Slmod Version 7.0 Manual

Revision: 0

Author: Speed

Special Thanks…

There were many folks who helped me test Slmod, and I could have never created it without the hours they devoted to helping to test various features and functionality. I tried to make the testing fun whenever possible, but often there's no way to avoid it being just tedious and boring.

Thanks to:

3 Squadron (especially Grimes, thanks!)

The 16th ACCW, including (but not limited to!):
16th Hollywood,
16th Mojeaux,

104th (especially Polecat)

138th (especially Hijack)

Eno from "Eno's Firehouse" fame

VBA Greywo1f

The Virtual Tactical Air Group, including (but not limited to!):
47th Box
47th Ghostrider (aka, Willem Dafoe- no matter how much you deny it, I know who you really are!)

To all of you (and any I forgot to mention), thank you very much for your time!

Table of Contents

1 - Slmod Overview

Server Lua Mod, "Slmod", is a multiplayer server mod created by "Speed". Despite the uninspired name, it provides a significant set of enhancements for DCS multiplayer servers, including multiplayer server administration tools, multiplayer stats tracking, enhanced multiplayer server operation, enhanced mission scripting capabilities, and several other useful features. <u>As it is a server mod, multiplayer clients do **not** need to have it installed to use it.</u>

Slmod includes the SlmodStats system. It keeps persistent stats on multiplayer clients who join the server, tracking their kills, deaths, friendly fire incidents, flight time, etc. It can save and display player stats from their entire history on the server, or from just a single mission alone. More information can be found in the SlmodStats section of this manual.

Slmod also includes the Slmod Admin Menu. This menu allows registered server admins to remotely manage the server through chat-based menu system. Server admins can kick players, ban players, unban players, add players to the list of server admins, remove players from the list of server admins, restart the mission, load a different mission, pause/unpause the game, etc. More info about this feature can be found in the Slmod Admin Menu section of this manual.

Using the data acquired through SlmodStats, Slmod can auto-kick or auto-ban players for team hitting, team-killing, team-collision-hitting, and/or team-collision-killing. This feature is encompassed in the new SlmodAutoAdmin system, and is highly customizable. For more on this feature see the SlmodAutoAdmin section of this manual.

Slmod accomplishes enhanced mission scripting capabilities by adding several dozen new Lua functions to the “Mission Scripting” Lua environment. These new Lua functions provide capabilities to detect game world conditions that are undetectable with traditional triggers, allow increased client interaction with the game environment, and to output dynamic trigger text messages or chat messages. The function listing and how to use them can be found in the Slmod Scripting Functions Listing section of this manual.

In addition, Slmod also adds the following features:
- a help menu
- a chat logging system
-Pause or unpause the server based on whether or not clients are connected;
-A server Message of the Day (MOTD) system;
-The capability to export, in real time, alive unit information and events;
-Slmod Coordinate Conversion Utility (chat-based coordinate converter capable of converting between four different coordinate systems);

As it is named, Slmod is a *server* mod. It does not need to be installed by clients. It also will *not* function in single player. As tracks are replayed using the single player .exe, Slmod may also break track replay if any of the Slmod mission scripting functions were used to trigger in-game events. If your mission does not use any Slmod mission scripting functions (the vast majority won't, especially now that we have a very capable mission scripting environment already built in), then you don't need to worry about this.

Keep in mind that Slmod is *NOT* supported by Eagle Dynamics. In order to obtain many of its capabilities, Slmod must utilize some Lua functionalities not officially supported by Eagle Dynamics, or even necessarily intended for user modding in the first place. Because of this, and because Eagle Dynamics is constantly upgrading and improving the game world, Eagle Dynamics cannot be held responsible for a patch causing Slmod to no longer function correctly. That said, I will do my very best to keep Slmod functioning through future patches. Most of the code that Slmod is dependent on is not likely to undergo drastic changes in the near future, there are always such things as bugs, and sometimes relatively minor changes can have major ripple-down effects.

2 - User File Structure and Configuration Options

2.1- User File Structure

By default, DCS World stores your local settings, missions, and log files in the following folder: <Windows Drive>/Users/<Your User Account Name>/Saved Games/DCS.

<u>Forthwith, this document will refer to this directory as **Saved Games/DCS**.</u>

After installing Slmod and running it for the first time, the Saved Games/DCS/Slmod folder will be created. Inside this folder, several files and folders will or can be created. The following gives a brief description of them.

Saved Games/DCS/Slmod/config.lua
Slmod configuration settings. See section 2.3 for a full description.

Saved Games/DCS/Slmod/ServerAdmins.lua
List of server admins for the Admin Tools.

Saved Games/DCS/Slmod/BannedClients.lua
List of clients banned by the Admin Menu and SlmodAutoAdmin

Saved Games/DCS/Slmod/SlmodStats.lua
SlmodStats server-specific multiplayer stats.

Saved Games/DCS/Slmod/Slmod.log
Slmod log file, useful for debugging.

Saved Games/DCS/Slmod/Missions/
Default file folder to obtain missions from for the "-admin load" command.

Saved Games/DCS/Slmod/Chat Logs/
If chat logging is enabled, then chat logs go here.

Saved Games/DCS/Slmod/Mission Stats/
If SlmodStats mission-specific stats files are enabled, they are saved here.

2.2- A Note on Menu/Text Formatting

Slmod generally has four different format options for text that is output to players. They are:

`'chat'`: Output message only as a chat message. Keep in mind that chat messages have a limited lifetime on the screen, and can only display one line at a time. Slmod will break the desired message down into its individual lines and display each as a separate chat message. Multiple lines of the same chat message may be flashed if the display time is set too long. The message appears as if it was a chat message you said, so it shows up in orange. It is possible to change this to white chat text, but orange seems to be more visible usually and grabs the eye better.

`'text'`: Display the message only as "white trigger text" (such as what the "message to all" trigger action shows), for the desired duration.

`'echo'`: Display the message as "white trigger text" for the desired duration and also send (or "echo") each line of the message as a chat message, each chat message sent one second apart.

`'both'`: The combination of both `'chat'` and `'text'` display modes.

2.3- Slmod Configuration File Options

The Slmod configuration options are stored in Saved Games/Slmod/config.lua. Keep in mind that this configuration file is a Lua file, so any changes you make to it need to conform to proper Lua syntax. If there is an error in your config.lua file, Slmod will revert to default settings.

NOTE #1:
The default text editor with Windows, *Notepad*, is a WORTHLESS PIECE OF GARBAGE!!! Before you edit the Slmod confg.lua file, do yourself a favor and download *Notepad++*! *Notepad++* is free text editor that has natural recognition of the syntax of many programming languages (including Lua), and is a lot better at actually displaying text too- having things like line numbers, proper recognition of new line characters, etc. It can be downloaded at http://notepad-plus-plus.org/.

NOTE #2:
New Slmod versions will often have new configuration options. Whenever a Slmod version that has new configuration options is installed and run on your computer, the config.lua file will be reverted to the new default, and your old config.lua file will be backed up to Saved Games/DCS/Slmod/configOLD.lua.

2.3.1- Slmod Admin Menu Options

The config.lua file contains a number of options. The current list of options is detailed below.

admin_tools

```
admin_tools = <boolean>
```

If this option is set true, then the Slmod Admin Menu will be enabled. See section 3 for more information on the Slmod Admin Menu.

This option is set to true by default.

admin_tools_mission_folder

```
admin_tools_mission_folder = <string>
```

(This option only has an effect if the admin_tools option is set to true.)
The "-admin load" option on the Server Admin Tools menu allows a server admin to load a mission from a set of missions contained in the Saved Games\DCS\Slmod\Missions folder. However, you can specify a DIFFERENT folder to use. One example where this is useful is if you tell the server to obtain missions from a folder shared through a file-sharing program like *Dropbox*. Now, your server will share its

missions folder with other computers, allowing very easy and fast remote access to the server's mission list.

This option is set to nil by default.

admin_display_mode

```
admin_display_mode = <string>
```

Controls the display mode of the Admin Tools menu. This value must be either 'chat', 'text', 'echo', or 'both'. See "2.2 A Note on Menu/Text Formatting" for an explanation of these display modes.

This option is set to 'text' by default.

admin_display_time

```
admin_display_time = <number>
```

This option controls the display time of the Admin Menu. The units are in seconds.

This option is set to 30 by default.

admin_register_password

```
admin_register_password = <string>
```

By default, the only way to add a server admin to the list of server admins is to use the "-admin add <player name>" option on the Admin Menu. However, the admin_register_password option allows clients to enter a password to register themselves as server admins.

If the admin_register_password is set to a string, then when a non-admin types in chat "-reg", their NEXT chat message will be considered a password entry, and not shown to the rest of the players. If their password entry matches the password set in `admin_register_password,` then they will be added to the list of server admins.

Note: because this is a STRING, you MUST have single quotes, double quotes, or double square brackets around your password. If your password contains quotes itself, then you need to "escape" out of them with the\ character. Remember, the config.lua file is a Lua file, and as such, you must follow proper Lua syntax!

So the following is ***INCORRECT***:

```
admin_register_password = password123
```

Because the config.lua file is a Lua file, and you did not put quotes around your password, you just told Slmod to use the value of the variable named "password123" as your password. Unless you previously set this variable equal to something, then the effect of this is the same as setting your password equal to nil.

The following shows three different ***CORRECT*** ways to set your password equal to the string of characters "password123":

```
admin_register_password = "password123"

admin_register_password = 'password123'

admin_register_password = [[password123]]
```

Also note that as yet, there is no limit on the number of incorrect password entries a client can attempt. So if you are going to use a password, make your password strong. However, all currently connected server admins will see a notification when someone enters the password wrong, and if chat logging is enabled, then these incorrect entries will be recorded on the chat log, of course.

This option is set to nil by default, which disables it.

2.3.2- SlmodStats options

enable_slmod_stats

`enable_slmod_stats = <boolean>`

If this option is set to true, SlmodStats start fully enabled.

If this option is set to false, it makes SlmodStats stats keeping start *mostly* disabled. No tracking of weapons/kills will occur, but new clients will still be added into stats as they join the server. However, if admin tools are enabled, then stats can be fully enabled at any time by a server admin using the toggle stats command on the admin tools menu.

This option is set to true by default.

enable_mission_stats

`enable_mission_stats = <boolean>`

If this option is set to true, stats are also kept for each individual mission flown. The in-game stats display can be toggled to mission mode by using the "-stats mode" command.

This option is set to false by default.

write_mission_stats_files

`write_mission_stats_files = <boolean>`

If this option is set true and enable_mission_stats is set true, then stats files for each mission will be written in real time in a file directory (see below).

This option is set to true by default.

mission_stats_files_dir

`mission_stats_files_dir = <string or nil>`

This option determines which directory mission stats files are written in (if mission stats files are enabled). If this option is set nil, then they are written in Saved Games/DCS/Slmod/Mission Stats. You can set this variable equal to a string value file directory to make the mission files be written in another directory.

This option is set to nil by default.

host_ucid

```
host_ucid = <string> or nil
```

Currently, Slmod does not have access to the game host's unique DCS account ID (their UCID). This option sets the game host's UCID to the correct value for use in SlmodStats. This option only really matters if the host computer is going to be used to participate in the mission, AND if you someday plan to try to unify your server stats with another server's stats (a feature that is not yet in Slmod, but is planned).

This option is set to nil by default.

host_name

```
host_name = <string> or nil
```

This option only matters if you specify a host ucid. This option sets the name under which the host computer is recorded in SlmodStats.

This option is set to nil by default.

stats_dir

```
stats_dir = <string> or nil
```

This option sets an alternate directory to be used to obtain the SlmodStats stats file from. Useful for future plans to allow multiple servers to combine stats.

This option is set to nil by default, which makes it obtain stats from Saved Games/DCS/Slmod.

2.3.3- SlmodAutoAdmin Settings

Please read section 4 of this manual for a detailed description of how the auto-kick/auto-ban system ("SlmodAutoAdmin") works.

2.3.3.1- General SlmodAutoAdmin Settings

These options affect all types of offenses.

autoAdmin.autoBanEnabled

```
autoAdmin.autoBanEnabled = <boolean>
```

This option turns on the automatic temp-ban feature of SlmodAutoAdmin.

This option is set to true by default.

autoAdmin.autoKickEnabled
`autoAdmin.autoKickEnabled = <boolean>`
This option turns on the automatic kick feature of SlmodAutoAdmin.

This option is set to true by default.

autoAdmin.autoBanLevel
`autoAdmin.autoBanLevel = <number>`
This option sets the penalty score required for a player to be auto-temp-banned.

This option is set to 100 by default.

autoAdmin.autoKickLevel
`autoAdmin.autoKickLevel = <number>`
This option sets the penalty score required for a player to be auto-kicked.

This option is set to 50 by default.

autoAdmin.flightHoursWeightFunction
`autoAdmin.flightHoursWeightFunction = <table>`
This option adjusts the penalty score of a player based on how many flight hours they have on the server. This option must be set to a table of points that define consecutive points on a curve. Each point contains a time and relative weight value. The unit for time is **hours**.

This option is set to
```
{
    [1] = {time = 0, weight = 1.4},
    [2] = {time = 3, weight = 1},
    [3] = {time = 10, weight = 0.7},
}
```
by default.

autoAdmin.tempBanLimit
`autoAdmin.tempBanLimit = <number or nil>`
This option allows a temp ban to become a permanent ban ("perma-ban") once the number of times a player has been temp-banned is equal to or greater than autoAdmin.tempBanLimit. This option can be set to nil, in which case, players will never be perma-banned by SlmodAutoAdmin.

This option is set to 2 by default.

autoAdmin.reallowLevel
`autoAdmin.reallowLevel = <number>`

This option sets the value that a temp-banned player's penalty score must decay to in order to be reallowed into the server.

This option is set to 75 by default.

autoAdmin.exemptionList

`autoAdmin.exemptionList = <table>`

This option allows you to specify a list of players who are immune to SlmodAutoAdmin actions. By default, all server admins are immune already.

This option is set to an empty table by default.

2.3.3.2- Offense-Specific Settings

There are currently four different offenses that SlmodAutoAdmin can punish players for committing. They are:
- team-hit
- team-kill
- team-collision-hit
- team-collision-kill.

The following settings exist for each offense type.

autoAdmin.<*offense type*>.enabled

`autoAdmin.<offense type>.enabled = <boolean>`

This option determines if `<offense type>` will be factored into a player's penalty score. If set to true, this offense type will be factored into a player's penalty score. If set to false, this offense type will be entirely ignored.

By default, only teamKill and teamCollisionKill are enabled.

autoAdmin.<*offense type*>.penaltyPointsAI

`autoAdmin.<offense type>.penaltyPointsAI = <number>`

This sets the base penalty points that are given to a player for each offense of `<offense type>` that is committed against an AI unit.

autoAdmin.<*offense type*>.penaltyPointsHuman

`autoAdmin.<offense type>.penaltyPointsHuman = <number>`

This sets the base penalty points that are given to a player for each offense of `<offense type>` that is committed against a human.

autoAdmin.<*offense type*>.decayFunction

`autoAdmin.<offense type>.decayFunction = <table>`

This option adjusts the penalty points a specific offense of type <*offense type*> is worth based on how long ago the offense occurred. This option must be set to a table of points that define consecutive points on a curve. Each point contains a time and relative weight value. The unit of time is **days**.

autoAdmin.<*offense type*>.minPeriodAI

`autoAdmin.<offense type>.minPeriodAI = <number>`

This option sets the minimum time, in seconds, between offenses of `<offense type>,` committed against AI units, that are counted towards a player's penalty score. The reason for this option can be clearly understood if you consider that a single honest mistake (such as misidentifying a friendly convoy as enemy) often leads to multiple offenses of the same type.

autoAdmin.<*offense type*>.minPeriodHuman

`autoAdmin.<offense type>.minPeriodHuman = <number>`

This option sets the minimum time, in seconds, between offenses of `<offense type>,` committed against human-controlled units, that are counted towards a player's penalty score. The reason for this option can be clearly understood if you consider that a single honest mistake often leads to multiple offenses of the same type. An example for the case of humans vs. humans would be collisions. Often, half a dozen (or more!) "hit" events are generated in a single aircraft-to-aircraft collision.

2.3.4- Miscellaneous Server Options

This section contains general server options that don't readily fall under other categories.

pause_when_empty

`pause_when_empty = <boolean>`

When this option is set true, the server will stay paused when only the server host is in the game. It will unpause when clients join.

