UnitCommandTypes.Enum.**Right_Click_Unit**
Value of 31.

BWAPI.UnitCommandTypes.Enum.**Halt_Construction**
Value of 32.

BWAPI.UnitCommandTypes.Enum.**Cancel_Construction**
Value of 33.

BWAPI.UnitCommandTypes.Enum.**Cancel_Addon**
Value of 34.

BWAPI.UnitCommandTypes.Enum.**Cancel_Train**
Value of 35.

BWAPI.UnitCommandTypes.Enum.**Cancel_Train_Slot**
Value of 36.

BWAPI.UnitCommandTypes.Enum.**Cancel_Morph**
Value of 37.

BWAPI.UnitCommandTypes.Enum.**Cancel_Research**
Value of 38.

BWAPI.UnitCommandTypes.Enum.**Cancel_Upgrade**
Value of 39.

BWAPI.UnitCommandTypes.Enum.**Use_Tech**
Value of 40.

BWAPI.UnitCommandTypes.Enum.**Use_Tech_Position**
Value of 41.

BWAPI.UnitCommandTypes.Enum.**Use_Tech_Unit**
Value of 42.

BWAPI.UnitCommandTypes.Enum.**Place_COP**
Value of 43.

BWAPI.UnitCommandTypes.Enum.**None**
Value of 44.

BWAPI.UnitCommandTypes.Enum.**Unknown**
Value of 45.

BWAPI.UnitCommandTypes.Enum.**MAX**
Value of 46.

UnitSizeTypes

Module containing unit size types.

See also:

BWAPI.UnitSizeType

Functions

static BWAPI.UnitSizeTypes.**allUnitSizeTypes**() → set
Retrieves a set of all UnitSizeTypes.

Returns Set of all UnitSizeTypes.

Return type *UnitSizeTypeset*

Constants

All constants are instances of the *UnitSizeType* class

BWAPI.UnitSizeTypes.**Independent**

BWAPI.UnitSizeTypes.**Small**

BWAPI.UnitSizeTypes.**Medium**

BWAPI.UnitSizeTypes.**Large**

BWAPI.UnitSizeTypes.**None**

BWAPI.UnitSizeTypes.**Unknown**

Enum

Enumeration of unit size types.

Important: Enum values rarely need to be used in Lua.

`BWAPI.UnitSizeTypes.Enum.`**`Independent`**
Value of 0.

`BWAPI.UnitSizeTypes.Enum.`**`Small`**
Value of 1.

`BWAPI.UnitSizeTypes.Enum.`**`Medium`**
Value of 2.

`BWAPI.UnitSizeTypes.Enum.`**`Large`**
Value of 3.

`BWAPI.UnitSizeTypes.Enum.`**`None`**
Value of 4.

`BWAPI.UnitSizeTypes.Enum.`**`Unknown`**
Value of 5.

`BWAPI.UnitSizeTypes.Enum.`**`MAX`**
Value of 6.

UnitTypes

Module containing unit types.

See also:

BWAPI.UnitType

Functions

static `BWAPI.UnitTypes.`**`allUnitTypes`**`()` → set
Retrieves a set of all UnitTypes.

Returns Set of all UnitTypes.

Return type *UnitTypeset*

static `BWAPI.UnitTypes.`**`allMacroTypes`**`()` → set
Retrieves the set of all macro unit types.

A macro type is a fake unit type, used by some functions. These include *AllUnits*, *Men*, *Buildings*, and *Factories*. The purpose of these types are to match the same ones used in Broodwar, also seen in the StarEdit map editor.

Returns Set of all macro UnitTypes.

