Mudlet Lua Function API Reference

Contents

Alphabetical Function Index

A, B, C, D, E, F, G, H, I, K, L, M, O, P, R, S, T, U, W

E

F

G

H

I

selectCurrentLine
selectSection
selectString
send
sendAll
sendGMCP
sendIrc
sendTelnetChannel102
setAreaName
setBackgroundColor
setBackgroundImage
setBgColor
setBold
setBorderBottom
setBorderColor
setBorderLeft
setBorderRight
setBorderTop
setConsoleBufferSize
setCustomEnvColor
setExit
setFgColor
setGauge
setGridMode
setItalics
setLabelClickCallback
setLabelStyleSheet
setLink
setMiniConsoleFontSize
setModulePriority
setPopup
setRoomArea
setRoomChar
setRoomCoordinates
setRoomEnv
setRoomIDbyHash
setRoomName
setRoomUserData
setRoomWeight
setTextFormat
setTriggerStayOpen
setUnderline
setWindowWrap
showCaptureGroups
showColors
showMultimatches
showToolBar
showWindow
spawn
speedwalk
startStopWatch
stopStopWatch

T

Mudlet System Variables

Mudlet defines several global Lua variables that are accessible from anywhere.

Built-in Lua Variables	
Variable Name	**Description**
comand	This variable holds the current user command. This is typically used in alias scripts.
line	This variable holds the content of the current line as being processed by the trigger engine. The engine runs all triggers on each line as it arrives from the MUD.
matches[n]	This Lua table is being used by Mudlet in the context of triggers that use Perl regular expressions. matches[1] holds the entire match, matches[2] holds the first capture group, matches[n] holds the nth-1 capture group. If the trigger uses the Perl style /g switch to evaluate all possible matches of the given regex within the current line, matches[n+1] will hold the second entire match, matches[n+2] the first capture group of the second match and matches[n+m] the m-th capture group of the second match.
multimatches[n][m]	This table is being used by Mudlet in the context of multiline triggers that use Perl regular expression. It holds the table matches[n] as described above for each Perl regular expression based condition of the multiline trigger. multimatches[5][4] may hold the 3rd capture group of the 5th regex in the multiline trigger. This way you can examine and process all relevant data within a single script.

Function Categories

Basic Essential Functions: These functions are generic functions used in normal scripting. These deal with mainly everyday things, like sending stuff and echoing to the screen.

Database Functions: A collection of functions for helping deal with the database.

Date & Time Functions: A collection of functions for handling Date & Time.

Display Functions: A collection of functions for displaying or formatting information on the screen.

File System Functions: A collection of functions for interacting with the file system.

Mapper Functions: A collection of functions that manipulate the mapper and it's related features.

Miscellaneous Functions: **Need verbiage here**

Scripting Object Functions: **Need verbiage here**

Networking Functions: A collection of functions for managing networking.

String Functions: These functions are used to manipulate strings.

Table Functions: These functions are used to manipulate tables. Through them you can add to tables, remove values, check if a value is present in the table, check the size of a table, and more.

UI Functions: These functions are used to construct custom user GUIs. They deal mainly with miniconsole/label/gauge creation and manipulation.

Basic Essentials

These functions are generic functions used in normal scripting. These deal with mainly everyday things, like sending stuff and echoing to the screen.

send

send(command, show on screen)

> This sends "command" directly to the network layer, skipping the alias matching. The optional second argument of type boolean (print) determines if the outgoing command is to be echoed on the screen.

See also: sendAll()

 **Note:** If you want your command to be checked as if it's an alias, use expandAlias() instead - send() will ignore them.

```
send( "Hello Jane" ) --echos the command on the screen
send( "Hello Jane", true ) --echos the command on the screen
send( "Hello Jane", false ) --does not echo the command on the screen

-- use a variable in the send:
send("kick "..target)
```

echo

echo(windowName, text)

> This function appends text at the end of the current line. The current cursor position is ignored. Use moveCursor() and insertText() if you want to print at a different cursor position. If the first argument is omitted the main console is used, otherwise the mini console windowName.

Example:

```
echo( "Hello world\n" ) -- writes "Hello world" to the main screen.
echo( "info", "Hello this is the info window" ) -- writes text to the mini
console named "info" if such a window exists
```

Database Functions

These database functions make using a database with Mudlet easier. They are optional, if you are an expert in SQL, you can use LuaSQL's sqlite driver directly within Mudlet - see it's manual here.

db:add

db:add(sheet reference, table1, …, tableN)

Adds one or more new rows to the specified sheet. If any of these rows would violate a UNIQUE index, a lua error will be thrown and execution will cancel. As such it is advisable that if you use a UNIQUE index, you test those values before you attempt to insert a new row.

Example

```
--Each table is a series of key-value pairs to set the values of the sheet,
--but if any keys do not exist then they will be set to nil or the default
value.
db:add(mydb.enemies, {name="Bob Smith", city="San Francisco"})
db:add(mydb.enemies,
     {name="John Smith", city="San Francisco"},
     {name="Jane Smith", city="San Francisco"},
     {name="Richard Clark"})
```

As you can see, all fields are optional.

db:aggregate

db:aggregate(field reference, aggregate function, query)

Returns the result of calling the specified aggregate function on the field and its sheet. The query is optional.

The supported aggregate functions are:

- COUNT - Returns the total number of records that are in the sheet or match the query.
- AVG - Returns the average of all the numbers in the specified field.
- MAX - Returns the highest number in the specified field.
- MIN - Returns the lowest number in the specified field.
- TOTAL - Returns the value of adding all the contents of the specified field.

Example

```
local mydb = db:get_database("my database")
echo(db:aggregate(mydb.enemies.name, "count"))
```

db:AND

db:AND(sub-expression1, …, sub-expressionN)

Returns a compound database expression that combines all of the simple expressions passed into it; these expressions should be generated with other db: functions such as db:eq, db:like, db:lt and the like.

This compound expression will only find items in the sheet if all sub-expressions match.

db:between

db:between(field reference, lower_bound, upper_bound)

Returns a database expression to test if the field in the sheet is a value between lower_bound and upper_bound. This only really makes sense for numbers and Timestamps.

db:create

db:create(database name, schema table)

Creates and/or modifies an existing database. This function is safe to define at a top-level of a Mudlet script: in fact it is reccommended you run this function at a top-level without any kind of guards. If the named database does not exist it will create it. If the database does exist then it will add any columns or indexes which didn't exist before to that database. If the database already has all the specified columns and indexes, it will do nothing.

The database will be called Database_<sanitized database name>.db and will be stored in the Mudlet configuration directory.

Database tables are called sheets consistantly throughout this documentation, to avoid confusion with Lua tables.

The schema table must be a Lua table array containing table dictionaries that define the structure and layout of each sheet

Example

```
local mydb = db:create("combat_log",
        {
                kills = {
                name = "",
                area = "",
                killed = db:Timestamp("CURRENT_TIMESTAMP"),
                _index = { {"name", "area"} }
                },
                enemies = {
                        name = "",
                        city = "",
                        reason = "",
                        enemied = db:Timestamp("CURRENT_TIMESTAMP"),
                        _index = { "city" },
                        _unique = { "name" },
                        _violations = "IGNORE"
                }
        })
```

The above will create a database with two sheets; the first is kills and is used to track every successful kill, with both where and when the kill happened. It has one index, a compound inde tracking the combination of name and area. The second sheet has two indexes, but one is unique: it isn't possible to add two items to the enemies sheet with the same name.

For sheets with unique indexes, you may specify a _violations key to indicate how the db layer handle cases where the unique index is violated. The options you may use are: * FAIL - the default. A hard error is thrown, cancelling the script. * IGNORE - The command that would add a record that violates uniqueness just fails silently. * REPLACE - The old record which matched the unique index is dropped, and the new one is added to replace it.

Returns a reference of an already existing database. This instance can be used to get references to the sheets (and from there, fields) that are defined within the database. You use these references to construct queries.

If a database has a sheet named enemies, you can obtain a reference to that sheet by simply

doing:

```
local mydb = db:get_database("my database")
local enemies_ref = mydb.enemieslocal name_ref = mydb.enemies.name
```

db:delete

db:delete(sheet reference, query)

Deletes rows from the specified sheet. The argument for query tries to be intelligent:

- If it is a simple number, it deletes a specific row by _row_id
- If it is a table that contains a _row_id (e.g., a table returned by db:get) it deletes just that record.
- Otherwise, it deletes every record which matches the query pattern which is specified as with b:get.
- If the query is simply true, then it will truncate the entire contents of the sheet.

Example

```
enemies = db:fetch(mydb.enemies)
db:delete(mydb.enemies, enemies[1])

db:delete(mydb.enemies, enemies[1]._row_id)
db:delete(mydb.enemies, 5)
db:delete(mydb.enemies, db:eq(mydb.enemies.city, "San Francisco"))
db:delete(mydb.enemies, true)
```

Those deletion commands will do in order:

1. one When passed an actual result table that was obtained from db:fetch, it will delete the record for that table.
2. two When passed a number, will delete the record for that _row_id. This example shows getting the row id from a table.
3. three As above, but this example just passes in the row id directly.
4. four Here, we will delete anything which matches the same kind of query as db:fetch uses--namely, anyone who is in the city of San Francisco.
5. five And finally, we will delete the entire contents of the enemies table.

db:eq

db:eq(field reference, value)

Returns a database expression to test if the field in the sheet is equal to the value.

db:exp

db:exp(string)

Returns the string as-is to the database.

Use this function with caution, but it is very useful in some circumstances. One of the most common of such is incrementing an existing field in a db:set() operation, as so:

```
db:set(mydb.enemies, db:exp("kills + 1"), db:eq(mydb.enemies.name, "Ixokai"))
```

This will increment the value of the kills field for the row identified by the name Ixokai.

But there are other uses, as the underlining database layer provides many functions you can call to do certain things. If you want to get a list of all your enemies who have a name longer then 10 characters, you may do:

```
db:fetch(mydb.enemies, db:exp("length(name) > 10"))
```

Again, take special care with this, as you are doing SQL syntax directly and the library can't help you get things right.

db:fetch

db:fetch(sheet reference, query, order_by, descending)

Returns a table array containing a table for each matching row in the specified sheet. All arguments but sheet are optional. If query is nil, the entire contents of the sheet will be returned.

Query is a string which should be built by calling the various db: expression functions, such as db:eq, db:AND, and such. You may pass a SQL WHERE clause here if you wish, but doing so is very dangerous. If you don't know SQL well, its best to build the expression.

Query may also be a table array of such expressions, if so they will be AND'd together implicitly.

The results that are returned are not in any guaranteed order, though they are usually the same order as the records were inserted. If you want to rely on the order in any way, you must pass a value to the order_by field. This must be a table array listing the columns you want to sort by. It can be { "column1" }, or { "column1", "column2" }

The results are returned in ascending (smallest to largest) order; to reverse this pass true into the final field.

Example

```
db:fetch(mydb.enemies, nil, {"city", "name"})
db:fetch(mydb.enemies, db:eq(mydb.enemies.city, "San Francisco"))
db:fetch(mydb.kills,
     {db:eq(mydb.kills.area, "Undervault"),
     db:like(mydb.kills.name, "%Drow%")}
)
```

The first will fetch all of your enemies, sorted first by the city they reside in and then by their name.

The second will fetch only the enemies which are in San Francisco.

The third will fetch all the things you've killed in Undervault which have Drow in their name.

db:gt

db:gt(field reference, value)

Returns a database expression to test if the field in the sheet is greater than to the value.

db:get_database

db:get_database(database_name)

Returns your database name.

Example

```
local mydb = db:get_database("my database")
```

db:gte

db:gte(field reference, value)

Returns a database expression to test if the field in the sheet is greater than or equal to the value.

db:in_

db:in_(field reference, table array)

Returns a database expression to test if the field in the sheet is one of the values in the table array.

First, note the trailing underscore carefully! It is required.

The following example illustrates the use of in_:

```
local mydb = db:get_database("my database")
local areas = {"Undervault", "Hell", "Purgatory"}

db:fetch(mydb.kills, db:in_(mydb.kills.area, areas))
```

This will obtain all of your kills which happened in the Undervault, Hell or Purgatory. Every db:in_ expression can be written as a db:OR, but that quite often gets very complex.

db:is_nil

db:is_nil(field reference, value)

Returns a database expression to test if the field in the sheet is nil.

db:is_not_nil

db:is_not_nil(field reference, value)

Returns a database expression to test if the field in the sheet is not nil.

db:like

db:like(field reference, pattern)

returns a database expression to test if the field in the sheet matches the specified pattern.

LIKE patterns are not case-sensitive, and allow two wild cards. The first is an underscore which matches any single one character. The second is a percent symbol which matches zero or more of any character.

LIKE with "_" is therefore the same as the "." regular expression.

LIKE with "%" is therefore the same as ".*" regular expression.

db:lt

db:lt(field reference, value)

Returns a database expression to test if the field in the sheet is less than the value.

db:lte

db:lte(field reference, value)

Returns a database expression to test if the field in the sheet is less than or equal to the value.

db:merge_unique

db:merge_unique(sheet reference, table array)

Merges the specified table array into the sheet, modifying any existing rows and adding any that don't exist.

This function is a convenience utility that allows you to quickly modify a sheet, changing existing rows and add new ones as appropriate. It ONLY works on sheets which have a unique index, and only when that unique index is only on a single field. For more complex situations you'll have to do the logic yourself.

The table array may contain tables that were either returned previously by db:fetch, or new tables that you've constructed with the correct fields, or any mix of both. Each table must have a value for the unique key that has been set on this sheet.

For example, consider this database

```
local mydb = db:create("peopledb",
     {
          friends = {
               name = "",
               race = "",
               level = 0,
               city = "",
               _index = { "city" },
               _unique = { "name" }
          }
);
```

Here you have a database with one sheet, which contains your friends, their race, level, and what city they live in. Let's say you want to fetch everyone who lives in San Francisco, you could do:

```
local results = db:fetch(mydb.friends, db:eq(mydb.friends.city, "San
Francisco"))
```

The tables in results are static, any changes to them are not saved back to the database. But after a major radioactive cataclysm rendered everyone in San Francisco a mutant, you could make changes to the tables as so:

```
for _, friend in ipairs(results) do
     friend.race = "Mutant"
end
```

If you are also now aware of a new arrival in San Francisco, you could add them to that existing table array:

```
results[#results+1] = {name="Bobette", race="Mutant", city="San Francisco"}
```

And commit all of these changes back to the database at once with:

```
db:merge_unique(mydb.friends, results)
```

The db:merge_unique function will change the city values for all the people who we previously fetched, but then add a new record as well.

db:not_between

db:not_between(field reference, lower_bound, upper_bound)

Returns a database expression to test if the field in the sheet is not a value between lower_bound and upper_bound. This only really makes sense for numbers and Timestamps.

db:not_eq

db:not_eq(field reference, value)

Returns a database expression to test if the field in the sheet is NOT equal to the value.

db:not_in

db:not_in(field reference, table array)

Returns a database expression to test if the field in the sheet is not one of the values in the table array.

See also: db:in_

db:not_like

db:not_like(field reference, pattern)

Returns a database expression to test if the field in the sheet does not match the specified pattern.
LIKE patterns are not case-sensitive, and allow two wild cards. The first is an underscore which matches any single one character. The second is a percent symbol which matches zero or more of any character.
LIKE with "_" is therefore the same as the "." regular expression.
LIKE with "%" is therefore the same as ".*" regular expression.

db:OR

db:OR(sub-expression1, sub-expression2)

Returns a compound database expression that combines both of the simple expressions passed into it; these expressions should be generated with other db: functions such as db:eq, db:like, db:lt and the like.

This compound expression will find any item that matches either the first or the second sub-expression.

db:set

db:set(field reference, value, query)

The db:set function allows you to set a certain field to a certain value across an entire sheet. Meaning, you can change all of the last_read fields in the sheet to a certain value, or possibly only the last_read fields which are in a certain city. The query argument can be any value

which is appropriate for db:fetch, even nil which will change the value for the specified column for EVERY row in the sheet.
For example, consider a situation in which you are tracking how many times you find a certain type of egg during Easter. You start by setting up your database and adding an Eggs sheet, and then adding a record for each type of egg.

Example

```
local mydb = db:create("egg database", {eggs = {color = "", last_found =
db.Timestamp(false), found = 0}})
        db:add(mydb.eggs,
                {color = "Red"},
                {color = "Blue"},
                {color = "Green"},
                {color = "Yellow"},
                {color = "Black"}
        )
```

Now, you have three columns. One is a string, one a timestamp (that ends up as nil in the database), and one is a number.
You can then set up a trigger to capture from the mud the string, "You pick up a (.*) egg!", and you end up arranging to store the value of that expression in a variable called "myegg".
To increment how many we found, we will do this:

```
myegg = "Red" -- We will pretend a trigger set this.
        db:set(mydb.eggs.found, db:exp("found + 1"), db:eq(mydb.eggs.color,
myegg))
        db:set(mydb.eggs.last_found, db.Timestamp("CURRENT_TIMESTAMP"),
db:eq(mydb.eggs.color, myegg))
```

This will go out and set two fields in the Red egg sheet; the first is the found field, which will increment the value of that field (using the special db:exp function). The second will update the last_found field with the current time.
Once this contest is over, you may wish to reset this data but keep the database around. To do that, you may use a more broad use of db:set as such:

```
db:set(mydb.eggs.found, 0)
db:set(mydb.eggs.last_found, nil)
```

db:update

db:update(sheet reference, table)

This function updates a row in the specified sheet, but only accepts a row which has been previously obtained by db:fetch. Its primary purpose is that if you do a db:fetch, then change the value of a field or tow, you can save back that table.

Example

```
local mydb = db:get_database("my database")
local bob = db:fetch(mydb.friends, db:eq(mydb.friends.name, "Bob"))[1]
bob.notes = "He's a really awesome guy."
db:update(mydb.friends, bob)
```

This obtains a database reference, and queries the friends sheet for someone named Bob. As this returns a table array containing only one item, it assigns that one item to the local variable

named bob. We then change the notes on Bob, and pass it into db:update() to save the changes back.

Date & Time Functions

A collection of functions for handling Date & Time.

datetime:parse

datetime:parse(source, format, as_epoch)

Parses the specified source string, according to the format if given, to return a representation of the date/time. If as_epoch is provided and true, the return value will be a Unix epoch — the number of seconds since 1970. This is a useful format for exchanging date/times with other systems. If as_epoch is false, then a Lua time table will be returned. Details of the time tables are provided in the Lua Manual.

