LUA: WoW Addon get namespace

```
local ADDON, NS = ...
```

“ADDON” will contain the name of the addon as per the folder name (a string) and “NS” is assigned a “local” table used to store functions, variables, objects etc. that will be passed to all (.toc rules) associated .lua files (modules) via their own ... parameters. It's a mechanism to avoid having to declare globals in order to access code and data across modules within an addon.

file1.lua

```
local ADDON, NS = ...  -- get the addon name and common table.
NS.SomeVar = 000                -- create an entry in the table.
function NS:SomeFunc(arg1)      -- create a function in the table.
    print(arg1)
end


-- ... all the other code in the file adding/using "NS" entries also used by other "modules".
```

file2.lua

```
local ADDON, NS = ...  -- get the addon name and common table.
NS:SomeFunc(NS.SomeVar) -- prints 000

-- ... all the other code in the file adding/using "NS" entries also used by other "modules".
```

“ADDON” and “NS” can be diffent names in each .lua file if that's the way you think, the magic is in the ... being two parameters, a string and a (Blizzard created “local”) table.

file3.lua

```
local MYNAME, MYTABLE = ... -- get the addon name and common table.
MYTABLE:SomeFunc(MYTABLE.SomeVar) -- still prints 000

-- ... all the other code in the file adding/using "MYTABLE" entries also used by other "modules".
```

There's nothing magical about the “local ADDON, NS = ...” line, and it's not issuing any special instructions, and Lua doesn't have any specific concept of “namespaces”. It's just a simple variable assignment.

Let's say your addon has 3 Lua files. Let's call those files File A and File B, and let's say they're loaded in that order. When WoW loads a Lua file, what basically happens is that the contents of the file become the contents of a function, and that function gets run immediately.

If your FileA says this:

```
local name = UnitName("player")
print("Hello " .. name .. "!")
```

... then WoW basically does this:

```
function FileA()
     local name = UnitName("player")
     print("Hello " .. name .. "!")
end

FileA()
```

This isn't exactly what happens, but it's an easy way to frame it so that the idea of passing values into the file makes sense:

```
local AddonTable = {}

function FileA(...)
     local name = UnitName("player")
     print("Hello " .. name .. "!")
end

FileA("MyAddon", AddonTable)

function FileB(...)
     -- some stuff here
end

FileB("MyAddon", AddonTable)
```

So now you can see it is actually just a simple matter of naming variables, and using variables that refer to strings and tables:

```
local AddonTable = {}

function FileA(...)
     local addonName, addonTable = ...

     local name = UnitName("player")
```

```
     addonTable.name = name
end

FileA("MyAddon", AddonTable)

function FileB(...)
     local addonName, addonTable = ...

     print("Hello " .. addonTable.name .. "!")
end

FileB("MyAddon", AddonTable)
```

No matter how many files are in your addon, they're all passed the same two values:

- a string naming your addon
- a reference to a table

The table doesn't contain any values by default, but it's just a table, so you can put whatever you want in it, and since all of your files are getting a reference to the same table, anything you add to the table from one file is accessible from the other files.

The primary purpose of this table is to let your addon's files share things without having to clutter up the global namespace. However, there's nothing stopping you from putting the table into the global namespace if you really want to:

```
local addonName, addonTable = ...
SomeGlobalNameForMyAddon = addonTable
```

Source: https://www.wowinterface.com/forums/showthread.php?t=49349

—
return to gimbo wiki home page