Return type *UnitTypeset*

Constants

All constants are instances of the *UnitType* class

Terran Ground Units

BWAPI.UnitTypes.**Terran_Firebat**

BWAPI.UnitTypes.**Terran_Ghost**

BWAPI.UnitTypes.**Terran_Goliath**

BWAPI.UnitTypes.**Terran_Marine**

BWAPI.UnitTypes.**Terran_Medic**

BWAPI.UnitTypes.**Terran_SCV**

BWAPI.UnitTypes.**Terran_Siege_Tank_Siege_Mode**

BWAPI.UnitTypes.**Terran_Siege_Tank_Tank_Mode**

BWAPI.UnitTypes.**Terran_Vulture**

BWAPI.UnitTypes.**Terran_Vulture_Spider_Mine**

Terran Air Units

BWAPI.UnitTypes.**Terran_Battlecruiser**

BWAPI.UnitTypes.**Terran_Dropship**

BWAPI.UnitTypes.**Terran_Nuclear_Missile**

BWAPI.UnitTypes.**Terran_Science_Vessel**

BWAPI.UnitTypes.**Terran_Valkyrie**

BWAPI.UnitTypes.**Terran_Wraith**

Terran Heroes

BWAPI.UnitTypes.**Hero_Alan_Schezar**

BWAPI.UnitTypes.**Hero_Alexei_Stukov**

BWAPI.UnitTypes.**Hero_Arcturus_Mengsk**

BWAPI.UnitTypes.**Hero_Edmund_Duke_Tank_Mode**

BWAPI.UnitTypes.**Hero_Edmund_Duke_Siege_Mode**

BWAPI.UnitTypes.**Hero_Gerard_DuGalle**

BWAPI.UnitTypes.**Hero_Gui_Montag**

BWAPI.UnitTypes.**Hero_Hyperion**

BWAPI.UnitTypes.**Hero_Jim_Raynor_Marine**

BWAPI.UnitTypes.**Hero_Jim_Raynor_Vulture**

BWAPI.UnitTypes.**Hero_Magellan**

BWAPI.UnitTypes.**Hero_Norad_II**

BWAPI.UnitTypes.**Hero_Samir_Duran**

BWAPI.UnitTypes.**Hero_Sarah_Kerrigan**

BWAPI.UnitTypes.**Hero_Tom_Kazansky**

BWAPI.UnitTypes.**Terran_Civilian**

Terran Buildings

BWAPI.UnitTypes.**Terran_Academy**

BWAPI.UnitTypes.**Terran_Armory**

BWAPI.UnitTypes.**Terran_Barracks**

BWAPI.UnitTypes.**Terran_Bunker**

BWAPI.UnitTypes.**Terran_Command_Center**

BWAPI.UnitTypes.**Terran_Engineering_Bay**

BWAPI.UnitTypes.**Terran_Factory**

BWAPI.UnitTypes.**Terran_Missile_Turret**

BWAPI.UnitTypes.**Terran_Refinery**

BWAPI.UnitTypes.**Terran_Science_Facility**

BWAPI.UnitTypes.**Terran_Starport**

BWAPI.UnitTypes.**Terran_Supply_Depot**

Terran Addons

BWAPI.UnitTypes.**Terran_Comsat_Station**

BWAPI.UnitTypes.**Terran_Control_Tower**

BWAPI.UnitTypes.**Terran_Covert_Ops**

BWAPI.UnitTypes.**Terran_Machine_Shop**

BWAPI.UnitTypes.**Terran_Nuclear_Silo**

BWAPI.UnitTypes.**Terran_Physics_Lab**

Terran Special Buildings

BWAPI.UnitTypes.**Special_Crashed_Norad_II**

BWAPI.UnitTypes.**Special_Ion_Cannon**

BWAPI.UnitTypes.**Special_Power_Generator**

BWAPI.UnitTypes.**Special_Psi_Disrupter**

Protoss Ground Units

BWAPI.UnitTypes.**Protoss_Archon**

BWAPI.UnitTypes.**Protoss_Dark_Archon**

BWAPI.UnitTypes.**Protoss_Dark_Templar**

BWAPI.UnitTypes.**Protoss_Dragoon**

BWAPI.UnitTypes.**Protoss_High_Templar**

BWAPI.UnitTypes.**Protoss_Probe**

BWAPI.UnitTypes.**Protoss_Reaver**

BWAPI.UnitTypes.**Protoss_Scarab**

BWAPI.UnitTypes.**Protoss_Zealot**

Protoss Air Units

BWAPI.UnitTypes.**Protoss_Arbiter**

BWAPI.UnitTypes.**Protoss_Carrier**

BWAPI.UnitTypes.**Protoss_Corsair**

BWAPI.UnitTypes.**Protoss_Interceptor**

BWAPI.UnitTypes.**Protoss_Observer**

BWAPI.UnitTypes.**Protoss_Scout**

BWAPI.UnitTypes.**Protoss_Shuttle**

Protoss Heroes

BWAPI.UnitTypes.**Hero_Aldaris**

BWAPI.UnitTypes.**Hero_Artanis**

BWAPI.UnitTypes.**Hero_Danimoth**

BWAPI.UnitTypes.**Hero_Dark_Templar**

BWAPI.UnitTypes.**Hero_Fenix_Dragoon**

BWAPI.UnitTypes.**Hero_Fenix_Zealot**

BWAPI.UnitTypes.**Hero_Gantrithor**

BWAPI.UnitTypes.**Hero_Mojo**

BWAPI.UnitTypes.**Hero_Raszagal**

BWAPI.UnitTypes.**Hero_Tassadar**

BWAPI.UnitTypes.**Hero_Tassadar_Zeratul_Archon**

BWAPI.UnitTypes.**Hero_Warbringer**

BWAPI.UnitTypes.**Hero_Zeratul**

Protoss Buildings

BWAPI.UnitTypes.**Protoss_Arbiter_Tribunal**

BWAPI.UnitTypes.**Protoss_Assimilator**

BWAPI.UnitTypes.**Protoss_Citadel_of_Adun**

BWAPI.UnitTypes.**Protoss_Cybernetics_Core**

BWAPI.UnitTypes.**Protoss_Fleet_Beacon**

BWAPI.UnitTypes.**Protoss_Forge**

BWAPI.UnitTypes.**Protoss_Gateway**

BWAPI.UnitTypes.**Protoss_Nexus**

BWAPI.UnitTypes.**Protoss_Observatory**

BWAPI.UnitTypes.**Protoss_Photon_Cannon**

BWAPI.UnitTypes.**Protoss_Pylon**

BWAPI.UnitTypes.**Protoss_Robotics_Facility**

BWAPI.UnitTypes.**Protoss_Robotics_Support_Bay**

BWAPI.UnitTypes.**Protoss_Shield_Battery**

BWAPI.UnitTypes.**Protoss_Stargate**

BWAPI.UnitTypes.**Protoss_Templar_Archives**

Protoss Special Buildings

BWAPI.UnitTypes.**Special_Khaydarin_Crystal_Form**

BWAPI.UnitTypes.**Special_Protoss_Temple**

BWAPI.UnitTypes.**Special_Stasis_Cell_Prison**

BWAPI.UnitTypes.**Special_Warp_Gate**

BWAPI.UnitTypes.**Special_XelNaga_Temple**

Zerg Ground Units

BWAPI.UnitTypes.**Zerg_Broodling**

BWAPI.UnitTypes.**Zerg_Defiler**

BWAPI.UnitTypes.**Zerg_Drone**

BWAPI.UnitTypes.**Zerg_Egg**

BWAPI.UnitTypes.**Zerg_Hydralisk**

BWAPI.UnitTypes.**Zerg_Infested_Terran**

BWAPI.UnitTypes.**Zerg_Larva**

BWAPI.UnitTypes.**Zerg_Lurker**

BWAPI.UnitTypes.**Zerg_Lurker_Egg**

BWAPI.UnitTypes.**Zerg_Ultralisk**

BWAPI.UnitTypes.**Zerg_Zergling**

Zerg Air Units

BWAPI.UnitTypes.**Zerg_Cocoon**

BWAPI.UnitTypes.**Zerg_Devourer**

BWAPI.UnitTypes.**Zerg_Guardian**

BWAPI.UnitTypes.**Zerg_Mutalisk**

BWAPI.UnitTypes.**Zerg_Overlord**

BWAPI.UnitTypes.**Zerg_Queen**

BWAPI.UnitTypes.**Zerg_Scourge**

Zerg Heroes

BWAPI.UnitTypes.**Hero_Devouring_One**

BWAPI.UnitTypes.**Hero_Hunter_Killer**

BWAPI.UnitTypes.**Hero_Infested_Duran**

BWAPI.UnitTypes.**Hero_Infested_Kerrigan**

BWAPI.UnitTypes.**Hero_Kukulza_Guardian**

BWAPI.UnitTypes.**Hero_Kukulza_Mutalisk**

BWAPI.UnitTypes.**Hero_Matriarch**

BWAPI.UnitTypes.**Hero_Torrasque**

BWAPI.UnitTypes.**Hero_Unclean_One**

BWAPI.UnitTypes.**Hero_Yggdrasill**

Zerg Buildings

BWAPI.UnitTypes.**Zerg_Creep_Colony**

BWAPI.UnitTypes.**Zerg_Defiler_Mound**

BWAPI.UnitTypes.**Zerg_Evolution_Chamber**

BWAPI.UnitTypes.**Zerg_Extractor**

BWAPI.UnitTypes.**Zerg_Greater_Spire**

BWAPI.UnitTypes.**Zerg_Hatchery**

BWAPI.UnitTypes.**Zerg_Hive**

BWAPI.UnitTypes.**Zerg_Hydralisk_Den**

BWAPI.UnitTypes.**Zerg_Infested_Command_Center**

BWAPI.UnitTypes.**Zerg_Lair**

BWAPI.UnitTypes.**Zerg_Nydus_Canal**

BWAPI.UnitTypes.**Zerg_Queens_Nest**

BWAPI.UnitTypes.**Zerg_Spawning_Pool**

BWAPI.UnitTypes.**Zerg_Spire**

BWAPI.UnitTypes.**Zerg_Spore_Colony**

BWAPI.UnitTypes.**Zerg_Sunken_Colony**

BWAPI.UnitTypes.**Zerg_Ultralisk_Cavern**

Zerg Special Buildings

BWAPI.UnitTypes.**Special_Cerebrate**

BWAPI.UnitTypes.**Special_Cerebrate_Daggoth**

BWAPI.UnitTypes.**Special_Mature_Chrysalis**

BWAPI.UnitTypes.**Special_Overmind**

BWAPI.UnitTypes.**Special_Overmind_Cocoon**

BWAPI.UnitTypes.**Special_Overmind_With_Shell**

Critters

BWAPI.UnitTypes.**Critter_Bengalaas**

BWAPI.UnitTypes.**Critter_Kakaru**

BWAPI.UnitTypes.**Critter_Ragnasaur**

BWAPI.UnitTypes.**Critter_Rhynadon**

BWAPI.UnitTypes.**Critter_Scantid**

BWAPI.UnitTypes.**Critter_Ursadon**

