Shared function

Call

> **Note:** Calls may incur a performance overhead - they are not equivalent in performance to calling functions in the same resource.

> **Important Note:** The sourceResource and sourceResourceRoot "hidden" variables are available even if you use exports.*:*

> **Important Note:** Using this function straight away on resource start might cause elements to not be passed properly, use setTimer in order to delay function execution and to avoid this issue.

This function is used to call a function from another resource (which must be running).

The function which you wish to call **must** first be exported within the resource's meta. For example:

```
<meta>
        <info author="jbeta" type="script" description="Scoreboard resource" />
        <script src="scoreboard_client.lua" type="client"/>
        <script src="scoreboard_exports.lua" type="server"/>

        <script src="scoreboard_http.lua" type="server"/>

        <export function="getScoreboardColumns" http="true" />
        <export function="getScoreboardRows" http="true" />

        <export function="addScoreboardColumn" type="server"/>
        <export function="removeScoreboardColumn" type="server"/>

        <export function="setPlayerScoreboardForced" type="server"/>
        <export function="setScoreboardForced" type="client"/>
</meta>
```

This enables other resources to call a function from this resource.

You cannot call a server function from the client or vice versa. See triggerServerEvent and triggerClientEvent for possibilities to do that.

There is an easier syntax replacing this function. For example, you can instead of:

```
call ( getResourceFromName ( "resource" ), "exportedFunction", 1, "2", "three" )
```

do much like a normal call:

```
exports.resource:exportedFunction ( 1, "2", "three" )
```

If the resource name contains illegal characters (such as hyphens), you can also do:

```
exports["resource-name"]:exportedFunction ( 1, "2", "three" )
```

Two extra "hidden" variables are passed to the exported function:

- **sourceResource** - The resource that called the exported function
- **sourceResourceRoot** - The resource root element of the resource which called the exported function.

Syntax

```
var... call ( resource theResource, string theFunction, [ arguments... ] )
```

OOP Syntax Help! I don't understand this!

Method: *resource:call(...)*

Required Arguments

- **theResource:** This is a resource pointer which refers to the resource you are calling a function from.
- **theFunction:** This is a string with the name of the function which you want to call.

Optional Arguments

NOTE: When using optional arguments, you might need to supply all arguments before the one you wish to use. For more information on optional arguments, see optional arguments.

- **arguments:** Any arguments you may want to pass to the function when it is called. Any number of arguments of can be specified, each being passed to the designated function.

Returns

Returns anything that the designated function has returned, if the function has no return, nil is returned. If the function does not exist, is not exported, or the call was not successful it will return false.

Syntax

```
exports["resource_name"]:exportedFunction([ arguments... ])

exports.resource_name:exportedFunction([ arguments... ])
```

Required Arguments

- **resource_name:** Resource name
- **exportedFunction:** The name of the function you want to call. Its **not** a string.

Optional Arguments

NOTE: When using optional arguments, you might need to supply all arguments before the one you wish to use. For more information on optional arguments, see optional arguments.

- **arguments:** Any arguments you may want to pass to the function when it is called. Any number of arguments of can be specified, each being passed to the designated function.

Returns

Returns anything that the designated function has returned, if the function has no return, nil is returned. If the function does not exist, is not exported, or the call was not successful it will return false.