Supported Format Codes

```
%b = Abbreviated Month Name
%B = Full Month Name
%d = Day of Month
%H = Hour (24-hour format)
%I = Hour (12-hour format, requires %p as well)
%p = AM or PM
%m = 2-digit month (01-12)
%M = 2-digit minutes (00-59)
%S = 2-digit seconds (00-59)
%y = 2-digit year (00-99), will automatically prepend 20 so 10 becomes 2010 and
not 1910.
%Y = 4-digit year.
```

getTime

getTime(returntype, format)

returntype takes a boolean value (in Lua anything but false or nil will translate to true). If false, the function will return a table in the following format:

```
{ 'min': #, 'year': #, 'month': #, 'day': #, 'sec': #, 'hour': #, 'msec': # }
```

If true, it will return the date and time as a string using a format passed to the format arg or the default of "yyyy.MM.dd hh:mm:ss.zzz" if none is supplied:

```
2012.02.18 00:52:52.489
```

Format expressions:

```
h               the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM
display)
hh              the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM
display)
H               the hour without a leading zero (0 to 23, even with AM/PM
display)
HH              the hour with a leading zero (00 to 23, even with AM/PM display)
m               the minute without a leading zero (0 to 59)
mm              the minute with a leading zero (00 to 59)
s               the second without a leading zero (0 to 59)
```

```
ss              the second with a leading zero (00 to 59)
z               the milliseconds without leading zeroes (0 to 999)
zzz             the milliseconds with leading zeroes (000 to 999)
AP or A         use AM/PM display. AP will be replaced by either "AM" or "PM".
ap or a         use am/pm display. ap will be replaced by either "am" or "pm".

d               the day as number without a leading zero (1 to 31)
dd              the day as number with a leading zero (01 to 31)
ddd             the abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses
QDate::shortDayName().
dddd            the long localized day name (e.g. 'Monday' to 'Qt::Sunday').
Uses QDate::longDayName().
M               the month as number without a leading zero (1-12)
MM              the month as number with a leading zero (01-12)
MMM             the abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses
QDate::shortMonthName().
MMMM            the long localized month name (e.g. 'January' to 'December').
Uses QDate::longMonthName().
yy              the year as two digit number (00-99)
yyyy            the year as four digit number
```

All other input characters will be ignored. Any sequence of characters that are enclosed in single quotes will be treated as text and not be used as an expression. Two consecutive single quotes ('') are replaced by a single single quote in the output.

Example

```
-- Get time as a table
@getTime()

-- Get time with default string
@getTime(true)

-- Get time without date and milliseconds
@getTime(true, "hh:mm:ss")
```

getTimestamp

getTimestamp(optional console_name, lineNumber)

Returns the timestamp string as it's seen when you enable the timestamps view (blue i button bottom right).

 **Note:** Available since 1.1.0-pre1

Example

```
--Echo the timestamp of the current line in a trigger:
echo(getTimestamp(getLineCount()))
```

Display Functions

A collection of functions for displaying or formatting information on the screen.

display

display(value)

This function will do it's best to show whatever you ask it (a number, string, table, function).

This function can be useful for seeing what values does a table have, for example. Note that this doesn't handle recursive references and will loop infinitely at the moment (Mudlet 2.0-test4). If a value is a string, it'll be in single quotes, and if it's a number, it won't be quoted.

Example

```
-- ask it to display a table
display({a = "somevalue", 1,2,3})
-- or some other target
display(target)
```

showColors

showColors(columns)

shows the named colors currently available in Mudlet's color table. These colors are stored in color_table, in table form. The format is color_table.colorName = {r,g,b}
See Also: bg(), fg(), cecho()

Parameters

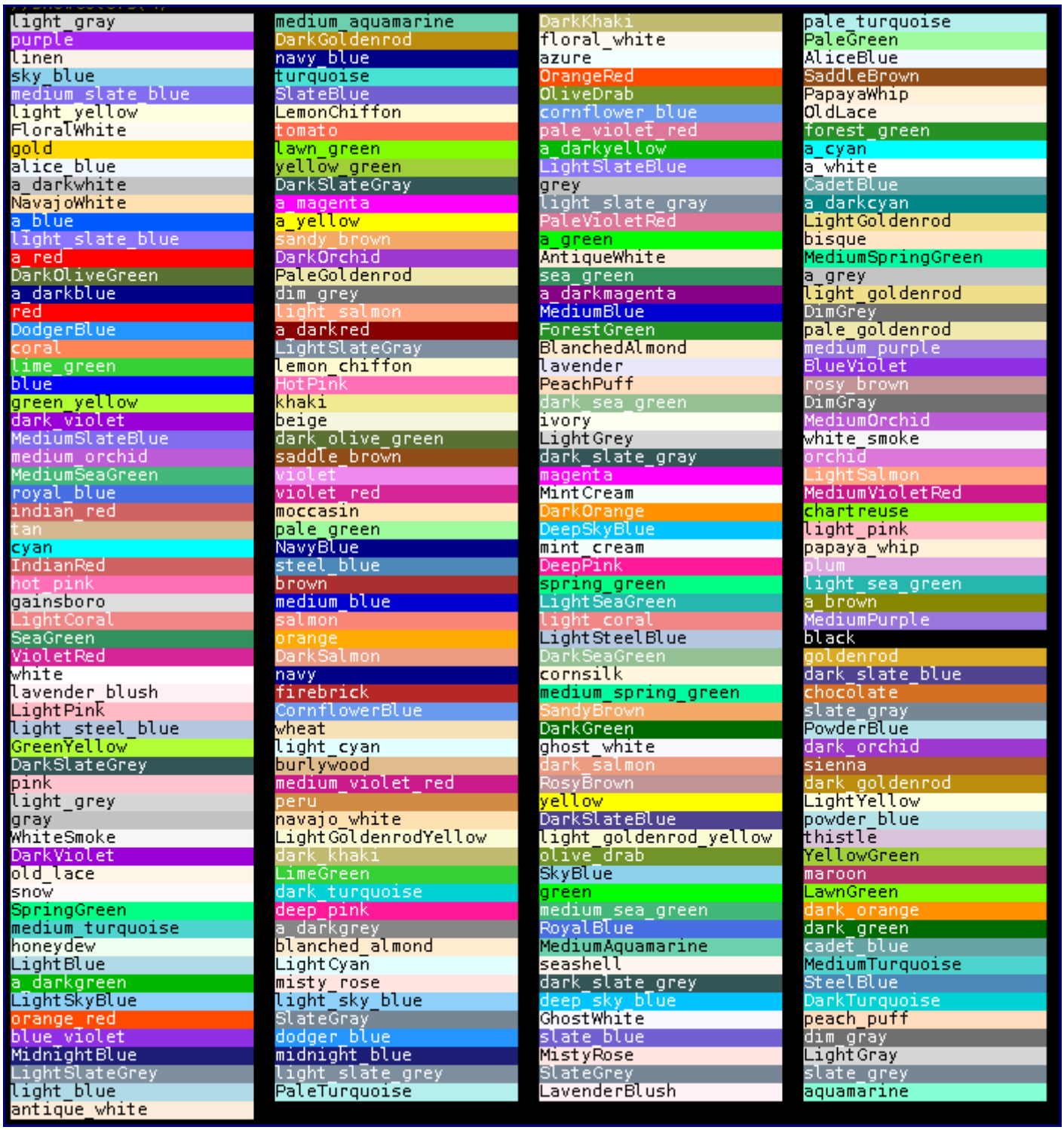
- *columns:*

  Number of columns to print the color table in. Passed as an integer number.

Example

```
showColors(4)
```

The output for this is:



wrapLine

wrapLine(windowName, lineNumber)

Wrap line lineNumber of mini console (window) windowName. This function will interpret \n characters, apply word wrap and display the new lines on the screen. This function may be necessary if you use deleteLine() and thus erase the entire current line in the buffer, but you want to do some further echo() calls after calling deleteLine(). You will then need to re-wrap the last line of the buffer to actually see what you have echoed and get you \n interpreted as newline characters properly. Using this function is no good programming practice and should be avoided. There are better ways of handling situations where you would call deleteLine() and echo afterwards.

Example

```
--This will effectively have the same result as a call to deleteLine() but the
buffer line will not be entirely removed.
--Consequently, further calls to echo() etc. sort of functions are possible
without using wrapLine() unnecessarily.

selectString(line,1);
replace("");
```

File System Functions

A collection of functions for interacting with the file system.

io.exists

io.exists()

Checks to see if a given file or folder exists.
If it exists, it'll return the Lua true boolean value, otherwise false.
Note: Behavior varies based on the user's operating system. Windows requires a specific file, while Linux will accept directories.
See lfs.attributes() for a cross-platform solution.

Example

```
-- This example works on Linux only
if io.exists("/home/vadi/Desktop") then
  echo("This folder exists!")
else
  echo("This folder doesn't exist.")
end

-- This example will work on both Windows and Linux.
if io.exists("/home/vadi/Desktop/file.tx") then
  echo("This file exists!")
else
  echo("This file doesn't exist.")
end
```

lfs.attributes

lfs.attributes(path)

Returns a table with detailed information regarding a file or directory, or nil if path is invalid.

Example

```
fileInfo = lfs.attributes("/path/to/file_or_directory")
if fileInfo then
    if fileInfo.mode == "directory" then
        echo("Path points to a directory.")
    elseif fileInfo.mode == "file" then
        echo("Path points to a file.")
    else
        echo("Path points to: "..fileInfo.mode)
    end
    display(fileInfo) -- to see the detailed information
```

```
else
    echo("The path is invalid (file/directory doesn't exist)")
end
```

Mapper Functions

These are functions that are to be used with the Mudlet Mapper. The mapper is designed to be MUD-generic - it only provides the display and pathway calculations, to be used in Lua scripts that are tailored to the MUD you're playing. For a collection of pre-made scripts and general mapper talk, visit the mapper section of the forums.

To register a script as a mapping one with Mudlet (so Mudlet knows the profile has one and won't bother the user when they open the map), please do this in your script:

```
mudlet = mudlet or {}; mudlet.mapper_script = true
```

addAreaName

areaID = addAreaName(areaName)

> Adds a new area name and returns the new area ID for the new name. If the name already exists, -1 is returned.
> See also: deleteArea(), addRoom()

Example

```
local newID = addAreaName("My house")

if newID == -1 then echo("That area name is already taken :(\n")
else echo("Created new area with the ID of "..newid..".\n") end
```

addMapEvent

addMapEvent(uniquename, event name, parent, display name, arguments)

> Adds a new entry to an existing mapper right-click entry. You can add one with addMapMenu. If there is no display name, it will default to the unique name (which otherwise isn't shown and is just used to differentiate this entry from others). *event name* is the Mudlet event that will be called when this is clicked on, and *arguments* will be passed to the handler function.
> See also: addMapMenu(), removeMapEvent(), getMapEvents()

Example

```
addMapEvent("room a", "onFavorite") -- will make a label "room a" on the map
menu's right click that calls onFavorite

addMapEvent("room b", "onFavorite", "Favorites", "Special Room!", 12345, "arg1",
"arg2", "argn")
```

The last line will make a label "Special Room!" under the "Favorites" menu that on clicking will send all the arguments.

addMapMenu

addMapEvent(uniquename, parent, display name)

Adds a new submenu to the right-click menu that opens when you right-click on the mapper. You can then add more submenus to it, or add entries with addMapEvent().
See also: addMapEvent(), removeMapEvent(), getMapEvents()

Example

```
addMapMenu("Favorites") -- will create a menu, favorites

addMapMenu("Favorites1234343", "Favorites", "Favorites")
```

The last line will create a submenu with the unique id *Favorites123*.. under Favorites with the display name of *Favorites*.

addRoom

addRoom(roomID)

Creates a new room with the given ID, returns true if the room was successfully created.

 **Note:** If you're not using incremental room IDs but room IDs stitched together from other factors or in-game hashes for room IDs - and your room IDs are starting off at 250+million numbers, you need to look into incrementally creating Mudlets room IDs with createRoomID() and associating your room IDs with Mudlets via setRoomIDbyHash() and getRoomIDbyHash(). The reason being is that Mudlet's A* pathfinding implementation from boost cannot deal with extremely large room IDs because the resulting matrices it creates for pathfinding are enormously huge.

See also: createRoomID()

Example

```
local newroomid = createRoomID()
addRoom(newroomid)
```

addSpecialExit

addSpecialExit(roomIDFrom, roomIDTo, command)

Creates a one-way from one room to another, that will use the given command for going through them.
See also: [#clearSpecialExits|clearSpecialExits()]]

Example

```
-- sample alias pattern: ^spe (\d+) (.*?)$
-- mmp.currentroom is your current room ID in this example
addSpecialExit(mmp.currentroom,tonumber(matches[2]), matches[3])
echo("\n SPECIAL EXIT ADDED TO ROOMID:"..matches[2].."", Command:"..matches[3])
centerview(mmp.currentroom)
```

centerview

centerview (roomID)

Centers the map view onto the given room ID. The map must be open to see this take effect. This function can also be used to see the map of an area if you know the number of a room there and the area and room are mapped.

clearRoomUserData

clearRoomUserData(roomID)

Clears all user data from a given room.
See also: setRoomUserData()

Example

```
clearRoomUserData(341)
```

clearSpecialExits

clearSpecialExits(roomID)

Removes all special exits from a room.
See also: addSpecialExit()

Example

```
clearSpecialExits(1337)

if #getSpecialExits(1337) == 0 then -- clearSpecialExits will neve fail on a
valid room ID, this is an example
  echo("All special exits successfully cleared from 1337.\n")
end
```

createMapLabel

labelID = createMapLabel(areaID, text, posx, posy, fgRed, fgGreen, fgBlue, bgRed, bgGreen, bgBlue)

Creates a visual label on the map for all z-levels at given coordinates, with the given background and foreground colors. It returns a label ID that you can use later for deleting it.

The coordinates 0,0 are in the middle of the map, and are in sync with the room coordinates - so using the x,y values of getRoomCoordinates() will place the label near that room.
See also: deleteMapLabel

Example

```
local labelid = createMapLabel( 50, "my map label", 0, 0, 255,0,0,23,0,0)
```

createMapper

createMapper(x, y, width, height)

> Creates a miniconsole window for mapper to render in, the with the given dimensions. You can only create one at a time at the moment.

Example

```
createMapper(0,0,300,300) -- creates a 300x300 mapper top-right of Mudlet
setBorderLeft(305) -- adds a border so text doesn't underlap the mapper display
```

createRoomID

usableId = createRoomID()

> Returns the lowest possible room ID you can use for creating a new room. If there are gaps in room IDs your map uses it, this function will go through the gaps first before creating higher IDs.
> See also: addRoom()

deleteArea

deleteArea(areaID)

> Deletes the given area, permanently. This will also delete all rooms in it!
> See also: addAreaName()

Example

```
deleteArea(23)
```

deleteMapLabel

deleteMapLabel(areaID, labelID)

> Deletes a map label from a specfic area.
> See also: createMapLabel()

Example

```
deleteMapLabel(50, 1)
```

deleteRoom

deleteRoom(roomID)

> Deletes an individual room, and unlinks all exits leading to and from it.

Example

```
deleteRoom(335)
```

getAreaRooms

getAreaRooms(area id)

Returns an indexed table with all rooms IDs for a given area ID (room IDs are values), or *nil* if no such area exists.

Note: On Mudlet versions prior to the 2.0 final release, this function would raise an error.

Example

```
-- using the sample findAreaID() function defined in the getAreaTable() example,
-- we'll define a function that echo's us a nice list of all rooms in an area
with their ID
function echoRoomList(areaname)
  local id, msg = findAreaID(areaname)
  if id then
    local roomlist, endresult = getAreaRooms(id), {}

    -- obtain a room list for each of the room IDs we got
    for _, id in ipairs(roomlist) do
      endresult[id] = getRoomName(id)
    end

    -- now display something half-decent looking
    cecho(string.format(
      "List of all rooms in %s (%d):\n", msg, table.size(endresult)))

    for roomid, roomname in pairs(endresult) do
      cecho(string.format(
        "%6s: %s\n", roomid, roomname))
    end
  elseif not id and msg then
    echo("ID not found; " .. msg)
  else
    echo("No areas matched the query.")
  end
end
```

getAreaTable

getAreaTable()

Returns a key(area name)-value(area id) table with all known areas and their IDs. There is an area with the name of *and an ID of 0 included in it, you should ignore that.*

Example

```
-- example function that returns the area ID for a given area

function findAreaID(areaname)
  local list = getAreaTable()

  -- iterate over the list of areas, matching them with substring match.
  -- if we get match a single area, then return it's ID, otherwise return
  -- 'false' and a message that there are than one are matches
```

```
  local returnid, fullareaname
  for area, id in pairs(list) do
    if area:find(areaname, 1, true) then
      if returnid then return false, "more than one area matches" end
      returnid = id; fullareaname = area
    end
  end

  return returnid, fullareaname
end

-- sample use:
local id, msg = findAreaID("blahblah")
if id then
  echo("Found a matching ID: " .. id")
elseif not id and msg then
  echo("ID not found; " .. msg)
else
  echo("No areas matched the query.")
end
```

getCustomEnvColorTable

envcolors = getCustomEnvColorTable()

> Returns a table with customized environments, where the key is the environment ID and the value is a indexed table of rgb values.

Example

```
{
  envid1 = {r,g,b},
  envid2 = {r,g,b}
}
```

getMapLabel

labelinfo = getMapLabels(areaID, labelID)

> Returns a key-value table describing that particular label in an area. Keys it contains are the *X*, *Y*, *Z* coordinates, *Height* and *Width*, and the *Text* it contains. If the label is an image label, then *Text* will be set to the *no text* string.

Example

```
lua getMapLabels(1658987)
table {
  1: 'no text'
  0: 'test'
}

lua getMapLabel(1658987, 0)
table {
  'Y': -2
  'X': -8
  'Z': 11
  'Height': 3.9669418334961
  'Text': 'test'
```

```
  'Width': 8.6776866912842
}

lua getMapLabel(1658987, 1)
table {
  'Y': 8
  'X': -15
  'Z': 11
  'Height': 7.2520666122437
  'Text': 'no text'
  'Width': 11.21900844574
}
```

getMapLabels

arealabels = getMapLabels(areaID)

Returns an indexed table (that starts indexing from 0) of all of the labels in the area, plus their label text. You can use the label id to deleteMapLabel() it.

Example

```
display(getMapLabels(43))
table {
  0: ''
  1: 'Waterways'
}

deleteMapLabel(43, 0)
display(getMapLabels(43))
table {
  1: 'Waterways'
}
```

getModulePriority

priority = getModulePriority(module name)

Returns the priority of a module as an integer. This determines the order modules will be loaded in - default is 0. Useful if you have a module that depends on another module being loaded first, for example.
See also: setModulePriority()

Example

```
getModulePriority("mudlet-mapper")
```

getPath

getPath(roomID from, roomID to)

Returns a boolean true/false if a path between two room IDs is possible. If it is, the global *speedWalkPath* table is set to all of the directions that have to be taken to get there, and the

global *speedWalkDir* table is set to all of the roomIDs you'll encounter on the way.

Example

```
-- check if we can go to a room - if yes, go to it
if getPath(34,155) then
  gotoRoom(155)
else
  echo("\nCan't go there!")
end
```

getRoomArea

areaID = getRoomArea(roomID)

Returns the area ID of a given room ID. To get the area name, you can check the area ID against the data given by getAreaTable() function, or use the getRoomAreaName() function.

Note: If the room ID does not exist, this function will raise an error.

Example

```
display("Area ID of room #100 is: "..getRoomArea(100))

display("Area name for room #100 is: "..getRoomAreaName(getRoomArea(100)))
```

getRoomAreaName

areaname = getRoomAreaName(areaID)

Returns the area name for a given area id.

Example

```
echo(string.format("room id #455 is in %s.", getRoomAreaName(getRoomArea(455))))
```

getRoomCoordinates

x,y,z = getRoomCoordinates(room ID)

Returns the room coordinates of the given room ID.