Resources

BWAPI.UnitTypes.**Resource_Mineral_Field**

BWAPI.UnitTypes.**Resource_Mineral_Field_Type_2**

BWAPI.UnitTypes.**Resource_Mineral_Field_Type_3**

BWAPI.UnitTypes.**Resource_Vespene_Geyser**

Spells

BWAPI.UnitTypes.**Spell_Dark_Swarm**

BWAPI.UnitTypes.**Spell_Disruption_Web**

BWAPI.UnitTypes.**Spell_Scanner_Sweep**

Beacons

BWAPI.UnitTypes.**Special_Protoss_Beacon**

BWAPI.UnitTypes.**Special_Protoss_Flag_Beacon**

BWAPI.UnitTypes.**Special_Terran_Beacon**

BWAPI.UnitTypes.**Special_Terran_Flag_Beacon**

BWAPI.UnitTypes.**Special_Zerg_Beacon**

BWAPI.UnitTypes.**Special_Zerg_Flag_Beacon**

Powerups

BWAPI.UnitTypes.**Powerup_Data_Disk**

BWAPI.UnitTypes.**Powerup_Flag**

BWAPI.UnitTypes.**Powerup_Khalis_Crystal**

BWAPI.UnitTypes.**Powerup_Khaydarin_Crystal**

BWAPI.UnitTypes.**Powerup_Mineral_Cluster_Type_1**

BWAPI.UnitTypes.**Powerup_Mineral_Cluster_Type_2**

BWAPI.UnitTypes.**Powerup_Protoss_Gas_Orb_Type_1**

BWAPI.UnitTypes.**Powerup_Protoss_Gas_Orb_Type_2**

BWAPI.UnitTypes.**Powerup_Psi_Emitter**

BWAPI.UnitTypes.**Powerup_Terran_Gas_Tank_Type_1**

BWAPI.UnitTypes.**Powerup_Terran_Gas_Tank_Type_2**

BWAPI.UnitTypes.**Powerup_Uraj_Crystal**

BWAPI.UnitTypes.**Powerup_Young_Chrysalis**

BWAPI.UnitTypes.**Powerup_Zerg_Gas_Sac_Type_1**

BWAPI.UnitTypes.**Powerup_Zerg_Gas_Sac_Type_2**

Traps

BWAPI.UnitTypes.**Special_Floor_Gun_Trap**

BWAPI.UnitTypes.**Special_Floor_Missile_Trap**

BWAPI.UnitTypes.**Special_Right_Wall_Flame_Trap**

BWAPI.UnitTypes.**Special_Right_Wall_Missile_Trap**

BWAPI.UnitTypes.**Special_Wall_Flame_Trap**

BWAPI.UnitTypes.**Special_Wall_Missile_Trap**

Doors

BWAPI.UnitTypes.**Special_Pit_Door**

BWAPI.UnitTypes.**Special_Right_Pit_Door**

BWAPI.UnitTypes.**Special_Right_Upper_Level_Door**

BWAPI.UnitTypes.**Special_Upper_Level_Door**

Special

BWAPI.UnitTypes.**Special_Cargo_Ship**

BWAPI.UnitTypes.**Special_Floor_Hatch**

BWAPI.UnitTypes.**Special_Independant_Starport**

BWAPI.UnitTypes.**Special_Map_Revealer**

BWAPI.UnitTypes.**Special_Mercenary_Gunship**

BWAPI.UnitTypes.**Special_Start_Location**

Macros

BWAPI.UnitTypes.**None**

BWAPI.UnitTypes.**AllUnits**

BWAPI.UnitTypes.**Men**

BWAPI.UnitTypes.**Buildings**

BWAPI.UnitTypes.**Factories**

BWAPI.UnitTypes.**Unknown**

Enum

Enumeration of unit types.

Important: Enum values rarely need to be used in Lua.

BWAPI.UnitTypes.Enum.**Terran_Marine**
: Value of 0.

BWAPI.UnitTypes.Enum.**Terran_Ghost**
: Value of 1.

BWAPI.UnitTypes.Enum.**Terran_Vulture**
: Value of 2.

BWAPI.UnitTypes.Enum.**Terran_Goliath**
: Value of 3.

BWAPI.UnitTypes.Enum.**Terran_Goliath_Turret**
: Value of 4.

BWAPI.UnitTypes.Enum.**Terran_Siege_Tank_Tank_Mode**
Value of 5.

BWAPI.UnitTypes.Enum.**Terran_Siege_Tank_Tank_Mode_Turret**
Value of 6.

BWAPI.UnitTypes.Enum.**Terran_SCV**
Value of 7.

BWAPI.UnitTypes.Enum.**Terran_Wraith**
Value of 8.

BWAPI.UnitTypes.Enum.**Terran_Science_Vessel**
Value of 9.

BWAPI.UnitTypes.Enum.**Hero_Gui_Montag**
Value of 10.

BWAPI.UnitTypes.Enum.**Terran_Dropship**
Value of 11.

BWAPI.UnitTypes.Enum.**Terran_Battlecruiser**
Value of 12.

BWAPI.UnitTypes.Enum.**Terran_Vulture_Spider_Mine**
Value of 13.

BWAPI.UnitTypes.Enum.**Terran_Nuclear_Missile**
Value of 14.

BWAPI.UnitTypes.Enum.**Terran_Civilian**
Value of 15.

BWAPI.UnitTypes.Enum.**Hero_Sarah_Kerrigan**
Value of 16.

BWAPI.UnitTypes.Enum.**Hero_Alan_Schezar**
Value of 17.

BWAPI.UnitTypes.Enum.**Hero_Alan_Schezar_Turret**
Value of 18.

BWAPI.UnitTypes.Enum.**Hero_Jim_Raynor_Vulture**
Value of 19.

BWAPI.UnitTypes.Enum.**Hero_Jim_Raynor_Marine**
Value of 20.

BWAPI.UnitTypes.Enum.**Hero_Tom_Kazansky**
Value of 21.

BWAPI.UnitTypes.Enum.**Hero_Magellan**
Value of 22.

BWAPI.UnitTypes.Enum.**Hero_Edmund_Duke_Tank_Mode**
Value of 23.

BWAPI.UnitTypes.Enum.**Hero_Edmund_Duke_Tank_Mode_Turret**
Value of 24.

BWAPI.UnitTypes.Enum.**Hero_Edmund_Duke_Siege_Mode**
Value of 25.

BWAPI.UnitTypes.Enum.**Hero_Edmund_Duke_Siege_Mode_Turret**
Value of 26.

BWAPI.UnitTypes.Enum.**Hero_Arcturus_Mengsk**
Value of 27.

BWAPI.UnitTypes.Enum.**Hero_Hyperion**
Value of 28.

BWAPI.UnitTypes.Enum.**Hero_Norad_II**
Value of 29.

BWAPI.UnitTypes.Enum.**Terran_Siege_Tank_Siege_Mode**
Value of 30.

BWAPI.UnitTypes.Enum.**Terran_Siege_Tank_Siege_Mode_Turret**
Value of 31.

BWAPI.UnitTypes.Enum.**Terran_Firebat**
Value of 32.

BWAPI.UnitTypes.Enum.**Spell_Scanner_Sweep**
Value of 33.

BWAPI.UnitTypes.Enum.**Terran_Medic**
Value of 34.

BWAPI.UnitTypes.Enum.**Zerg_Larva**
Value of 35.

BWAPI.UnitTypes.Enum.**Zerg_Egg**
Value of 36.

BWAPI.UnitTypes.Enum.**Zerg_Zergling**
Value of 37.

BWAPI.UnitTypes.Enum.**Zerg_Hydralisk**
Value of 38.

BWAPI.UnitTypes.Enum.**Zerg_Ultralisk**
Value of 39.

BWAPI.UnitTypes.Enum.**Zerg_Broodling**
Value of 40.

BWAPI.UnitTypes.Enum.**Zerg_Drone**
Value of 41.

BWAPI.UnitTypes.Enum.**Zerg_Overlord**
Value of 42.

BWAPI.UnitTypes.Enum.**Zerg_Mutalisk**
Value of 43.

BWAPI.UnitTypes.Enum.**Zerg_Guardian**
Value of 44.

BWAPI.UnitTypes.Enum.**Zerg_Queen**
Value of 45.

BWAPI.UnitTypes.Enum.**Zerg_Defiler**
Value of 46.

BWAPI.UnitTypes.Enum.**Zerg_Scourge**
Value of 47.

BWAPI.UnitTypes.Enum.**Hero_Torrasque**
Value of 48.

BWAPI.UnitTypes.Enum.**Hero_Matriarch**
Value of 49.

BWAPI.UnitTypes.Enum.**Zerg_Infested_Terran**
Value of 50.

BWAPI.UnitTypes.Enum.**Hero_Infested_Kerrigan**
Value of 51.

BWAPI.UnitTypes.Enum.**Hero_Unclean_One**
Value of 52.

BWAPI.UnitTypes.Enum.**Hero_Hunter_Killer**
Value of 53.

BWAPI.UnitTypes.Enum.**Hero_Devouring_One**
Value of 54.

BWAPI.UnitTypes.Enum.**Hero_Kukulza_Mutalisk**
Value of 55.

BWAPI.UnitTypes.Enum.**Hero_Kukulza_Guardian**
Value of 56.

BWAPI.UnitTypes.Enum.**Hero_Yggdrasill**
Value of 57.

BWAPI.UnitTypes.Enum.**Terran_Valkyrie**
Value of 58.

BWAPI.UnitTypes.Enum.**Zerg_Cocoon**
Value of 59.

BWAPI.UnitTypes.Enum.**Protoss_Corsair**
Value of 60.

BWAPI.UnitTypes.Enum.**Protoss_Dark_Templar**
Value of 61.

BWAPI.UnitTypes.Enum.**Zerg_Devourer**
Value of 62.

BWAPI.UnitTypes.Enum.**Protoss_Dark_Archon**
Value of 63.

BWAPI.UnitTypes.Enum.**Protoss_Probe**
Value of 64.

BWAPI.UnitTypes.Enum.**Protoss_Zealot**
Value of 65.

BWAPI.UnitTypes.Enum.**Protoss_Dragoon**
Value of 66.

BWAPI.UnitTypes.Enum.**Protoss_High_Templar**
Value of 67.

BWAPI.UnitTypes.Enum.**Protoss_Archon**
Value of 68.

BWAPI.UnitTypes.Enum.**Protoss_Shuttle**
Value of 69.

BWAPI.UnitTypes.Enum.**Protoss_Scout**
Value of 70.

BWAPI.UnitTypes.Enum.**Protoss_Arbiter**
Value of 71.

BWAPI.UnitTypes.Enum.**Protoss_Carrier**
Value of 72.

BWAPI.UnitTypes.Enum.**Protoss_Interceptor**
Value of 73.

BWAPI.UnitTypes.Enum.**Hero_Dark_Templar**
Value of 74.

BWAPI.UnitTypes.Enum.**Hero_Zeratul**
Value of 75.