This option is set to false by default.

enable_pvp_kill_messages

`enable_pvp_kill_messages = <boolean>`

When this option is set true, chat message announcements will occur when a player scores a PvP kill in SlmodStats. Note that PvP kills follow their own special rules (see the section on SlmodStats), and also will work correctly in many situations where the normal server event messages fail (such as a player in a fatally-damaged aircraft leaving the server before his aircraft crashes). This option requires that SlmodStats be enabled.

This option is set to true by default.

enable_team_hit_messages

`enable_team_hit_messages = <boolean>`

When this option is set true, chat message announcements will occur when a player hits a friendly unit with his weapons or aircraft. This option requires that SlmodStats be enabled.

This option is set to true by default.

enable_team_kill_messages

```
enable_team_kill_messages = <boolean>
```

When this option is set true, chat message announcements will occur when a player kills a friendly unit with his weapons or aircraft. This option requires that SlmodStats be enabled.

This option is set to true by default.

chat_log

```
chat_log = <boolean>
```

When this option is set true, all chat messages will be recorded in a chat log that is saved in Saved Games/DCS/Slmod/Chat Logs. The chat logs are date and time stamped.

This option is set to true by default.

log_team_hits

```
log_team_hits = <boolean>
```

When this option is set true, team hits are logged in the chat log as well. Obviously, this setting requires chat_log = true.

This option set true to default.

log_team_kills

```
log_team_kills = <boolean>
```

When this option is set true, team kills are logged in the chat log as well. Obviously, this setting requires chat_log = true.

This option is set to true by default.

MOTD_enabled

```
MOTD_enabled = <boolean>
```

When this option is set true, it enables the server Message Of The Day (MOTD). This is a message delivered to players when they join an aircraft.

This option is set to true by default.

custom_MOTD

```
custom_MOTD = <string> or nil
```

This option allows you to specify your own Message of the Day instead of using the default Slmod MOTD. Your MOTD must be a Lua string.

This option is set to nil by default, which enables the default Slmod MOTD.

MOTD_display_time

```
MOTD_display_time = <number>
```

This option specifies how long the MOTD is shown for (in seconds).

This option is set to 30 by default.

MOTD_display_mode

`MOTD_display_mode = <string>`

This option controls the display mode of the MOTD. This value must be either 'chat', 'text', 'echo', or 'both'. See "2.2 A Note on Menu/Text Formatting" for an explanation of these display modes.

This option is set to 'chat' by default.

MOTD_show_POS_and_PTS

`MOTD_show_POS_and_PTS = <boolean>`

This option controls whether or not the keystrokes for the Parallel Options System and Parallel Tasking System are shown in the MOTD.

This option is set to false by default.

2.3.5- Parallel Tasking System Options

PTS_list_display_mode

`PTS_list_display_mode = <string>`

This option controls the display mode of the Parallel Tasking System task list. This value must be either 'chat', 'text', 'echo', or 'both'. See "2.2 A Note on Menu/Text Formatting" for an explanation of these display modes.

This option is set to 'text' by default.

PTS_list_display_time

`PTS_list_display_time = <number>`

This option sets the display time of the Parallel Tasking System task list (in seconds).

This option is set to 30 by default.

PTS_task_display_mode

`PTS_task_display_mode = <string>`

This option controls the display mode of the Parallel Tasking System tasks. This value must be either 'chat', 'text', 'echo', or 'both'. See "2.2 A Note on Menu/Text Formatting" for an explanation of these display modes.

This option is set to 'text' by default.

PTS_task_display_time

`PTS_task_display_time = <number>`

This option sets the display time of the Parallel Tasking System tasks (in seconds).

This option is set to 40 by default.

2.3.6- Parallel Options System Options

POS_list_display_mode

`POS_list_display_mode = <string>`

This option controls the display mode of the Parallel Options System options list. This value must be either 'chat', 'text', 'echo', or 'both'. See "2.2 A Note on Menu/Text Formatting" for an explanation of these display modes.

This option is set to 'text' by default.

POS_list_display_time

`POS_list_display_time = <number>`

This option sets the display time of the Parallel Options System options list (in seconds).

This option is set to 30 by default.

2.3.7- Miscellaneous Options

export_world_objs

`export_world_objs = <boolean>`

Set to export_world_objs to true to export a list of active world units to Saved Games/DCS/Slmod/activeUnits.txt. These units are updated roughly every 5 seconds.

This option is set to false by default.

events_output

`events_output = <boolean>`

Set this option to true to output game events in real time to Saved Games/DCS/Slmod/slmodEvents.txt.

This option is set to false by default.

coord_converter

`coord_converter = <boolean>`

Set this option to true to enable the coordinate conversion menu. See the Slmod Coordinate Conversion Menu section of this guide for more detail.

This option is set to true by default.

debugger

`debugger = <boolean>`

Set debugger to true to enable the Slmod debugger utility. The Slmod debugger utility reports Lua *syntax* errors ONLY. Currently, it only reports Lua syntax errors in triggered action Lua scripts. Any Lua syntax errors found are reported at the beginning of the mission, when started in multiplayer.

This option is set to false by default.

Unless someone actually still wants this feature, it will probably end up getting deprecated in future versions of Slmod.

debugger_outfile

```
debugger_outfile = <string>
```

If the debugger is active and it finds Lua syntax errors, these errors are saved to a file in to <Drive>/Users/<Your account name>/Saved Games/DCS <whatever>/Logs. debugger_outfile controls the name of this file. By default, the name of this file is: “Lua_Syntax_Errors.txt”

udp_port

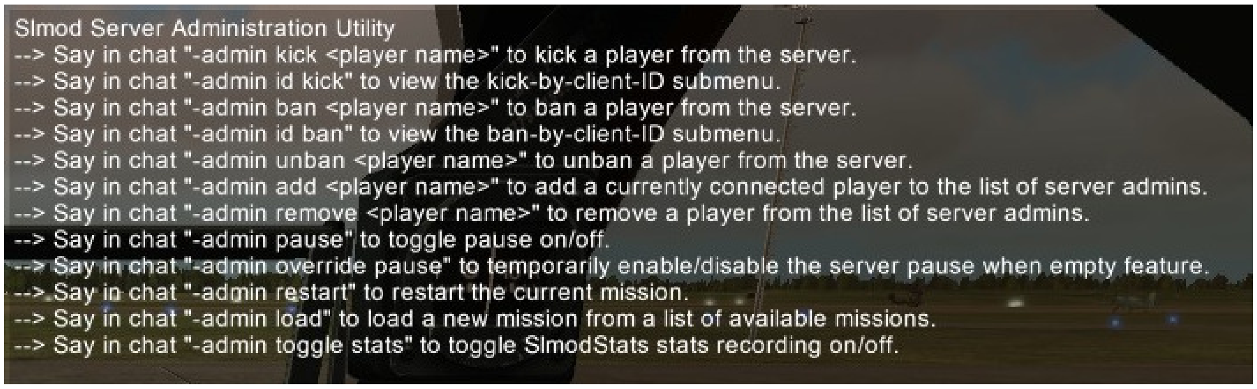
```
udp_port = <number>
```

Slmod mission scripting commands are passed to Slmod net code through a localhost UDP port. This option allows you to set which UDP port to use.

This option is set to 52146 by default.

3- Slmod Admin Menu

Slmod provides a server administration menu to give multiplayer server hosts more control over their server, and allow remote access (for those who are registered administrators) to functions like server pausing, mission reloading, mission loading, kicking, banning, etc. By default, the Admin Menu is enabled.



```
Slmod Server Administration Utility
--> Say in chat "-admin kick <player name>" to kick a player from the server.
--> Say in chat "-admin id kick" to view the kick-by-client-ID submenu.
--> Say in chat "-admin ban <player name>" to ban a player from the server.
--> Say in chat "-admin id ban" to view the ban-by-client-ID submenu.
--> Say in chat "-admin unban <player name>" to unban a player from the server.
--> Say in chat "-admin add <player name>" to add a currently connected player to the list of server admins.
--> Say in chat "-admin remove <player name>" to remove a player from the list of server admins.
--> Say in chat "-admin pause" to toggle pause on/off.
--> Say in chat "-admin override pause" to temporarily enable/disable the server pause when empty feature.
--> Say in chat "-admin restart" to restart the current mission.
--> Say in chat "-admin load" to load a new mission from a list of available missions.
--> Say in chat "-admin toggle stats" to toggle SlmodStats stats recording on/off.
```

Figure 1: the Admin Menu (shown as trigger text).

For information on config file settings for the Server Admin Menu, see section "2.3.1- Slmod Admin Menu Options".

3.1- Admin Menu Commands

Like all Slmod menus, users interact with the Admin Menu through chat messages.
These are the Admin Menu commands/options:

-admin
If you're a server admin, typing this command into chat shows the Admin Menu.

-admin kick *<player name>*
By typing this command into chat, a server admin can kick the player whose name matches *<player name>*. The player may rejoin at any time.

-admin id kick
By typing this command into chat, a server admin will view a submenu that shows all currently connected clients, with each client's client ID number specified. By typing into chat "-admin id kick *<ID number>*", the server admin can then kick the specified client from the server.

-admin ban *<player name>*
By typing this command into chat, a server admin can permanently ban the player whose name matches *<player name>*.

-admin id ban
By typing this command into chat, a server admin will view a submenu that shows all currently connected clients, with each client's client ID number specified. By typing into chat "-admin id ban *<ID number>*", the server admin can then permanently ban the specified client from the server.

-admin unban *<player name>*
By typing this command into chat a server admin can unban a previously banned player whose name matched *<player name>*. If multiple banned players exist with this same name, all of them will be unbanned.

Note: You can manually edit the ban list by opening the Saved Games/Slmod/BannedClients.lua file in a text editor such as *Notepad++.*

-admin add *<player name>*
By typing this command into chat, a server admin can grant a currently connected client whose name matches *<player name>* server admin privileges, allowing them to use the Admin Menu.

-admin remove *<player name>*
By typing this command into chat, a server admin can revoke the server administrator privileges of the players whose names matched *<player name>*. If multiple administrators exist with this same name, all of them will have their admin privileges revoked.

Note: You can manually edit the list of server admins by opening the Saved Games/Slmod/ServerAdmins.lua file in a text editor such as *Notepad++.*

-admin pause
By typing this command into chat, a server admin can pause or unpause the server.

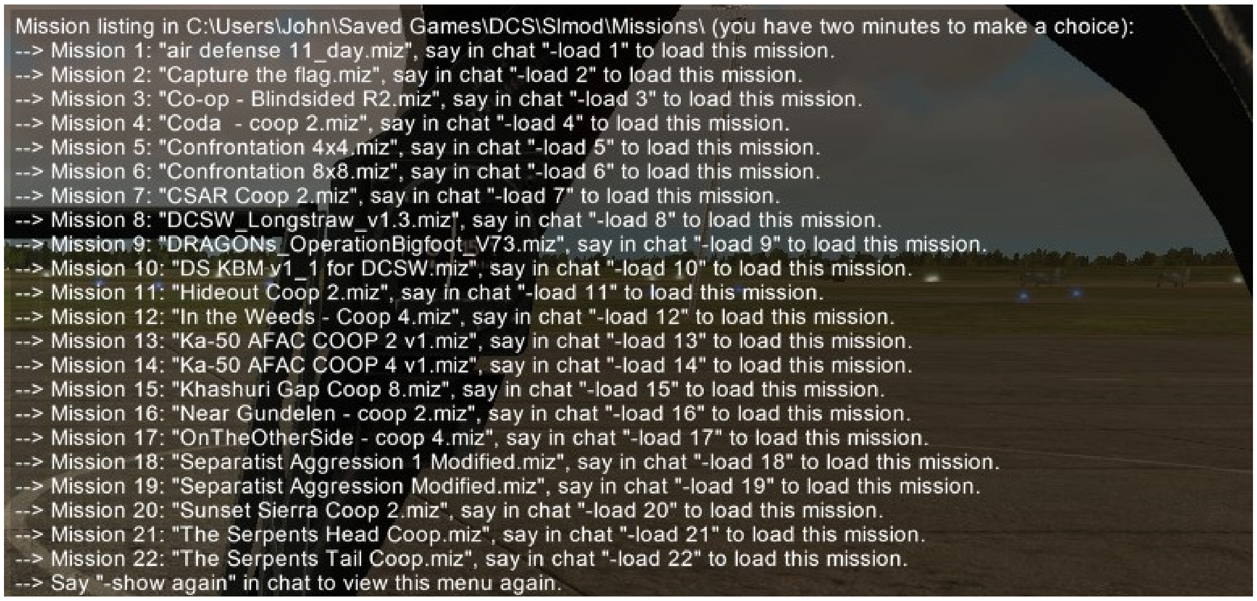
-admin override pause
By typing this command into chat, a server admin can override the pause-when-empty server feature. This command is only available if the pause-when-empty feature is enabled.

-admin restart
By typing this command into chat, a server admin will restart the current mission.

-admin load
By typing this command into chat, a server admin will open a submenu that will (by default) display a list of the missions that are located in Saved Games/DCS/Slmod/Missions/. Each mission will be given a number. The server admin who requested this submenu can then select a mission to load by typing in "-load *<mission number>*" into chat. Two minutes will be given to make a selection before the menu expires.

Instead of missions coming from the default Saved Games/DCS/Slmod/Missions/ directory, you may tell Slmod to use an alternate directory for missions by using the `admin_tools_mission_folder` option in the config.lua file. This folder could even be a shared folder, which is especially useful for virtual squadrons- sharing your mission with your squad's dedicated server can be as simple as uploading it to a shared folder on YOUR computer! Or, if your squad uses multiple dedicated servers, you can have all servers share the same missions folder.

An example of what this load mission submenu looks like is shown in Figure 2.



```
Mission listing in C:\Users\John\Saved Games\DCS\Slmod\Missions\ (you have two minutes to make a choice):
--> Mission 1: "air defense 11_day.miz", say in chat "-load 1" to load this mission.
--> Mission 2: "Capture the flag.miz", say in chat "-load 2" to load this mission.
--> Mission 3: "Co-op - Blindsided R2.miz", say in chat "-load 3" to load this mission.
--> Mission 4: "Coda  - coop 2.miz", say in chat "-load 4" to load this mission.
--> Mission 5: "Confrontation 4x4.miz", say in chat "-load 5" to load this mission.
--> Mission 6: "Confrontation 8x8.miz", say in chat "-load 6" to load this mission.
--> Mission 7: "CSAR Coop 2.miz", say in chat "-load 7" to load this mission.
--> Mission 8: "DCSW_Longstraw_v1.3.miz", say in chat "-load 8" to load this mission.
--> Mission 9: "DRAGONs_OperationBigfoot_V73.miz", say in chat "-load 9" to load this mission.
--> Mission 10: "DS KBM v1_1 for DCSW.miz", say in chat "-load 10" to load this mission.
--> Mission 11: "Hideout Coop 2.miz", say in chat "-load 11" to load this mission.
--> Mission 12: "In the Weeds - Coop 4.miz", say in chat "-load 12" to load this mission.
--> Mission 13: "Ka-50 AFAC COOP 2 v1.miz", say in chat "-load 13" to load this mission.
--> Mission 14: "Ka-50 AFAC COOP 4 v1.miz", say in chat "-load 14" to load this mission.
--> Mission 15: "Khashuri Gap Coop 8.miz", say in chat "-load 15" to load this mission.
--> Mission 16: "Near Gundelen - coop 2.miz", say in chat "-load 16" to load this mission.
--> Mission 17: "OnTheOtherSide - coop 4.miz", say in chat "-load 17" to load this mission.
--> Mission 18: "Separatist Aggression 1 Modified.miz", say in chat "-load 18" to load this mission.
--> Mission 19: "Separatist Aggression Modified.miz", say in chat "-load 19" to load this mission.
--> Mission 20: "Sunset Sierra Coop 2.miz", say in chat "-load 20" to load this mission.
--> Mission 21: "The Serpents Head Coop.miz", say in chat "-load 21" to load this mission.
--> Mission 22: "The Serpents Tail Coop.miz", say in chat "-load 22" to load this mission.
--> Say "-show again" in chat to view this menu again.
```

Figure 2: The load mission submenu.

-admin toggle stats
By typing this command into chat, a server admin will toggle the on/off state of SlmodStats stats recording. This is useful if you want to have some consensual team-killing or something silly like that, or if you only want to record stats during specific missions/times while leaving stats recording turned off for the rest of the time. For more on SlmodStats, see the SlmodStats portion of this manual.