Example

```
local x,y,z = getRoomCoordinates(roomID)
echo("Room Coordinates for "..roomID..":")
echo("\n     X:"..x)
echo("\n     Y:"..y)
echo("\n     Z:"..z)
```

getRoomEnv

envID = getRoomEnv(roomID)

Returns the environment ID of a room. The mapper does not store environment names, so you'd need to keep track of which ID is what name yourself.

Example

```
funtion checkID(id)
  echo(strinf.format("The env ID of room #%d is %d.\n", id, getRoomEnv(id)))
end
```

getRoomExits

getRoomExits (roomID)

Returns the currently known non-special exits for a room in an key-index form: *exit* = *exitroomid*, ie:

Example

```
table {
  'northwest': 80
  'east': 78
}
```

getRoomIDbyHash

roomID = getRoomIDbyHash(hash)

Returns a room ID that is associated with a given hash in the mapper. This is primarily for MUDs that make use of hashes instead of room IDs (like Avalon.de MUD). *-1* is returned if no room ID matches the hash.

Example

```
-- example taken from http://forums.mudlet.org/viewtopic.php?f=13&t=2177
_id1 = getRoomIDbyHash( "5dfe55b0c8d769e865fd85ba63127fbc" );
if _id1 == -1 then
  _id1 = createRoomID()
  setRoomIDbyHash( _id1, "5dfe55b0c8d769e865fd85ba63127fbc" )
  addRoom( _id )
  setRoomCoordinates( _id1, 0, 0, -1 )
end
```

getRoomName

roomName = getRoomName(roomID)

Returns the room name for a given room id.

Example

```
echo(string.format("The name of the room id #455 is %s.", getRoomname(455))
```

getRooms

rooms = getRooms()

Returns the list of **all** rooms in the map in an area in roomid - room name format.

Example

```
-- simple, raw viewer for rooms in an area
display(getRooms())
```

getRoomsByPosition

getRoomsByPosition(areaID, x,y,z)

Returns an indexed table of all rooms at the given coordinates in the given area, or an empty one if there are none. This function can be useful for checking if a room exists at certain coordinates, or whenever you have rooms overlapping.

Example

```
-- sample script to determine a room nearby, given a relative direction from the
current room
local otherroom
if matches[2] == "" then
  local w = matches[3]
  local ox, oy, oz, x,y,z = getRoomCoordinates(mmp.currentroom)
  local has = table.contains
  if has({"west", "left", "w", "l"}, w) then
    x = (x or ox) - 1; y = (y or oy); z = (z or oz)
  elseif has({"east", "right", "e", "r"}, w) then
    x = (x or ox) + 1; y = (y or oy); z = (z or oz)
  elseif has({"north", "top", "n", "t"}, w) then
    x = (x or ox); y = (y or oy) + 1; z = (z or oz)
  elseif has({"south", "bottom", "s", "b"}, w) then
    x = (x or ox); y = (y or oy) - 1; z = (z or oz)
  elseif has({"northwest", "topleft", "nw", "tl"}, w) then
    x = (x or ox) - 1; y = (y or oy) + 1; z = (z or oz)
  elseif has({"northeast", "topright", "ne", "tr"}, w) then
    x = (x or ox) + 1; y = (y or oy) + 1; z = (z or oz)
  elseif has({"southeast", "bottomright", "se", "br"}, w) then
    x = (x or ox) + 1; y = (y or oy) - 1; z = (z or oz)
  elseif has({"southwest", "bottomleft", "sw", "bl"}, w) then
    x = (x or ox) - 1; y = (y or oy) - 1; z = (z or oz)
  elseif has({"up", "u"}, w) then
    x = (x or ox); y = (y or oy); z = (z or oz) + 1
  elseif has({"down", "d"}, w) then
    x = (x or ox); y = (y or oy); z = (z or oz) - 1
  end

  local carea = getRoomArea(mmp.currentroom)
  if not carea then mmp.echo("Don't know what area are we in.") return end

  otherroom = select(2, next(getRoomsByPosition(carea,x,y,z)))

  if not otherroom then
    mmp.echo("There isn't a room to the "..w.." that I see - try with an exact
room id.") return
  else
```

```
    mmp.echo("The room "..w.." of us has an ID of "..otherroom)
  end
```

getRoomUserData

data = getRoomUserData(roomID, key (as a string))

> Returns the user data stored at a given room with a given key, or "" if none is stored. Use setRoomUserData() function for setting the user data.

Example

```
display(getRoomUserData(341, "visitcount"))
```

getRoomWeight

room weight = getRoomWeight(roomID)

> Returns the weight of a room. By default, all new rooms have a weight of 1.
> See also: setRoomWeight()

Example

```
display("Original weight of room 541: "..getRoomWeight(541)
setRoomWeight(541, 3)
display("New weight of room 541: "..getRoomWeight(541)
```

getSpecialExits

exits = getSpecialExits(roomID)

> Returns a roomid - command table of all special exits in the room. If there are no special exits in the room, the table returned will be empty.

Example

```
getSpecialExits(1337)

-- results in:
--[[
table {
  12106: 'faiglom nexus'
}
]]
```

getSpecialExitsSwap

exits = getSpecialExitsSwap(roomID)

> Very similar to getSpecialExits() function, but returns the rooms in the command - roomid style.

gotoRoom

gotoRoom (roomID)

Speedwalks you to the given room from your current room if it is able and knows the way. You must turn the map on for this to work, else it will return "(mapper): Don't know how to get there from here :(".

hasExitLock

status = hasExitLock(roomID, direction)

Returns *true* or *false* depending on whenever a given exit leading out from a room is locked. *direction* right now is a number that corresponds to the direction:

```
exitmap = {
  n = 1,
  north = 1,
  ne = 2,
  northeast = 2,
  nw = 3,
  northwest = 3,
  e = 4,
  east = 4,
  w = 5,
  west = 5,
  s = 6,
  south = 6,
  se = 7,
  southeast = 7,
  sw = 8,
  southwest = 8,
  u = 9,
  up = 9,
  d = 10,
  down = 10,
  ["in"] = 11,
  out = 12
}
```

Example

```
-- check if the east exit of room 1201 is locked
display(hasExitLock(1201, 4))
```

See also: lockExit()

hasSpecialExitLock

status = hasSpecialExitLock(from roomID, to roomID, command)

Returns *true* or *false* depending on whenever a given exit leading out from a room is locked. *command* is the action to take to get through the gate.

```
-- lock a special exit from 17463 to 7814 that uses the 'enter feather' command
lockSpecialExit(17463, 7814, 'enter feather', true)

-- see if it is locked: it will say 'true', it is
```

```
display(hasSpecialExitLock(17463, 7814, 'enter feather'))
```

highlightRoom

highlightRoom(id, r1,g1,b1,r2,g2,b2, radius, alpha1, alpha2)

> Highlights a room with the given color, which will override it's environment color. If you use two different colors, then there'll be a shading from the center going outwards that changes into the other color. *highlightRadius* is the radius for the highlight circle - if you don't want rooms beside each other to over lap, use *1* as the radius. *alphaColor1* and *alphaColor2* are transparency values from 0 (completely transparent) to 255 (not transparent at all).
>
> See also: unHighlightRoom()

Note: Available since Mudlet 2.0 final release

```
-- color room #351 red to blue
local r,g,b = unpack(color_table.red)
local br,bg,bb = unpack(color_table.blue)
highlightRoom(351, r,g,b,br,bg,bb, 1, 255, 255)
```

loadMap

boolean = loadMap(file location)

> Loads the map file from a given location. The map file must be in Mudlet's format (not XML or any other) - saved with saveMap().
>
> Returns a boolean for whenever the loading succeeded. Note that the mapper must be open, or this will fail.
>
> See also: saveMap()

```
loadMap("/home/user/Desktop/Mudlet Map.dat")
```

lockExit

lockExit(roomID, direction, lock = true/false)

> Locks a given exit from a room (which means that unless all exits in the incoming room are locked, it'll still be accessible). Direction at the moment is only set as a number, and here's a listing of them:

```
exitmap = {
  n = 1,
  north = 1,
  ne = 2,
  northeast = 2,
  nw = 3,
  northwest = 3,
  e = 4,
  east = 4,
  w = 5,
  west = 5,
```

```
   s = 6,
   south = 6,
   se = 7,
   southeast = 7,
   sw = 8,
   southwest = 8,
   u = 9,
   up = 9,
   d = 10,
   down = 10,
   ["in"] = 11,
   out = 12
 }
```

Example

```
-- lock the east exit of room 1201 so we never path through it
lockExit(1201, 4, true)
```

See also: hasExitLock()

lockRoom

lockRoom (roomID, lock = true/false)

Locks a given room id from future speed walks (thus the mapper will never path through that room).
See also: roomLocked()

Example

```
lockRoom(1, true) -- locks a room if from being walked through when
speedwalking.
lockRoom(1, false) -- unlocks the room, adding it back to possible rooms that
can be walked through.
```

lockSpecialExit

lockSpecialExit (from roomID, to roomID, special exit command, lock = true/false)

Locks a given special exit, leading from one specific room to another that uses a certain command from future speed walks (thus the mapper will never path through that special exit).
See also: hasSpecialExitLock(), lockExit(), lockRoom()

Example

```
lockSpecialExit(1,2,'enter gate', true) -- locks the special exit that does
'enter gate' to get from room 1 to room 2
lockSpecialExit(1,2,'enter gate', false) -- unlocks the said exit
```

removeMapEvent

removeMapEvent (event name)

Removes the given menu entry from a mappers right-click menu. You can add custom ones with addMapEvent().
See also: addMapEvent(), addMapMenu(), getMapEvents()

Example

```
addMapEvent("room a", "onFavorite") -- add one to the general menu
removeMapEvent("room a") -- removes the said menu
```

roomExists

roomExists(roomID)

Returns a boolean true/false depending if the room with that ID exists (is created) or not.

roomLocked

locked = roomLocked(roomID)

Returns true or false whenever a given room is locked.
See also: lockRoom()

Example

```
echo(string.format("Is room #4545 locked? %s.", roomLocked(4545) and "Yep" or
"Nope"))
```

saveMap

saveMap(location)

Saves the map to a given location, and returns true on success. The location folder needs to be already created for save to work.

See also: loadMap()

Example

```
local savedok = saveMap(getMudletHomeDir().."/my fancy map.dat")
if not savedok then
  echo("Couldn't save :(\n")
else
  echo("Saved fine!\n")
end
```

searchRoom

searchRoom (room name)

Searches for rooms that match (by case-insensitive, substring match) the given room name. It returns a key-value table in form of *roomid* = *roomname*, like so:

Example

```
display(searchRoom("master"))

--[[ would result in:
table {
  17463: 'in the branches of the Master Ravenwood'
  3652: 'master bedroom'
  10361: 'Hall of Cultural Masterpieces'
  6324: 'Exhibit of the Techniques of the Master Artisans'
  5340: 'office of the Guildmaster'
  (...)
  2004: 'office of the guildmaster'
  14457: 'the Master Gear'
  1337: 'before the Master Ravenwood Tree'
}
]]
```

If no rooms are found, then an empty table is returned.

setAreaName

setAreaName(areaID, newName)

> Renames an existing area to the new name.

Example

```
setAreaName(2, "My city")
```

setCustomEnvColor

setCustomEnvColor(environmentID, r,g,b,a)

> Creates, or overrides an already created custom color definition for a given environment ID. Note that this will not work on the default environment colors - those are adjustable by the user in the preferences. You can however create your own environment and use a custom color definition for it.

 **Note:** Numbers 1-16 and 257-272 are reserved by Mudlet. 257-272 are the default colors the user can adjust in mapper settings, so feel free to use them if you'd like - but don't overwrite them with this function.

Example

```
setRoomEnv(571, 200) -- change the room's environment ID to something arbitrary,
like 200
local r,g,b = unpack(color_table.blue)
setCustomEnvColor(200, r,g,b, 255) -- set the color of environmentID 200 to blue
```

setExit

setExit(from roomID, to roomID, direction)

> Creates a one-way exit from one room to another using a standard direction - which can be

either one of *n, ne, nw, e, w, s, se, sw, u, d, in, out*, or a number which represents a direction.

Returns *false* if the exit creation didn't work.

Example

```
-- alias pattern: ^exit (\d+) (\w+)$

if setExit(mmp.currentroom, tonumber(matches[2]),matches[3]) then
echo("\nExit set to room:"..matches[2]..",
Direction:"..string.upper(matches[3]))
centerview(mmp.currentroom)
else
mmp.echo("Failed to set the exit.") end
```

This function can also delete exits from a room if you use it like so: setExit(from roomID, -1, direction)

Which will delete an outgoing exit in the specified direction from a room.

```
-- locate the room on the other end, so we can unlink it from there as well if
necessary
local otherroom
if getRoomExits(getRoomExits(mmp.currentroom)[dir])[mmp.ranytolong(dir)] then
  otherroom = getRoomExits(mmp.currentroom)[dir]
end

if setExit(mmp.currentroom, -1, dir) then
  if otherroom then
    if setExit(otherroom, -1, mmp.ranytolong(dir)) then
      mmp.echo(string.format("Deleted the %s exit from %s (%d).",
        dir, getRoomName(mmp.currentroom), mmp.currentroom))
     else mmp.echo("Couldn't delete the incoming exit.") end
  else
    mmp.echo(string.format("Deleted the one-way %s exit from %s (%d).",
      dir, getRoomName(mmp.currentroom), mmp.currentroom))
  end
else
  mmp.echo("Couldn't delete the outgoing exit.")
end
```

You can use these numbers for setting the directions as well:

```
exitmap = {
  n = 1,
  north = 1,
  ne = 2,
  northeast = 2,
  nw = 3,
  northwest = 3,
  e = 4,
  east = 4,
  w = 5,
  west = 5,
  s = 6,
  south = 6,
  se = 7,
  southeast = 7,
  sw = 8,
  southwest = 8,
  u = 9,
  up = 9,
```

```
  d = 10,
  down = 10,
  ["in"] = 11,
  out = 12
}
```

setGridMode

setGridMode(area, true/false)

Enables grid/wilderness view mode for an area - this will cause the rooms to lose their visible exit connections, like you'd see on compressed ASCII maps, both in 2D and 3D view mode.

Example

```
setGridMode(55,true) -- set area with ID 55 to be in grid mode
```

setModulePriority

setModulePriority(module name, priority #)

Sets the module priority on a given module as a number - the module priority determines the order modules are loaded in, which can be helpful if you have ones dependent on each other. This can also be set from the module manager window.
See also: getModulePriority()

```
setModulePriority("mudlet-mapper", 1)
```

setRoomArea

setRoomArea(roomID, newAreaID)

Assigns the given room to a new area. This will have the room be visually moved into the area as well.

setRoomChar

setRoomChar(roomID, character)

Designed for an area in grid mode, this will set a single character to be on a room. You can use "_" to clear it.

Example

```
setRoomChar(431, "#")

setRoomChar(123. "$")
```

setRoomCoordinates

setRoomCoordinates(roomID, x, y, z)

Sets the given room ID to be at the following coordinates visually on the map, where *z* is the up/down level.

 **Note:** 0,0,0 is the center of the map.

Example

```
-- alias pattern: ^set rc (-?\d+) (-?\d+) (-?\d+)$
local x,y,z = getRoomCoordinates(previousRoomID)
local dir = matches[2]

if dir == "n" then
  y = y+1
elseif dir == "ne" then
  y = y+1
  x = x+1
elseif dir == "e" then
  x = x+1
elseif dir == "se" then
  y = y-1
  x = x+1
elseif dir == "s" then
  y = y-1
elseif dir == "sw" then
  y = y-1
  x = x-1
elseif dir == "w" then
  x = x-1
elseif dir == "nw" then
  y = y+1
  x = x-1
elseif dir == "u" or dir == "up" then
  z = z+1
elseif dir == "down" then
  z = z-1
end
setRoomCoordinates(roomID,x,y,z)
centerview(roomID)
```

You can make them relative as well:

```
-- alias pattern: ^src (\w+)$

local x,y,z = getRoomCoordinates(previousRoomID)
local dir = matches[2]

if dir == "n" then
  y = y+1
elseif dir == "ne" then
  y = y+1
  x = x+1
elseif dir == "e" then
  x = x+1
elseif dir == "se" then
  y = y-1
  x = x+1
elseif dir == "s" then
  y = y-1
elseif dir == "sw" then
  y = y-1
  x = x-1
elseif dir == "w" then
```

```
  x = x-1
elseif dir == "nw" then
  y = y+1
  x = x-1
elseif dir == "u" or dir == "up" then
  z = z+1
elseif dir == "down" then
  z = z-1
end
setRoomCoordinates(roomID,x,y,z)
centerview(roomID)
```

setRoomEnv

setRoomEnv(roomID, newEnvID)

> Sets the given room to a new environment ID. You don't have to use any functions to create it - can just set it right away.

Example

```
setRoomEnv(551, 34) -- set room with the ID of 551 to the environment ID 34
```

setRoomIDbyHash

setRoomIDbyHash(roomID, hash)

> Sets the hash to be associated with the given roomID. See also getRoomIDbyHash().

setRoomName

setRoomName(roomID, newName)

> Renames an existing room to a new name.

Example

```
setRoomName(534, "That evil room I shouldn't visit again.")
lockRoom(534, true) -- and lock it just to be safe
```

setRoomUserData

setRoomUserData(roomID, key (as a string), value (as a string))

> Stores information about a room under a given key. Similar to Lua's key-value tables, except only strings may be used here. One advantage of using userdata is that it's stored within the map file itself - so sharing the map with someone else will pass on the user data. You can have as many keys as you'd like.
>
> Returns true if successfully set.
> See also: clearRoomUserData()

Example

```
-- can use it to store room descriptions...
setRoomUserData(341, "description", "This is a plain-looking room.")

-- or whenever it's outdoors or not...
setRoomUserData(341, "ourdoors", "true")

-- how how many times we visited that room
local visited = getRoomUserData(341, "visitcount")
visited = (tonumber(visited) or 0) + 1
setRoomUserData(341, "visitcount", tostring(visited))

-- can even store tables in it, using the built-in yajl.to_string function
setRoomUserData(341, "some table", yajl.to_string({name = "bub", age = 23}))
display("The denizens name is: "..yajl.to_value(getRoomUserData(341, "some
table")).name)
```

setRoomWeight

setRoomWeight(roomID, weight)

Sets a weight to the given roomID. By default, all rooms have a weight of 0 - the higher the weight is, the more likely the room is to be avoided for pathfinding. For example, if travelling across water rooms takes more time than land ones - then you'd want to assign a weight to all water rooms, so they'd be avoided if there are possible land pathways.

To completely avoid a room, make use of lockRoom().

See also: getRoomWeight()

Example

```
setRoomWeight(1532, 3) -- avoid using this room if possible, but don't
completely ignore it
```

speedwalk

speedwalk(dirString, backwards, delay)

A speedwalking function will work on cardinal+ordinal directions (n, ne, e, etc.) as well as u (for up), d (for down), in and out. It can be called to execute all directions directly after each other, without delay, or with a custom delay, depending on how fast your mud allows you to walk. It can also be called with a switch to make the function reverse the whole path and lead you backwards.

Call the function by doing: `speedwalk("YourDirectionsString", true/false, delaytime)`

The delaytime parameter will set a delay between each move (set it to 0.5 if you want the script to move every half second, for instance). It is optional: If you don't indicate it, the script will send all direction commands right after each other. (If you want to indicate a delay, you -have- explicitly indicate true or false for the reverse flag.)