BWAPI.UnitTypes.Enum.**Hero_Tassadar_Zeratul_Archon**
Value of 76.

BWAPI.UnitTypes.Enum.**Hero_Fenix_Zealot**
Value of 77.

BWAPI.UnitTypes.Enum.**Hero_Fenix_Dragoon**
Value of 78.

BWAPI.UnitTypes.Enum.**Hero_Tassadar**
Value of 79.

BWAPI.UnitTypes.Enum.**Hero_Mojo**
Value of 80.

BWAPI.UnitTypes.Enum.**Hero_Warbringer**
Value of 81.

BWAPI.UnitTypes.Enum.**Hero_Gantrithor**
Value of 82.

BWAPI.UnitTypes.Enum.**Protoss_Reaver**
Value of 83.

BWAPI.UnitTypes.Enum.**Protoss_Observer**
Value of 84.

BWAPI.UnitTypes.Enum.**Protoss_Scarab**
Value of 85.

BWAPI.UnitTypes.Enum.**Hero_Danimoth**
Value of 86.

BWAPI.UnitTypes.Enum.**Hero_Aldaris**
Value of 87.

BWAPI.UnitTypes.Enum.**Hero_Artanis**
Value of 88.

BWAPI.UnitTypes.Enum.**Critter_Rhynadon**
Value of 89.

BWAPI.UnitTypes.Enum.**Critter_Bengalaas**
Value of 90.

BWAPI.UnitTypes.Enum.**Special_Cargo_Ship**
Value of 91.

BWAPI.UnitTypes.Enum.**Special_Mercenary_Gunship**
Value of 92.

BWAPI.UnitTypes.Enum.**Critter_Scantid**
Value of 93.

BWAPI.UnitTypes.Enum.**Critter_Kakaru**
Value of 94.

BWAPI.UnitTypes.Enum.**Critter_Ragnasaur**
Value of 95.

BWAPI.UnitTypes.Enum.**Critter_Ursadon**
Value of 96.

BWAPI.UnitTypes.Enum.**Zerg_Lurker_Egg**
Value of 97.

BWAPI.UnitTypes.Enum.**Hero_Raszagal**
Value of 98.

BWAPI.UnitTypes.Enum.**Hero_Samir_Duran**
Value of 99.

BWAPI.UnitTypes.Enum.**Hero_Alexei_Stukov**
Value of 100.

BWAPI.UnitTypes.Enum.**Special_Map_Revealer**
Value of 101.

BWAPI.UnitTypes.Enum.**Hero_Gerard_DuGalle**
Value of 102.

BWAPI.UnitTypes.Enum.**Zerg_Lurker**
Value of 103.

BWAPI.UnitTypes.Enum.**Hero_Infested_Duran**
Value of 104.

BWAPI.UnitTypes.Enum.**Spell_Disruption_Web**
Value of 105.

BWAPI.UnitTypes.Enum.**Terran_Command_Center**
Value of 106.

BWAPI.UnitTypes.Enum.**Terran_Comsat_Station**
Value of 107.

BWAPI.UnitTypes.Enum.**Terran_Nuclear_Silo**
Value of 108.

BWAPI.UnitTypes.Enum.**Terran_Supply_Depot**
Value of 109.

BWAPI.UnitTypes.Enum.**Terran_Refinery**
Value of 110.

BWAPI.UnitTypes.Enum.**Terran_Barracks**
Value of 111.

BWAPI.UnitTypes.Enum.**Terran_Academy**
Value of 112.

BWAPI.UnitTypes.Enum.**Terran_Factory**
Value of 113.

BWAPI.UnitTypes.Enum.**Terran_Starport**
Value of 114.

BWAPI.UnitTypes.Enum.**Terran_Control_Tower**
Value of 115.

BWAPI.UnitTypes.Enum.**Terran_Science_Facility**
Value of 116.

BWAPI.UnitTypes.Enum.**Terran_Covert_Ops**
Value of 117.

BWAPI.UnitTypes.Enum.**Terran_Physics_Lab**
Value of 118.

BWAPI.UnitTypes.Enum.**Unused_Terran1**
Value of 119.

BWAPI.UnitTypes.Enum.**Terran_Machine_Shop**
Value of 120.

BWAPI.UnitTypes.Enum.**Unused_Terran2**
Value of 121.

BWAPI.UnitTypes.Enum.**Terran_Engineering_Bay**
Value of 122.

BWAPI.UnitTypes.Enum.**Terran_Armory**
Value of 123.

BWAPI.UnitTypes.Enum.**Terran_Missile_Turret**
Value of 124.

BWAPI.UnitTypes.Enum.**Terran_Bunker**
Value of 125.

BWAPI.UnitTypes.Enum.**Special_Crashed_Norad_II**
Value of 126.

BWAPI.UnitTypes.Enum.**Special_Ion_Cannon**
Value of 127.

BWAPI.UnitTypes.Enum.**Powerup_Uraj_Crystal**
Value of 128.

BWAPI.UnitTypes.Enum.**Powerup_Khalis_Crystal**
Value of 129.

BWAPI.UnitTypes.Enum.**Zerg_Infested_Command_Center**
Value of 130.

BWAPI.UnitTypes.Enum.**Zerg_Hatchery**
Value of 131.

BWAPI.UnitTypes.Enum.**Zerg_Lair**
Value of 132.

BWAPI.UnitTypes.Enum.**Zerg_Hive**
Value of 133.

BWAPI.UnitTypes.Enum.**Zerg_Nydus_Canal**
Value of 134.

BWAPI.UnitTypes.Enum.**Zerg_Hydralisk_Den**
Value of 135.

BWAPI.UnitTypes.Enum.**Zerg_Defiler_Mound**
Value of 136.

BWAPI.UnitTypes.Enum.**Zerg_Greater_Spire**
Value of 137.

BWAPI.UnitTypes.Enum.**Zerg_Queens_Nest**
Value of 138.

BWAPI.UnitTypes.Enum.**Zerg_Evolution_Chamber**
Value of 139.

BWAPI.UnitTypes.Enum.**Zerg_Ultralisk_Cavern**
Value of 140.

BWAPI.UnitTypes.Enum.**Zerg_Spire**
Value of 141.

BWAPI.UnitTypes.Enum.**Zerg_Spawning_Pool**
Value of 142.

BWAPI.UnitTypes.Enum.**Zerg_Creep_Colony**
Value of 143.

BWAPI.UnitTypes.Enum.**Zerg_Spore_Colony**
Value of 144.

BWAPI.UnitTypes.Enum.**Unused_Zerg1**
Value of 145.

BWAPI.UnitTypes.Enum.**Zerg_Sunken_Colony**
Value of 146.

BWAPI.UnitTypes.Enum.**Special_Overmind_With_Shell**
Value of 147.

BWAPI.UnitTypes.Enum.**Special_Overmind**
Value of 148.

BWAPI.UnitTypes.Enum.**Zerg_Extractor**
Value of 149.

BWAPI.UnitTypes.Enum.**Special_Mature_Chrysalis**
Value of 150.

BWAPI.UnitTypes.Enum.**Special_Cerebrate**
Value of 151.

BWAPI.UnitTypes.Enum.**Special_Cerebrate_Daggoth**
Value of 152.

BWAPI.UnitTypes.Enum.**Unused_Zerg2**
Value of 153.

BWAPI.UnitTypes.Enum.**Protoss_Nexus**
Value of 154.

BWAPI.UnitTypes.Enum.**Protoss_Robotics_Facility**
Value of 155.

BWAPI.UnitTypes.Enum.**Protoss_Pylon**
Value of 156.

BWAPI.UnitTypes.Enum.**Protoss_Assimilator**
Value of 157.

BWAPI.UnitTypes.Enum.**Unused_Protoss1**
Value of 158.

BWAPI.UnitTypes.Enum.**Protoss_Observatory**
Value of 159.

BWAPI.UnitTypes.Enum.**Protoss_Gateway**
Value of 160.

BWAPI.UnitTypes.Enum.**Unused_Protoss2**

BWAPI.UnitTypes.Enum.**Protoss_Photon_Cannon**
Value of 162.

BWAPI.UnitTypes.Enum.**Protoss_Citadel_of_Adun**
Value of 163.

BWAPI.UnitTypes.Enum.**Protoss_Cybernetics_Core**
Value of 164.

BWAPI.UnitTypes.Enum.**Protoss_Templar_Archives**
Value of 165.

BWAPI.UnitTypes.Enum.**Protoss_Forge**
Value of 166.

BWAPI.UnitTypes.Enum.**Protoss_Stargate**
Value of 167.