3.2- Registering Server Admins

As already mentioned, the "-admin add <*player name*>" can be used to add players to the list of server admins. However, there are two other ways to add server admins. First, server admins can be added by adding their UCIDs and names to the Saved Games/DCS/Slmod/ServerAdmins.lua file.

The final way of registering server admins is to use the admin_register_password option in Saved Games/DCS/Slmod/config.lua. By setting this variable to a string value, this string value becomes the server admin registration password.

In order to register as a server admin using this option, an unregistered client needs to type "-reg" in chat. The client will then be prompted to enter the password to register as a server admin. Whatever the client types next will not be shown in chat, even if he gets the password wrong. If the client successfully enters a chat message that matches the password, he will be registered as a server administrator.

For a more detailed description of how to set up this password, please see section "2.3.1 Server Admin Menu Options".

3.3- Manually Banning/Unbanning a Player

Occasionally, it may be desirable to manually ban or unban someone. To do so, you need to edit the Saved Games/DCS/Slmod/BannedClients.lua file. To manually ban someone by UCID, find the slmod_banned_ucids table inside BannedClients.lua, and add an entry that has an index of the player's UCID, and a value that is the player's name, such as:

```
["ab04135debb24c40032489ad03"] = "Douchebag",
```

To manually ban someone by IP, find the slmod_banned_ips table inside BannedClients.lua, and add an entry that has an index of the player's IP, and a value that is the player's name, such as:

```
["102.99.118.92"] = "Douchebag",
```

If you should want to manually unban someone, you need to find their entries in the slmod_banned_ucids and slmod_banned_ips table and simply delete them.

WARNING:

Be sure to back-up your ban list before you restart DCS after making a manual edit of your BannedClients.lua file. If you create a Lua error, Slmod will erase your BannedClients.lua file and replace it with an empty one!

4- SlmodAutoAdmin Auto-kick/Auto-ban System

Slmod now includes a highly customizable and capable auto-kicking/auto-banning system. This system, SlmodAutoAdmin, uses SlmodStats to evaluate a player's "criminal history" when they connect and when they commit an offense. Based on the player's past history of offenses, the player may be cleared to play on the server, or could face disciplinary actions such as being kicked, being temporarily banned, or even being permanently banned. SlmodAutoAdmin takes into account such factors as the client's flight time on the server, how many offenses were committed of each type, how long ago these offenses were, and whether or not these offenses were clustered together in time.

It is important to note that a player can commit offenses even if he is not currently connected to the server. Should a player team-damage a unit, disconnect from the server, and the unit he team-damaged later crash or otherwise fail to return to base, then player will receive a team kill, even while not being connected to the server.

4.1- Understanding Penalty Score

SlmodAutoAdmin makes decisions on player punishments based on the player's "penalty score". The penalty score is a numerical value that is calculated based on settings derived from the config.lua file, and on a player's history of transgressions as recorded in SlmodStats. Each offense is worth a certain number of penalty points, and all these penalty points added together make up the player's penalty score.

There are four basic types of offenses- team-hit, team-kill, team-collision-hit, and team-collision-kill. The amount of penalty points a specific offense is worth at a specific time is related the "base" number of points the offense type is worth, how long ago the offense occurred, and how many flight hours the player has on the server. This value, which we will call P_s, is given with the equation

$$P_s = F_h(p_t) \cdot F_d(t - t_o) \cdot P_B$$

where:

F_h is the flight hours weight function, a piecewise function of player flight hours (where player flight hours is p_t) as defined in `autoAdmin.flightHoursWeightFunction;`

F_d is the decay curve for the specific offense type, a piecewise function of time since the offense occurred (where the current time is t and the time of the offense is t_o) as described in `autoAdmin.<`*`offense type`*`>.decayFunction;`

P_B is the base number of points this particular type of offense is worth as described in `autoAdmin.<`*`offense type`*`>.penaltyPointsAI` or `autoAdmin.<`*`offense type`*`>.penaltyPointsHuman.`

In general, the value of penalty points each specific offense a player commits is worth is independent of any other offenses a player commits. The exception is in the case of multiple offenses of the same type occurring within a short time span. In this case, events of the same offense type against the same kind of unit (AI or human) can ONLY be counted once every `autoAdmin.<`*`offense type`*`>.minPeriodAI` or `autoAdmin.<`*`offense type`*`>.minPeriodHuman` seconds.

The default flight hours weight function as defined in `autoAdmin.flightHoursWeightFunction` is shown in Figure 3, and the default decay functions for all four offense types is shown in Figure 4. As implied, these are just the default curves; you may modify your config.lua file and change them to whatever you want.

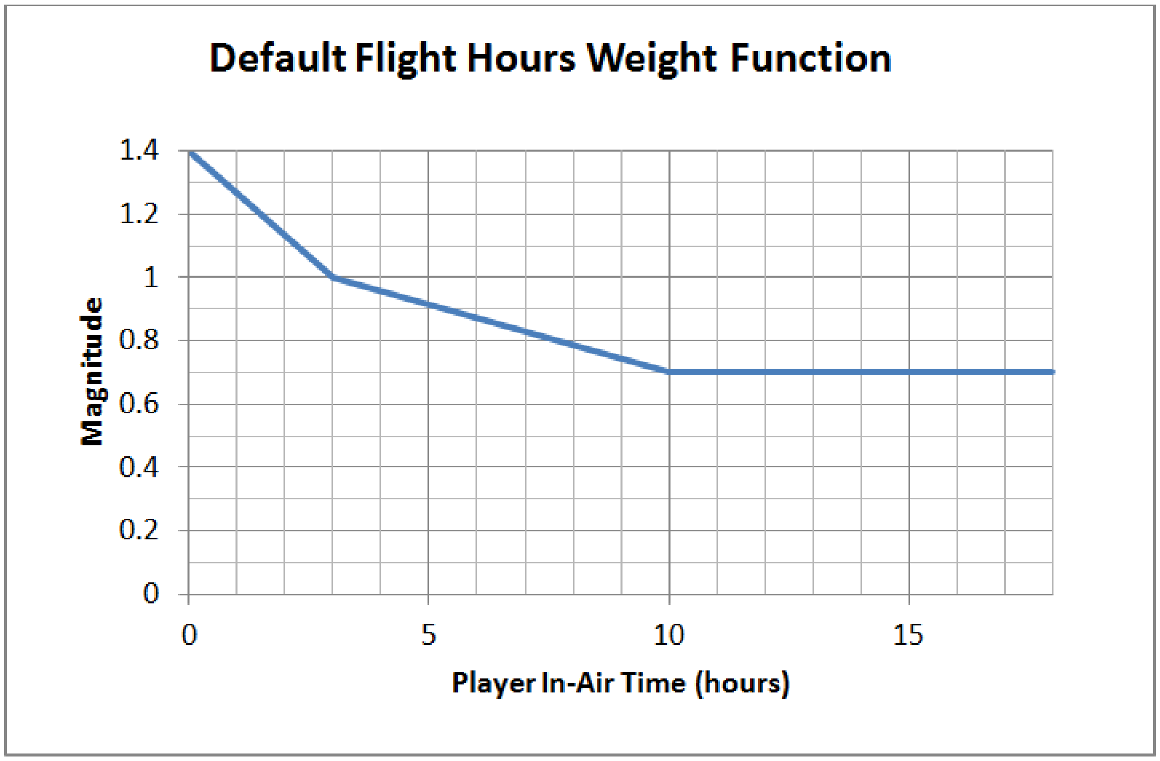


Figure 3: Default flight hours weight function.

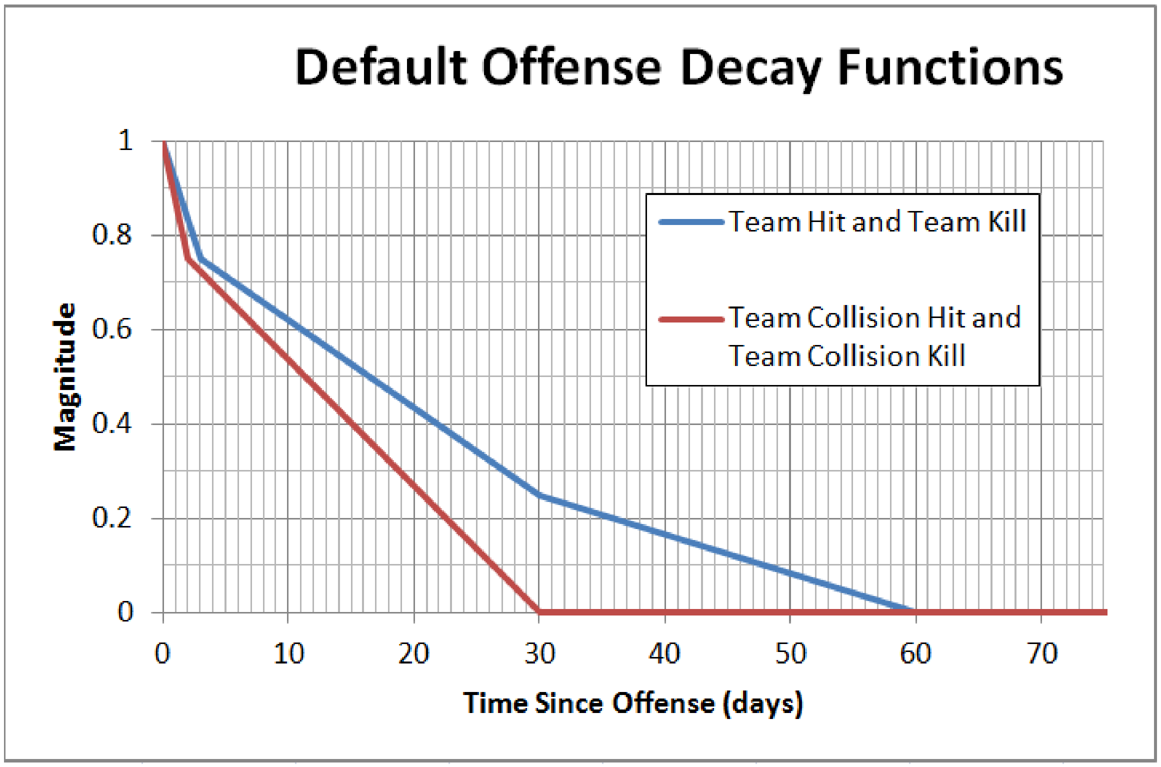


Figure 4: default offense decay functions.

Let's work a few examples to demonstrate how penalty score is calculated. Assume that all variables are set to their default values.

First, for simplicity, let's assume we have a player who has commited only one offense, a single team-kill of a human. Since we are using the default settings in these examples, the team-hit that resulted in the team-kill will not be counted. The player has 1.5 flight hours on the server, and the team-kill occurred 3.0 days ago.

To calculate this player's current penalty score, first observe Figure 3. At 1.5 flight hours, the value of the flight hours weight function is 1.2. Next, by inspection of Figure 4, we see that the value of the team-kill decay function at 3.0 days is 0.75. Finally, see that the base number of penalty points a team-kill of a human is worth (`autoAdmin.teamKill.penaltyPointsHuman`) is 30. So the number of penalty points this player has is $1.2 \times 0.75 \times 30 = 27$.

Next, let's look at a much more complex example. Consider a player with the following stats/offenses:

- Flight hours: 9.0
- Offense #1
  - Type: team-collision-kill of a human
  - Time since offense: 35 days
- Offense #2
  - Type: team-kill of human

    - Time since offense: 10 days
- Offense #3
    - Type: team-kill of AI
    - Time since offense: 5 days
- Offense #4
    - Type: team-kill of an AI
    - Time since offense: 4.99983 days (in other words- 15 seconds after Offense #3)
- Offense #5
    - Type: team-kill of an AI
    - Time since offense: 4.99965 days (in other words- 30 seconds after Offense #3)

First, since it is the same value for all offenses, let us find the value of the flight hours weight function for this player. Inspecting Figure 3, we see that the value of the flight hours weight function at 9.0 hours of flight time is about 0.74.

Next, let us calculate the penalty points Offense #1 is worth (the team-collision-kill of a human). Inspecting Figure 4, we see that no calculation is required; the default decay function for team-collision-kill reaches a value of zero at 30 days, and the offense happened 35 days ago. Thus, this offense is worth zero penalty points.

Next, calculate the value of penalty points that Offense #2 is worth (the team-kill of a human). The offense happened 10 days ago; by inspecting Figure 4, we see that the value of the team-kill decay curve at 10 days is about 0.62. Since the base penalty points of a team-kill of a human is 30, and remembering that the flight hours weight function value for this player is currently 0.74, this offense is worth $0.74 \times 0.62 \times 30 = 13.8$ penalty points.

Finally, let us calculate the penalty points value for the three team-kills on AI units that happened about 5 days ago. First, inspecting Figure 4 shows that the value of the team-kill decay curve at 5.0 days is about 0.71. Since the base penalty points of a team-kill of an AI unit is 18, and the flight hours weight function value for this player is 0.74, the first of these three AI team-kills is worth $0.74 \times 0.71 \times 18 = 9.5$ penalty points.

The next team-kill of an AI unit that happens at 4.99983 days would be worth about the same amount if it wasn't for the `autoAdmin.teamKill.minPeriodAI` setting. Note that by default, this value is 20 seconds, and 4.99983 days is only 15 seconds after 5.0 days. <u>Thus, the second team-kill is ignored.</u> However, the last of the three team-kills of AI units occurs at 4.99965 days, which is 30 seconds after the last team-kill on an AI unit that was counted (the team-kill that occurred at 5.0 days). Since this time interval is greater than `autoAdmin.teamKill.minPeriodAI`, this team kill *does* count, and is worth about 9.5 penalty points, as previously counted.

Thus, this player's total penalty score would currently be at $13.8 + 2 \times 9.5 = 32.8$ penalty points.

4.2- Punishments and Punishment Thresholds

Now that an understanding of how penalty points work is established, a meaningful discussion can be made of how SlmodAutoAdmin hands out punishments based on these penalty scores.
Based on the server's `autoAdmin.autoKickEnabled`, `autoAdmin.autoBanEnabled`, and `autoAdmin.tempBanLimit` config.lua settings, players may be auto-kicked, auto-banned, or auto-perma-banned. (In this discussion "auto-banned" denotes an automatic temporary ban, and "auto-perma-banned" denotes an automatic permanent ban.)

If `autoAdmin.autoKickEnabled` is set true in the config.lua file, then a player can be auto-kicked in two different situations:

1) The player commits an offense that pushes his penalty score over the value set by `autoAdmin.autoKickLevel`

2) The player commits an offense, and his penalty score is *already* in excess of `autoAdmin.autoKickLevel.`

By default, `autoAdmin.autoKickLevel` is set to 50.

If `autoAdmin.autoBanEnabled` is set true in the config.lua file, then a player will be auto-banned in two different situations:

1) The player commits an offense that pushes his penalty score over the value set by `autoAdmin.autoBanLevel;`

2) Upon the player connecting, if the player's penalty score is found to be in excess of `autoAdmin.autoBanLevel`, and the player is not already banned.

A player will become unbanned once his penalty score decays below the value set by `autoAdmin.reallowLevel,` unless the player has been perma-banned.

Finally, a player will be auto-perma-banned should the number of times he has been auto-banned equal or exceed `autoAdmin.tempBanLimit.` Note that as of Slmod v 7.0, there is no decay rate on the counter that keeps track of the number of times a player has been temp banned. This is certainly a feature that may be worth investigating as time goes on.

4.3- Exemptions

Players registered as server admins that are able to access the Slmod Admin Menu are exempt from SlmodAutoAdmin actions/punishments. Additionally, the `autoAdmin.exemptionList` allows you to define a list of players who are exempt from SlmodAutoAdmin disciplinary actions. The players must be entered into this Lua table, and be indexed by their UCID, a string value. The value this UCID index is set to does not matter, as long as it is not false or nil. It is recommended that you set it equal to the player's string name, as shown in the following example:

```
autoAdmin.exemptionList = {
    ["14da0413fe23c41b2495f2f257"] = "flowyerG",
    ["a50f741d91ef2478213b42023f"] = "deepS",
    ["ab04135debb24c40032489ad03"] = "semirG",
}
```

4.4- Un-Perma-Banning Someone

Players who get perma-banned get entered into the Saved Games/DCS/Slmod/BannedClients.lua file, and are banned by both IP and UCID. Players who have been auto-perma-banned by SlmodAutoAdmin will be will have an entry that says ["bannedBy"] = "autoban". If you want to un-perma-ban a player, just delete their entries in the slmod_banned_ips and slmod_banned_ucids tables.