The "YourDirectionsString" contains your list of directions and steps (e.g.: "2n, 3w, u, 5ne"). Numbers indicate the number of steps you want it to walk in the direction specified after it. The directions must be separated by anything other than a letter that can appear in a direction itself. (I.e. you can separate with a comma, spaces, the letter x, etc. and any such combinations, but you cannot separate by the letter "e", or write two directions right next to each other with nothing in-between, such as "wn". If you write a number before every direction, you don't need any further separator. E.g. it's perfectly acceptable to write "3w1ne2e".) The function is not case-sensitive.

If your Mud only has cardinal directions (n,e,s,w and possibly u,d) and you wish to be able to write directions right next to each other like "enu2s3wdu", you'll have to change the pattern slightly. (See the link at the beginning of my post for something like that.)

Likewise, if your Mud has any other directions than n, ne, e, se, s, sw, w, nw, u, d, in, out, the function must be adapted to that.

Example

```
speedwalk("16d1se1u")
-- Will walk 16 times down, once southeast, once up. All in immediate
succession.

speedwalk("2ne,3e,2n,e")
-- Will walk twice northeast, thrice east, twice north, once east. All in
immediate succession.

speedwalk("IN N 3W 2U W", false, 0.5)
-- Will walk in, north, thrice west, twice up, west, with half a second delay
between every move.

speedwalk("5sw - 3s - 2n - w", true)
-- Will walk backwards: east, twice south, thrice, north, five times northeast.
All in immediate succession.

speedwalk("3w, 2ne, w, u", true, 1.25)
-- Will walk backwards: down, east, twice southwest, thrice east, with 1.25
seconds delay between every move.
```

Note: The probably most logical usage of this would be to put it in an alias. For example, have the pattern *^/(.+)$* execute: `speedwalk(matches[2], false, 0.7)` And have *^//(.+)$* execute: `speedwalk(matches[2], true, 0.7)`

Or make aliases like: *^banktohome$* to execute

```
speedwalk("2ne,e,ne,e,3u,in", true, 0.5)
```

unHighlightRoom

unHighlightRoom(roomID)

Unhighlights a room if it was previously highlighted and restores the rooms original environment color.
See also: highlightRoom()

Note: Available since Mudlet 2.0 final release

Example

```
unHighlightRoom(4534)
```

Miscellaneous Functions

feedTriggers

feedTriggers(text)

This function will have Mudlet parse the given text as if it came from the MUD - one great application is trigger testing. You can use \n to represent a new line - you also want to use it before and after the text you're testing, like so:

```
feedTriggers("\nYou sit yourself down.\n")
```

The function also accept ANSI color codes that are used in MUDs. A sample table can be found here.

Example

```
feedTriggers("\nThis is \27[1;32mgreen\27[0;37m, \27[1;31mred\27[0;37m, \
27[46mcyan background\27[0;37m," ..
"\27[32;47mwhite background and green foreground\27[0;37m.\n")
```

expandAlias

expandAlias(command,true/false)

Runs the command as if it was from the command line - so aliases are checked and if none match, it's sent to the the game. If the second argument is false, it will hide the command from being echoed back in your buffer. Defaults to true.

Example

```
expandAlias("t rat")

-- don't echo the command
expandAlias("t rat", false)
```

Note: If you want to be using the *matches* table after calling *expandAlias*, you should save it first as *local oldmatches = matches* before calling *expandAlias*, since *expandAlias* will overwrite it after using it again.

feedTriggers

feedTriggers(text)

This function will have Mudlet parse the given text as if it came from the MUD - one great application is trigger testing. You can use \n to represent a new line - you also want to use it before and after the text you're testing, like so:

```
feedTriggers("\nYou sit yourself down.\n")
```

The function also accept ANSI color codes that are used in MUDs. A sample table can be found here.

Example

```
feedTriggers("\nThis is \27[1;32mgreen\27[0;37m, \27[1;31mred\27[0;37m, \
27[46mcyan background\27[0;37m," ..
"\27[32;47mwhite background and green foreground\27[0;37m.\n")
```

getMudletHomeDir

getMudletHomeDir()

Returns the current home directory of the current profile. This can be used to store data, save statistical information, or load resource files from packages.

Example

```
-- save a table
table.save(getMudletHomeDir().."/myinfo.dat", myinfo)

-- or access package data. The forward slash works even on Windows fine
local path = getMudletHomeDir().."/mypackagename"
```

playSoundFile

playSoundFile(fileName)

This function plays a sound file. On 2.0, it can play most sound formats and up to 4 sounds simulaneously.

Parameters

- *fileName:*

  Exact path of the sound file.

Example

```
-- play a sound in Windows
playSoundFile([[C:\My folder\boing.wav]])

-- play a sound in Linux
playSoundFile([[/home/myname/Desktop/boingboing.wav]])

-- play a sound from a package
playSoundFile(getMudletHomeDir().. [[/mypackage/boingboing.wav]])
```

registerAnonymousEventHandler

registerAnonymousEventHandler(event name, function name)

Registers a function to an event handler, not requiring you to set one up via s cript.

At the moment, it's not possible to use handlers inside namespaces, or unregister them.

Example

```
-- example taken from the God Wars 2 (http://godwars2.org) Mudlet UI - forces
```

```
the window to keep to a certain size
function keepStaticSize()
  setMainWindowSize(1280,720)
end -- keepStaticSize

registerAnonymousEventHandler("sysWindowResizeEvent", "keepStaticSize")
```

spawn

spawn(read function, process to spawn)

> Spawns a process and opens a communicatable link with it - *read function* is the function you'd like to use for reading output from the process, and *t* is a table containing functions specific to this connection - *send(data)*, *true/false = isRunning()*, and *close()*.

Example

```
-- simple example on a program that quits right away, but prints whatever it
gets using the 'display' function
local f = spawn(display, "ls")
display(f.isRunning())
f.close()
```

Mudlet Object Functions

appendCmdLine

appendCmdLine()

> Appends text to the main input line.

Example

```
-- adds the text "55 backpacks" to whatever is currently in the input line
appendCmdLine("55 backpacks")

-- makes a link, that when clicked, will add "55 backpacks" to the input line
echoLink("press me", "appendCmdLine'55 backpack'", "Press me!")
```

clearCmdLine

> clearCmdLine()
> Clears the input line of any text that's been entered.

Example

```
-- don't be evil with this!
clearCmdLine()
```

createStopWatch

createStopWatch()

This function creates a stop watch. It is high resolution time measurement tool. Stop watches can be started, stopped, reset and asked how much time has passed since the stop watch has been started.

 **Note:** it's best to re-use stopwatch IDs if you can - Mudlet at the moment does not delete them, so creating more and more would use more memory.

Returns: The ID of a high resolution clock with milliseconds to measure time more accurately.

Example

In a global script you create all stop watches that you need in your system and store the respective stopWatch-IDs in global variables:

```
fightStopWatch = createStopWatch() -- you store the watchID in a global variable
to access it from anywhere
```

Then you can start the stop watch in some trigger/alias/script with:

```
startStopWatch( fightStopWatch )
```

To stop the watch and measure its time in e.g. a trigger script you can write:

```
fightTime = stopStopWatch( fightStopWatch )
echo( "The fight lasted for " .. fightTime .. " seconds." )
resetStopWatch( fightStopWatch )
```

You can also measure the elapsed time without having to stop the stop watch with getStopWatchTime.

disableAlias

disableAlias(name)

Disables/deactivates the alias by it’s name. If several aliases have this name, they’ll all be disabled.

Parameters

- *name:*

The name of the alias. Passed as a string.

Examples

```
--Disables the alias called 'my alias'
disableAlias("my alias")
```

disableKey

disableKey(name)

Disables key a key (macro) or a key group. When you disable a key group, all keys within the group will be implicitly disabled as well.

Parameters

- *name:*

  The name of the key or group to identify what you'd like to disable.

Examples

```
-- you could set multiple keys on the F1 key and swap their use as you wish by
disabling and enabling them
disableKey("attack macro")
disableKey("jump macro")
enableKey("greet macro")
```

disableTimer

disableTimer(name)

Disables a timer from running it's script when it fires - so the timer cycles will still be happening, just no action on them. If you'd like to permanently delete it, use killTrigger instead.

Parameters

- *name:*

  Expects the timer ID that was returned by tempTimer on creation of the timer or the name of the timer in case of a GUI timer.

Example

```
--Disables the timer called 'my timer'
disableTimer("my timer")
```

disableTrigger

disableTrigger(name)

Disables a trigger that was previously enabled.

Parameters

- *name:*

  Expects the trigger ID that was returned by tempTrigger or other temp*Trigger variants, or the name of the trigger in case of a GUI trigger.

Example

```
-- Disables the trigger called 'my trigger'
disableTrigger("my trigger")
```

enableAlias

enableAlias(name)

Enables/activates the alias by it's name. If several aliases have this name, they'll all be enabled.

Parameters

- *name:*

  Expects the alias ID that was returned by tempTrigger on creation of the alias or the name of the alias in case of a GUI alias.

Example

```
--Enables the alias called 'my alias'
enableAlias("my alias")
```

enableKey

enableKey(name)

Enables a key (macro) or a group of keys (and thus all keys within it that aren't explicitly deactivated).

Parameters

- *name:*

  The name of the group that identifies the key.

```
-- you could use this to disable one key set for the numpad and activate another
disableKey("fighting keys")
enableKey("walking keys")
```

enableTimer

enableTimer(name)

Enables or activates a timer that was previously disabled.

Parameters

- *name:*

  Expects the timer ID that was returned by tempTimer on creation of the timer or the name of the timer in case of a GUI timer.

```
-- enable the timer called 'my timer' that you created in Mudlets timers section
enableTimer("my timer")

-- or disable & enable a tempTimer you've made
timerID = tempTimer(10, [[echo("hi!")]])

-- it won't go off now
disableTimer(timerID)
-- it will continue going off again
enableTimer(timerID)
```

enableTrigger

enableTrigger(name)

Enables or activates a trigger that was previously disabled.

Parameters

- *name:*

  Expects the trigger ID that was returned by tempTrigger or by any other temp*Trigger variants, or the name of the trigger in case of a GUI trigger.

```
-- enable the trigger called 'my trigger' that you created in Mudlets triggers
section
enableTrigger("my trigger")

-- or disable & enable a tempTrigger you've made
triggerID = tempTrigger("some text that will match in a line", [[echo("hi!")]])

-- it won't go off now when a line with matching text comes by
disableTrigger(triggerID)

-- and now it will continue going off again
enableTrigger(triggerID)
```

exists

exists(name, type)

Tells you how many things of the given type exist.

Parameters

- *name:*

  The name or the id returned by tempTimer to identify the item.

- *type:*

  The type can be 'alias', 'trigger', or 'timer'.

Example

```
echo("I have " .. exists("my trigger", "trigger") .. " triggers called 'my
trigger'!")
```

You can also use this alias to avoid creating duplicate things, for example:

```
-- this code doesn't check if an alias already exists and will keep creating new
aliases
permAlias("Attack", "General", "^aa$", [[send ("kick rat")]])

-- while this code will make sure that such an alias doesn't exist first
-- we do == 0 instead of 'not exists' because 0 is considered true in Lua
if exists("Attack", "alias") == 0 then
    permAlias("Attack", "General", "^aa$", [[send ("kick rat")]])
end
```

getButtonState

getButtonState()

This function can only be used inside a toggle button script

Returns *2* if button is checked, and *1* if it's not.

Example

```
checked = getButtonState();
if checked == 1 then
    hideExits()
else
    showExits()
end;
```

invokeFileDialog

invokeFileDialog(fileOrFolder, dialogTitle)

Opens a file chooser dialog, allowing the user to select a file or a folder visually. The function returns the selected path or "" if there was none chosen.

Parameters

- *fileOrFolder: true* for file selection, *false* for folder selection.
- *dialogTitle*: the code to do when the timer is up - wrap it in [[ ]], or provide a Lua function

Examples

```
function whereisit()
  local path = invokeFileDialog(false, "Where should we save the file? Select a
folder and click Open")

  if path == "" then return nil else return path end
end
```

isActive

isActive(name, type)

You can use this function to check if something, or somethings, are active.

Parameters

- *name:*

  The name or the id returned by tempTimer to identify the item.

- *type:*

  The type can be 'alias', 'trigger', or 'timer'.

Example

```
echo("I have " .. isActive("my trigger", "trigger") .. " currently active
trigger(s) called 'my trigger'!")
```

isPrompt

isPrompt()

Returns true or false depending on if the current line being processed is a prompt. This infallible feature is available for MUDs that supply GA events (to check if yours is one, look to bottom-right of the main window - if it doesn't say <No GA>, then it supplies them).

Example use could be as a Lua function, making closing gates on a prompt real easy.

Example

```
-- make a trigger pattern with 'Lua function', and this will trigger on every
prompt!
return isPrompt()
```

killAlias

killAlias(name)

Deletes an alias with the given name. If several aliases have this name, they'll all be deleted.

Parameters

- *name:*

  The name or the id returned by tempTimer to identify the alias.

```
--Deletes the alias called 'my alias'
killAlias("my alias")
```

killTimer

killTimer(id)

Deletes a tempTimer.

Note: Non-temporary timers that you have set up in the GUI cannot be deleted with this function. Use disableTimer() and enableTimer() to turn them on or off.

Parameters

- *id:* the ID returned by tempTimer.

  Returns true on success and false if the timer id doesn't exist anymore (timer has already fired) or the timer is not a temp timer.

Example

```
-- create the timer and remember the timer ID
timerID = tempTimer(10, [[echo("hello!")]])

-- delete the timer
if killTimer(timerID) then echo("deleted the timer") else echo("timer is already
deleted") end
```

killTrigger

killTrigger(id)

Deletes a tempTrigger.

Parameters

- *id:*

  The ID returned by tempTimer to identify the item. ID is a string and not a number.

  Returns true on success and false if the trigger id doesn't exist anymore (trigger has already fired) or the trigger is not a temp trigger.

permAlias

permAlias(name, parent, regex, lua code)

Creates a persistent alias that stays after Mudlet is restarted and shows up in the Script Editor.

Parameters

- *name:*

  The name you'd like the alias to have.

- *parent:*

  The name of the group, or another alias you want the trigger to go in - however if such a group/alias doesn't exist, it won't do anything. Use "" to make it not go into any groups.

- *regex:*

  The pattern that you'd like the alias to use.

- *lua code:*

  The script the alias will do when it matches.

Example

```
-- creates an alias called "new alias" in a group called "my group"
permAlias("new alias", "my group", "^test$", [[echo ("say it works! This alias
will show up in the script editor too.")]])
```

 **Note:** Mudlet by design allows duplicate names - so calling permAlias with the same name will keep creating new aliases. You can check if an alias already exists with the exists function.

permGroup

permGroup(name, itemtype)

Creates a new group of a given type at the root level (not nested in any other groups). This group will persist through Mudlet restarts.

Parameters

- *name:*

  The name of the new group you want to create.

- *itemtype:*

  The name of the timer, trigger, or alias.

 **Note:** Added to Mudlet in the 2.0 final release.

```
--create a new trigger group
permGroup("Combat triggers", "trigger")

--create a new alias group only if one doesn't exist already
if exists("Defensive aliases", "alias") == 0 then
  permGroup("Defensive aliases", "alias")
end
```

permRegexTrigger

permRegexTrigger(name, parent, pattern, lua code)

Creates a persistent trigger with a *regex* pattern that stays after Mudlet is restarted and shows up in the Script Editor.

Parameters

- *name* is the name you'd like the trigger to have.
- *parent* is the name of the group, or another trigger you want the trigger to go in - however if such a group/trigger doesn't exist, it won't do anything. Use "" to make it not go into any groups.
- *pattern table* is a table of patterns that you'd like the trigger to use - it can be one or many.
- *lua code* is the script the trigger will do when it matches.

Example

```
-- Create a regex trigger that will match on the prompt to record your status
permRegexTrigger("Prompt", "", {"^(\d+)h, (\d+)m"}, [[health =
tonumber(matches[2]; mana = tonumber(matches[3])]]
```

 **Note:** Mudlet by design allows duplicate names - so calling permRegexTrigger with the same name will keep creating new triggers. You can check if a trigger already exists with the exists() function.

permSubstringTrigger

permSubstringTrigger(name, parent, pattern, lua code)

Creates a persistent trigger with a substring pattern that stays after Mudlet is restarted and shows up in the Script Editor.

Parameters

- *name* is the name you'd like the trigger to have.
- *parent* is the name of the group, or another trigger you want the trigger to go in - however if such a group/trigger doesn't exist, it won't do anything. Use "" to make it not go into any groups.

- *pattern table* is a table of patterns that you'd like the trigger to use - it can be one or many.
- *lua code* is the script the trigger will do when it matches.

Example

```
-- Create a trigger to highlight the word "pixie" for us
permSubstringTrigger("Highlight stuff", "General", {"pixie"},
[[selectString(line, 1) bg("yellow") resetFormat()]])

-- Or another trigger to highlight several different things
permSubstringTrigger("Highlight stuff", "General", {"pixie", "cat", "dog",
"rabbit"},
[[selectString(line, 1) fg ("blue") bg("yellow") resetFormat()]])
```

 **Note:** Mudlet by design allows duplicate names - so calling permSubstringTrigger with the same name will keep creating new triggers. You can check if a trigger already exists with the exists() function.

permTimer

permTimer(name, parent, seconds, lua code)

Creates a persistent timer that stays after Mudlet is restarted and shows up in the Script Editor.

Parameters

- *name*

  Is the name of the timer.

- *parent*

  Is the name of the timer group you want the timer to go in..

- *seconds*

  Is a number specifying a delay after which the timer will do the lua code you give it as a string.

- *lua code* is the code with string you are doing this to.

Example

```
permTimer("my timer", "first timer group", 4.5, [[send ("my timer that's in my
first timer group fired!")]])
```

 **Note:** Mudlet by design allows duplicate names - so calling permTimer with the same name will keep creating new timers. You can check if a timer already exists with the exists() function.

printCmdLine

printCmdLine(text)

Replaces the current text in the input line, and sets it to the given text.

```
printCmdLine("say I'd like to buy ")
```

raiseEvent

raiseEvent(event_name, arg-1, … arg-n)

Raises the event event_name. The event system will call the main function (the one that is called exactly like the script name) of all such scripts that have registered event handlers. If an event is raised, but no event handler scripts have been registered with the event system, the event is ignored and nothing happens. This is convenient as you can raise events in your triggers, timers, scripts etc. without having to care if the actual event handling has been implemented yet - or more specifically how it is implemented. Your triggers raise an event to tell the system that they have detected a certain condition to be true or that a certain event has happened. How - and if - the system is going to respond to this event is up to the system and your trigger scripts don't have to care about such details. For small systems it will be more convenient to use regular function calls instead of events, however, the more complicated your system will get, the more important events will become because they help reduce complexity very much.

The corresponding event handlers that listen to the events raised with raiseEvent() need to use the script name as function name and take the correct number of arguments.