BWAPI.UnitTypes.Enum.**Special_Stasis_Cell_Prison**
Value of 168.

BWAPI.UnitTypes.Enum.**Protoss_Fleet_Beacon**
Value of 169.

BWAPI.UnitTypes.Enum.**Protoss_Arbiter_Tribunal**
Value of 170.

BWAPI.UnitTypes.Enum.**Protoss_Robotics_Support_Bay**
Value of 171.

BWAPI.UnitTypes.Enum.**Protoss_Shield_Battery**
Value of 172.

BWAPI.UnitTypes.Enum.**Special_Khaydarin_Crystal_Form**
Value of 173.

BWAPI.UnitTypes.Enum.**Special_Protoss_Temple**
Value of 174.

BWAPI.UnitTypes.Enum.**Special_XelNaga_Temple**
Value of 175.

BWAPI.UnitTypes.Enum.**Resource_Mineral_Field**
Value of 176.

BWAPI.UnitTypes.Enum.**Resource_Mineral_Field_Type_2**
Value of 177.

BWAPI.UnitTypes.Enum.**Resource_Mineral_Field_Type_3**
Value of 178.

BWAPI.UnitTypes.Enum.**Unused_Cave**
Value of 179.

BWAPI.UnitTypes.Enum.**Unused_Cave_In**
Value of 180.

BWAPI.UnitTypes.Enum.**Unused_Cantina**
Value of 181.

BWAPI.UnitTypes.Enum.**Unused_Mining_Platform**
Value of 182.

BWAPI.UnitTypes.Enum.**Unused_Independant_Command_Center**
Value of 183.

BWAPI.UnitTypes.Enum.**Special_Independant_Starport**
Value of 184.

BWAPI.UnitTypes.Enum.**Unused_Independant_Jump_Gate**
Value of 185.

BWAPI.UnitTypes.Enum.**Unused_Ruins**
Value of 186.

BWAPI.UnitTypes.Enum.**Unused_Khaydarin_Crystal_Formation**
Value of 187.

BWAPI.UnitTypes.Enum.**Resource_Vespene_Geyser**
Value of 188.

BWAPI.UnitTypes.Enum.**Special_Warp_Gate**
Value of 189.

BWAPI.UnitTypes.Enum.**Special_Psi_Disrupter**
Value of 190.

BWAPI.UnitTypes.Enum.**Unused_Zerg_Marker**
Value of 191.

BWAPI.UnitTypes.Enum.**Unused_Terran_Marker**
Value of 192.

BWAPI.UnitTypes.Enum.**Unused_Protoss_Marker**
Value of 193.

BWAPI.UnitTypes.Enum.**Special_Zerg_Beacon**
Value of 194.

BWAPI.UnitTypes.Enum.**Special_Terran_Beacon**
Value of 195.

BWAPI.UnitTypes.Enum.**Special_Protoss_Beacon**
Value of 196.

BWAPI.UnitTypes.Enum.**Special_Zerg_Flag_Beacon**
Value of 197.

BWAPI.UnitTypes.Enum.**Special_Terran_Flag_Beacon**
Value of 198.

BWAPI.UnitTypes.Enum.**Special_Protoss_Flag_Beacon**
Value of 199.

BWAPI.UnitTypes.Enum.**Special_Power_Generator**
Value of 200.

BWAPI.UnitTypes.Enum.**Special_Overmind_Cocoon**
Value of 201.

BWAPI.UnitTypes.Enum.**Spell_Dark_Swarm**
Value of 202.

BWAPI.UnitTypes.Enum.**Special_Floor_Missile_Trap**
Value of 203.

BWAPI.UnitTypes.Enum.**Special_Floor_Hatch**
Value of 204.

BWAPI.UnitTypes.Enum.**Special_Upper_Level_Door**
Value of 205.

BWAPI.UnitTypes.Enum.**Special_Right_Upper_Level_Door**
Value of 206.

BWAPI.UnitTypes.Enum.**Special_Pit_Door**
Value of 207.

BWAPI.UnitTypes.Enum.**Special_Right_Pit_Door**
Value of 208.

BWAPI.UnitTypes.Enum.**Special_Floor_Gun_Trap**
Value of 209.

BWAPI.UnitTypes.Enum.**Special_Wall_Missile_Trap**
Value of 210.

BWAPI.UnitTypes.Enum.**Special_Wall_Flame_Trap**
Value of 211.

BWAPI.UnitTypes.Enum.**Special_Right_Wall_Missile_Trap**
Value of 212.

BWAPI.UnitTypes.Enum.**Special_Right_Wall_Flame_Trap**
Value of 213.

BWAPI.UnitTypes.Enum.**Special_Start_Location**
Value of 214.

BWAPI.UnitTypes.Enum.**Powerup_Flag**
Value of 215.

BWAPI.UnitTypes.Enum.**Powerup_Young_Chrysalis**
Value of 216.

BWAPI.UnitTypes.Enum.**Powerup_Psi_Emitter**
Value of 217.

BWAPI.UnitTypes.Enum.**Powerup_Data_Disk**
Value of 218.

BWAPI.UnitTypes.Enum.**Powerup_Khaydarin_Crystal**
Value of 219.

BWAPI.UnitTypes.Enum.**Powerup_Mineral_Cluster_Type_1**
Value of 220.

BWAPI.UnitTypes.Enum.**Powerup_Mineral_Cluster_Type_2**
Value of 221.

BWAPI.UnitTypes.Enum.**Powerup_Protoss_Gas_Orb_Type_1**
Value of 222.

BWAPI.UnitTypes.Enum.**Powerup_Protoss_Gas_Orb_Type_2**
Value of 223.

BWAPI.UnitTypes.Enum.**Powerup_Zerg_Gas_Sac_Type_1**
Value of 224.

BWAPI.UnitTypes.Enum.**Powerup_Zerg_Gas_Sac_Type_2**
Value of 225.

BWAPI.UnitTypes.Enum.**Powerup_Terran_Gas_Tank_Type_1**
Value of 226.

BWAPI.UnitTypes.Enum.**Powerup_Terran_Gas_Tank_Type_2**
Value of 227.

BWAPI.UnitTypes.Enum.**None**
Value of 228.

BWAPI.UnitTypes.Enum.**AllUnits**
Value of 229.

BWAPI.UnitTypes.Enum.**Men**
Value of 230.

BWAPI.UnitTypes.Enum.**Buildings**
Value of 231.

BWAPI.UnitTypes.Enum.**Factories**
Value of 232.

BWAPI.UnitTypes.Enum.**Unknown**
Value of 233.

BWAPI.UnitTypes.Enum.**MAX**
Value of 234.

UpgradeTypes

Module containing upgrade types.

See also:

BWAPI.UpgradeType

Functions

static BWAPI.UpgradeTypes.**allUpgradeTypes**() → set
Retrieves a set of all UpgradeTypes.

Returns Set of all UpgradeTypes.