WARNING:

Be sure to back-up your ban list before you restart DCS after making a manual edit of your BannedClients.lua file. If you create a Lua error, Slmod will erase your BannedClients.lua file and replace it with an empty one!

5- SlmodStats

Slmod v7+ includes SlmodStats, a multiplayer statistics tracking system. SlmodStats stores multiplayer statistics across hosting sessions, allowing your server to build up a database of player statistics. SlmodStats tracks the following data:

- Flight time and total time in each aircraft (and total time in Combined Arms)
- Number of kills
- Types of kills (such as Ground Unit -> Tanks or Helicopter -> Attack)
- Number of crashes, ejections, and deaths
- Number of friendly fire hits and kills
- Number of fair PvP kills
- Types of and number of weapons fired, and the number of hits and kills with these weapons

The data that SlmodStats collects is saved to the hard drive between and *during* hosting sessions. If the host crashes to desktop, no data will be lost. SlmodStats also is capable of saving data for each mission flown. These files can also be saved to the hard drive. The saved data is in the form of a serialized Lua table, and thus can be reloaded into anything else that has access to a Lua interpreter. This includes external websites.

What sets SlmodStats apart from other stat systems that have come before is that it exists within DCS. This allows the stats to be displayed *in game* to clients using trigger text or chat messages- no more logging into external websites to see your stats (though, as already stated, nothing prevents you from loading the SlmodStats data into an external website too!). Thus, SlmodStats includes a built-in stats display menu accessed through the "-stats" chat command.

Additionally, being part of DCS itself allows SlmodStats to gather more information than previous stats systems could. This gives it a lot of room for expansion and enhancement in the future, too. It resolves the "killed by building" bugs, and makes it possible to even credit players who are offline with kills (should something they have damaged later die as a result of their damage to it, after they have already logged out).

Finally, note that SlmodStats tracks player stats by their unique DCS account UCID number. Thus, a player's name does not matter. In fact, SlmodStats even tracks what names you've used on the server… so be warned!

PLEASE SEE SECTION 2.3.2 FOR SlmodStats config.lua FILE OPTIONS!

5.1- Stats Types

As of Slmod v7.0, there are two kinds of stats: "server" stats and "mission" stats. Server stats are stats that are recorded across the server's entire history. They show all statistics that have been recorded since SlmodStats started keeping records. By default, these are the only stats that are kept.

Mission stats, on the other hand, are stats that are only recorded for the time that a single mission is run. By default, mission stats are disabled, but they may be enabled by setting `enable_mission_stats` to true in the config.lua file.

5.2- Enabling/Disabling SlmodStats

As already mentioned in *3.1- Admin Menu Commands*, the -admin toggle stats command allows an admin to enable or disable stats tracking. If stats tracking is disabled, clients will not be awarded any stats. However, SlmodStats is only partially dormant in this mode- new clients that join are still added to the stats tables (they just don't accumulate any stats), and you can still view the stats menus. By setting the `enable_slmod_stats` option to false in the config.lua file, you can make SlmodStats start in this mode by default.

5.3- Stats Files

By default, SlmodStats keeps track of server stats in Saved Games/DCS/Slmod/SlmodStats.lua. When DCS is loaded, Slmod executes the contents of this file as Lua code and attempts to get the resulting "stats" table that is created. If the "stats" table is successfully loaded from this file, it re-serializes the stats table and outputs it back into Saved Games/DCS/Slmod/SlmodStats.lua. If the "stats" table is not successfully recovered (due to a bug in Slmod, DCS, or manual user edits of the file), it backs up the old SlmodStats.lua file to SlmodStats_BACKUP.lua and creates a new SlmodStats.lua file.

As DCS is still far from being completely crash-free over long time spans, it is important that SlmodStats update the SlmodStats.lua file during mission runtime to avoid the loss of player stats. Serializing large Lua tables and saving them to the hard drive demands a lot of computer resources. Thus, instead of reserializing the stats table and resaving it to the hard drive, Slmod just keeps the SlmodStats.lua file open and appends onto it single lines of Lua that modify the original stats table. This happens in real time as clients' stats change. The take-away point is this: you will not lose stats if the server crashes to desktop!

If `enable_mission_stats` and `write_mission_stats_files` are both set to true in the config.lua file, then Slmod will also save stats files for each mission that is played. By default, these files are saved in Saved Games/DCS/Slmod/Mission Stats/. The files are labeled by mission name, date, and time, and, like the server stats, they are Lua files.

If you want to read in the stats data from your stats files into an external program, you need to open and "dofile" them with a Lua interpreter. Lua interpreters are available for all major operating systems.

Slmod also allows you save your stats files (both mission and server stats files) in alternate directories using the `mission_stats_files_dir` and `stats_ dir` options in the config.lua file. The alternate directory(ies) could even be an FTP folder or Dropbox folder.

5.4- PvP Kill Rules

SlmodStats applies a special rule set to score Player-vs.-Player (PvP) kills and deaths. SlmodStats assumes that the natural "food chain" in PvP combat is this:
Fighters > Attack Aircraft > Helicopters.
You are only awarded a PvP kill if you kill someone on your level of the "food chain" or *higher*, and you are only awarded a PvP death if you are killed by someone on your level of the "food chain" or *lower*.

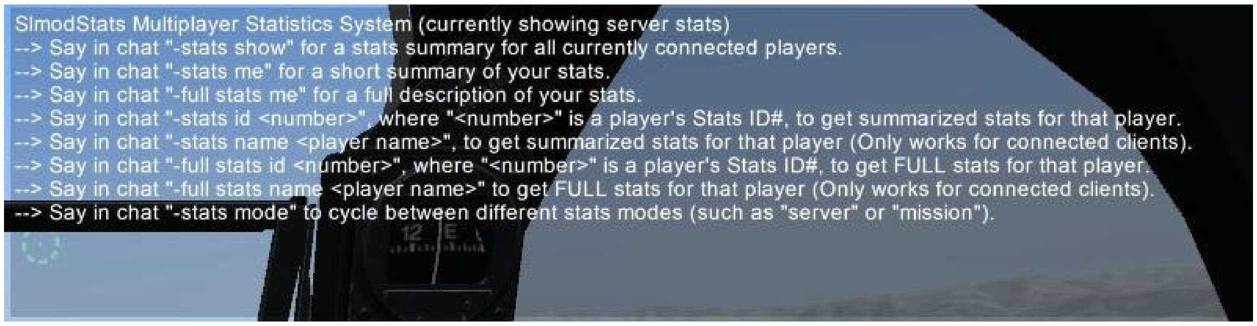
Here are some examples:
Ka-50 #1 kills Ka-50 #2 - PvP kill given to Ka-50 #1, PvP death given to Ka-50 #2.
Su-27 kills Ka-50 - No PvP kills or deaths given
A-10C kills Su-27 - PvP kill given to A-10C, PvP death given to Su-27
A-10C kills Ka-50 - No PvP kills or deaths given
F-15C kills Su-27 - PvP kill given to F-15C, PvP death given to Su-27

With this system in place, players flying attack aircraft and helicopters are not punished with poor PvP records for choosing an aircraft that is at an extreme disadvantage in an air-to-air scenario.

SlmodStats applies an additional rule to PvP kills as well- the hit that kills the player must occur in the air. Thus, no PvP kills or deaths are awarded for killing players on the ground!

5.5- The Stats Menu

Figure 5 shows the main stats menu, accessed with the "-stats" chat command.



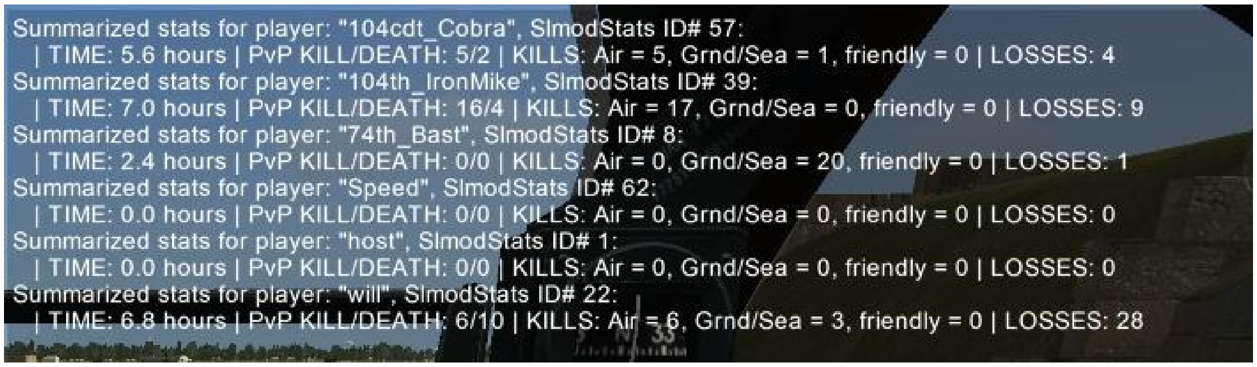
```
SlmodStats Multiplayer Statistics System (currently showing server stats)
--> Say in chat "-stats show" for a stats summary for all currently connected players.
--> Say in chat "-stats me" for a short summary of your stats.
--> Say in chat "-full stats me" for a full description of your stats.
--> Say in chat "-stats id <number>", where "<number>" is a player's Stats ID#, to get summarized stats for that player.
--> Say in chat "-stats name <player name>", to get summarized stats for that player (Only works for connected clients).
--> Say in chat "-full stats id <number>", where "<number>" is a player's Stats ID#, to get FULL stats for that player.
--> Say in chat "-full stats name <player name>" to get FULL stats for that player (Only works for connected clients).
--> Say in chat "-stats mode" to cycle between different stats modes (such as "server" or "mission").
```

Figure 5: The main stats menu, accessed with "-stats".

The following commands interact with the stats menu:

-stats
Typing this command into chat shows the main stats menu, as seen in Figure 5.

-stats show
Typing this into chat gives you a summary of the stats of all currently connected players. Figure 6 shows what this looks like.


```
Summarized stats for player: "104cdt_Cobra", SlmodStats ID# 57:
  | TIME: 5.6 hours | PvP KILL/DEATH: 5/2 | KILLS: Air = 5, Grnd/Sea = 1, friendly = 0 | LOSSES: 4
Summarized stats for player: "104th_IronMike", SlmodStats ID# 39:
  | TIME: 7.0 hours | PvP KILL/DEATH: 16/4 | KILLS: Air = 17, Grnd/Sea = 0, friendly = 0 | LOSSES: 9
Summarized stats for player: "74th_Bast", SlmodStats ID# 8:
  | TIME: 2.4 hours | PvP KILL/DEATH: 0/0 | KILLS: Air = 0, Grnd/Sea = 20, friendly = 0 | LOSSES: 1
Summarized stats for player: "Speed", SlmodStats ID# 62:
  | TIME: 0.0 hours | PvP KILL/DEATH: 0/0 | KILLS: Air = 0, Grnd/Sea = 0, friendly = 0 | LOSSES: 0
Summarized stats for player: "host", SlmodStats ID# 1:
  | TIME: 0.0 hours | PvP KILL/DEATH: 0/0 | KILLS: Air = 0, Grnd/Sea = 0, friendly = 0 | LOSSES: 0
Summarized stats for player: "will", SlmodStats ID# 22:
  | TIME: 6.8 hours | PvP KILL/DEATH: 6/10 | KILLS: Air = 6, Grnd/Sea = 3, friendly = 0 | LOSSES: 28
```
Figure 6: Summarized stats for currently connected clients as commanded by "-stats show".

-stats me

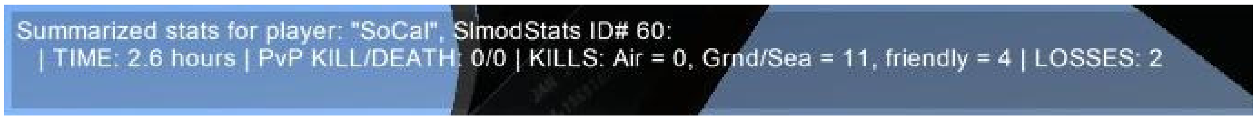
Typing this command into chat gives you a summary of your stats only. Figure 7 shows what a single summarized stats looks like.

-full stats me

Typing this command into chat gives you a full report of your own stats. The report comes in two pages- the first page gives kills, flight times, friendly fire hits/kills, and the second page gives weapons fired data. Figure 8 and Figure 9 show what these two pages look like.

-stats id <SlmodStats id #>

Each player is assigned a unique SlmodStats id #. By typing this command into chat, you can get a summarized stats page for a player based on their SlmodStats id #. Figure 7 shows what this looks like.

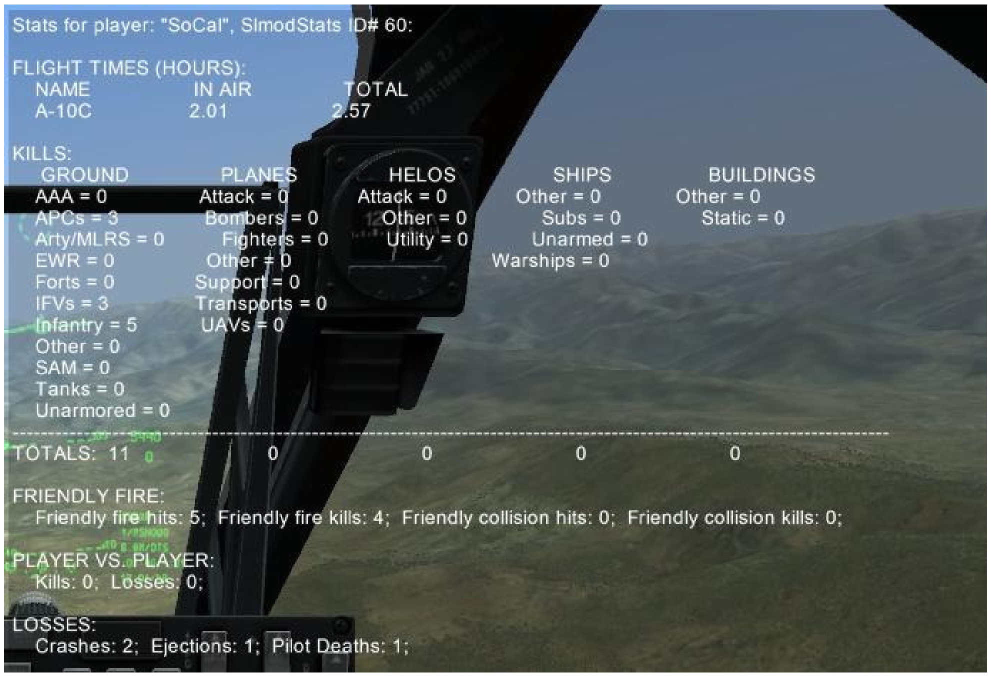

```
Summarized stats for player: "SoCal", SlmodStats ID# 60:
  | TIME: 2.6 hours | PvP KILL/DEATH: 0/0 | KILLS: Air = 0, Grnd/Sea = 11, friendly = 4 | LOSSES: 2
```
Figure 7: A summarized stats page for a player.

-stats name <player name>

This option allows you to get a summarized stats page for a *currently connected* player by specifying that player's name. Figure 7 shows what this looks like.