Example

raiseEvent("fight") a correct event handler function would be: myScript(event_name). In this example raiseEvent uses minimal arguments, name the event name. There can only be one event handler function per script, but a script can still handle multiple events as the first argument is always the event name. So you can call your own special handlers for individual events. The reason behind this is that you should rather use many individual scripts instead of one huge script that has all your function code etc. Scripts can be organized very well in trees and thus help reduce complexity on large systems.

remember

remember("variable")
This function flags a variable to be saved by Mudlet's variable persistence system.

Parameters

- *variable*

  Variable that you are saving. Can be a table or regular variable. Name must be passed as a string.

Example

```
remember("table_Weapons")
remember("var_EnemyHeight")
```

Variables are automatically unpacked into the global namespace when the profile is loaded.
They are saved to "SavedVariables.lua" when the profile is closed or saved.

resetStopWatch

resetStopWatch(watchID)

This function resets the time to 0:0:0.0, but does not start the stop watch. You can start it with startStopWatch → createStopWatch

setConsoleBufferSize

setConsoleBufferSize(consoleName, linesLimit, sizeOfBatchDeletion)

Sets the maximum number of lines can a buffer (main window or a miniconsole) can hold.

Parameters

- *consoleName:*

  The name of the window

- *linesLimit:*

  Sets the amount of lines the buffer should have.

 **Note:** Mudlet performs extremely efficiently with even huge numbers, so your only limitation is your computers memory (RAM).

- *sizeOfBatchDeletion:*

  Specifies how many lines should Mudlet delete at once when you go over the limit - it does it in bulk because it's efficient to do so.

Example

```
-- sets the main windows size to 5 million lines maximum - which is more than
enough!
setConsoleBufferSize("main", 5000000, 1000)
```

setTriggerStayOpen

setTriggerStayOpen(name, number)

Sets for how many more lines a trigger script should fire or a chain should stay open after the trigger has matched - so this allows you to extend or shorten the *fire length* of a trigger. The main use of this function is to close a chain when a certain condition has been met.

Parameters

- *name:* The name of the trigger which has a fire length set (and which opens the chain).
- *number*: 0 to close the chain, or a positive number to keep the chain open that much longer.

Examples

```
-- if you have a trigger that opens a chain (has some fire length) and you'd
like it to be closed
-- on the next prompt, you could make a trigger inside the chain with a Lua
function pattern of:
return isPrompt()
-- and a script of:
```

```
setTriggerStayOpen("'''Parent trigger name'''", 0)
-- to close it on the prompt!
```

startStopWatch

startStopWatch(watchID)
: Starts the stop watch. → createStopWatch()

stopStopWatch

stopStopWatch(watchID)
: Stops the stop watch and returns the elapsed time in milliseconds in form of 0.001. → createStopWatch()
Returns time as a number

tempAlias

aliasID = tempAlias(regex, code to do)
: Creates a temporary alias - temporary in the sense that it won't be saved when Mudlet restarts (unless you re-create it). The alias will go off as many times as it matches, it is not a one-shot alias. The function returns an ID for subsequent enableAlias(), disableAlias() and killAlias() calls.

Parameters

- *regex:* Alias pattern in regex.
- *code to do:* The code to do when the alias fires - wrap it in [[ ]].

Examples

```
myaliasID = tempAlias("^hi$", [[send ("hi") echo ("we said hi!")]])

-- you can also delete the alias later with:
killAlias(myaliasID)
```

tempBeginOfLineTrigger

tempBeginOfLineTrigger(part of line, code to do)
: Creates a trigger that will go off whenever the part of line it's provided with matches the line right from the start (doesn't matter what the line ends with). This trigger isn't temporary in the sense that it'll go off only once (it'll go off as often as it matches), but rather it won't be saved when Mudlet is closed. The function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Parameters

- *part of line:* Start of the line that you'd like to match.
- *code to do:* The code to do when the trigger fires - wrap it in [[ ]].

Examples

```
mytriggerID = tempBeginOfLineTrigger("Hello", [[echo("We matched!")]])

--[[ now this trigger will match on any of these lines:
Hello
Hello!
Hello, Bob!

but not on:
Oh, Hello
Oh, Hello!
]]
```

tempColorTrigger

tempColorTrigger(foregroundColor, backgroundColor, code)

Makes a color trigger that triggers on the specified foreground and background color. Both colors need to be supplied in form of these simplified ANSI 16 color mode codes. The function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Parameters

- *foregroundColor:* The foreground color you'd like to trigger on.
- *backgroundColor*: The background color you'd like to trigger on.
- *code*: The code you'd like the trigger to run, as a string.

Color codes

```
0 = default text color
1 = light black
2 = dark black
3 = light red
4 = dark red
5 = light green
6 = dark green
7 = light yellow
8 = dark yellow
9 = light blue
10 = dark blue
11 = light magenta
12 = dark magenta
13 = light cyan
14 = dark cyan
15 = light white
16 = dark white
```

Examples

```
-- This script will re-highlight all text in blue foreground colors on a black
background with a red foreground color
-- on a blue background color until another color in the current line is being
met. temporary color triggers do not
-- offer match_all or filter options like the GUI color triggers because this is
rarely necessary for scripting.
-- A common usage for temporary color triggers is to schedule actions on the
basis of forthcoming text colors in a particular context.
tempColorTrigger(9,2,[[selectString(matches[1],1); fg("red"); bg("blue");]] );
```

tempExactMatchTrigger

tempExactMatchTrigger(exact line, code to do)

Creates a trigger that will go off whenever the line from the game matches the provided line exactly (ends the same, starts the same, and looks the same). You don't need to use any of the regex symbols here (^ and $). This trigger isn't temporary in the sense that it'll go off only once (it'll go off as often as it matches), but rather it won't be saved when Mudlet is closed. The function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Parameters

- *exact line:* Exact line you'd like to match.
- *code to do:* The code to do when the trigger fires - wrap it in [[ ]].

Examples

```
mytriggerID = tempExactMatchTrigger("You have recovered balance on all limbs.",
[[echo("We matched!")]])
```

tempLineTrigger

tempLineTrigger(from, howMany, LuaCode)

Temporary trigger that will fire on *n* consecutive lines following the current line. This is useful to parse output that is known to arrive in a certain line margin or to delete unwanted output from the MUD - the trigger does not require any patterns to match on. The function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Returns trigger ID as a string.

Note: You can use this ID to enable/disable or kill this trigger later on.

Example

```
--Will fire 3 times with the line from the MUD.
tempLineTrigger( 1, 3, )

--Will fire 20 lines after the current line and fire twice on 2 consecutive
lines.
tempLineTrigger( 20, 2, )
```

tempRegexTrigger

tempRegexTrigger(regex, code to do)

Creates a temporary regex trigger that executes the code whenever it matches. The function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Parameters

- *regex:* The regular expression that lines will be matched on.
- *code to do*: The code to do when the timer is up - wrap it in [[ ]].

Examples

```
-- create a non-duplicate trigger that matches on any line and calls a function
html5log = html5log or {}
if html5log.trig then killTrigger(html5log.trig) end
html5log.trig = tempRegexTrigger("^", "html5log.recordline()")
```

tempTimer

tempTimer(time, code to do)

Creates a temporary one-shot timer and returns the timer ID, which you can use with enableTimer(), disableTimer() and killTimer() functions. You can use 2.3 seconds or 0.45 etc. After it has fired, the timer will be deactivated and destroyed, so it will only go off once. See the Technical Manual here for a more detailed introduction to tempTimer.

Parameters

- *time:* The time in seconds for which to set the timer for - you can use decimals here for precision. The timer will go off *x* given seconds after you make it.
- *code to do*: The code to do when the timer is up - wrap it in [[ ]], or provide a Lua function.

Examples

```
-- wait half a second and then run the command
tempTimer( 0.5, [[send("kill monster")]] )

-- or an another example - two ways to 'embed' variable in a code for later:
local name = matches[2]
tempTimer(2, [[send("hello, ]]..name..[[ !")]])
-- or:
tempTimer(2, function()
  send("hello, "..name)
end)
```

Note: Double brackets, e.g: [[ ]] can be used to quote strings in Lua. The difference to the usual `" " quote syntax is that `[[ ]] also accepts the character ". Consequently, you don't have to escape the " character in the above script. The other advantage is that it can be used as a multiline quote, so your script can span several lines.

Note: Lua code that you provide as an argument is compiled from a string value when the timer fires. This means that if you want to pass any parameters by value e.g. you want to make a function call that uses the value of your variable myGold as a parameter you have to do things like this:

```
tempTimer( 3.8, [[echo("at the time of the tempTimer call I had ]] .. myGold ..
[[ gold.")]] )

-- tempTimer also accepts functions (and thus closures) - which can be an easier
way to embed variables and make the code for timers look less messy:

local variable = matches[2]
tempTimer(3, function () send("hello, " .. variable) end)
```

tempTrigger

tempTrigger(substring, code to do)

Creates a temporary substring trigger that executes the code whenever it matches. The

function returns the trigger ID for subsequent enableTrigger(), disableTrigger() and killTrigger() calls. The trigger will go off multiple times until you disable or destroy it.

Parameters

- *substring:*: The substring to look for - this means a part of the line. If your provided text matches anywhere within the line from the game, the trigger will go off.
- *code to do*: The code to do when the timer is up - wrap it in [[ ]].

Example:

```
-- this example will highlight the contents of the "target" variable.
-- It will also delete the previous trigger it made when you call it again, so
you're only ever highlighting one name
if id then killTrigger(id) end
id = tempTrigger(target, [[selectString("]] .. target .. [[", 1) fg("gold")
resetFormat()]])

-- a simpler trigger to replace "hi" with "bye" whenever you see it
tempTrigger("hi", [[selectString("hi", 1) replace("bye")]])
```

tempButton

tempButton(group name, button text, orientation)

Creates a temporary button.

Parameters

- *group name:*: The toolbar to place the button into.
- *button text*: The text to place on the button.
- *orientation*: a number, 0 - horizontal orientation for the button, or 1 - vertical orientation for the button.

Networking Functions

A collection of functions for managing networking.

disconnect

disconnect()

Disconnects you from the game right away. Note that this will *not* properly log you out of the game.

Example

```
disconnect()
```

downloadFile

downloadFile(saveto, url)

Downloads the resource at the given url into the saveto location on disk. This does not pause the script until the file is downloaded - instead, it lets it continue right away and downloads in the background. When a download is finished, the sysDownloadDone event is raised (with the

saveto location as the argument), or when a download fails, the sysDownloadError event is raised with the reason for failure. You may call downloadFile multiple times and have multiple downloads going on at once - but they aren't guaranteed to be downloaded in the same order that you started them in.

Note: Requires Mudlet 2.0+

Example

```
-- this example will check the Imperian homepage to see how many players are on
right now

-- in an alias, download the Imperian homepage for stats processing
downloadFile(getMudletHomeDir().."/page.html", "http://www.imperian.com/")

-- then create a new script with the name of downloaded_file, add the event
handler
-- for sysDownloadDone, and use this to parse the webpage and display the amount
function downloaded_file(_, filename)
  -- is the file that downloaded ours?
  if not filename:match("page", 1, true) then return end

  -- parse our ownloaded file for the player count
  io.input(filename)
  local s = io.read("*all")
  local pc = s:match([[<a href='players.php%?search=who'>(%d+)</a>]])
  display("Imperian has "..tostring(pc).." players on right now.")
  io.open():close()
  os.remove(filename)
end
```

getNetworkLatency

getNetworkLatency()

Returns the last measured response time between the sent command and the server reply e.g. 0.058 (=58 milliseconds lag) or 0.309 (=309 milliseconds). Also known as server lag.

Example

Need example

openUrl

openUrl (url)

Opens the default OS browser for the given URL.

Example

```
openUrl("http://google.com")
openUrl("www.mudlet.org")
```

reconnect

reconnect()

Force-reconnects (so if you're connected, it'll disconnect) you to the game.

Example

```
-- you could trigger this on a log out message to reconnect, if you'd like
reconnect()
```

sendAll

sendAll(list of things to send, [echo back or not])

send()'s a list of things to the game. If you'd like the commands not to be shown, include *false* at the end.

Example

```
-- instead of using many send() calls, you can use one sendAll
sendAll("outr paint", "outr canvas", "paint canvas")
-- can also have the commands not be echoed
sendAll("hi", "bye", false)
```

sendGMCP

sendGMCP(command)

Sends a GMCP message to the server. The IRE document on GMCP has information about what can be sent, and what tables it will use, etcetera.

See Also: Scripting Manual: GMCP Scripting

Example

```
--This would send "Core.KeepAlive" to the server, which resets the timeout
sendGMCP("Core.KeepAlive")

--This would send a request for the server to send an update to the
gmcp.Char.Skills.Groups table.
sendGMCP("Char.Skills.Get {}")

--This would send a request for the server to send a list of the skills in the
--vision group to the gmcp.Char.Skills.List table.

sendGMCP([[Char.Skills.Get { "group": "vision"}]])

--And finally, this would send a request for the server to send the info for
--hide in the woodlore group to the gmcp.Char.Skills.Info table

sendGMCP([[Char.Skills.Get { "group": "woodlore", "name": "hide"}]])
```

sendIrc

sendIrc(channel, message)

Sends a message to an IRC channel or person. You must have the IRC window open, and if speaking to a channel, be joined in that channel. IRC currently only works on the freenode network and password-protected channels aren't supported.

Parameters

- *channel:*

  The channel to send the message to. Can be #<channelname> to send to a channel, or <person name> to send to a person. Passed as a string.

- *message:*

  The message to send. Passed as a string.

Example

```
--This would send "hello from Mudlet!" to the channel #mudlet on freenode.net
sendIrc("#mudlet", "hello from Mudlet!")
--This would send "identify password" in a private message to Nickserv on
freenode.net
sendIrc("Nickserv", "identify password")
```

sendTelnetChannel102

sendTelnetChannel102(msg)

Sends a message via the 102 subchannel back to the MUD (that's used in Aardwolf). The msg is in a two byte format - see `help telopts` in Aardwolf on how that works.

Example

```
-- turn prompt flags on:
sendTelnetChannel102("\52\1")

-- turn prompt flags off:
sendTelnetChannel102("\52\2")
```

String Functions

string.byte

string.byte(string [, i [, j]])
mystring:byte([, i [, j]])

Returns the internal numerical codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for `i` is `1`; the default value for `j` is `i`.
Note that numerical codes are not necessarily portable across platforms.
See also: string.char

Example

```
--The following call will return the ASCII values of "A", "B" and "C"
a, b, c = string.byte("ABC", 1, 3)

echo(a .. " - " .. b .. " - " .. c) -- echos "65 - 66 - 67"
```

string.char

string.char(···)

Receives zero or more integers. Returns a string with length equal to the number of

arguments, in which each character has the internal numerical code equal to its corresponding argument.

Note: Numerical codes are not necessarily portable across platforms.

See also: string.byte

Example

```
--The following call will return the string "ABC" corresponding to the ASCII
values 65, 66, 67
mystring = string.char(65, 66, 67)
```

string.cut

string.cut(string, maxLen)

Cuts string to the specified maximum length.
Returns the modified string.

Parameters

- *string:*

  The text you wish to cut. Passed as a string.

- *maxLen:*

  The maximum length you wish the string to be. Passed as an integer number.

Example

```
--The following call will return 'abc' and store it in myString
mystring = string.cut("abcde", 3)
--You can easily pad string to certain length. Example below will print 'abcde
' e.g. pad/cut string to 10 characters.
local s = "abcde"
s = string.cut(s .. "          ", 10)   -- append 10 spaces
echo("'" .. s .. "'")
```

string.dump

string.dump()

Need information here!!!

Example

Need example

string.enclose

string.enclose(String)

Wraps a string with [[ ]]
Returns the altered string.

Parameters

- *String:*

  The string to enclose. Passed as a string.

Example

```
--This will echo '[[Oh noes!]]' to the main window
echo("'" .. string.enclose("Oh noes!") .. "'")
```

string.ends

string.ends(String, Suffix)

Test if string is ending with specified suffix.
Returns true or false.
See also: string.starts

Parameters

- *String:*

  The string to test. Passed as a string.

- *Suffix:*

  The suffix to test for. Passed as a string.

Example

```
--This will test if the incoming line ends with "in bed" and if not will add it
to the end.
if not string.ends(line, "in bed") then
  echo("in bed\n")
end
```

string.find

string.find()

Need description

Example

Need example

string.findPattern

string.findPattern(text, pattern)

Return first matching substring or nil.

Parameters

- *text:*

  The text you are searching the pattern for.

- *pattern:*

The pattern you are trying to find in the text.

Example

Following example will print: "I did find: Troll" string.

```
local match = string.findPattern("Troll is here!", "Troll")
if match then
   echo("I did find: " .. match)
end
```

This example will find substring regardless of case.

```
local match = string.findPattern("Troll is here!",
string.genNocasePattern("troll"))
if match then
    echo("I did find: " .. match)
end
```

- Return value:

  nil or first matching substring

See also: string.genNocasePattern()

string.format

string.format()
Need description here.
Example

Need example

string.genNocasePattern

string.genNocasePattern(s)
Generate case insensitive search pattern from string.

Parameters

- s:

Example
Following example will generate and print "123[aA][bB][cC]" string.

```
echo(string.genNocasePattern("123abc"))
```

- Return value:

  case insensitive pattern string

string.gfind

string.gfind()
Need description here.

Example

Need example

string.gmatch

string.gmatch()
: Need description here.

Example

Need example

string.gsub

string.gsub()
: Need description here.

Example

Need example

string.len

string.len(String)
mystring:len()
: Receives a string and returns its length. The empty string "" has length 0. Embedded zeros are counted, so "a\000bc\000" has length 5.

Parameters

- *String:*

  The string you want to find the length of. Passed as a string.

Example

Need example

string.lower

string.lower(String)
mystring:lower()
: Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged. The definition of what an uppercase letter is depends on the current locale.
  See also: string.upper

Example

Need example

string.match

string.match()

Need description here.

Example

Need example

string.rep

string.rep(String, n)
mystring:rep(n)

Returns a string that is the concatenation of `n` copies of the string `String`.

Example

Need example

string.reverse

string.reverse(string)
mystring:reverse()

Returns a string that is the string `string` reversed.

Parameters

- *string:*

  The string to reverse. Passed as a string.

Example

```
mystring = "Hello from Lua"
echo(mystring:reverse()) -- displays 'auL morf olleH'
```

string.split

string.split(string, delimiter)
myString:split(delimiter)

Splits a string into a table by the given delimiter. Can be called against a string (or variable holding a string) using the second form above.
Returns a table containing the split sections of the string.

Parameters

- *string:*

  The string to split. Parameter is not needed if using second form of the syntax above. Passed as a string.

- *delimiter:*

  The delimiter to use when splitting the string. Passed as a string.

Example

```
-- This will split the string by ", " delimiter and print the resulting table to
the main window.
names = "Alice, Bob, Peter"
name_table = string.split(names, ", ")
display(name_table)

--The alternate method
names = "Alice, Bob, Peter"
name_table = names:split(", ")
display(name_table)
```

Either method above will print out:
table {
1: 'Alice'
2: 'Bob'
3: 'Peter'
}

string.starts

string.starts(string, prefix)
Test if string is starting with specified prefix.
Returns true or false
See also: string.ends

Parameters

- *string:*

  The string to test. Passed as a string.

- *prefix:*

  The prefix to test for. Passed as a string.