Return type *UpgradeTypeset*

Constants

All constants are instances of the *UpgradeType* class

Terran Upgrades

BWAPI.UpgradeTypes.**Terran_Infantry_Armor**

BWAPI.UpgradeTypes.**Terran_Vehicle_Plating**

BWAPI.UpgradeTypes.**Terran_Ship_Plating**

BWAPI.UpgradeTypes.**Terran_Infantry_Weapons**

BWAPI.UpgradeTypes.**Terran_Vehicle_Weapons**

BWAPI.UpgradeTypes.**Terran_Ship_Weapons**

BWAPI.UpgradeTypes.**U_238_Shells**

BWAPI.UpgradeTypes.**Ion_Thrusters**

BWAPI.UpgradeTypes.**Titan_Reactor**

BWAPI.UpgradeTypes.**Ocular_Implants**

BWAPI.UpgradeTypes.**Moebius_Reactor**

BWAPI.UpgradeTypes.**Apollo_Reactor**

BWAPI.UpgradeTypes.**Colossus_Reactor**

BWAPI.UpgradeTypes.**Caduceus_Reactor**

BWAPI.UpgradeTypes.**Charon_Boosters**

Zerg Upgrades

BWAPI.UpgradeTypes.**Zerg_Carapace**

BWAPI.UpgradeTypes.**Zerg_Flyer_Carapace**

BWAPI.UpgradeTypes.**Zerg_Melee_Attacks**

BWAPI.UpgradeTypes.**Zerg_Missile_Attacks**

BWAPI.UpgradeTypes.**Zerg_Flyer_Attacks**

BWAPI.UpgradeTypes.**Ventral_Sacs**

BWAPI.UpgradeTypes.**Antennae**

BWAPI.UpgradeTypes.**Pneumatized_Carapace**

BWAPI.UpgradeTypes.**Metabolic_Boost**

BWAPI.UpgradeTypes.**Adrenal_Glands**

BWAPI.UpgradeTypes.**Muscular_Augments**

BWAPI.UpgradeTypes.**Grooved_Spines**

BWAPI.UpgradeTypes.**Gamete_Meiosis**

BWAPI.UpgradeTypes.**Metasynaptic_Node**

BWAPI.UpgradeTypes.**Chitinous_Plating**

BWAPI.UpgradeTypes.**Anabolic_Synthesis**

Protoss Upgrades

BWAPI.UpgradeTypes.**Protoss_Ground_Armor**

BWAPI.UpgradeTypes.**Protoss_Air_Armor**

BWAPI.UpgradeTypes.**Protoss_Ground_Weapons**

BWAPI.UpgradeTypes.**Protoss_Air_Weapons**

BWAPI.UpgradeTypes.**Protoss_Plasma_Shields**

BWAPI.UpgradeTypes.**Singularity_Charge**

BWAPI.UpgradeTypes.**Leg_Enhancements**

BWAPI.UpgradeTypes.**Scarab_Damage**

BWAPI.UpgradeTypes.**Reaver_Capacity**

BWAPI.UpgradeTypes.**Gravitic_Drive**

BWAPI.UpgradeTypes.**Sensor_Array**

BWAPI.UpgradeTypes.**Gravitic_Boosters**

BWAPI.UpgradeTypes.**Khaydarin_Amulet**

BWAPI.UpgradeTypes.**Apial_Sensors**

BWAPI.UpgradeTypes.**Gravitic_Thrusters**

BWAPI.UpgradeTypes.**Carrier_Capacity**

BWAPI.UpgradeTypes.**Khaydarin_Core**

BWAPI.UpgradeTypes.**Argus_Jewel**

BWAPI.UpgradeTypes.**Argus_Talisman**

Misc

BWAPI.UpgradeTypes.**Upgrade_60**

BWAPI.UpgradeTypes.**None**

BWAPI.UpgradeTypes.**Unknown**

Enum

Enumeration of upgrade types.

Important: Enum values rarely need to be used in Lua.

BWAPI.UpgradeTypes.Enum.**Terran_Infantry_Armor**
Value of 0.

BWAPI.UpgradeTypes.Enum.**Terran_Vehicle_Plating**
Value of 1.

BWAPI.UpgradeTypes.Enum.**Terran_Ship_Plating**
Value of 2.

BWAPI.UpgradeTypes.Enum.**Zerg_Carapace**
Value of 3.

BWAPI.UpgradeTypes.Enum.**Zerg_Flyer_Carapace**
Value of 4.

BWAPI.UpgradeTypes.Enum.**Protoss_Ground_Armor**
Value of 5.

BWAPI.UpgradeTypes.Enum.**Protoss_Air_Armor**
Value of 6.

BWAPI.UpgradeTypes.Enum.**Terran_Infantry_Weapons**
Value of 7.

BWAPI.UpgradeTypes.Enum.**Terran_Vehicle_Weapons**
Value of 8.

BWAPI.UpgradeTypes.Enum.**Terran_Ship_Weapons**
Value of 9.

BWAPI.UpgradeTypes.Enum.**Zerg_Melee_Attacks**
Value of 10.

BWAPI.UpgradeTypes.Enum.**Zerg_Missile_Attacks**
Value of 11.

BWAPI.UpgradeTypes.Enum.**Zerg_Flyer_Attacks**
Value of 12.

BWAPI.UpgradeTypes.Enum.**Protoss_Ground_Weapons**
Value of 13.

BWAPI.UpgradeTypes.Enum.**Protoss_Air_Weapons**
Value of 14.

BWAPI.UpgradeTypes.Enum.**Protoss_Plasma_Shields**
Value of 15.

BWAPI.UpgradeTypes.Enum.**U_238_Shells**
Value of 16.

BWAPI.UpgradeTypes.Enum.**Ion_Thrusters**
Value of 17.

BWAPI.UpgradeTypes.Enum.**Titan_Reactor**
Value of 19.

BWAPI.UpgradeTypes.Enum.**Ocular_Implants**
Value of 20.

BWAPI.UpgradeTypes.Enum.**Moebius_Reactor**
Value of 21.

BWAPI.UpgradeTypes.Enum.**Apollo_Reactor**
Value of 22.

BWAPI.UpgradeTypes.Enum.**Colossus_Reactor**
Value of 23.

BWAPI.UpgradeTypes.Enum.**Ventral_Sacs**
Value of 24.

BWAPI.UpgradeTypes.Enum.**Antennae**
Value of 25.

BWAPI.UpgradeTypes.Enum.**Pneumatized_Carapace**
Value of 26.

BWAPI.UpgradeTypes.Enum.**Metabolic_Boost**
Value of 27.

BWAPI.UpgradeTypes.Enum.**Adrenal_Glands**
Value of 28.

BWAPI.UpgradeTypes.Enum.**Muscular_Augments**
Value of 29.

BWAPI.UpgradeTypes.Enum.**Grooved_Spines**
Value of 30.

BWAPI.UpgradeTypes.Enum.**Gamete_Meiosis**
Value of 31.

BWAPI.UpgradeTypes.Enum.**Metasynaptic_Node**
Value of 32.

BWAPI.UpgradeTypes.Enum.**Singularity_Charge**
Value of 33.

BWAPI.UpgradeTypes.Enum.**Leg_Enhancements**
Value of 34.

BWAPI.UpgradeTypes.Enum.**Scarab_Damage**
Value of 35.

BWAPI.UpgradeTypes.Enum.**Reaver_Capacity**
Value of 36.

BWAPI.UpgradeTypes.Enum.**Gravitic_Drive**
Value of 37.

BWAPI.UpgradeTypes.Enum.**Sensor_Array**
Value of 38.

BWAPI.UpgradeTypes.Enum.**Gravitic_Boosters**
Value of 39.

BWAPI.UpgradeTypes.Enum.**Khaydarin_Amulet**
Value of 40.

BWAPI.UpgradeTypes.Enum.**Apial_Sensors**
Value of 41.

BWAPI.UpgradeTypes.Enum.**Gravitic_Thrusters**
Value of 42.

BWAPI.UpgradeTypes.Enum.**Carrier_Capacity**
Value of 43.

BWAPI.UpgradeTypes.Enum.**Khaydarin_Core**
Value of 44.

BWAPI.UpgradeTypes.Enum.**Argus_Jewel**
Value of 47.

BWAPI.UpgradeTypes.Enum.**Argus_Talisman**
Value of 49.

BWAPI.UpgradeTypes.Enum.**Caduceus_Reactor**
Value of 51.

BWAPI.UpgradeTypes.Enum.**Chitinous_Plating**
Value of 52.

BWAPI.UpgradeTypes.Enum.**Anabolic_Synthesis**
Value of 53.

BWAPI.UpgradeTypes.Enum.**Charon_Boosters**
Value of 54.

BWAPI.UpgradeTypes.Enum.**Upgrade_60**
Value of 55.

BWAPI.UpgradeTypes.Enum.**None**
Value of 56.

BWAPI.UpgradeTypes.Enum.**Unknown**
Value of 57.

BWAPI.UpgradeTypes.Enum.**MAX**
Value of 58.

WeaponTypes

Module containing weapon types.

See also:

BWAPI.WeaponType

Functions

static BWAPI.WeaponTypes.**allWeaponTypes**() → set
Retrieves a set of all WeaponTypes.

Returns Set of all WeaponTypes.

Return type *WeaponTypeset*

static BWAPI.WeaponTypes.**normalWeaponTypes**() → set
Retrieves a set of all defined normal WeaponTypes. This set contains all weapons that are not used for abilities.

Returns Set of normal WeaponTypes.

Return type *WeaponTypeset*

static BWAPI.WeaponTypes.**specialWeaponTypes**() → set
Retrieves a set of all defined special WeaponTypes. This set contains all weapons that are used exclusively for special unit abilities.

Returns Set of special WeaponTypes.