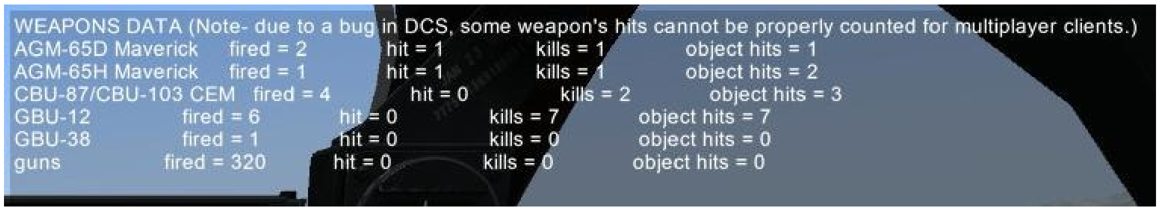
-full stats id <SlmodStats id #>

Each player is assigned a unique SlmodStats id #. By typing this command into chat, you can get a full stats report for a player based on their SlmodStats id #. The report comes in two pages- the first page gives kills, flight times, friendly fire hits/kills, and the second page gives weapons fired data. Figure 8 and Figure 9 show what these two pages look like.



```
Stats for player: "SoCal", SlmodStats ID# 60:

FLIGHT TIMES (HOURS):
   NAME              IN AIR          TOTAL
   A-10C             2.01            2.57

KILLS:
    GROUND            PLANES         HELOS          SHIPS          BUILDINGS
   AAA = 0           Attack = 0     Attack = 0     Other = 0      Other = 0
   APCs = 3          Bombers = 0    Other = 0      Subs = 0       Static = 0
   Arty/MLRS = 0     Fighters = 0   Utility = 0    Unarmed = 0
   EWR = 0           Other = 0                     Warships = 0
   Forts = 0         Support = 0
   IFVs = 3          Transports = 0
   Infantry = 5      UAVs = 0
   Other = 0
   SAM = 0
   Tanks = 0
   Unarmored = 0
-----------------------------------------------------------------------------------------
TOTALS:  11          0              0              0              0

FRIENDLY FIRE:
   Friendly fire hits: 5;  Friendly fire kills: 4;  Friendly collision hits: 0;  Friendly collision kills: 0;

PLAYER VS. PLAYER:
   Kills: 0;  Losses: 0;

LOSSES:
   Crashes: 2;  Ejections: 1;  Pilot Deaths: 1;
```

Figure 8: Full stats report for a player, page 1.



```
WEAPONS DATA (Note- due to a bug in DCS, some weapon's hits cannot be properly counted for multiplayer clients.)
AGM-65D Maverick     fired = 2        hit = 1       kills = 1       object hits = 1
AGM-65H Maverick     fired = 1        hit = 1       kills = 1       object hits = 2
CBU-87/CBU-103 CEM   fired = 4        hit = 0       kills = 2       object hits = 3
GBU-12               fired = 6        hit = 0       kills = 7       object hits = 7
GBU-38               fired = 1        hit = 0       kills = 0       object hits = 0
guns                 fired = 320      hit = 0       kills = 0       object hits = 0
```

Figure 9: Full stats report for a player, page 2.

-full stats name <player name>

This option allows you to get a full stats report for a *currently connected* player by specifying that player's name. Figure 8 and Figure 9 show what a full stats report looks like.

-stats mode

This option cycles through the available stats types. As of Slmod v7.0, there are only two stats types: "server" and "mission". The main stats menu will display at the top what stats type you are currently viewing. No matter if you have mission stats enabled or not, the default stats type/mode will be server stats.

5.6- Server Team Hit/Team Kill/PvP Kill Messages

Using the team hit, team kill, and PvP kill data collected by SlmodStats, Slmod now allows server-wide chat messages to be broadcast whenever a player team hits, team kills, or scores a PvP kill. As these messages are created using data from SlmodStats, SlmodStats must be active and recording for these messages to work. As already mentioned, these messages are enabled/disabled through the `enable_team_hit_messages`, `enable_team_kill_messages`, and `enable_pvp_kill_messages` options in the config.lua file.

You might wonder, why have these messages in the first place when the game already outputs them? The answer is- the game does NOT always output kill messages. First of all, in a default DCS multiplayer kill message, it does not tell whether or not the kill was friendly fire.

Secondly, many servers turn off event callbacks in the network.cfg file, disabling all event messages for clients. Having these Slmod kill messages allows these servers to still display a friendly fire message, but not display a "<player> killed <unit>" message when the killed unit was an actual enemy.

Finally, many times the default kill messages fail, especially in PvP. SlmodStats kill messages should be a lot more reliable than the default kill messages. As an example, consider this: if a player is hit by another player's weapons and is fatally injured, should he disconnect from the server before he crashes, no default DCS multiplayer kill message will appear! However, if the Slmod PvP kill messages are enabled, when the player disconnects, the Slmod PvP kill message will appear.

6- Miscellaneous Features

6.1- Chat Log

If enabled with the `chat_log` option in the config.lua file, Slmod will save a chat log to Saved Games/DCS/Slmod/Chat Logs/. Each chat log file is date and time stamped, and contains all chat messages sent throughout the entire hosting session, even ones that were Slmod commands (so be careful when sharing your chat logs if your server has password registration of server admins enabled!). Additionally, there are two additional options that modify the chat logs- if the `log_team_hits` option is enabled, then team-hits will be logged in the chat log, and if the `log_team_kills` option is enabled, then team-kills will also be logged in the chat log.

Stay tuned, upgrades to the chat logging feature are planned in future updates of Slmod!

6.2- Help Menu

Slmod now contains a help menu that gives the active menus, how many items are on them, and how to access them. Access the help menu by saying "-help" in chat.

6.3- Server Message of the Day (MOTD)

Slmod now contains a server Message of the Day (MOTD) system. The MOTD message is sent to clients when they join a slot, and can be enabled and disabled using the `MOTD_enabled` option in the config.lua file. The MOTD contains the following information:

- A welcome line, customizable via the `custom_MOTD` option in the config.lua file;
- Slmod version and chat command for the Slmod Help Menu;
- Whether SlmodStats is enabled or disabled;
- Whether SlmodAutoAdmin is enabled for autokick/autoban, and what offenses will get you kicked/banned;
- Keystroke for the coordinate converter menu;
- (Optional) keystrokes for the Parallel Options Menu and Parallel Tasking Menu (enable or disable this with `MOTD_show_POS_and_PTS);`
- (If applicable) keystroke to access the Admin Menu;
- Default keystroke for multiplayer chat in the module the player is currently playing.

As of Slmod v 7.0, the MOTD is delivered when the player selects a slot, not when they actually take control of an aircraft. By default, the MOTD is sent to the player as a series of chat messages. While there is a config.lua file to change the MOTD message into trigger text, DCS 1.2.4 made it impossible to send a trigger text message to a player unless the player actually has an aircraft. Thus for now, make sure to leave the MOTD display mode as 'chat'.

6.4- Pause on Empty

By setting the `pause_when_empty` option to true in the config.lua file, you can make the server stay paused when clients are not connected. When clients do connect, the server will unpause. As a server admin, you may temporarily override this feature and assume manual pause control by using the "-admin override pause" command.

As of Slmod v7.0, the server becomes unpaused the moment clients become connected. At some point in the future, this will probably be changed so that the server will stay paused until a client actually joins a slot.

6.5- Export World Objects

By setting `export_world_objs` to true in the config.lua file, Slmod will create a file containing all active units in Saved Games/DCS/Slmod/activeUnits.txt. The data takes the form of a serialized Lua table.

6.6- Export Events

By setting `events_output` to true in the config.lua file, Slmod will create a file, updated in real time, of all game events. This file's directory and name is Saved Games/DCS/Slmod/slmodEvents.txt. The event system used is the same one used by the debrief.lua module; however, Slmod may insert some additional information into the events in some cases. The file is a serialized Lua table.

7- Slmod Coordinate Conversion Utility

Slmod v6.0 introduced the Slmod coordinate conversion utility. The Slmod coordinate conversion utility allows quick conversion between the major coordinate systems used in game: Lat/Lon, MGRS, bearing and range from aircraft (BR), and bearing and range from bullseye. It was included to help the various aircraft of DCS world work together. For example, the Su-25T, lacking sophisticated navigational and targeting avionics will be able to take a Lat/Lon coordinate given to it by a ME trigger or multiplayer client and convert it to a bearing and range from their aircraft.

The help "menu" of the coordinate conversion utility can be seen by typing
"-conv" into chat. An extended help menu can be seen by typing "-conv help" into chat.

The Slmod coordinate conversion utility uses the object-oriented SlmodMenu system to be case-insensitive, and even to a certain extent, spelling-insensitive.

7.1- How to Use the Coordinate Converter

To use the Slmod coordinate conversion utility, simply say in chat:
"-conv" + <required for BR and Bulls, optional for LL/MGRS *coordinate type*> + <*coordinate*> + <optional "to"> + <*coordinate type*>". Your chat message will be suppressed (only you will see it), and the converted coordinate will be sent privately to you.

Accepted *coordinate type* values are:
"LL" – latitude, longitude
"MG" – military grid reference system
"BR" – bearing and range from your aircraft
"BUL" - bearing and range from bullseye

Accepted *coordinate* formats are as follows:

Lat/Lon:
<optional "LL" > + <N/S> + <degrees coordinate> + <E/W> + <degrees coordinate>
<optional "LL" > + <degrees coordinate> + <N/S> + <degrees coordinate> + <E/W>
<optional "LL"> + "<degrees coordinate>+ <N/S> + <E/W> + <degrees coordinate>"

<degrees coordinate> can be formatted in the following ways:

Degrees:
Example:
42.21143

Degrees, minutes:
Example:
42 12.686'

Degrees, minutes, seconds:

Example:
42 12’ 41.1”

If no ‘ or “ is given to designate minute or seconds, then it is assumed it was supposed to be where spaces were placed. Pay attention to how the coordinate converter interprets your coordinates (it will tell you how it interpreted them).

The coordinate converter will only make limited assumptions as to what you mean by a coordinate. If you mess up badly enough, such as:

N 42 34’212”, E 039 42’553”

then the coordinate converter will **not** know how to interpret that. The “coordinate” above is not a valid coordinate- you cannot have more than 60 seconds! See this website:
http://zonalandeducation.com/mmts/trigonometryRealms/degMinSec/degMinSec.htm
if you don’t understand why.

MGRS:
<optional “MG” + space> + <UTM Grid zone> + <required space> + <MGRS Digraph> + <optional space> + <optional < Easting> + <optional space> + < Northing>>

The only required space is after the UTM grid zone. The accuracy of the easting and northing must be the same, and must be between zero and five digits each.

Some examples:
38T KM 45521 55321
37T EJ443112
38T NM
38T LM 44 53

BR:
<required “BR”> + <heading in degrees> + <required “for”> + <range in nm>

Some examples:
br 335 for 75
Br 6 For 23
BR 97 FoR 51

Bullseye:
<required “bul”> + <heading in degrees> + <required “for”> + <range in nm>

Some examples:
bullseye 060 for 37
bull 141 FOR 78
BuLlS 15 FoR 132

Full examples:

Bringing together the information presented above, the following are full chat messages that would work with the coordinate conversion utility.

<u>Lat/Lon examples:</u>
"-conv 42 34N 44 15E to mgrs"
"42 34N 44 15E" will be interpreted as 42 34'N, 44 15'E, and converted to MGRS.

"-CONVERT 42 14.914'N 040 37.379'E To Bul"
"42 14.914'N 040 37.379'E" will be interpreted as 42 14.914'N, 40 37.379'E, and converted to bearing and range from bullseye.

"-conv LL N 43 13 39 E 038 55 31 MG"
"N 43 13 39 E 038 55 31" will be interpreted as 43 13' 39"N, 038 55' 31"E, and converted to MGRS. Note in this example the use of the optional coordinate type "LL" in front of the Lat/Lon coordinates. Also, note that the coordinate converter is able to guess that the desired coordinate type to convert to is "MGRS", despite only "MG" being entered into the chat message. Optional "to" is left off.

<u>MGRS examples:</u>

"-convert 38T KM 58382 77418 BR"
Converts the MGRS coordinate to bearing and range from your aircraft. Optional "to" is left off.

"-CONVRET 37T EJ81 tO ll"
Converts the MGRS coordinate 37T EJ81 to lat/lon. Note that after the first four letters of "convert", the spelling does not matter. In this case, "convert" has been misspelled, and the command still works!

<u>Bullseye examples:</u>
"-conv Bul 42 for 163 LL"
Converts bullseye 042 for 163 to lat/lon. Optional "to" is left off.

"-convert bulls 292 for 79 to BR"
Converts bullseye 292 for 79 to bearing and range from your aircraft.

<u>Bearing/Range examples:</u>
"-conv Br 33 for 45 MGRS"
Converts bearing/range 33 for 45 to MGRS. Optional "to" is left off.

"-CoNvsfsafascsadfasdf br 164 FOr 75 tO Ll"
Converts bearing/range 164 for 75 to lat/lon. Again, this example illustrates that only the first four letters count in "convert", and that the commands are not case-sensitive.

8- Using Slmod Mission Scripting Functions

8.1- Creating Unit Tables

Many Slmod functions require tables of unit names. In Slmodbeta v1 to Slmodbeta v5.3, all unit names had to manually be entered into these tables by hand. This became very tedious when a large number of units names was required.

To alleviate this problem, the following unit table "short-cut" commands can be used:

Character sequence + name commands:
"[-u]<unit name>" - subtract this unit from the table
"[g]<group name>" - add this group's units to the table
"[-g]<group name>" - subtract this group's units from the table
"[c]<country name>" - add this country's units to the table
"[-c]<country name>" - subtract this country's units from the table

Stand-alone identifiers
"[all]" – add all units to the table
"[-all]" – remove all units from the table
"[blue]" - add all blue coalition units to the table
"[-blue]" - subtract all blue coalition units from the table
"[red]" - add all red coalition units to the table
"[-red]" - subtract all red coalition units from the table

Compound identifiers:
"[c][helicopter]<country name>" - add all of this country's helicopters to the table
"[-c][helicopter]<country name>" - subtract all of this country's helicopters from the table
"[c][plane]<country name>" - add all of this country's planes to the table
"[-c][plane]<country name>" - subtract all of this country's planes from the table
"[c][ship]<country name>" - add all of this country's ships to the table
"[-c][ship]<country name>" - subtract all of this country's ships from the table
"[c][vehicle]<country name>" - add all of this country's vehicles to the table
"[-c][vehicle]<country name>" - subtract all of this country's vehicles from the table

"[all][helicopter]" - add all helicopters to the table
"[-all][helicopter]" - subtract all helicopters from the table
"[all][plane]" - add all planes to the table
"[-all][plane]" - subtract all planes from the table
"[all][ship]" - add all ships to the table
"[-all][ship]" - subtract all ships from the table
"[all][vehicle]" - add all vehicles to the table
"[-all][vehicle]" - subtract all vehicles from the table

"[blue][helicopter]" - add all blue coalition helicopters to the table
"[-blue][helicopter]" - subtract all blue coalition helicopters from the table
"[blue][plane]" - add all blue coalition planes to the table

"[-blue][plane]" - subtract all blue coalition planes from the table
"[blue][ship]" - add all blue coalition ships to the table
"[-blue][ship]" - subtract all blue coalition ships from the table
"[blue][vehicle]" - add all blue coalition vehicles to the table
"[-blue][vehicle]" - subtract all blue coalition vehicles from the table

"[red][helicopter]" - add all red coalition helicopters to the table
"[-red][helicopter]" - subtract all red coalition helicopters from the table
"[red][plane]" - add all red coalition planes to the table
"[-red][plane]" - subtract all red coalition planes from the table
"[red][ship]" - add all red coalition ships to the table
"[-red][ship]" - subtract all red coalition ships from the table
"[red][vehicle]" - add all red coalition vehicles to the table
"[-red][vehicle]" - subtract all red coalition vehicles from the table

Country names for the "[c]<country name>" and "[-c]<country name>" short-cuts:

Turkey
Norway
The Netherlands
Spain
UK
Denmark
USA
Georgia
Germany
Belgium
Canada
France
Israel
Ukraine
Russia
South Osetia
Abkhazia
Italy

Do NOT use a '[u]' notation for single units. Single units are referenced the same way as before: simply input their names as strings.

These unit tables are evaluated in order, and you cannot subtract a unit from a table before it is added. For example,

```
{'[blue]', '[-c]Georgia'}
```

will evaluate to all of blue coalition except those units owned by the country named “Georgia”; however:

```
{'[-c]Georgia', '[blue]'}
```

will evaluate to all of the units in blue coalition, because the addition of all units owned by blue coalition occurred AFTER the subtraction of all units owned by Georgia.