Example

```
--The following will see if the line begins with "You" and if so will print a
statement at the end of the line
if string.starts(line, "You") then
  echo("====oh you====\n")
end
```

string.sub

string.sub()
Need description here.
Example

Need example

string.title

string.title(string)
string:title()

Capitalizes the first character in a string.
Returns the altered string.

Parameters

- *string:*

  The string to modify. Not needed if you use the second form of the syntax above.

Example

```
--Variable testname is now Anna.
testname = string.title("anna")
--Example will set test to "Bob".
test = "bob"
test = test:title()
```

string.trim

string.trim(string)

Trims string, removing all 'extra' white space at the beginning and end of the text.
Returns the altered string.

Parameters

- *string:*

  The string to trim. Passed as a string.

Example

```
--This will print 'Troll is here!', without the extra spaces.
local str = string.trim("  Troll is here!  ")
echo("'" .. str .. "'")
```

string.upper

string.upper(string)
mystring:upper()

Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged. The definition of what a lowercase letter is depends on the current locale.

Parameters

- *string:*

  The string you want to change to uppercase

Example

```
-- displays 'RUN BOB RUN'
local str = string.upper("run bob run")
```

See also: string.lower

Table Functions

table.complement

table.complement (set1, set2)

Returns a table that is the relative complement of the first table with respect to the second table. Returns a complement of key/value pairs.

Parameters

- *table1:*
- *table2:*

table.concat

table.concat(table, delimiter, startingindex, endingindex)

Joins a table into a string. Each item must be something which can be transformed into a string.
Returns the joined string.
See also: string.split

Parameters

- **table:**

  The table to concatenate into a string. Passed as a table.

- **delimiter:**

  Optional string to use to separate each element in the joined string. Passed as a string.

- **startingindex:**

  Optional parameter to specify which index to begin the joining at. Passed as an integer.

- **endingindex:**

  Optional parameter to specify the last index to join. Passed as an integer.

Examples

```
--This shows a basic concat with none of the optional arguments
testTable = {1,2,"hi","blah",}
testString = table.concat(testTable)
--testString would be equal to "12hiblah"

--This example shows the concat using the optional delimiter
testString = table.concat(testTable, ", ")
--testString would be equal to "1, 2, hi, blah"

--This example shows the concat using the delimiter and the optional starting
index
testString = table.concat(testTable, ", ", 2)
```

```
--testString would be equal to "2, hi, blah"

--And finally, one which uses all of the arguments
testString = table.concat(testTable, ", ", 2, 3)
--testString would be equal to "2, hi"
```

table.contains

table.contains (t, value)

Determines if a table contains a value as a key or as a value (recursive).
Returns true or false

Parameters

- **t:**

  The table in which you are checking for the presence of the value.

- **value:**

  The value you are checking for within the table.

Example

```
local test_table = { "value1", "value2", "value3", "value4" }
if table.contains(test_table, "value1") then
   echo("Got value 1!")
else
   echo("Don't have it. Sorry!")
end
```

This example would always echo the first one, unless you remove value1 from the table.

table.foreach

table.intersection

table.insert

table.insert(table, [pos,] value)

Inserts element **value** at position **pos** in **table**, shifting up other elements to open space, if necessary. The default **value** for **pos** is n+1, where n is the length of the table, so that a call table.insert(t,x) inserts x at the end of table t.
See also: table.remove

Parameters

- **table:**

  The table in which you are inserting the value

- **pos:**

  Optional argument, determining where the value will be inserted.

- **value:**

  The variable that you are inserting into the table. Can be a regular variable, or even a table or function*.

 **Note:** Inserting a function into a table is not good coding practice, and will not turn out how you think it would.

table.index_of

table.is_empty

table.is_empty(table)
Check if a table is devoid of any values.

Parameters

- **table:**

  The table you are checking for values.

table.load

table.load(location, table)
Load a table from an external file into mudlet.
See also: table.save

Parameters

- **location:**

  Where you are loading the table from. Can be anywhere on your computer.

- **table:**

  The table that you are loading into - it must exist already.

  Example:

```
-- This will load the table mytable from the lua file mytable present in your
Mudlet Home Directory.
mytable = {}
table.load(getMudletHomeDir().."/mytable.lua", mytable) -- using / is OK on
Windows too.
-- You can load a table from anywhere on your computer, but it's preferable to
have them consolidated somewhere connected to Mudlet.
```

table.maxn

table.maxn(Table)
Returns the largest positive numerical index of the given table, or zero if the table has no positive numerical indices. (To do its job this function does a linear traversal of the whole table.)

table.n_union

table.n_complement

table.n_intersection

table.pickle

table.remove

table.remove(table, value_position)

Remove a value from an indexed table, by the values position in the table.
See also: table.insert

Parameters

- **table**

  The indexed table you are removing the value from.

- **value_position**

  The indexed number for the value you are removing.

Example

```
testTable = { "hi", "bye", "cry", "why" }
table.remove(testTable, 1) -- will remove hi from the table
-- new testTable after the remove
testTable = { "bye", "cry", "why" }
-- original position of hi was 1, after the remove, position 1 has become bye
-- any values under the removed value are moved up, 5 becomes 4, 4 becomes 3,
etc
```

 **Note:** To remove a value from a key-value table, it's best to simply change the value to nil.

```
testTable = { test = "testing", go = "boom", frown = "glow" }
table.remove(testTable, test) -- this will error
testTable.test = nil -- won't error
testTable["test"] = nil -- won't error
```

table.save

table.save(location, table)

Save a table into an external file in **location**.
See also: table.load

Parameters

- **location:**

  Where you want the table file to be saved. Can be anywhere on your computer.

- **table:**

The table that you are saving to the file.

Example:

```
-- Saves the table mytable to the lua file mytable in your Mudlet Home Directory
table.save(getMudletHomeDir().."/mytable.lua", mytable)
```

table.sort

table.sort(Table [, comp])

Sorts table elements in a given order, in-place, from `Table[1]` to `Table[n]`, where `n` is the length of the table.

If `comp` is given, then it must be a function that receives two table elements, and returns true when the first is less than the second (so that not `comp(a[i+1],a[i])` will be true after the sort). If `comp` is not given, then the standard Lua operator < is used instead.

The sort algorithm is not stable; that is, elements considered equal by the given order may have their relative positions changed by the sort.

table.size

table.size (t)

Gets the actual size of non-index based tables.
Returns a number.

Parameters

- *t:*

The table you are checking the size of.

 **Note:** For index based tables you can get the size with the # operator: This is the standard Lua way of getting the size of index tables i.e. ipairs() type of tables with numerical indices. To get the size of tables that use user defined keys instead of automatic indices (pairs() type) you need to use the function table.size() referenced above.

```
local test_table = { "value1", "value2", "value3", "value4" }
myTableSize = #test_table
-- This would return 4.
local myTable = { 1 = "hello", "key2" = "bye", "key3" = "time to go" }
table.size(myTable)
-- This would return 3.
```

table.setn

table.unpickle

table.update

table.union

UI Functions

appendBuffer

appendBuffer(name)

Pastes the previously copied rich text (including text formats like color etc.) into user window name.
See also: paste()

Parameters

- *name:*

  The name of the user window to paste into. Passed as a string.

Example

```
--selects and copies an entire line to user window named "Chat"
selectCurrentLine()
copy()
appendBuffer("Chat")
```

bg

bg(colorName)

Changes the background color of the text. Useful for highlighting text.
See Also: fg(), setBgColor()

Parameters

- *colorName:*

  The name of the color to set the background to. 

Example

```
--This would change the background color of the text on the current line to
magenta
selectCurrentLine()
bg("magenta")
```

calcFontSize

calcFontSize(fontSize)

Used to calculate the number of pixels wide and high a character would be on a mini console at fontSize.
Returns two numbers, width/height
See Also: setMiniConsoleFontSize(), getMainWindowSize()

Parameters

- *fontSize:*

  The font size you are wanting to calculate pixel sizes for. Passed as an integer number.

Example

```
--this snippet will calculate how wide and tall a miniconsole designed to hold 4
lines of text 20 characters wide
--would need to be at 9 point font, and then changes miniconsole Chat to be that
size
local width,height = calcFontSize(9)
width = width * 20
height = height * 4
resizeWindow("Chat", width, height)
```

cecho

cecho(window, text)

Echoes text that can be easily formatted with colour tags.
See Also: decho(), hecho()

Parameters

- *window:*

  Optional - the window name to echo to - can either be none or "main" for the main window, or the miniconsoles name.

- *text:*

  The text to display, with color names inside angle brackets <>, ie <*red*>. If you'd like to use a background color, put it after a double colon : - <*:red*>. You can use the <*reset*> tag to reset to the default color. You can select any from this list: 

Example

```
cecho("Hi! This text is <red>red, <blue>blue, <green> and green.")

cecho("<:green>Green background on normal foreground. Here we add an
<ivory>ivory foreground.")

cecho("<blue:yellow>Blue on yellow text!")

cecho("myinfo", "<green>All of this text is green in the myinfo miniconsole.")
```

cinsertText

cinsertText(window, text)

inserts text at the current cursor position, with the possibility for color tags.
See Also: cecho()

Parameters

- *window:*

  Optional - the window name to echo to - can either be none or "main" for the main window, or the miniconsoles name.

- *text:*

  The text to display, with color names inside angle brackets <>, ie <*red*>. If you'd like to use a background color, put it after a double colon : - <*:red*>. You can use the <*reset*> tag to reset to the default color. You can select any from this list: 

Example

```
cinsertText("Hi! This text is <red>red, <blue>blue, <green> and green.")

cinsertText("<:green>Green background on normal foreground. Here we add an
<ivory>ivory foreground.")

cinsertText("<blue:yellow>Blue on yellow text!")

cinsertText("myinfo", "<green>All of this text is green in the myinfo
miniconsole.")
```

clearUserWindow

clearUserWindow(name)

Clears the window or miniconsole with the name given as argument.

Parameters

- *name:*

  The name of the user window to clear. Passed as a string.

Example

```
--This would clear a user window, or miniconsole with the name "Chat"
clearUserWindow("Chat")
```

clearWindow

clearWindow([optional name])

Clears the window or miniconsole with the name given as argument (removes all text from it). If you don't give it a name, it will clear the main window.
See also: clearUserWindow()

Parameters

- name:

  The name of the user window to clear. Passed as a string.

Example

```
--This would clear a label, user window, or miniconsole with the name "Chat"
clearWindow("Chat")

-- this can clear your whole main window - needs 2.0-test3+
clearWindow()
```

copy

copy()

Copies the current selection to the clipboard. This function operates on rich text, i. e. the selected text including all its format codes like colors, fonts etc. in the clipboard until it gets overwritten by another copy operation. example: This script copies the current line on the main screen to a user window (mini console) named chat and gags the output on the main screen.
See Also: selectString(), selectCurrentLine()

Parameters

None

Example

```
selectString( line )
copy()
appendBuffer("chat")
replace("This line has been moved to the chat window!")
```

createBuffer

createBuffer(name)

Creates a named buffer for formatted text, much like a miniconsole, but the buffer is not intended to be shown on the screen - use it for formatting text or storing formatted text.

Parameters

- *name:*

  The name of the buffer to create.

Example

```
--This creates a named buffer called "scratchpad"
createBuffer("scratchpad")
```

createConsole

createConsole(consoleName, fontSize, charsPerLine, numberOfLines, Xpos, Ypos)

Makes a new miniconsole. The background will be black, and the text color white.

Parameters

- *consoleName:*

  The name of your new miniconsole. Passed as a string.

- *fontSize:*

  The font size to use for the miniconsole. Passed as an integer number.

- *charsPerLine:*

  How many characters wide to make the miniconsole. Passed as an integer number.

- *numberOfLines:*

  How many lines high to make the miniconsole. Passed as an integer number.

- *Xpos:*

  X position of miniconsole. Measured in pixels, with 0 being the very left. Passed as an integer number.

- *Ypos:*

  Y position of miniconsole. Measured in pixels, with 0 being the very top. Passed as an integer number.

Example

```
-- this will create a console with the name of "myConsoleWindow", font size 8,
80 characters wide,
-- 20 lines high, at coordinates 300x,400y
createConsole("myConsoleWindow", 8, 80, 20, 200, 400)
```

createGauge

createGauge(name, width, Xpos, Ypos, gaugeText, r, g, b)
createGauge(name, width, Xpos, Ypos, gaugeText, colorName)

Creates a gauge that you can use to express completion with. For example, you can use this as your healthbar or xpbar.
See also: moveGauge(), setGauge(), setGaugeText()

Parameters

- *name:*

  The name of the gauge. Must be unique, you can not have two or more gauges with the same name. Passed as a string.

- *width:*

The width of the gauge, in pixels. Passed as an integer number.

- *height:*

  The height of the gauge, in pixels. Passed as an integer number.

- *Xpos:*

  X position of gauge. Measured in pixels, with 0 being the very left. Passed as an integer number.

- *Ypos:*

  Y position of gauge. Measured in pixels, with 0 being the very top. Passed as an integer number.

- *gaugeText:*

  Text to display on the gauge. Passed as a string, unless you do not wish to have any text, in which case you pass nil

- *r:*

  The red component of the gauge color. Passed as an integer number from 0 to 255

- *g:*

  The green component of the gauge color. Passed as an integer number from 0 to 255

- *b:*

  The blue component of the gauge color. Passed as an integer number from 0 to 255

- *colorName:*

  the name of color for the gauge. Passed as a string.

Example

```
-- This would make a gauge at that's 300px width, 20px in height, located at
Xpos and Ypos and is green.
-- The second example is using the same names you'd use for something like
[[fg]]() or [[bg]]().
createGauge("healthBar", 300, 20, 30, 300, nil, 0, 255, 0)
createGauge("healthBar", 300, 20, 30, 300, nil, "green")


-- If you wish to have some text on your label, you'll change the nil part and
make it look like this:
createGauge("healthBar", 300, 20, 30, 300, "Now with some text", 0, 255, 0)
-- or
createGauge("healthBar", 300, 20, 30, 300, "Now with some text", "green")
```

Note

```
--If you want to put text on the back of the gauge when it's low, use an echo
with the gaugeName_back.
echo("gaugeName_back", "This is a test of putting text on the back of the
gauge!")
```

createLabel

createLabel(name, Xpos, Ypos, width, height, fillBackground)

Creates a highly manipulable overlay which can take some css and html code for text formatting. Labels are clickable, and as such can be used as a sort of button. Labels are meant for small variable or prompt displays, messages, images, and the like. You should not use them for larger text displays or things which will be updated rapidly and in high volume, as they are much slower than miniconsoles.
Returns true or false.
See also: hideWindow(), showWindow(), resizeWindow(), setLabelClickCallback(), setTextFormat(), setTextFormat(), setMiniConsoleFontSize(), setBackgroundColor(), getMainWindowSize(), calcFontSize()

Parameters

- *name:*

  The name of the label. Must be unique, you can not have two or more labels with the same name. Passed as a string.

- *Xpos:*

  X position of the label. Measured in pixels, with 0 being the very left. Passed as an integer number.

- *Ypos:*

  Y position of the label. Measured in pixels, with 0 being the very top. Passed as an integer number.

- *width:*

  The width of the label, in pixels. Passed as an integer number.

- *height:*

  The height of the label, in pixels. Passed as an integer number.

- *fillBackground:*

  Whether or not to display the background. Passed as either 1 or 0. 1 will display the background color, 0 will not.

Example

```
--This example creates a transparent overlay message box to show a big warning
message "You are under attack!" in the middle
--of the screen. Because the background color has a transparency level of 150
(0-255, with 0 being completely transparent
--and 255 non-transparent) the background text can still be read through. The
message box will disappear after 2.3 seconds.
local width, height = getMainWindowSize();
createLabel("messageBox",(width/2)-300,(height/2)-100,250,150,1);
resizeWindow("messageBox",500,70);
moveWindow("messageBox", (width/2)-300,(height/2)-100 );
setBackgroundColor("messageBox", 150,100,100,200);
echo("messageBox", [[<p style="font-size:35px"><b><center><font color="red">You
are under attack!</font></center></b></p>]] );
```

```
showWindow("messageBox");
tempTimer(2.3, [[hideWindow("messageBox")]] ) -- close the warning message box
after 2.3 seconds
```

createMiniConsole

createMiniConsole(name, posX, posY, width, height)

Opens a miniconsole window inside the main window of Mudlet. This is the ideal fast colored text display for everything that requires a bit more text, such as status screens, chat windows, etc.
Returns true or false.
See also: createLabel(), hideWindow(), showWindow(), resizeWindow(), setTextFormat(), moveWindow(), setMiniConsoleFontSize(), handleWindowResizeEvent(), setBorderTop(), setWindowWrap(), getMainWindowSize(), calcFontSize()

Parameters

- *name:*

  The name of the miniconsole. Must be unique, you can not have two or more miniconsoles with the same name. Passed as a string.

- *Xpos:*

  X position of the miniconsole. Measured in pixels, with 0 being the very left. Passed as an integer number.

- *Ypos:*

  Y position of the miniconsole. Measured in pixels, with 0 being the very top. Passed as an integer number.

- *width:*

  The width of the miniconsole, in pixels. Passed as an integer number.

- *height:*

  The height of the miniconsole, in pixels. Passed as an integer number.

Example

```
--This script would create a mini text console called "sys" and write with
yellow foreground color and blue background color
--"Hello World".


-- set up the small system message window in the top right corner
-- determine the size of your screen
WindowWidth = 0;
WindowHeight = 0;
WindowWidth, WindowHeight = getMainWindowSize();

createMiniConsole("sys",WindowWidth-650,0,650,300)
setBackgroundColor("sys",85,55,0,255)
setMiniConsoleFontSize("sys", 8)
-- wrap lines in window "sys" at 40 characters per line
```

```
setWindowWrap("sys", 40)
-- set default font colors and font style for window "sys"
setTextFormat("sys",0,35,255,50,50,50,0,0,0)

echo("sys","Hello world!")
```

decho

decho ([name of console,] text)

Color changes can be made using the format <FR,FG,FB:BR,BG,BB> where each field is a number from 0 to 255. The background portion can be omitted using <FR,FG,FB> or the foreground portion can be omitted using <:BR,BG,BB>. Arguments 2 and 3 set the default fore and background colors for the string using the same format as is used within the string, sans angle brackets, e.g. *decho("<50,50,0:0,255,0>test")*.

Parameters

- *text:*

  The text that you'd like to echo with embedded color tags. Tags take the RGB values only, see below for an explanation.

- *name of console*

  Optional name of the console to echo to. Defaults to main.

Example

```
decho("<50,50,0:0,255,0>test")

decho("miniconsolename", "<50,50,0:0,255,0>test")
```

deleteLine

deleteLine()

Deletes the current line under the user cursor. This is a high speed gagging tool and is very good at this task, but is only meant to be use when a line should be omitted entirely in the output. If you echo() to that line it will not be shown, and lines deleted with deleteLine() are simply no longer rendered. For replacing text, replace() is the proper option - doing *selectCurrentLine(); replace(""); cecho("new line!\n")* is better.
See Also: replace(), wrapLine()

 **Note:** you do **not** need to put anything between () - it just deletes the line it is used on.

Example

```
-- deletes the line - just put this command into the big script box. Keep the
case the same -
-- it has to be deleteLine(), not Deleteline(), deleteline() or anything else
deleteLine()

--This example creates a temporary line trigger to test if the next line is a
prompt, and if so gags it entirely.
--This can be useful for keeping a pile of prompts from forming if you're
gagging chat channels in the main window
```

```
--Note: isPrompt() only works on servers which send a GA signal with their
prompt.
tempLineTrigger(1, 1, [[if isPrompt() then deleteLine() end]])

-- example of deleting multiple lines:
deleteLine()                            -- delete the current line
moveCursor(0,getLineNumber()-1)  -- move the cursor back one line
deleteLine()                            -- delete the previous line now
```

deselect

deselect([optional window name])

This is used to clear the current selection (to no longer have anything selected). Should be used after changing the formatting of text, to keep from accidentally changing the text again later with another formatting call.

