Megadroid's Mission Mod

Tutorial 2 - Timers

http://www.megadroidsmod.com

Contents

Introduction

When making a mission it is common to require something to happen after a set period of time. You can use the *Timers* library; it will manage all the details for you and inform you when the specified time period has transpired by calling a function you provide.

As well as providing a time period you can also provide an identifier and arguments to be passed to the callback function. You can then take different actions based on them instead of having to store them elsewhere.

Definitions

This section will cover all of the definitions you need to know in order to use the *Timers* library.

Timer Creation Function

To create a Timer we call the function *Timers.add*. Here is the definition of the function.

```
function Timers.add( id, interval, object, callback, parameter )
```

As you can see the *Timers.add* function takes several arguments. Here's what each one does:

Name	Type	Description
id	String	A name for the timer. Can be used in the callback to identify which timer has fired
interval	Number	The time to wait before firing the timer.
object	Table	If the callback function is part of an object (a table) - it is a : function - this should be the table instance. If the callback is standalone, this should be nil.
callback	Function	The callback function that should be called when the timer triggers.
parameter	Any	An argument to pass to the callback function. This can be of any type. If not required, pass nil.

Callback Signature

The callback function should accept two arguments - the id of the timer and the parameter, as shown below.

```
function FunctionName( id, arg )
```

or

```
function TableName:FunctionName( id, arg )
```

Callback Result

A callback should return a value to indicate if the timer should be repeated or if it should be destroyed. A callback function can also return a new parameter to replace the current parameter if required.

```
CallbackResult = { Remove, Repeat }
```

Creating a Timer

This example shows using a callback to count. You can of course do more complex actions inside callbacks (anything you like in fact), but this demonstrates all the features of callbacks well.

```
--Our callback function. You can call the arguments whatever you
--want, as long as they are there.
function testCallbackFunction( id, count )
	--Increment the counter
	count = count + 1
	print("Callback function has been called "..count.." times" )
	--We want the timer to repeat, but we want to use a new argument -
	--in this case, the incremented count variable
	return CallbackResult.Repeat, count
end

function TestMission:init( )
	--Execution starts from this point
	--We pass nil for object because the function is not part
	--of a table. We pass 0 as an argument so we can start
	--counting in the callback.
	Timers.add( "test", 2.5, nil, testCallbackFunction, 0 )
end
```

Repeat or Remove?

Having done what we wanted to do inside our callback function we have one last decision to make - do we want the Timer to repeat or not? If there in an action we want to happen every 5 seconds we can create a Timer and return *CallbackResult.Repeat* from the callback function. The Timer will be reset and will trigger again in another 5 seconds.

If we only want our Timer to fire once, we can return *CallbackResult.Remove* instead. The Timer will be destroyed and will not trigger again. Using *Timers* and *CallbackResult* well can save lots of organisation work when your mission.

Sample Mission

In this sample mission we use two timers. One timer (createNewShip) is part of the TimersMission table. It uses values stored in the instance of the table (self.ships) and creates a random ship. It repeats every 3 seconds. The second timer prints "Hello" once after 13 seconds and then stops.

```lua
TimersMission = { }
initMetatable( TimersMission )
inherit( TimersMission, MMMApp )

function TimersMission.new( )
	local timersMission = { }
	setmetatable( timersMission, TimersMission.mt )
	return timersMission
end

function TimersMission:init( )
	--Used by the createShip timer.
	self.ships = { "fconst.odf", "fscout.odf", "ffreight.odf" }

	--Creating a timer that is part of a table - we pass self as the table instance.
	--Not using any arguments so we are passing nil for that.
	Timers.add( "createShip", 3, self, TimersMission.createNewShip, nil )

	--Timer that prints a message after a bit.
	Timers.add( "info", 13, nil, printMessage, "Hello" )
end

--Timer function that is a member of the self.ships field of the TestMission
--table instance.
function TimersMission:createNewShip( id, arg )
	--Choose a random ship to create
	local shipIndex = math.random( #self.ships )
	--Create that ship at the centre of the map on team 1
	Entities.add( self.ships[ shipIndex ], 1, Vector.new( 3200, 0, 3200 ) )
	--We want this to repeat
	return CallbackResult.Repeat
end

--Timer to print a message.
function printMessage( id, arg )
	print( arg )
	return CallbackResult.Remove
end

MMMApp.register( TimersMission.new() )
```

Conclusion

That's the end of this tutorial on the Timers library. Hopefully you now understand more about how Timers work and how they can help you create missions easier. If you want more help or have any questions remember to go to the forum at http://www.megadroidsmod.com/forum.