Return type *WeaponTypeset*

Constants

All constants are instances of the *WeaponType* class

Normal Weapons

BWAPI.WeaponTypes.**Gauss_Rifle**

BWAPI.WeaponTypes.**Gauss_Rifle_Jim_Raynor**

BWAPI.WeaponTypes.**C_10_Canister_Rifle**

BWAPI.WeaponTypes.**C_10_Canister_Rifle_Sarah_Kerrigan**

BWAPI.WeaponTypes.**C_10_Canister_Rifle_Samir_Duran**

BWAPI.WeaponTypes.**C_10_Canister_Rifle_Infested_Duran**

BWAPI.WeaponTypes.**C_10_Canister_Rifle_Alexei_Stukov**

BWAPI.WeaponTypes.**Fragmentation_Grenade**

BWAPI.WeaponTypes.**Fragmentation_Grenade_Jim_Raynor**

BWAPI.WeaponTypes.**Spider_Mines**

BWAPI.WeaponTypes.**Twin_Autocannons**

BWAPI.WeaponTypes.**Twin_Autocannons_Alan_Schezar**

BWAPI.WeaponTypes.**Hellfire_Missile_Pack**

BWAPI.WeaponTypes.**Hellfire_Missile_Pack_Alan_Schezar**

BWAPI.WeaponTypes.**Arclite_Cannon**

BWAPI.WeaponTypes.**Arclite_Cannon_Edmund_Duke**

BWAPI.WeaponTypes.**Fusion_Cutter**

BWAPI.WeaponTypes.**Gemini_Missiles**

BWAPI.WeaponTypes.**Gemini_Missiles_Tom_Kazansky**

BWAPI.WeaponTypes.**Burst_Lasers**

BWAPI.WeaponTypes.**Burst_Lasers_Tom_Kazansky**

BWAPI.WeaponTypes.**ATS_Laser_Battery**

BWAPI.WeaponTypes.**ATS_Laser_Battery_Hero**

BWAPI.WeaponTypes.**ATS_Laser_Battery_Hyperion**

BWAPI.WeaponTypes.**ATA_Laser_Battery**

BWAPI.WeaponTypes.**ATA_Laser_Battery_Hero**

BWAPI.WeaponTypes.**ATA_Laser_Battery_Hyperion**

BWAPI.WeaponTypes.**Flame_Thrower**

BWAPI.WeaponTypes.**Flame_Thrower_Gui_Montag**

BWAPI.WeaponTypes.**Arclite_Shock_Cannon**

BWAPI.WeaponTypes.**Arclite_Shock_Cannon_Edmund_Duke**

BWAPI.WeaponTypes.**Longbolt_Missile**

BWAPI.WeaponTypes.**Claws**

BWAPI.WeaponTypes.**Claws_Devouring_One**

BWAPI.WeaponTypes.**Claws_Infested_Kerrigan**

BWAPI.WeaponTypes.**Needle_Spines**

BWAPI.WeaponTypes.**Needle_Spines_Hunter_Killer**

BWAPI.WeaponTypes.**Kaiser_Blades**

BWAPI.WeaponTypes.**Kaiser_Blades_Torrasque**

BWAPI.WeaponTypes.**Toxic_Spores**

BWAPI.WeaponTypes.**Spines**

BWAPI.WeaponTypes.**Acid_Spore**

BWAPI.WeaponTypes.**Acid_Spore_Kukulza**

BWAPI.WeaponTypes.**Glave_Wurm**

BWAPI.WeaponTypes.**Glave_Wurm_Kukulza**

BWAPI.WeaponTypes.**Seeker_Spores**

BWAPI.WeaponTypes.**Subterranean_Tentacle**

BWAPI.WeaponTypes.**Suicide_Infested_Terran**

BWAPI.WeaponTypes.**Suicide_Scourge**

BWAPI.WeaponTypes.**Particle_Beam**

BWAPI.WeaponTypes.**Psi_Blades**

BWAPI.WeaponTypes.**Psi_Blades_Fenix**

BWAPI.WeaponTypes.**Phase_Disruptor**

BWAPI.WeaponTypes.**Phase_Disruptor_Fenix**

BWAPI.WeaponTypes.**Psi_Assault**

BWAPI.WeaponTypes.**Psionic_Shockwave**

BWAPI.WeaponTypes.**Psionic_Shockwave_TZ_Archon**

BWAPI.WeaponTypes.**Dual_Photon_Blasters**

BWAPI.WeaponTypes.**Dual_Photon_Blasters_Mojo**

BWAPI.WeaponTypes.**Dual_Photon_Blasters_Artanis**

BWAPI.WeaponTypes.**Anti_Matter_Missiles**

BWAPI.WeaponTypes.**Anti_Matter_Missiles_Mojo**

BWAPI.WeaponTypes.**Anti_Matter_Missiles_Artanis**

BWAPI.WeaponTypes.**Phase_Disruptor_Cannon**

BWAPI.WeaponTypes.**Phase_Disruptor_Cannon_Danimoth**

BWAPI.WeaponTypes.**Pulse_Cannon**

BWAPI.WeaponTypes.**STS_Photon_Cannon**

BWAPI.WeaponTypes.**STA_Photon_Cannon**

BWAPI.WeaponTypes.**Scarab**

BWAPI.WeaponTypes.**Neutron_Flare**

BWAPI.WeaponTypes.**Halo_Rockets**

BWAPI.WeaponTypes.**Corrosive_Acid**

BWAPI.WeaponTypes.**Subterranean_Spines**

BWAPI.WeaponTypes.**Warp_Blades**

BWAPI.WeaponTypes.**Warp_Blades_Hero**

BWAPI.WeaponTypes.**Warp_Blades_Zeratul**

BWAPI.WeaponTypes.**Independant_Laser_Battery**

BWAPI.WeaponTypes.**Twin_Autocannons_Floor_Trap**

BWAPI.WeaponTypes.**Hellfire_Missile_Pack_Wall_Trap**

BWAPI.WeaponTypes.**Flame_Thrower_Wall_Trap**

BWAPI.WeaponTypes.**Hellfire_Missile_Pack_Floor_Trap**

Special Weapons

BWAPI.WeaponTypes.**Yamato_Gun**

BWAPI.WeaponTypes.**Nuclear_Strike**

BWAPI.WeaponTypes.**Lockdown**

BWAPI.WeaponTypes.**EMP_Shockwave**

BWAPI.WeaponTypes.**Irradiate**

BWAPI.WeaponTypes.**Parasite**

BWAPI.WeaponTypes.**Spawn_Broodlings**

BWAPI.WeaponTypes.**Ensnare**

BWAPI.WeaponTypes.**Dark_Swarm**

BWAPI.WeaponTypes.**Plague**

BWAPI.WeaponTypes.**Consume**

BWAPI.WeaponTypes.**Stasis_Field**

BWAPI.WeaponTypes.**Psionic_Storm**

BWAPI.WeaponTypes.**Disruption_Web**

BWAPI.WeaponTypes.**Restoration**

BWAPI.WeaponTypes.**Mind_Control**

BWAPI.WeaponTypes.**Feedback**

BWAPI.WeaponTypes.**Optical_Flare**

BWAPI.WeaponTypes.**Maelstrom**

Misc

BWAPI.WeaponTypes.**None**

BWAPI.WeaponTypes.**Unknown**

Enum

Enumeration of weapon types.

Important: Enum values rarely need to be used in Lua.

BWAPI.WeaponTypes.Enum.**Gauss_Rifle**
Value of 0.

BWAPI.WeaponTypes.Enum.**Gauss_Rifle_Jim_Raynor**
Value of 1.

BWAPI.WeaponTypes.Enum.**C_10_Canister_Rifle**
Value of 2.

BWAPI.WeaponTypes.Enum.**C_10_Canister_Rifle_Sarah_Kerrigan**
Value of 3.

BWAPI.WeaponTypes.Enum.**Fragmentation_Grenade**
Value of 4.

BWAPI.WeaponTypes.Enum.**Fragmentation_Grenade_Jim_Raynor**
Value of 5.

BWAPI.WeaponTypes.Enum.**Spider_Mines**
Value of 6.

BWAPI.WeaponTypes.Enum.**Twin_Autocannons**
Value of 7.

BWAPI.WeaponTypes.Enum.**Hellfire_Missile_Pack**
Value of 8.

BWAPI.WeaponTypes.Enum.**Twin_Autocannons_Alan_Schezar**
Value of 9.

BWAPI.WeaponTypes.Enum.**Hellfire_Missile_Pack_Alan_Schezar**
Value of 10.

BWAPI.WeaponTypes.Enum.**Arclite_Cannon**
Value of 11.

BWAPI.WeaponTypes.Enum.**Arclite_Cannon_Edmund_Duke**
Value of 12.

BWAPI.WeaponTypes.Enum.**Fusion_Cutter**
Value of 13.

BWAPI.WeaponTypes.Enum.**Gemini_Missiles**
Value of 15.

BWAPI.WeaponTypes.Enum.**Burst_Lasers**
Value of 16.