Parameters

- *name:*

  The name of the buffer/miniConsole to stop having anything selected in. This is an optional argument, if name is not provided the main window will have its selection cleared.

Example

```
--This will change the background on an entire line in the main window to red,
and then properly clear the selection to keep further
--changes from effecting this line as well.
selectCurrentLine()
bg("red")
deselect()
```

echoLink

echoLink([windowName], text, command, hint, [bool use_current_format or defaultLinkFormat])

Echos a piece of text as a clickable link, at the end of the current selected line - similar to echo().

Parameters

- *text:*

  text to display in the echo. Same as a normal echo().

- *command:*

  lua code to do when the link is clicked.

- *hint:*

  text for the tooltip to be displayed when the mouse is over the link.

- *window:*

  if true, then the link will use the current selection style (colors, underline, etc). If missing or false, it will use the default link style - blue on black underlined text.

Example

```
-- echo a link named 'press me!' that'll send the 'hi' command to the game
echoLink("press me!", [[send("hi")]], "This is a tooltip")

-- do the same, but send this link to a miniConsole
echoLink("my miniConsole", "press me!", [[send("hi")]], "This is a tooltip")
```

echoUserWindow

echoUserWindow(windowName)

This function will print text to both mini console windows, dock windows and labels. It is outdated however - echo() instead.

echoPopup

echoPopup([window], text, {commands}, {hints}, [current or default format])

Creates text with a left-clickable link, and a right-click menu for more options at the end of the current line, like echo. The added text, upon being left-clicked, will do the first command in the list. Upon being right-clicked, it'll display a menu with all possible commands. The menu will be populated with hints, one for each line.

Parameters

- *window:*

  Optional - the window to echo to - use either *main* or omit for the main window, or the miniconsoles name otherwise.

- *name:*

  the name of the console to operate on. If not using this in a miniConsole, use "main" as the name.

- *{lua code}:*

  a table of lua code strings to do. ie,

  ```
  {[[send("hello")]], [[echo("hi!"]]}
  ```

- *{hints}:*

  a table of strings which will be shown on the popup and right-click menu. ie,

  ```
  {"send the hi command", "echo hi to yourself"}
  ```

- *current or default format:*

  a boolean value for using either the current formatting options (colour, underline, italic) or the link default (blue underline).

Example

```
-- Create some text as a clickable with a popup menu:
echoPopup("activities to do", {[[send "sleep"]], [[send "sit"]], [[send
```

```
"stand"]]}, {"sleep", "sit", "stand"})
```

fg

fg(colorName)

If used on a selection, sets the foreground color to *colorName* - otherwise, it will set the color of the next text-inserting calls (*echo(), insertText, echoLink()*, and others)
See Also: bg(), setBgColor()

Parameters

- *colorName:*

  The name of the color to set the foreground to - list of possible names: 

Example

```
--This would change the color of the text on the current line to green
selectCurrentLine()
fg("green")
resetFormat()

--This will echo red, green, blue in their respective colors
fg("red")
echo("red ")
fg("green")
echo("green ")
fg("blue")
echo("blue ")
resetFormat()
```

getBgColor

getBgColor(windowName)

This function returns the rgb values of the background color of the first character of the current selection on mini console (window) windowName. If windowName is omitted Mudlet will use the main screen.

Parameters

- *windowName:*

  A window to operate on - either a miniconsole or the main window.

Example

```
local r,g,b;
selectString("troll",1)
r,g,b = getBgColor()
if r == 255 and g == 0 and b == 0 then
    echo("HELP! troll is highlighted in red letters, the monster is
aggressive!\n");
end
```

getColorWildcard

getColorWildcard(ansi color number)

This function, given an ANSI color number (list), will return all strings on the current line that match it.

Parameters

- *ansi color number:*

  A color number (list) to match.

Example

```
-- we can run this script on a line that has the players name coloured
differently to easily capture it from
-- anywhere on the line
local match = getColorWildcard(14)

if match then
  echo("\nFound "..match.."!")
else
  echo("\nDidn't find anyone.")
end
```

getColumnNumber

getColumnNumber()

Gets the absolute column number of the current user cursor.

Parameters

None

Example

Need example

getCurrentLine

getCurrentLine()

Returns the content of the current line under the user cursor in the buffer. The Lua variable **line** holds the content of *getCurrentLine()* before any triggers have been run on this line. When triggers change the content of the buffer, the variable line will not be adjusted and thus hold an outdated string. *line = getCurrentLine()* will update line to the real content of the current buffer. This is important if you want to copy the current line after it has been changed by some triggers. *selectString(line,1)* will return false and won't select anything because line no longer equals *getCurrentLine()*. Consequently, *selectString(getCurrentLine(), 1)* is what you need.

Parameters

None

Example

Need example

getFgColor

getFgColor(windowName)

This function returns the rgb values of the color of the first character of the current selection on mini console (window) windowName. If windowName is omitted Mudlet will use the main screen.

Parameters

- *windowName:*

  A window to operate on - either a miniconsole or the main window.

Example

```
local r,g,b;
selectString("troll",1)
r,g,b = getFgColor()
if r == 255 and g == 0 and b == 0 then
    echo("HELP! troll is written in red letters, the monster is aggressive!\n");
end
```

getLineCount

getLineCount()
Gets the absolute amount of lines in the current console buffer

Parameters

None

Example

Need example

getLines

getLines(from_line_number, to_line_number)

Returns a section of the content of the screen text buffer. Returns a Lua table with the content of the lines on a per line basis. The form value is *result = {relative_linenumber = line}*.

Absolute line numbers are used.

Parameters

- *from_line_number:*

  First line number

- *to_line_number:*

  End line number

Example

```
-- retrieve & echo the last line:
echo(getLines(getLineNumber()-1, getLineNumber())[1])

-- find out which server and port you are connected to (as per Mudlet settings
dialog):
local t = getLines(0, getLineNumber())

local server, port

for i = 1, #t do
  local s, p = t[i]:match("looking up the IP address of server:(.-):(%d+)")
  if s then server, port = s, p break end
end

display(server)
display(port)
```

getLineNumber

getLineNumber()

Returns the absolute line number of the current user cursor. The cursor by default is on the current line the triggers are processing - which you can move around with moveCursor() and moveCursorEnd(). This function can come in handy in combination when using with moveCursor() and getLines().

Example

```
-- use getLines() in conjuction with getLineNumber() to check if the previous
line has a certain word
if getLines(getLineNumber()-1, getLineNumber())[1]:find("attacks") then
echo("previous line had the word 'attacks' in it!\n") end
```

getMainConsoleWidth

getMainConsoleWidth()

Returns a single number; the width of the main console (MuD output) in pixels.

Parameters

None

Example

```
-- Save width of the main console to a variable for future use.
consoleWidth = getMainConsoleWidth()
```

hasFocus

hasFocus()

Returns *true* or *false* depending if Mudlet's main window is currently in focus (ie, the user isn't focused on another window, like a browser). This can be useful for determining whenever your script should call for attention or not, for example.

Parameters
None

Example

```
if attacked and not hasFocus() then
  runaway()
else
  fight()
end
```

getMainWindowSize

getMainWindowSize()
Returns two numbers, the width and height in pixels.

Parameters
None

Example

```
--this will get the size of your main mudlet window and save them
--into the variables mainHeight and mainWidth
mainWidth, mainHeight = getMainWindowSize()
```

getStopWatchTime

getStopWatchTimer(watchID)
Returns the time (milliseconds based) in form of 0.058 (= clock ran for 58 milliseconds before it was stopped). Please note that after the stopwatch is stopped, retrieving the time will not work - it's only valid while it is running.
See also: createStopWatch()

Returns a number

Parameters

- *watchID*

The ID number of the watch.

Example

```
-- an example of showing the time left on the stopwatch
teststopwatch = teststopwatch or createStopWatch()
startStopWatch(teststopwatch)
echo("Time on stopwatch: "..getStopWatchTime(teststopwatch))
tempTimer(1, [[echo("Time on stopwatch: "..getStopWatchTime(teststopwatch))]])
tempTimer(2, [[echo("Time on stopwatch: "..getStopWatchTime(teststopwatch))]])
stopStopWatch(teststopwatch)
```

handleWindowResizeEvent

handleWindowResizeEvent()

(*depreciated*) This function is depreciated and should not be used; it's only documented here for historical reference - use the sysWindowResizeEvent() event instead.

The standard implementation of this function does nothing. However, this function gets called whenever the main window is being manually resized. You can overwrite this function in your own scripts to handle window resize events yourself and e. g. adjust the screen position and size of your mini console windows, labels or other relevant GUI elements in your scripts that depend on the size of the main Window. To override this function you can simply put a function with the same name in one of your scripts thus overwriting the original empty implementation of this

Parameters

None

Example

```
function handleWindowResizeEvent()
   -- determine the size of your screen
   WindowWidth=0;
   WindowHeight=0;
   WindowWidth, WindowHeight = getMainWindowSize();

   -- move mini console "sys" to the far right side of the screen whenever the
screen gets resized
   moveWindow("sys",WindowWidth-300,0)
end
```

hasFocus

hasFocus()

Returns true or false depending on if the main Mudlet window is in focus. By focus, it means that the window is selected and you can type in the input line and etc. Not in focus means that the window isn't selected, some other window is currently in focus.

Parameters

None

Example

Need example

hecho

hecho(window, text)

Echoes text that can be easily formatted with colour tags in the hexadecimal format.
See Also: decho(), cecho()

Parameters

- *window:*

  Optional - the window name to echo to - can either be none or "main" for the main window, or the miniconsoles name.

- *text:*

The text to display, with color changes made within the string using the format |cFRFGFB,BRBGBB where FR is the foreground red value, FG is the foreground green value, FB is the foreground blue value, BR is the background red value, etc., BRBGBB is optional. |r can be used within the string to reset the colors to default.

Example

```
hecho("|ca00040black!")
```

hideToolBar

hideToolBar(name)

Hides the toolbar with the given name name and makes it disappear. If all toolbars of a tool bar area (top, left, right) are hidden, the entire tool bar area disappears automatically.

Parameters

- *name:*

  name of the button group to display

Example

```
hideToolBar("my offensive buttons")
```

hideWindow

hideWindow(name)

This function hides a mini console label. To show it again, use showWindow().

See also: createMiniConsole(), createLabel()

Parameters

None

Example

Need example

insertLink

insertLink([windowName], text, command, hint, [bool use_current_format or defaultLinkFormat])

Inserts a piece of text as a clickable link at the current cursor position - similar to insertText().

Parameters

- *text:*

  text to display in the echo. Same as a normal echo().

- *command:*

  lua code to do when the link is clicked.

- *hint:*

  text for the tooltip to be displayed when the mouse is over the link.

- *window:*

  if true, then the link will use the current selection style (colors, underline, etc). If missing or false, it will use the default link style - blue on black underlined text.

Example

Need example

insertPopup

insertPopup([windowName], text, {commands}, {hints}, [current or default format])

Creates text with a left-clickable link, and a right-click menu for more options exactly where the cursor position is, similar to insertText(). The inserted text, upon being left-clicked, will do the first command in the list. Upon being right-clicked, it'll display a menu with all possible commands. The menu will be populated with hints, one for each line.

Parameters

- *window:*

  Optional - the window to echo to - use either *main* or omit for the main window, or the miniconsoles name otherwise.

- *name:*

  the name of the console to operate on. If not using this in a miniConsole, use "main" as the name.

- *{lua code}:*

  a table of lua code strings to do. ie,

  ```
  {[[send("hello")]], [[echo("hi!"]]}
  ```

  .

- *{hints}:*

  a table of strings which will be shown on the popup and right-click menu. ie,

  ```
  {"send the hi command", "echo hi to yourself"}
  ```

  .

- *current or default format:*

  a boolean value for using either the current formatting options (colour, underline, italic) or the link default (blue underline).

Example

```
-- Create some text as a clickable with a popup menu:
insertPopup("activities to do", {[[send "sleep"]], [[send "sit"]], [[send
"stand"]]}, {"sleep", "sit", "stand"})
```

insertText

insertText([optional windowName], text)

Inserts text at cursor postion in window - unlike echo(), which inserts the text at the end of the last line in the buffer (typically the one being processed by the triggers). You can use moveCursor() to move the cursor into position first.

insertHTML() also does the same thing as insertText, if you ever come across it.

See also: cinsertText()

Parameters

- *window:*

  The window to insert the text to.

- *text:*

  The text you will insert into the current cursor position.

Example

```
-- move the cursor to the end of the previous line and insert some text

-- move to the previous line
moveCursor(0, getLineNumber()-1)
-- move the end the of the previous line
moveCursor(#getCurrentLine(), getLineNumber())

fg("dark_slate_gray")
insertText(' <- that looks nice.')

deselect()
resetFormat()
moveCursorEnd()
```

isAnsiBgColor

isAnsiBgColor(ansiBgColorCode)

This function tests if the first character of **the current selection** has the background color specified by ansiBgColorCode.

Parameters

- *ansiBgColorCode:*

  A color code to test for, possible codes are:

```
0 = default text color
1 = light black
2 = dark black
```

```
3 = light red
4 = dark red
5 = light green
6 = dark green
7 = light yellow
8 = dark yellow
9 = light blue
10 = dark blue
11 = light magenta
12 = dark magenta
13 = light cyan
14 = dark cyan
15 = light white
16 = dark white
```

Example

```
selectString( matches[1], 1 )
if isAnsiBgColor( 5 ) then
    bg("red");
    resetFormat();
    echo("yes, the background of the text is light green")
else
    echo( "no sorry, some other backgroundground color" )
end
```

 **Note:** matches[1] holds the matched trigger pattern - even in substring, exact match, begin of line substring trigger patterns or even color triggers that do not know about the concept of capture groups. Consequently, you can always test if the text that has fired the trigger has a certain color and react accordingly. This function is faster than using getFgColor() and then handling the color comparison in Lua.

isAnsiFgColor

isAnsiFgColor(ansiFgColorCode)

This function tests if the first character of **the current selection** has the foreground color specified by ansiFgColorCode.

Parameters

- *ansiFgColorCode:*

  A color code to test for, possible codes are:

```
0 = default text color
1 = light black
2 = dark black
3 = light red
4 = dark red
5 = light green
6 = dark green
7 = light yellow
8 = dark yellow
9 = light blue
10 = dark blue
11 = light magenta
12 = dark magenta
13 = light cyan
14 = dark cyan
```

```
15 = light white
16 = dark white
```

Example

```
selectString( matches[1], 1 )
if isAnsiFgColor( 5 ) then
    bg("red");
    resetFormat();
    echo("yes, the text is light green")
else
    echo( "no sorry, some other foreground color" )
end
```

 **Note:** matches[1] holds the matched trigger pattern - even in substring, exact match, begin of line substring trigger patterns or even color triggers that do not know about the concept of capture groups. Consequently, you can always test if the text that has fired the trigger has a certain color and react accordingly. This function is faster than using getFgColor() and then handling the color comparison in Lua.

moveCursor

moveCursor([optional windowName], x, y)

Moves the user cursor of the window windowName, or the main window, to the absolute point (x,y). This function returns false if such a move is impossible e.g. the coordinates don't exist. To determine the correct coordinates use getLineNumber(), getColumnNumber() and getLastLineNumber(). The trigger engine will always place the user cursor at the beginning of the current line before the script is run. If you omit the windowName argument, the main screen will be used.

Returns *true* or *false* depending on if the cursor was moved to a valid position. Check this before doing further cursor operations - because things like deleteLine() might invalidate this.

Parameters

- *windowName:*

  The window you are going to move the cursor in.

- *x:*

  The horizontal axis in the window - that is, the line number.

- *y:*

  The vertical axis in the window - that is, the letter position within the line.

Example

```
-- move cursor to the start of the previous line and insert -<(
-- the first 0 means we want the cursor right at the start of the line,
-- and getLineNumber()-1 means we want the cursor on the current line# - 1 which
-- equals to the previous line
moveCursor(0, getLineNumber()-1)
insertText("-<(")
```

```
-- now we move the cursor at the end of the previous line. Because the
-- cursor is on the previous line already, we can use #getCurrentLine()
-- to see how long it is. We also just do getLineNumber() because
getLineNumber()
-- returns the current line # the cursor is on
moveCursor(#getCurrentLine(), getLineNumber())
insertText(")>-")

-- finally, reset it to the end where it was after our shenaningans - other
scripts
-- could expect the cursor to be at the end
moveCursorEnd()

-- a more complicated example showing how to work with Mudlet functions

-- set up the small system message window in the top right corner
-- determine the size of your screen
local WindowWidth, WindowHeight = getMainWindowSize()

-- define a mini console named "sys" and set its background color
createMiniConsole("sys",WindowWidth-650,0,650,300)
setBackgroundColor("sys",85,55,0,255)

-- you *must* set the font size, otherwise mini windows will not work properly
setMiniConsoleFontSize("sys", 12)
-- wrap lines in window "sys" at 65 characters per line
setWindowWrap("sys", 60)
-- set default font colors and font style for window "sys"
setTextFormat("sys",0,35,255,50,50,50,0,0,0)
-- clear the window
clearUserWindow("sys")

moveCursorEnd("sys")
setFgColor("sys", 10,10,0)
setBgColor("sys", 0,0,255)
echo("sys", "test1---line1\n<this line is to be deleted>\n<this line is to be
deleted also>\n")
echo("sys", "test1---line2\n")
echo("sys", "test1---line3\n")
setTextFormat("sys",158,0,255,255,0,255,0,0,0);
--setFgColor("sys",255,0,0);
echo("sys", "test1---line4\n")
echo("sys", "test1---line5\n")
moveCursor("sys", 1,1)

-- deleting lines 2+3
deleteLine("sys")
deleteLine("sys")

-- inserting a line at pos 5,2
moveCursor("sys", 5,2)
setFgColor("sys", 100,100,0)
setBgColor("sys", 255,100,0)
insertText("sys","############## line inserted at pos 5/2 ##############")

-- inserting a line at pos 0,0
moveCursor("sys", 0,0)
selectCurrentLine("sys")
setFgColor("sys", 255,155,255)
setBold( "sys", true );
setUnderline( "sys", true )
setItalics( "sys", true )
insertText("sys", "------- line inserted at: 0/0 -----\n")
```

```
setBold( "sys", true )
setUnderline( "sys", false )
setItalics( "sys", false )
setFgColor("sys", 255,100,0)
setBgColor("sys", 155,155,0)
echo("sys", "*** This is the end. ***\n")
```

moveCursorEnd

moveCursorEnd(windowName)

Moves the cursor to the end of the buffer. "main" is the name of the main window, otherwise use the name of your user window.
See Also: moveCursor()

Returns true or false

Parameters

- *windowName:*

  The name of your user window.