More examples:

```
{'[blue]', '[-c]Georgia', '[-g]Hawg 1'}
--[[Slmod will turn this table into a unit table holding all the units
of blue coalition, minus the units belonging to Georgia, and minus
the units in the group named "Hawg 1".]]

{'[g]arty1', '[-u]arty1_AD', '[g]arty2', '[-u]arty2_AD', 'Shark 11' }
--[[Evaluates to a table containing all of the units in the groups
named
"arty1" and "arty2", MINUS the units named "arty1_AD" and "arty2_AD,
PLUS the unit named "Shark 11".]]
```

Full example:
A good example is how previously, it was very possible to make a fratricide-detecting script with the slmod.units_hitting function. However, all the units that might be detected as being hit by friendlies would need to be added in a massive table of unit names, which could contain hundreds of entries. The new unit table short-cuts MASSIVELY simplify such a task:

```
slmod.units_hitting({'Hawg 11'}, {'[blue]', '[-u]Hawg 11'}, 101, -1,
'BLUE ON BLUE! BLUE ON BLUE!', '', 5, 'text')
--[[Detects any time Hawg 11 hits any unit on blue team (besides
itself) with any weapon.  Displays the text
"BLUE ON BLUE! BLUE ON BLUE!" for 5 seconds and sets flag 101 when
this event occurs.]]
```

8.2- Slmod alternate input variable specification method

Slmod now supports an alternate, and improved, method of specifying input variables in function calls. In previous versions of Slmod, the only way to specify variables was to input all variables into the function in a specific order, such as seen in the units_hitting example directly above. Some functions have large numbers of optional variables (such as units_hitting), and with this older method, even optional variables you didn't care about had to be specified in some manner; for example, stopflag is commonly set to -1 if you do not want to use it.

Slmod v6.0+ supports a new way to specify variables. In the new method, all variables are put inside a single Lua table. Each variable is set to a table key identical to the variable names specified in this guide. Further, the Lua programming language contains special support for this method of specifying variables for functions, by allowing you to drop the parentheses from the function call when the function is only being passed a single table as its input variables. Basically:

OLD METHOD:

function(value1, value2, value3 **)**

NEW METHOD:
function { variable1name = value1, variable2name = value2, variable3name = value3 **}**
Note the use of curly brackets (the Lua table constructor operators) instead of parenthesis!

For example, the previous friendly-fire detecting slmod.units_hitting function call goes from this:

```
slmod.units_hitting({'Hawg 11'}, {'[blue]', '[-u]Hawg 11'}, 101, -1,
'BLUE ON BLUE! BLUE ON BLUE!', '', 5, 'text')
```
OLD METHOD

to this:

```
slmod.units_hitting {init_units = {'Hawg 11'}, tgt_units = {'[blue]',
'[-u]Hawg 11'}, flag  = 101, text = 'BLUE ON BLUE! BLUE ON BLUE!',
display_time = 5, display_mode = 'text'}
```
NEW METHOD

With the new method, your code becomes easier to read, the "order" that you specify variables no longer matters, and you do not have to specify optional variables you do not want to use anymore just to "get to" an optional variable you DO want to specify.

And finally, of course, the table of input variables can be declared in any way you would normally declare a table, for example, this is one way:

```
do
    local friendly_fire_tbl = {
        text = 'BLUE ON BLUE! BLUE ON BLUE!',
        display_mode = 'text',
        display_time = 5,
        tgt_units = {'[blue]', '[-u]Hawg 11'},
        init_units = {'Hawg 11'},
        flag  = 101,
        }
    slmod.units_hitting(friendly_fire_tbl)
end
```

The above example is a bit more readable than the previous ones, wouldn't you agree? Of course, I "scrambled" the "order" of the table a bit in the above example, just to show that it doesn't matter (in fact, there is no "order" to the table at all, other than there are table "keys" (such as "text" and "display_mode").

Of course, the old way to specify input variables for functions remains, but some new Slmod functions introduced in the future could potentially have more and more optional input variables, to the point where the old method becomes even more cumbersome and impractical.

9- Slmod Function Listing

9.1- Key:

Bold: function name you have to use in mission scripting environment (i.e., in a group's triggered actions)
Blue bold text: the type of the required variable (i.e., "string", "number", "table", etc.)
Blue Italics: the name of the required variable (used for reference within Slmod documentation only)
Green bold text: the type of the optional variable (i.e., "string", "number", "table", etc.)
Green Italics: the name of the optional variable (used for reference within Slmod documentation only)

9.2- Message input and output functions

9.2.1- slmod.chat_cmd

slmod.chat_cmd (string *cmd_text*, **number** *flag*, **number** *stopflag*, **string** *coa* **)**
Quick Description:
The function sets a flag when a specific line of text is entered as a multiplayer chat message. The function only needs to be told to run once, after which, it will continuously evaluate the chat history until either *stopflag* becomes true or the mission ends.
Quick notes:
If *stopflag* is not specified, then the function can never be stopped from executing its search for the chat message *cmd_text* (which will hardly ever be a problem).
coa has allowed values of `'red', 'blue', or 'all'.`
If *coa* is not specified, it defaults to `'all'.`

When you want to specify *coa* but do not want to specify any *stopflag* whatsoever, a good value to use for *stopflag* is -1 (negative one).
Examples:

```
slmod.chat_cmd('-Axeman resume patrol', 1101, -1, 'blue')
--[[After this script is first run, if anyone on blue team says
"-Axeman resume patrol" in chat, the flag 1101 will be set to true.]]-
-
```

9.2.2- slmod.msg_out

slmod.msg_out(string *text*, **number** *display_time*, **string** *display_mode*, **string** *coa* **)**
Quick Description:
This function displays the text *text* to players on one or both coalitions as either a chat message and/or trigger text (as determined by *display_mode* and *coa*), for a duration of time determined by *display_time*. This function is the basic message output function of Slmod. It is capable of outputting messages in chat, trigger text, or a combination of the two (as set by the *display_mode* optional input variable).
Quick notes:
display_mode has allowed values of `'echo', 'chat', 'text', or 'both'.`
The meaning of the display modes are as follows:

`'chat'`: Output message only as a chat message. Keep in mind that chat messages have a limited lifetime on the screen. To cope with this, *display_time* will be divided by the number of lines in the message (determined by counting new line characters). The result will then be divided by six, and then will be rounded down to the nearest whole number not less than 1. This final result will determine the number of times each line flashes on the screen. The time interval between the display of each line will be the result of *display_time* divided by the number of lines, rounded downward to the nearest whole number not less than one. So with *display_time* set to 40, and a 4 line message, then:
Each line will be displayed: 40/4/6 = 1.667, rounded down = **1 time.**
The interval between showing each line will be: 40/4 = 10, rounded down = **10 seconds.**

`'text'`: Display the message only as "white trigger text" (such as what the "message to all" trigger action shows), for the duration specified by *display_time*.

`'echo'`: Display the message as "white trigger text" for the duration specified by *display_time*, and also send (or "echo") each line of the message as a chat message, each chat message sent one second apart. This is the default setting for msg_out and any other msg_out-based function. This gives the benefits of both types of messages- the persistence of "white trigger text" on your screen, and the ability to recall the message by pressing the "tab" key and showing your chat history.

`'both'`: The combination of both `'chat'` and `'text'` display modes.

If *display_mode* is not specified, it defaults to `'echo'`.
coa has allowed values of `'red'`, `'blue'`, or `'all'`.
If *coa* is not specified, it defaults to `'all'`.

Examples:
```
slmod.msg_out('Hawg1, destroy enemy bunker at 38T KM384323', 40)
-- letting display_mode default to 'echo' and coalition to 'all'


slmod.msg_out('Hawg1, destroy enemy bunker at 38T KM384323', 40,
'chat', 'blue')
-- Only show message in chat, and only show it to blue team
```

9.2.3- slmod.msg_out_w_unit

slmod.msg_out_w_unit(string *text*, **string** *unit_name*, **number** *display_time*, **string** *display_mode*, **string** *coa***)**

Quick Description:
This function is identical to msg_out, except that it is capable of putting the multiplayer name of a unit within its message, dependent on where the magic characters "%s" are placed.

Quick notes:
display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.
If *display_mode* is not specified, it defaults to `'echo'`.
coa has allowed values of `'red'`, `'blue'`, or `'all'`.
If *coa* is not specified, it defaults to `'all'`.

Example:

```
slmod.msg_out_w_unit('You have hit friendlies three times, %s.  It\'s
time for you to die.', 'Hawg31', 10, 'text')

--[[ Example message for the unit with the name of "Hawg31".
The function automatically changes the "%s" in the string into to the
multiplayer name of "Hawg31".  Message will display for 10 seconds,
as a trigger-text type message.  The optional variable coa is not
specified, so it will default to "all".  Note the use of the \ to
escape the single quote inside the string.]]
```

9.2.4- slmod.CA_chatmsg

slmod.CA_chatmsg(string *text*, string *coa*)

Quick Description:

This function outputs *text* as a chat message to all clients playing Combined Arms on the coalition specified by *coa*. The message is not seen by anyone else.

Quick notes:

coa has allowed values of `'red'`, `'blue'`, or `'all'`.

If *coa* is not specified, it defaults to `'all'`.

9.2.5- slmod.msg_MGRS

slmod.msg_MGRS(string *text*, table *units*, number *numdigits*, number *display_time*, string *display_mode*, string *coa*)

Quick Description:

This function is capable of outputting the message *text*, followed by the average MGRS coordinates of a group of units. Additionally, if the sequence of characters “%s” (without the quotes) is placed in *text*, then the coordinates will be inserted there instead of at the end of *text*.

Quick notes:

units is a table of all the names of the units you want to get the average MGRS of. All table entries must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

numdigits must have a value of either 2, 4, 6, 8, or 10.

display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, it defaults to `'echo'`.

coa has allowed values of `'red'`, `'blue'`, or `'all'`.

If *coa* is not specified, it defaults to `'all'`.

Example:

```
slmod.msg_MGRS('Hawgs, engage enemy armor at: ', {'tanks1_1',
'tanks1_2', 'tanks1_3', 'tanks1_4', 'tanks1_5', 'tanks1_6'}, 6, 40,
'text', 'blue')
--[[Example function call with the optional variables "display_mode"
and
"coa" specified.  Gives a 6 digit MGRS grid (50 meter accuracy).]]--
```

9.2.6- slmod.msg_LL

slmod.msg_LL(string *text*, **table** *units*, **number** *precision*, **number** *display_time*, **string** *display_mode*, **string** *coa* **)**

Quick Description:

This function is capable of outputting the message *text*, followed by the average latitude and longitude coordinates of a group of units. Alternatively, if the sequence of characters "%s" (without the quotes) is placed in *text*, then the coordinates will be inserted there instead of at the end of *text*.

Quick notes:

units is a table of all the names of the units you want to get the average lat/long of. All table entries must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

precision is the number of places after the minutes decimal place you want to round to. Allowed values are -1, 0, 1, 2, 3.

display_mode has allowed values of `'echo', 'chat', 'text', or 'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, it defaults to `'echo'`.

coa has allowed values of `'red', 'blue', or 'all'`.

If *coa* is not specified, it defaults to `'all'`.

Example:

```
slmod.msg_LL('Sharks, engage enemy armor at: ', {'M1_1_1', 'M1_1_2',
'M1_1_3', 'M1_1_4'}, 2, 40, 'echo', 'red')
--[[Example function call with the optional variables "display_mode"
and
"coa" specified.  Gives a set of lat/long coordinates rounded to the
nearest 1/100 of a minute.]]--
```

9.2.7- slmod.msg_leading

slmod.msg_leading(string *coordtype*, **string** *text*, **table** *units*, **string** *direction*, **number** *radius*, **number** *precision*, **number** *display_time*, **string** *display_mode*, **string** *coa* **)**

Quick Description:

Displays *text* followed by the lat long or MGRS coordinates of the units within a radius *radius* of the unit in *units* most in the direction specified by *direction*. Alternatively, if the sequence of characters "%s" (without the quotes) is placed in *text*, then the coordinates will be inserted there instead of at the end of *text*.

Quick notes:

coordtype must be either 'LL' or 'MGRS'.

units is a table of all the names of the units. All table entries must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

precision is the number of places after the minutes decimal place you want to round to if *coordtype* is 'LL'. If *coordtype* is 'MGRS', then this is the number of digits in grid.

Allowed values are -1, 0, 1, 2, or 3 for *coordtype* 'LL'

Allowed values are 2, 4, 6, 8, or 10 for *coordtype* 'MGRS' *direction* has the allowed values of `'N', 'E', 'W', or 'S'.`

display_mode has allowed values of `'echo', 'chat', 'text', or 'both'.` See the description for the **slmod.msg_out** function for the details of how these four different display modes work.
If *display_mode* is not specified, it defaults to `'echo'.`
coa has allowed values of `'red', 'blue', or 'all'.`
If *coa* is not specified, it defaults to `'all'.`

Example:
```
slmod.msg_leading('LL', 'Sharks, engage US forces attacking our
eastern flank. The closest US armor to our flank is near: ',
{'M1_1_1', 'M1_1_2', 'M1_1_3', 'M1_1_4', 'M1_2_1', 'M1_2_2', 'M1_2_3',
'M1_2_4', 'M1_3_1', 'M1_3_2', 'M1_3_3', 'M1_3_4'}, 'W', 2000, 1, 40,
'chat', 'red')
--Example of the msg_leading function both optional variables
specified
```

9.3- Parallel Tasking System Functions

The parallel tasking system allows you to give one or more tasks to players. The tasking list can be viewed by typing in "-stl", "-STL", or "SHOW TASK LIST" as a multiplayer chat message. Individual tasks can be viewed by typing in "-st#", "-ST#", or "TASK # GET UPDATE" as a multiplayer chat message, where "#" is the task number. Task numbers are automatically assigned based on the order in which they are added to the tasking list. Tasks must be removed via a slmod.remove_task, slmod.remove_red_task, or slmod.remove_blue_task function call once the task is completed. Each coalition has its own task list.

9.3.1- slmod.add_task

slmod.add_task(number ***id*****, string** ***task_type*****, string** ***description*****, string** ***text*****, table** ***units*****, number** ***precision*****, string** ***direction*****, number** ***radius*** **)**

Quick Description:
If *id* is unique, then the function adds a task to the parallel tasking system for both coalitions. If *id* is the same as the *id* of a pre-existing task, this function will modify the pre-existing task. *description* is the description of the task within the parallel tasking system task list. *text* is the task text. If the task gives coordinates, and the sequence of characters "%s" (without the quotes) is placed in *text*, then the coordinates will be inserted there instead of at the end of *text*.
Quick Notes:
id is a unique, identifying number for each task, NOT THE TASK NUMBER.
task_type has allowed values of `'msg_out', 'msg_LL', 'msg_MGRS', 'msg_leading_LL', or 'msg_leading_MGRS'.`
If *task_type* is `'msg_out'`, then no optional variables should be specified. If *task_type* is `'msg_LL'`, or `'msg_MGRS'` then the optional variables *units* and *precision* must be specified. If the task type is `'msg_leading_LL', or 'msg_leading_MGRS'`, then all optional variables must be specified.
units is a table of all the names of the units. All table entries must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

When *task_type* is `'msg_LL'` or `'msg_leading_LL'` then allowed values of *precision* are -1, 0, 1, 2, or 3. When *task_type* is `'msg_MGRS'` or `'msg_leading_MGRS'` then allowed values of *precision* are 2, 4, 6, 8, or 10.

direction has the allowed values of `'N', 'E', 'W', or 'S'.`

radius will default to 3000 (meters) if not specified for a leading type task.

Examples:
```
slmod.add_task(101, 'msg_out', 'Destroy enemy HQ', 'Destroy enemy HQ
at: 38T KM509078')
-- Adding a msg_out type task to the task list

slmod.add_task(1032, 'msg_MGRS', 'Destroy enemy armor', 'We have
spotted enemy armor near: ', {'tanks1_1', 'tanks1_2', 'tanks1_3',
'tanks1_4', 'tanks1_5', 'tanks1_6', 'tanks1_7', 'tanks1_8'}, 6)
-- An example of msg_MGRS task

slmod.add_task(112, 'msg_leading_LL', 'Destroy advancing enemy armor',
'Sharks, destroy the attacking enemy armor! The nearest enemy armor is
at: ', {'tanks1_1', 'tanks1_2', 'tanks1_3', 'tanks1_4', 'tanks1_5',
'tanks2_1', 'tanks2_2', 'tanks2_3', 'tanks2_4', 'tanks2_5',
'tanks3_1', 'tanks3_2', 'tanks3_3', 'tanks3_4', 'tanks3_5',
'tanks4_1', 'tanks4_2', 'tanks4_3', 'tanks4_4', 'tanks4_5',
'tanks5_1', 'tanks5_2', 'tanks5_3', 'tanks5_4', 'tanks5_5',
'tanks6_1', 'tanks6_2', 'tanks6_3', 'tanks6_4', 'tanks6_5'}, 1, 'N',
2000)
-- An example of msg_leading_LL task
```

9.3.2- slmod.add_red_task

slmod.add_red_task(number *id*, **string** *task_type*, **string** *description*, **string** *text*, **table** *units*, **number** *precision*, **string** *direction*, **number** *radius* **)**
Quick Description:
This function is exactly the same as **slmod.add_task**, except that it only adds a task to the red coalition.