BWAPI.WeaponTypes.Enum.**Gemini_Missiles_Tom_Kazansky**
Value of 17.

BWAPI.WeaponTypes.Enum.**Burst_Lasers_Tom_Kazansky**
Value of 18.

BWAPI.WeaponTypes.Enum.**ATS_Laser_Battery**
Value of 19.

BWAPI.WeaponTypes.Enum.**ATA_Laser_Battery**
Value of 20.

BWAPI.WeaponTypes.Enum.**ATS_Laser_Battery_Hero**
Value of 21.

BWAPI.WeaponTypes.Enum.**ATA_Laser_Battery_Hero**
Value of 22.

BWAPI.WeaponTypes.Enum.**ATS_Laser_Battery_Hyperion**
Value of 23.

BWAPI.WeaponTypes.Enum.**ATA_Laser_Battery_Hyperion**
Value of 24.

BWAPI.WeaponTypes.Enum.**Flame_Thrower**
Value of 25.

BWAPI.WeaponTypes.Enum.**Flame_Thrower_Gui_Montag**
Value of 26.

BWAPI.WeaponTypes.Enum.**Arclite_Shock_Cannon**
Value of 27.

BWAPI.WeaponTypes.Enum.**Arclite_Shock_Cannon_Edmund_Duke**
Value of 28.

BWAPI.WeaponTypes.Enum.**Longbolt_Missile**
Value of 29.

BWAPI.WeaponTypes.Enum.**Yamato_Gun**
Value of 30.

BWAPI.WeaponTypes.Enum.**Nuclear_Strike**
Value of 31.

BWAPI.WeaponTypes.Enum.**Lockdown**
Value of 32.

BWAPI.WeaponTypes.Enum.**EMP_Shockwave**
Value of 33.

BWAPI.WeaponTypes.Enum.**Irradiate**
Value of 34.

BWAPI.WeaponTypes.Enum.**Claws**
Value of 35.

BWAPI.WeaponTypes.Enum.**Claws_Devouring_One**
Value of 36.

BWAPI.WeaponTypes.Enum.**Claws_Infested_Kerrigan**
Value of 37.

BWAPI.WeaponTypes.Enum.**Needle_Spines**
Value of 38.

BWAPI.WeaponTypes.Enum.**Needle_Spines_Hunter_Killer**
Value of 39.

BWAPI.WeaponTypes.Enum.**Kaiser_Blades**
Value of 40.

BWAPI.WeaponTypes.Enum.**Kaiser_Blades_Torrasque**
Value of 41.

BWAPI.WeaponTypes.Enum.**Toxic_Spores**
Value of 42.

BWAPI.WeaponTypes.Enum.**Spines**
Value of 43.

BWAPI.WeaponTypes.Enum.**Acid_Spore**
Value of 46.

BWAPI.WeaponTypes.Enum.**Acid_Spore_Kukulza**
Value of 47.

BWAPI.WeaponTypes.Enum.**Glave_Wurm**
Value of 48.

BWAPI.WeaponTypes.Enum.**Glave_Wurm_Kukulza**
Value of 49.

BWAPI.WeaponTypes.Enum.**Seeker_Spores**
Value of 52.

BWAPI.WeaponTypes.Enum.**Subterranean_Tentacle**
Value of 53.

BWAPI.WeaponTypes.Enum.**Suicide_Infested_Terran**
Value of 54.

BWAPI.WeaponTypes.Enum.**Suicide_Scourge**
Value of 55.

BWAPI.WeaponTypes.Enum.**Parasite**
Value of 56.

BWAPI.WeaponTypes.Enum.**Spawn_Broodlings**
Value of 57.

BWAPI.WeaponTypes.Enum.**Ensnare**
Value of 58.

BWAPI.WeaponTypes.Enum.**Dark_Swarm**
Value of 59.

BWAPI.WeaponTypes.Enum.**Plague**
Value of 60.

BWAPI.WeaponTypes.Enum.**Consume**
Value of 61.

BWAPI.WeaponTypes.Enum.**Particle_Beam**
Value of 62.

BWAPI.WeaponTypes.Enum.**Psi_Blades**
Value of 64.

BWAPI.WeaponTypes.Enum.**Psi_Blades_Fenix**
Value of 65.

BWAPI.WeaponTypes.Enum.**Phase_Disruptor**
Value of 66.

BWAPI.WeaponTypes.Enum.**Phase_Disruptor_Fenix**
Value of 67.

BWAPI.WeaponTypes.Enum.**Psi_Assault**
Value of 69.

BWAPI.WeaponTypes.Enum.**Psionic_Shockwave**
Value of 70.

BWAPI.WeaponTypes.Enum.**Psionic_Shockwave_TZ_Archon**
Value of 71.

BWAPI.WeaponTypes.Enum.**Dual_Photon_Blasters**
Value of 73.

BWAPI.WeaponTypes.Enum.**Anti_Matter_Missiles**
Value of 74.

BWAPI.WeaponTypes.Enum.**Dual_Photon_Blasters_Mojo**
Value of 75.

BWAPI.WeaponTypes.Enum.**Anti_Matter_Missiles_Mojo**
Value of 76.

BWAPI.WeaponTypes.Enum.**Phase_Disruptor_Cannon**
Value of 77.

BWAPI.WeaponTypes.Enum.**Phase_Disruptor_Cannon_Danimoth**
Value of 78.

BWAPI.WeaponTypes.Enum.**Pulse_Cannon**
Value of 79.

BWAPI.WeaponTypes.Enum.**STS_Photon_Cannon**
Value of 80.

BWAPI.WeaponTypes.Enum.**STA_Photon_Cannon**
Value of 81.

BWAPI.WeaponTypes.Enum.**Scarab**
Value of 82.

BWAPI.WeaponTypes.Enum.**Stasis_Field**
Value of 83.

BWAPI.WeaponTypes.Enum.**Psionic_Storm**
Value of 84.

BWAPI.WeaponTypes.Enum.**Warp_Blades_Zeratul**
Value of 85.

BWAPI.WeaponTypes.Enum.**Warp_Blades_Hero**
Value of 86.

BWAPI.WeaponTypes.Enum.**Platform_Laser_Battery**
Value of 92.

BWAPI.WeaponTypes.Enum.**Independant_Laser_Battery**
Value of 93.

BWAPI.WeaponTypes.Enum.**Twin_Autocannons_Floor_Trap**
Value of 96.

BWAPI.WeaponTypes.Enum.**Hellfire_Missile_Pack_Wall_Trap**
Value of 97.

BWAPI.WeaponTypes.Enum.**Flame_Thrower_Wall_Trap**
Value of 98.

BWAPI.WeaponTypes.Enum.**Hellfire_Missile_Pack_Floor_Trap**
Value of 99.

BWAPI.WeaponTypes.Enum.**Neutron_Flare**
Value of 100.

BWAPI.WeaponTypes.Enum.**Disruption_Web**
Value of 101.

BWAPI.WeaponTypes.Enum.**Restoration**
Value of 102.

BWAPI.WeaponTypes.Enum.**Halo_Rockets**
Value of 103.

BWAPI.WeaponTypes.Enum.**Corrosive_Acid**
Value of 104.

BWAPI.WeaponTypes.Enum.**Mind_Control**
Value of 105.

BWAPI.WeaponTypes.Enum.**Feedback**
Value of 106.

BWAPI.WeaponTypes.Enum.**Optical_Flare**
Value of 107.

BWAPI.WeaponTypes.Enum.**Maelstrom**
Value of 108.

BWAPI.WeaponTypes.Enum.**Subterranean_Spines**
Value of 109.

BWAPI.WeaponTypes.Enum.**Warp_Blades**
Value of 111.

BWAPI.WeaponTypes.Enum.**C_10_Canister_Rifle_Samir_Duran**
Value of 112.

BWAPI.WeaponTypes.Enum.**C_10_Canister_Rifle_Infested_Duran**
Value of 113.

BWAPI.WeaponTypes.Enum.**Dual_Photon_Blasters_Artanis**
Value of 114.

BWAPI.WeaponTypes.Enum.**Anti_Matter_Missiles_Artanis**
Value of 115.

BWAPI.WeaponTypes.Enum.**C_10_Canister_Rifle_Alexei_Stukov**
Value of 116.

BWAPI.WeaponTypes.Enum.**None**
Value of 130.

BWAPI.WeaponTypes.Enum.**Unknown**
Value of 131.

BWAPI.WeaponTypes.Enum.**MAX**
Value of 132.

CHAPTER 9

Indices and tables

- genindex
- search

Lua Module Index

b

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z