Example

Need example

moveGauge

moveGauge(gaugeName, newX, newY)

Moves a gauge created with createGauge to the new x,y coordinates. Remember the coordinates are relative to the top-left corner of the output window.

Parameters

- *gaugeName:*

  The name of your gauge

- *newX:*

  The horizontal pixel location

- *newY:*

  The vertical pixel location

Example

```
-- This would move the health bar gauge to the location 1200, 400
moveGauge("healthBar", 1200, 400)
```

moveWindow

moveWindow(name, x, y)

This function moves window name to the given x/y coordinate. The main screen cannot be moved. Instead you'll have to set appropriate border values → preferences to move the main screen e.g. to make room for chat or information mini consoles, or other GUI elements. In the future moveWindow() will set the border values automatically if the name parameter is omitted.

See Also: createMiniConsole(), createLabel(), handleWindowResizeEvent(), resizeWindow(), setBorderTop()

Parameters

- *name:*

  The name of your winow

- *newX:*

  The horizontal pixel location

- *newY:*

  The vertical pixel location

openUserWindow

openUserWindow(name)

Opens a user dockable console window for user output e.g. statistics, chat etc. If a window of such a name already exists, nothing happens. You can move these windows, dock them, make them into notebook tabs or float them. Note that they do currently have a bug in a sense that they will inherit your main windows borders. The windows position cannot be adjusting via scripting yet at the moment, and the layout won't be remembered next time Mudlet is open.

Parameters

- *name:* The name of your window

Examples

```
openUserWindow("My floatig window")
cecho("My floatig window", "<red>hello <blue>bob!")
```

paste

paste(windowName)

Pastes the previously copied text including all format codes like color, font etc. at the current user cursor position. The copy() and paste() functions can be used to copy formated text from the main window to a user window without losing colors e. g. for chat windows, map windows etc.

Parameters

- *windowName:*

  The name of your window

pasteUserWindow

pasteUserWindow(windowName)

Need description here

Parameters

- *windowName:*

  The name of your window

prefix

prefix(text)

Prefixes text at the beginning of the current line when used in a trigger.

Parameters

- *text:*

  The information you want to prefix

Example

```
-- Prefix the hours, minutes and seconds onto our prompt even though Mudlet has
a button for that
prefix(os.date("%H:%M:%S "))
```

replace

replace([windowName,] newtext)

Replaces the currently selected text with the new text. To select text, use selectString(), selectSection() or a similar function.

Note: If you'd like to delete/gag the whole line, use deleteLine().

Note: This won't preserve existing colours by default - however it's easy to make it, see example below.

Parameters

- *windowName:* optional name of window (a miniconsole)
- *with:* the new text to display.

Example

```
-- replace word "troll" with "cute trolly"
selectString("troll",1)
replace("cute trolly")

-- replace the whole line
selectCurrentLine()
replace("Out with the old, in with the new!")

-- if you'd like to keep the original colouring instead of applying your own,
you can do this:
if selectString("There", 1) ~= -1 then
```

```
  setBgColor(getBgColor())
  setFgColor(getFgColor())
  replace("Here")
end
```

replaceAll

replaceAll(what, with)

Replaces all occurrences of *what* in the current line with *with*.

Parameters

- *what:* the text to replace
- *with:* the new text to have in place

Examples

```
-- replace all occurrences of the word "south" in the line with "north"
replaceAll("south", "north")

-- replace all occurrences of the text that the variable "target" has
replaceAll(target, "The Bad Guy")
```

resizeWindow

resizeWindow(name,width,height)

Resizes a mini console or label

See also: createMiniConsole(), createLabel(), handleWindowResizeEvent(), resizeWindow(), setBorderTop()

selectCaptureGroup

selectCaptureGroup(groupNumber)

Selects the content of the capture group number in your Perl regular expression

Example

Perl Requlr expression e.g. "you have (\d+) Euro".

```
--If you want to color the amount of money you have green you do:

selectCaptureGroup(1);
setFgColor(0,255,0)
```

selectSection

selectSection(from, how long)

Selects the specified parts of the line starting *from* the left and extending to the right for however *how long*. The line starts from 0.

Returns true if the selection was successful, and false if the line wasn't actually long enough or the selection couldn't be done in general.

Example

```
-- select and colour the first character in the line red
if selectSection(0,1) then fg("red") end

-- select and colour the second character green (start selecting from the first
character, and select 1 character)
if selectSection(1,1) then fg("green") end

-- select and colour three character after the first two grey (start selecting
from the 2nd character for 3 characters long)
if selectSection(2,3) then fg("grey") end
```

selectString

selectString(text, number_of_match)

Selects a substring from the line where the user cursor is currently positioned - allowing you to edit selected text (apply colour, make it be a link, copy to other windows or other things).

You can move the user cursor with moveCursor(). When a new line arrives from the MUD, the user cursor is positioned at the beginning of the line. However, if one of your trigger scripts moves the cursor around you need to take care of the cursor position yourself and make sure that the cursor is in the correct line if you want to call one of the select functions. To deselect text, see deselect().

Returns position in line or -1 on error (text not found in line)

 **Note:** To prevent selection of random data use the error return if not found like this:

```
if selectString( "big monster", 1 ) > -1 then setFgColor( 255,0,0) end
```

setBgColor

setBgColor([windowName], r,g,b)

Sets the current text background color in the main window unless windowName parameter given. If you have selected text prior to this call, the selection will be highlighted otherwise the current text background color will be changed. If you set a foreground or background color, the color will be used until you call resetFormat() on all further print commands.

See also: cecho()

Parameters

- *windowName:*

  Optional parameter set the current text background color in windowname given.

- *r:*

  The red component of the gauge color. Passed as an integer number from 0 to 255

- *g:*

  The green component of the gauge color. Passed as an integer number from 0 to 255

- *b:*

  The blue component of the gauge color. Passed as an integer number from 0 to 255

Example

```
--highlights the first occurrence of the string "Tom" in the current line with a
red background color.
selectString( "Tom", 1 )
setBgColor( 255,0,0 )

--prints "Hello" on red background and "You" on blue.
setBgColor(255,0,0)
echo("Hello")
setBgColor(0,0,255)
echo(" You!")
resetFormat()
```

setBold

setBold(windowName, bool)

Sets the current text font to bold (true) or non-bold (false) mode. If the windowName parameters omitted, the main screen will be used.

setFgColor

setFgColor([windowName],r, g, b)

Sets the current text foreground color in the main window unless windowName parameter given.

- *windowName:*

  Optional parameter set the current text background color in windowname given.

- *r:*

  The red component of the gauge color. Passed as an integer number from 0 to 255

- *g:*

  The green component of the gauge color. Passed as an integer number from 0 to 255

- *b:*

  The blue component of the gauge color. Passed as an integer number from 0 to 255

Example

```
--highlights the first occurrence of the string "Tom" in the current line with a
red foreground color.
selectString( "Tom", 1 )
setFgColor( 255,0,0 )
```

setGauge

setGauge(gaugeName, currentValue, maxValue, gaugeText)

Use this function when you want to change the gauges look according to your values. Typical usage would be in a prompt with your current health or whatever value, and throw in some variables instead of the numbers.

Example

```
--Change the looks of the gauge named healthBar and make it
--fill to half of its capacity. The height is always remembered.
setGauge("healthBar", 200, 400)

--If you wish to change the text on your gauge, you'd do the following:
setGauge("healthBar", 200, 400, "some text")
```

setItalics

setItalics(windowName, bool)

Sets the current text font to italics/non-italics mode. If the windowName parameters omitted, the main screen will be used.

setMiniConsoleFontSize

setMiniConsoleFontSize(name, fontSize)

Sets the font size of the mini console. see also: createMiniConsole(), createLabel()

setTextFormat

setTextFormat(windowName, r1, g1, b1, r2, g2, b2, bold, underline, italics)

Sets current text format of window windowName: foreground color(r1,g1,b1), background color(r2,g2,b2), bold(1/0), underline(1/0), italics(1/0) A more convenient way to control the text format in a mini console is to use setFgColor(windowName, r,g,b), setBold(windowName, true), setItalics(windowName, true), setUnderline(windowName, true) etc. → createMiniConsole, setBold, setBgColor, setFgColor, setItalics, setUnderline

Example

```
--This script would create a mini text console and write with yellow foreground
color and blue background color "This is a test".
createMiniConsole( "con1", 0,0,300,100);
setTextFormat("con1",0,0,255,255,255,0,1,1,1);
echo("con1","This is a test")
```

setUnderline

setUnderline(windowName, bool)

Sets the current text font to underline/non-underline mode. If the windowName parameters omitted, the main screen will be used.

setWindowWrap

setWindowWrap(windowName, wrapAt)

sets at what position in the line the console or miniconsole will start word wrap

showCaptureGroups

showCaptureGroups()

Lua debug function that highlights in random coolors all capture groups in your trigger regex on the screen. This is very handy if you make complex regex and want to see what really matches in the text. This function is defined in LuaGlobal.lua.

Example

Make a trigger with the regex (\w+) and call this function in a trigger. All words in the text will be highlighted in random colors.

showMultimatches

showMultimatches()

Lua helper function to show you what the table multimatches[n][m] contains. This is very useful when debugging multiline triggers - just doing showMultimatches() in your script will make it give the info.

showWindow

showWindow(name)

Shows user window name.

replaceWildcard

replaceWildcard(which, replacement)

Replaces the given wildcard (as a number) with the given text. Equivalent to doing:

```
selectString(matches[2], 1)
replace("text")
```

Parameters

- *which:*

  Wildcard to replace.

- *replacement:*

  Text to replace the wildcard with.

Example

```
replaceWildcard(2, "hello") -- on a perl regex trigger of ^You wave (goodbye)\.
$, it will make it seem like you waved hello
```

resetFormat

resetFormat()

Resets the colour/bold/italics formatting. Always use this function when done adjusting formatting, so make sure what you've set doesn't 'bleed' onto other triggers/aliases.

Parameters

None

Example

```
-- select and set the 'Tommy' to red in the line
if selectString("Tommy", 1) ~= -1 then fg("red") end

-- now reset the formatting, so our echo isn't red
resetFormat()
echo(" Hi Tommy!")

-- another example: just highlighting some words
for _, word in ipairs{"he", "she", "her", "their"} do
  if selectString(word, 1) ~= -1 then
    bg("blue")
  end
end
resetFormat()
```

selectCurrentLine

selectCurrentLine()

Selects the content of the current line that the cursor at. By default, the cursor is at the start of the current line that the triggers are processing, but you can move it with the moveCursor() function. **Note:** This selects the whole line, including the linebreak - so it has a subtle difference from the slightly slower *selectString(line, 1)* selection method.
See Also: selectString(), getCurrentLine()

Parameters

None

Example

```
-- color the whole line green!
selectCurrentLine()
fg("green")
deselect()
resetFormat()
```

setBackgroundColor

setBackgroundColor(labelName, red, green, blue, transparency)

Sets rgb color values and the transparency for the given window. Colors are from 0 to 255 (0 being black), and transparency is from - to 255 (0 being completely transparent).

Note: Transparency only works on labels, not miniConsoles for efficiency reasons.

Parameters

- *labelName:*

  The name of the label to change it's background color.

- *red:*

  Amount of red to use, from 255 (full) to 0 (none).

- *green:*

  Amount of green to use, from 255 (full) to 0 (none).

- *blue:*

  Amount of red to use, from 255 (full) to 0 (none).

- *transparency:*

  Amount of transparency to use, from 255 (fully opaque) to 0 (fully transparent).

Example

```
-- make a red label that's somewhat transparent
setBackgroundColor("some label",255,0,0,200)
```

setBackgroundImage

setBackgroundImage(labelName, imageLocation)

Loads an image file (png) as a background image for a label. This can be used to display clickable buttons in combination with setLabelClickCallback() and such.

Note you can only do this for labels, not miniconsoles.

Note you can also load images via setLabelStyleSheet().

Parameters

- *labelName:*

  The name of the label to change it's background color.

- *imageLocation:*

  The full path to the image location. It's best to use [[ ]] instead of "" for it - because for Windows paths, backslashes need to be escaped.

Example

```
-- give the top border a nice look
setBackgroundImage("top bar", [[/home/vadi/Games/Mudlet/games/top_bar.png]])
```

setBorderBottom

setBorderBottom(size)

Sets the size of the bottom border of the main window in pixels. A border means that the MUD text won't go on it, so this gives you room to place your graphical elements there. See Also: setBorderColor()

Parameters

- *size:*

  Height of the border in pixels - with 0 indicating no border.

Example

```
setBorderLeft(100)
```

setBorderColor

setBorderColor(r,g,b)

Sets the color of the main windows border that you can create either with setBorderTop(), setBorderBottom(), setBorderLeft(), setBorderRight(), or via the main window settings.
See Also: setBorderTop(), setBorderBottom(), setBorderLeft(), setBorderRight()

Parameters

- *r:*

  Amount of red to use, from 0 to 255.

- *g:*

  Amount of green to use, from 0 to 255.

- *b:*

  Amount of blue to use, from 0 to 255.

Example

```
-- set the border to be completely blue
setBorderColor(0, 0, 255)

-- or red, using a name
setBorderColor( unpack(color_table.red) )
```

setBorderLeft

setBorderLeft(size)

Sets the size of the left border of the main window in pixels. A border means that the MUD text won't go on it, so this gives you room to place your graphical elements there.
See Also: setBorderColor()

Parameters

- *size:*

  Width of the border in pixels - with 0 indicating no border.

Example

```
setBorderLeft(100)
```

setBorderRight

setBorderRight(size)

Sets the size of the right border of the main window in pixels. A border means that the MUD

text won't go on it, so this gives you room to place your graphical elements there.
See Also: setBorderColor()

Parameters

- *size:*

  Width of the border in pixels - with 0 indicating no border.

Example

```
setBorderRight(100)
```

setBorderTop

setBorderTop(size)

Sets the size of the top border of the main window in pixels. A border means that the MUD text won't go on it, so this gives you room to place your graphical elements there.
See Also: setBorderColor()

Parameters

- *size:*

  Height of the border in pixels - with 0 indicating no border.

Example

```
setBorderTop(100)
```

setLabelClickCallback

setLabelClickCallback(labelName, luaFunctionName, optional any amount of arguments)

Specifies a Lua function to be called if the user clicks on the label/image. This function can pass any number of string or integer number values as additional parameters. These parameters are then used in the callback - thus you can associate data with the label/button.

Parameters

- *labelName:*

  The name of the label to attach a callback function to.

- *luaFunctionName:*

  The Lua function name to call, as a string - it must be registered as a global function, and not inside any namespaces (tables).

- *optional any amount of arguments:*

  Y position of the miniconsole. Measured in pixels, with 0 being the very top. Passed as an integer number.

Example

```
function onClickGoNorth()
  echo("the north button was clicked!")
end

setLabelClickCallback( "compassNorthImage", "onClickGoNorth" )
```

setLink

setLink(command, tooltip)

Turns the selected() text into a clickable link - upon being clicked, the link will do the command code. Tooltip is a string which will be displayed when the mouse is over the selected text.

Parameters

- *command:*

  command to do when the text is clicked

- *tooltip:*

  tooltip to show when the mouse is over the text - explaining what would clicking do

Example

```
-- you can clickify a lot of things to save yourself some time - for example,
you can change
--  the line where you receive a message to be clickable to read it!
-- prel regex trigger:
-- ^You just received message #(\w+) from \w+\.$
-- script:
selectString(matches[2], 1)
setUnderline(true) setLink([[send("msg read ]]..matches[2]..[[")]], "Read
#"..matches[2])
resetFormat()
```

setLabelStyleSheet

setLabelStyleSheet(label, markup)

Applies Qt style formatting to a label via a special markup language.

Parameters

- *label:*

  The name of the label to be formatted (passed when calling createLabel).

- *markup:*

  The string instructions, as specified by the Qt Style Sheet reference.

References

http://doc.qt.nokia.com/4.7/stylesheet-reference.html#list-of-properties

Example

```
-- This creates a label with a white background and a green border, with the
text "test"
-- inside.
createLabel("test", 50, 50, 100, 100, 0)
setLabelStyleSheet("test", [[
        background-color: white;
        border: 10px solid green;
        font-size: 12px;
        ]])
echo("test", "test")

-- This creates a label with a single image, that will tile or clip depending on
the
-- size of the label. To use this example, please supply your own image.
createLabel("test5", 50, 353, 164, 55, 0)
setLabelStyleSheet("test5", [[
        background-image:
url(C:/Users/Administrator/.config/mudlet/profiles/Midkemia
Online/Vyzor/MkO_logo.png);
        ]])

-- This creates a label with a single image, that can be resized (such as during
a
-- sysWindowResizeEvent). To use this example, please supply your own image.
createLabel("test9", 215, 353, 100, 100, 0)
setLabelStyleSheet("test9", [[
        border-image:
url(C:/Users/Administrator/.config/mudlet/profiles/Midkemia
Online/Vyzor/MkO_logo.png);
        ]])
```

setPopup

setPopup(name, {lua code}, {hints})

Turns the selected() text into a left-clickable link, and a right-click menu for more options. The selected text, upon being left-clicked, will do the first command in the list. Upon being right-clicked, it'll display a menu with all possible commands. The menu will be populated with hints, one for each line.

Parameters

- *name:*

  the name of the console to operate on. If not using this in a miniConsole, use "main" as the name.

- *{lua code}:*

  a table of lua code strings to do. ie,

  ```
  {[[send("hello")]], [[echo("hi!"]]}
  ```

- *{hints}:*

  a table of strings which will be shown on the popup and right-click menu. ie,

  ```
  {"send the hi command", "echo hi to yourself"}
  ```

  .

Example

```
-- In a `Raising your hand in greeting, you say "Hello!"` exact match trigger,
-- the following code will make left-clicking on `Hello` show you an echo, while
right-clicking
-- will show some commands you can do.

selectString("Hello", 1)
setPopup("main", {[[send("bye")]], [[echo("hi!")]]}, {"left-click or right-click
and do first item to send bye", "click to echo hi"})
```

showToolBar

showToolBar(name)

Makes a toolbar (a button group) appear on the screen.

Parameters

- *name:*

  name of the button group to display

Example

```
showToolBar("my offensive buttons")
```

wrapLine

wrapLine(windowName, lineNumber)

wraps the line specified by *lineNumber* of mini console (window) *windowName*. This function will interpret \n characters, apply word wrap and display the new lines on the screen. This function may be necessary if you use deleteLine() and thus erase the entire current line in the buffer, but you want to do some further echo() calls after calling deleteLine(). You will then need to re-wrap the last line of the buffer to actually see what you have echoed and get your \n interpreted as newline characters properly. Using this function is no good programming practice and should be avoided. There are better ways of handling situations where you would call deleteLine() and echo afterwards e.g.:

```
selectString(line,1)
replace("")
```

This will effectively have the same result as a call to deleteLine() but the buffer line will not be entirely removed. Consequently, further calls to echo() etc. sort of functions are possible without using wrapLine() unnecessarily.

See Also: replace(), deleteLine()

Parameters

- *windowName:*

  The miniconsole or the main window (use *main* for the main window)

- *lineNumber:*

The line number which you'd like re-wrapped.

Example

```
-- re-wrap the last line in the main window
wrapLine("main", getLineCount())
```