9.3.3- slmod.add_blue_task

slmod.add_blue_task(number *id*, **string** *task_type*, **string** *description*, **string** *text*, **table** *units*, **number** *precision*, **string** *direction*, **number** *radius* **)**
Quick Description:
This function is exactly the same as **slmod.add_task**, except that it only adds a task to the blue coalition.

9.3.4- slmod.remove_task

slmod.remove_task (number *id* **)**
Quick Description:
This function removes the task with the id of *id* from both red and blue coalition task lists.
Example:
```
slmod.remove_task(101)
```

```
-- removes task 101 from both red and blue task lists.
```

9.3.5- slmod.remove_red_task

slmod.remove_red_task (number *id*)

Quick Description:

This function removes the task with the id of *id* from the red coalition's task list.

9.3.6- slmod.remove_blue_task

slmod.remove_blue_task (number *id*)

Quick Description:

This function removes the task with the id of *id* from the blue coalition's task list.

9.3.7- slmod.show_task

slmod.show_task(number *id*, number *display_time*, string *display_mode*)

Quick Description:

You can manually trigger the showing of task with id *id* with this function. It shows each coalition their task with id *id*.

Quick Notes:

If *display_time* is not specified, it defaults to 40.

display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, it defaults to `'echo'`.

9.3.8- slmod.show_red_task

slmod.show_red_task(number *id*, number *display_time*, string *display_mode*)

Quick Description:

This function is exactly the same as slmod.show_task, except that it only shows the red coalition task with id *id* to red coalition members.

9.3.9- slmod.show_blue_task

slmod.show_blue_task(number *id*, number *display_time*, string *display_mode*)

Quick Description:

This function is exactly the same as slmod.show_task, except that it only shows the blue coalition task with id *id* to blue coalition members.

9.3.10- slmod.show_task_list

slmod.show_task_list()

Quick Description:

This function allows you to manually trigger the showing of the task list for both coalitions.

9.3.11- slmod.show_red_task_list

slmod.show_red_task_list()

Quick Description:

This function allows you to manually trigger the showing of the task list for red coalition only.

9.3.12- slmod.show_blue_task_list

slmod.show_blue_task_list()

Quick Description:

This function allows you to manually trigger the showing of the task list for blue coalition only.

9.3.13- slmod.set_PTS_list_output

slmod.set_PTS_list_output(number *display_time*, string *display_mode*, string *coa*)

Quick Description:

Changes the display settings of the task list for the coalition specified by *coa*. The default settings for the task lists are a *display_time* of 40, and a *display_mode* of `'echo'`.

Quick Notes:

display_mode has allowed values of `'echo', 'chat', 'text', or 'both'.` See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, then the function only acts to change the *display_time* variable (for both coalitions, since if *display_mode* is not specified, then *coa* can't be, either).

coa has allowed values of `'red', 'blue', or 'all'.`

If *coa* is not specified, it defaults to `'all'`- the changes will be applied to both coalition's task lists.

Examples:

```
slmod.set_PTS_list_output(40, 'text')
-- Change the tasking list to display as white trigger text for 40
sec,
-- the change is appled to both coalition's task lists.


slmod.set_PTS_list_output(40, 'chat', 'blue')
-- Set blue's tasking list to display as chat messages only, will
follow
-- the rules set forth for the 'chat' message mode (see
slmod.msg_out).
```

9.3.14- slmod.set_PTS_task_output

slmod.set_PTS_task_output(number *display_time*, string *display_mode*, string *coa*)

Quick Description:

This function follows the same rules as **slmod.set_PTS_list_output**, except that instead of the task list, this function modifies the display settings of the tasking messages themselves. See the description of **slmod.set_PTS_list_output** for more information.

9.4- Parallel Options System Functions

As of DCS World 1.2.3, the F10 “other” radio options menu does not work for multiplayer clients. The slmod.chat_cmd function can be used to replace this missing functionality, but chat commands end up being unique for each mission, and an entered chat message had to exactly match the command text defined by the mission creator. The parallel options system was added to streamline the entry of chat commands by offering a standardized command format and command display system. Each option will set a flag, as set by the mission creator.
To see the list of options available, a player must type “-sol”, “-SOL”, or “SHOW OPTIONS LIST” in chat. To select a particular item on the options list, a player must type in chat “-op#”, “-OP#”, or “OPTION #”, where “#” is the option number on the options list.

9.4.1- slmod.add_option

slmod.add_option(number *id*, string *description*, number *flag*)
Quick Description:
If *id* is unique, then this function adds an option to the parallel options system for both coalitions. If *id* is the same as the *id* of a pre-existing option, this function will modify the pre-existing option instead. *description* is the description of the option on the parallel options system list. *flag* is the flag that will be set when the option is selected.
If the option with *id* already exists, then this function will over-write the old one.
Quick Notes:
id is a unique, identifying number for each option on each coalition’s option list, NOT THE OPTION NUMBER ON THE OPTION LIST. There can be only one option per *id* number per coalition. So, for example, an option with an *id* of 103 could exist for both coalitions simultaneously, or for only one coalition.
Examples:

```
slmod.add_option(1004, 'Tanks hold position', 50)
-- Add an option to both coalition's option lists with the description
-- of "Tanks hold position" and that will set flag 50 when activated.
```

9.4.2- slmod.add_red_option

slmod.add_red_option(number *id*, string *description*, number *flag*)
Quick Description:
This function is exactly the same as **slmod.add_option**, except that it only adds an option to the red coalition’s option list.

9.4.3- slmod.add_blue_option

slmod.add_blue_option(number *id*, string *description*, number *flag*)
Quick Description:
This function is exactly the same as **slmod.add_option**, except that it only adds an option to the blue coalition’s option list.

9.4.4- slmod.remove_option

slmod.remove_option(number *id*)
Quick Description:

This function removes the option with the id of *id* from both red and blue coalition option lists.
<u>Example:</u>
```
slmod.remove_option(1001)
-- removes option with id of 1001 from both red and blue option lists.
```

9.4.5- slmod.remove_red_option
slmod.remove_red_option(number *id* **)**
<u>Quick Description:</u>
This function removes the option with the id of *id* from the red coalition's option list.

9.4.6- slmod.remove_blue_option
slmod.remove_blue_option(number *id* **)**
<u>Quick Description:</u>
This function removes the option with the id of *id* from the blue coalition's option list.

9.4.7- slmod.show_option_list
slmod.show_option_list( )
<u>Quick Description:</u>
If, for some reason, you want a mission trigger to force the display of the option list at some point, use this function. This function shows the option list to both red and blue coalitions.

9.4.8- slmod.show_red_option_list
slmod.show_red_option_list( )
<u>Quick Description:</u>
If, for some reason, you want a mission trigger to force the display of the option list at some point, use this function. This function shows the option list to red coalition only.

9.4.9- slmod.show_blue_option_list
slmod.show_blue_option_list( )
<u>Quick Description:</u>
If, for some reason, you want a mission trigger to force the display of the option list at some point, use this function. This function shows the option list to blue coalition only.

9.4.10- slmod.set_POS_output
slmod.set_POS_output(number *display_time*, **string** *display_mode*, **string** *coa* **)**
<u>Quick Description:</u>
Changes the display settings of the options list for the coalition specified by *coa*. The default settings for the option lists are a *display_time* of 40, and a *display_mode* of `'echo'`.
<u>Quick Notes:</u>
display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, then the function only acts to change the *display_time* variable (for both coalitions, since if *display_mode* is not specified, then *coa* can't be, either).
coa has allowed values of `'red', 'blue', or 'all'`.
If *coa* is not specified, it defaults to `'all'`- the changes will be applied to both coalition's option lists.

Examples:
```
slmod.set_POS_output(5, 'chat')
-- Change the option list display to chat messages for both coalitions

slmod.set_POS_output(40, 'echo', 'red')
-- Restore the option list display settings to default for red
coalition
```

9.5- Events-based functions

9.5.1- slmod.mapobj_destroyed

slmod.mapobj_destroyed(number or table *id*, number *flag*, number *percent*)
Quick Description:
If *id* is a number, then the function sets flag *flag* when the map object with the id of *id* is destroyed.
If *id* is a table (of numbers), then flag *flag* gets set true when one of the map objects whose id is specified in *id* gets destroyed.
If *percent* is specified and *id* is a table (of numbers), then the flag is only set true when the percentage of map objects destroyed, out of the total number of map object ids listed in *id*, exceeds *percent*.
The function only needs to be told to run once, after which, it will run for the rest of the mission.
Quick Notes:
id is either a number or a table of numbers, which represent map object unique id numbers. Get ids for map objects out of debrief.log after destroying them in a test mission.
If *percent* is not specified, it defaults to 0.
Examples:
```
slmod.mapobj_destroyed(11943631, 175)
-- sets flag 175 when map object with id #11943631 is destroyed

slmod.mapobj_destroyed({12243946, 11776432, 7684, 5567754, 112385433,
6643563}, 15, 33)
-- sets flag 15 when two or more (33% or more) of the map objects are
-- destroyed.
```

9.5.2- slmod.mapobj_dead_in_zone

slmod.mapobj_dead_in_zone(string *zone*, number *flag*, number *stopflag*, number *numdead*)
Quick Description:
If *numdead* is not specified or is 0, then the function sets the flag *flag* when a map object has been destroyed within the zone named *zone* (the name of the zone is the same name you gave the zone in the mission editor).
If *numdead* is greater than 0, then the flag gets set true only after the number of map objects destroyed within *zone* exceeds the value of *numdead*.

The function only needs to be told to run once, after which, it will be re-evaluated once every second until either *stopflag* becomes true or the mission ends.

Quick Notes:

When you want to specify *numdead* but do not want to specify any *stopflag* whatsoever, a good value to use for *stopflag* is -1 (negative one).

Examples:

```
slmod.mapobj_dead_in_zone('bridge1', 150)
-- sets flag 150 true when a map object dies within the zone named
"bridge1"

slmod.mapobj_dead_in_zone('enemy airbase', 201, 1000, 8)
--[[ sets flag 201 true when more than 8 map objects are destroyed
within the zone named "enemy airbase".  Setting flag 1000 to true
will cause the function to stop its operation.]]--
```

9.5.3- slmod.units_firing

slmod.units_firing(table *init_units*, **number** *flag*, **number** *stopflag*, **table** *weapons*, **string** *text*, **string** *display_units*, **number** *display_time*, **string** *display_mode*, **string** *coa* **)**

Quick Description:

Sets flag *flag* true when any one of the unit(s) listed in *init_units* fires a weapon. The optional table variable *weapons* allows you to specify the names of the weapons you want to detect (use the same weapons names as you find in the events, which can be viewed in your debrief.log file after a mission is over). The optional variables *text*, *display_units*, *display_time*, *display_mode*, and *coa* allow you to optionally output a message when a weapons launch of the proper type is detected. The function only needs to be told to run once, after which, it will continuously evaluate the mission events until either *stopflag* becomes true or the mission ends.

Quick Notes:

All the table entries in *init_units* must be strings. These are the names of the units as you named them in the mission editor. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

If you never want the function to stop running (it does use relatively little computational overhead), then a good value for *stopflag* is -1.

All the table entries in *weapons* must be strings. These are the names of the weapons as seen in the debrief.log file. To find out the names of the weapons you must use, you have to fire or get an AI to fire the weapon in a mission. Next, open C:\Users\<Your Account Name>\Saved Games\DCS\Logs\debrief.log. Find the weapon firing event, and copy the name of the weapon out of that.

If you don't want to specify a specific weapons type, but you need to include the *weapons* variable so that you can specify optional variables after *weapons*, then make *weapons* this: `{'all'}`

display_units allows you to replace any %s character sequences in *text* with the initiating unit and/or the weapon fired. Allowable values are:

- `''` – an empty string, fill in no %s sequences.
- `'i'` – the first %s in *text* becomes the initiating unit's multiplayer name.
- `'w'` – the first %s in *text* becomes the weapon name.

`'iw'` – the first %s in *text* becomes the initiating unit's multiplayer name, and the second %s becomes the weapon name.
`'wi'` – the first %s in *text* becomes the weapon name, and the second %s becomes the initiating unit's multiplayer name.

display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.
If *display_mode* is not specified, it defaults to `'echo'`.
coa has allowed values of `'red'`, `'blue'`, or `'all'`.
If *coa* is not specified, it defaults to `'all'`.

Examples:

```
slmod.units_firing({'M1_1', 'M1_2', 'M1_3', 'M1_4'}, 1701)
--[[VERY simple example.  Sets flag 1701 to true every time one of
the units named either 'M1_1', 'M1_2', 'M1_3', or 'M1_4' fires
any weapon.]]--


slmod.units_firing({'Buff11'}, 211, -1, {'all'}, 'Buff 11: MISSILE
AWAY!', '', 5, 'chat')
--[[A simple example.  When the unit named "Buff11" fires any weapon,
then the chat message "Buff 11: MISSILE AWAY!" appears, and flag
211 becomes true]]--

slmod.units_firing({'Ford 11', 'Ford 21', 'Ford 31', 'Ford 41',
'Springfield 11', 'Springfield 21', 'Springfield 31'}, 2601, -1,
{"AGM-88 HARM"}, '%s: MAGNUM!', 'i', 5, 'chat')
--[[A more advanced example.  When one of the units named 'Ford 11'
through 'Ford 41' or 'Springfield 11' through 'Springfield 31'
fires an AGM-88, then flag 2601 is set to true and a chat message
appears.  The chat message says which unit fired the weapon,
followed by "MAGNUM!".  The flag can be used with trigger logic
to play a sound file.]]--

slmod.units_firing({'Chevy 1', 'Chevy 2', 'Chevy 3', 'Chevy 4', 'Chevy
5', 'Chevy 6', 'Chevy 7', 'Chevy 8'}, 2501, -1, {'CBU_87', 'CBU_97',
'CBU_103', 'CBU_105', 'GBU_10', 'GBU_12', 'GBU-31', 'GBU-38', 'Mk_82',
'MK_82AIR', 'Mk_84'}, '%s: BOMBS AWAY! (%s)', 'iw', 5, 'chat', 'blue')
--[[An advanced example, ideal for usage on a public A-10C server.
Whenever any of the units named "Chevy 1" through "Chevy 8" drop
any kind of bomb, flag 2501 becomes true (so that, for instance,
a sound file can be played), and a chat message appears.  The chat
message first gives the multiplayer name of the guy who dropped the
bomb, then says "BOMBS AWAY!", then in parenthesis says what kind of
bomb was dropped.  For instance, the chat message may be:

16th Speed: BOMBS AWAY! (GBU_12)
The chat message only appears to those on the blue coalition.
]]
```

9.5.4- slmod.units_hitting

slmod.units_hitting(table *init_units*, **table** *tgt_units*, **number** *flag*, **number** *stopflag*, **string** *text*, **string** *display_units*, **number** *display_time*, **string** *display_mode*, **string** *coa* **)**

Quick Description:

Sets flag *flag* whenever one of the units listed in *init_units* hits one of the units listed in *tgt_units*. The optional variables *text*, *display_units*, *display_time*, *display_mode*, and *coa* allow you to optionally output a message of some sort when the proper "hit"-type event occurs. The function only needs to be told to run once, after which, it will continuously evaluate the mission events until either *stopflag* becomes true or the mission ends.

Quick Notes:

All the table entries in *init_units* must be strings. These are the names of the units as you named them in the mission editor. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

All the table entries in *tgt_units* must be strings. These are the names of the units as you named them in the mission editor. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

If you never want the function to stop running (it does use relatively little computational overhead), then a good value for *stopflag* is -1.

display_units allows you to replace any %s character sequences in *text* with the initiating unit and/or the unit that was hit (the "target unit"). Allowable values are:

`''` – an empty string, fill in no %s sequences.

`'i'` – the first %s in *text* becomes the initiating unit's multiplayer name.

`'t'` – the first %s in *text* becomes the target unit's multiplayer name.

`'it'` – the first %s in *text* becomes the initiating unit's multiplayer name, and the second %s becomes the target unit's multiplayer name.

`'ti'` – the first %s in *text* becomes the target unit's multiplayer name, and the second %s becomes the initiating unit's multiplayer name.

display_mode has allowed values of `'echo'`, `'chat'`, `'text'`, or `'both'`. See the description for the **slmod.msg_out** function for the details of how these four different display modes work.

If *display_mode* is not specified, it defaults to `'echo'`.

coa has allowed values of `'red'`, `'blue'`, or `'all'`.

If *coa* is not specified, it defaults to `'all'`.

Examples:

```
slmod.units_hitting({'Zu-23_1'}, {'C-130_1'}, 79)
--[[A simple example.  When the unit named "C-130_1" gets hit by the
unit named "Zu-23_1", then flag 79 gets set to true.]]

slmod.units_hitting({'Chevy 1'}, {'Chevy 2', 'Chevy 3', 'Chevy 4',
'Chevy 5', 'Chevy 6', 'Chevy 7', 'Chevy 8'}, 2711, -1, '%s HIT %s!
STOP TEAM KILLING OR ELSE!!!', 'it', 5, 'chat')
```

```
--[[A more advanced example.  Flag 2711 is set to true when the unit
named "Chevy 1" team-hits another human player (assuming all the
"humanable" units are comprised of "Chevy 1" through "Chevy 8")]]


slmod.units_hitting({'Chevy 1', 'Chevy 2', 'Chevy 3', 'Chevy 4'},
{'uzi1_1', 'uzi1_2', 'uzi1_3', 'seals1_1', 'seals1_2', 'seals1_3',
'seals1_4', 'seals1_5', 'seals1_6', 'seals2_1', 'seals2_2',
'seals2_3', 'seals2_4', 'seals2_5', 'seals2_6' }, 2701, -1, '%s, YOUR
FIRING ON FRIENDLIES!!!', 'i', 5, 'chat')
--[[A more advanced example, a "blue on blue humiliation script".
Any time one of the units listed in the second table is hit by one
of the units in the first table (we'll assume that the first table
in this example is all the names of the "humanable" A-10 units),
then set flag 2701 (so, for example, a sound file like "BLUE ON
BLUE! BLUE ON BLUE!" can be played), and output a chat message
telling everyone who just fired on friendly units.]]
```

9.5.5- slmod.units_killed_by

slmod.units_killed_by(table *dead_units*, **table** *killer_units*, **number** *flag*, **number** *stopflag*, **number** *last_to_hit*, **number** *time_limit* **)**

Description:

This function sets flag *flag* when one of the units listed in *dead_units* is killed by one of the units listed in *killer_units*, within the restrictions set by the optional variables *last_to_hit* and *time_limit*. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

last_to_hit sets a limit on which units that were last to hit a unit before it died are considered killers of that unit. For example, if *last_to_hit* is set to 1, the only the last unit to hit the unit that died is considered a killer of that unit. However, if *last_to_hit* is set to 3, then the last three units to hit a unit before it died are considered killers of that unit. If *last_to_hit* is not specified, then all hit events are considered (effectively, *last_to_hit* becomes infinity).

time_limit sets a limit on the amount of time that can pass between a hit event, and the unit dying, for the function to consider the event as a killed event. The units of *time_limit* are seconds. For example, if *time_limit* is set to 600, a unit *dead_units* dies, but was last hit by any unit in *killer_units* 900 seconds previously, then there is no way that the flag *flag* will be set true by the slmod.units_killed_by function.

9.5.6- slmod.units_crashed

slmod.units_crashed(table *units*, **number** *flag*, **number** *stopflag* **)**

Description:

The function sets flag *flag* when one of the units listed in *units* crashes. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

Only air units can crash. An air unit only “crashes” when it is *completely* destroyed- this is different than when it “dies”! An air unit can be still be airborne, dead, but not yet crashed.

9.5.7- slmod.pilots_dead

slmod.pilots_dead(table ***units*****, number** ***flag*****, number** ***stopflag*** **)**

Description:

The function sets flag *flag* when the pilot of one of the air units listed in *units* dies. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

As of DCS 1.1.1.1, there were multiplayer synchronization issues with ejected pilots. Ejected pilots were local objects only. Thus, a multiplayer client might eject a second before he hits the ground, and see his pilot not have enough time to get his parachute open and, yet to the host, the pilot of the multiplayer client aircraft will disappear the moment the ejection process starts, and thus the pilot will *not* die. So this function would fail to detect the instance of a too-late ejection that also results in pilot death.

9.5.8- slmod.units_ejected

slmod.units_ejected(table ***units*****, number** ***flag*****, number** ***stopflag*** **)**

Description:

This function sets flag *flag* when the pilot of one of the air units listed in *units* ejects from the aircraft. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

slmod.pilots_ejected can be used as an alternate way of referring to this function; both names refer to the same function.

9.6- Weapons tracking functions

The weapon tracking functions were introduced with Slmodv6.0, and finally represent a solution to the problem of detecting when weapons enter or impact in zones. These functions can be used for a variety of exciting possibilities, such as:

-Having ground units turn off their AI and “take cover” when rounds start landing close to them
-Weapons accuracy competitions
-Having enemies react when fire comes near them (such as making a unit react when rounds are fired “across its bow”)

Considerable effort has been placed into making these functions run efficiently. Testing has shown that noticeable decreases in game performance for the host shouldn’t occur until there are hundreds or thousands of weapons simultaneously airborne, **and** hundreds of zones. Thus, you shouldn’t have to fear using these functions with GAU-8 or other rapid-fire weapons.

9.6.1- slmod.weapons_impacting_in_zones

slmod.weapons_impacting_in_zones(table *init_units*, **table** *zones*, **table** *weapons*, **number** *flag*, **number** *stopflag* **)**

Description:

This function sets flag *flag* to true when one of the weapon types listed in *weapons* that was fired by one of the units listed in *init_units* impacts into one of the zones listed in *zones*. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

The weapon names are the same as used by the slmod.units_firing function and all the other weapons-tracking functions; thus, to find out the names for the weapons you must use, you have to fire or get an AI to fire the weapon in a mission. Next, open C:\Users\<Your Account Name>\Saved Games\DCS\Logs\debrief.log. Find the weapon firing event, and copy the name of the weapon out of that.

The names of the zones listed in the table *zones* are the exact same as what they are named in the mission editor.

9.6.2- slmod.weapons_impacting_in_moving_zones

slmod.weapons_impacting_in_moving_zones(table *init_units*, **table** *zone_units*, **number** *radius*, **table** *weapons*, **number** *flag*, **number** *stopflag* **)**

Description:

This function sets flag *flag* to true when one of the weapon types listed in *weapons* that was fired by one of the units listed in *init_units* impacts within a radius of *radius* around one of the units listed in *zone_units*. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

You do not use mission editor zones for this function. The moving zones are wholly defined by *zone_units* and *radius*. The units of *radius* are meters.

The weapon names are the same as used by the slmod.units_firing function and all the other weapons-tracking functions; thus, to find out the names for the weapons you must use, you have to fire or get an AI to fire the weapon in a mission. Next, open C:\Users\<Your Account Name>\Saved Games\DCS\Logs\debrief.log. Find the weapon firing event, and copy the name of the weapon out of that.

The names of the zones listed in the table *zones* are the exact same as what they are named in the mission editor. These names must be strings, of course.

9.6.3- slmod.weapons_in_zones

slmod.weapons_in_zones(table *init_units*, **table** *zones*, **table** *weapons*, **number** *flag*, **number** *stopflag*, **string** *zone_type* **)**

Description:

This function sets flag *flag* to true when one of the weapon types listed in *weapons* that was fired by one of the units listed in *init_units* is inside one of the zones listed in *zones*. The optional variable *zone_type* sets the shape of the zone. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.

Notes:

The names of the zones listed in the table *zones* are the exact same as what they are named in the mission editor. These names must be strings, of course.
The optional variable *zone_type* defines the shape of the zones listed in *zones*. *zone_type* has the following allowed values:
`'cylinder'` or `'c'` or `'C'` or `'cylindrical'`: each zone is a cylindrical zone that extends from negative to positive infinity in altitude.
`'sphere'` or `'s'` or `'S'` or `'spherical'`: each zone is a spherical zone.
If *zone_type* is not defined, it defaults to `'cylinder'`.
The weapon names are the same as used by the slmod.units_firing function and all the other weapons-tracking functions; thus, to find out the names for the weapons you must use, you have to fire or get an AI to fire the weapon in a mission. Next, open C:\Users\<Your Account Name>\Saved Games\DCS\Logs\debrief.log. Find the weapon firing event, and copy the name of the weapon out of that.

9.6.4- slmod.weapons_in_ moving_zones

slmod.weapons_in_ moving_zones(table *init_units*, table *zone_units*, number *radius*, table *weapons*, number *flag*, number *stopflag*, string *zone_type*)
Description:
This function sets flag *flag* to true when one of the weapon types listed in *weapons* that was fired by one of the units listed in *init_units* is within a zone defined around one of the units listed in *zone_units*. The variable *radius* controls the size of the zone(s) and the optional variable *zone_type* controls the shape. Once called, this function evaluates continuously until either the mission ends, or *stopflag* is set true.
Notes:
You do not use mission editor zones for this function. The moving zones are wholly defined by *zone_units*, *radius*, and *zone_type*.
The optional variable *zone_type* defines the shape of the zones that are created around each unit in *zone_units*. *zone_type* has the following allowed values:
`'cylinder'` or `'c'` or `'C'` or `'cylindrical'`: each zone is a cylindrical zone that extends from negative to positive infinity in altitude.
`'sphere'` or `'s'` or `'S'` or `'spherical'`: each zone is a spherical zone.
If *zone_type* is not defined, it defaults to `'cylinder'`.
The weapon names are the same as used by the slmod.units_firing function and all the other weapons-tracking functions; thus, to find out the names for the weapons you must use, you have to fire or get an AI to fire the weapon in a mission. Next, open C:\Users\<Your Account Name>\Saved Games\DCS\Logs\debrief.log. Find the weapon firing event, and copy the name of the weapon out of that.

9.7- Unit property and position-based functions

9.7.1- slmod.num_dead_gt

slmod.num_dead_gt(table *units*, number *numdead*, number *flag*, number *stopflag*)
Quick Description:
This function operates similarly to the “GROUP DEAD MORE THAN” trigger condition, except it can span multiple groups of units, or just subsets units within an arbitrary number of groups. If the number of

units listed by name within *units* that are dead exceeds *numdead*, then flag *flag* is set true. The function only needs to be told to run once, after which, it will continuously the alive/dead status of the units within *units* until either *stopflag* becomes true or the mission ends.

Quick Notes:

This function does not work on multiplayer client aircraft (they are always evaluated as dead).

units is a table of all the names of the units. All table entries must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

If *stopflag* is not specified, then the continuous evaluation of this function can never be stopped.

Example:

```
slmod.num_dead_gt({'tanks1_1', 'tanks1_2', 'tanks1_3', 'tanks1_4',
'tanks1_5', 'tanks2_1', 'tanks2_2', 'tanks2_3', 'tanks2_4',
'tanks2_5', 'tanks3_1', 'tanks3_2', 'tanks3_3', 'tanks3_4',
'tanks3_5'}, 8, 99)
-- Set flag 99 true when more than 8 tanks are destroyed from the
-- list of 15 tanks.
```

9.7.2- slmod.units_LOS

slmod.units_LOS(table *unitset1*, **number** *altoffset1*, **table** *unitset2*, **number** *altoffset2*, **number** *flag*, **number** *stopflag*, **number** *interval*, **number** *checks*, **number** *radius* **)**

Description:

This function is a basic line-of-sight function. It sets flag *flag* when one of the units from *unitset1* is line of sight to one of the units from *unitset2*. *altoffset1* is the "vision" height for all the units in *unitset1* and *altoffset2* is the "vision" height for all the units in *unitset2* (this may become optionally a table a numbers in some future Slmod version). The optional variable *stopflag* specifies a flag number that, if set true, stops the function from operating. The optional variable *radius* allows you to set a maximum distance between units (in meters) beyond which line of sight (LOS) checking will not take place.
This function can become computationally expensive (i.e., if improperly set, it can bog down your computer), so the optional variables *interval* and *checks* can be specified. *interval* is the time period between function evaluations (in seconds). *checks* is the number of times that land height is checked between two units to determine if there is LOS.

Notes:

This function DOES work on multiplayer client aircraft!

All table entries in *unitset1* and *unitset2* must be strings. THE UNITS DO NOT HAVE TO BE FROM THE SAME GROUP. As with all tables of unit names, there are absolutely no group restrictions.

If *interval* is not specified, it defaults to 1.

If *checks* is not specified, it defaults to 1000.

If *radius* is not specified, it defaults to infinity (the DCS world is flat so you can get LOS between two units even if they are on the ground and on opposite sides of the map).

If you never want the function to stop running then a good value for *stopflag* is -1. NOTE: In the pre-release slmodbeta5 version released on the ED forums, the functionality of *stopflag* is slightly bugged. Please make sure you are updated to slmodbeta5_2 or later!

Examples:

```
slmod.units_LOS({'MPclient1'}, 0, {'AWACS1'}, 4, 101, -1, 6, 1000,
165000)
--[[An example of what part of a "radar detection" type logic
might look like.  If the unit named "MPclinet1" is LOS to the
"AWACS1" unit, then flag 101 gets set true.  The LOS evaluation
can never be stopped (stopflag is -1), it is evaluated once every
6 seconds, it does 1000 checks of terrain height between "MPclient1"
and "AWACS1", and it evaluates for LOS out to a range of 165000
meters.]]--


slmod.units_LOS({'us1_1', 'us1_2', 'us1_3', 'us1_4', 'us1_5'}, 3,
{'ru1_1', 'ru1_2', 'ru1_3', 'ru1_4', 'ru1_5'}, 3, 11, -1, 10, 200,
8000)
--[[Set flag 11 if one of the units in the "us1" group is LOS to
one of the units in the "ru1" group (assuming all these are in the
same group).  The LOS evaluation can never be stopped (stopflag is -
1),
it is evaluated once every 10 seconds, it does 200 checks of terrain
height between each unit, and only evaluates for LOS out to a range
of 8000 meters.]]—
```

9.7.3- slmod.units_in_moving_zones

slmod.units_in_moving_zones(table *units*, **table** *zone_units*, **number** *radius*, **number** *flag*, **number** *stopflag*, **string** *zone_type*, **number** *req_num*, **number** *interval* **)**

Description:

This function sets flag *flag* to true when one or more units in *units* comes within a zone centered on one of the units listed in *zone_units*. *radius* and *zone_type* control the size and shape of the zones that are drawn around each unit in *zone_units*, while *interval* controls the frequency of the checks and *req_num* controls the number of units that must be within a zone for flag *flag* to be set true. Once called, this function loops forever until either the mission ends, or *stopflag* is set true.

Notes:

The units of *radius* are meters.

The optional variable *zone_type* defines the shape of the zones that are created around each unit in *zone_units*. *zone_type* has the following allowed values:

`'cylinder'` or `'c'` or `'C'` or `'cylindrical'`: each zone is a cylindrical zone that extends from negative to positive infinity in altitude.

`'sphere'` or `'s'` or `'S'` or `'spherical'`: each zone is a spherical zone.

If *zone_type* is not defined, it defaults to `'cylinder'`.

req_num controls the number of units in *units* that must come within a zone around one of the units in *zone_units* for the flag *flag* to be set true. If *req_num* is not specified, it defaults to 1.

interval controls the time between each in-zone check. The units of *interval* are seconds. If *interval* is not specified, it defaults to 1. It is desirable to reduce the frequency of interval when there are many units being checked; for example, if 250 units are being checked if they are in a zone around 250 other units, that is 250*250 = 125,000 checks. For such a large number of units, it might be necessary to increase interval to avoid a periodic micro-pause for the host every second.

Additionally, for such a large number of in-zone checks, it might be desirable to break these checks down into several groups; for example, these 125,000 checks could be divided into five groups, and five different calls could be made:

slmod.units_in_moving_zones for units 1-50 and zone units 1-250; *interval* = 5.

One second later, start another units_in_moving_zones call:

slmod.units_in_moving_zones for units 51-100 and zone units 1-250; *interval* = 5.

One second after that, start another units_in_moving_zones call:
slmod.units_in_moving_zones for units 101-150 and zone units 1-250; *interval* = 5.
One second after that, start another units_in_moving_zones call:
slmod.units_in_moving_zones for units 151-200 and zone units 1-250; *interval* = 5.
One second after that, start another units_in_moving_zones call:
slmod.units_in_moving_zones for units 201-251 and zone units 1-250; *interval* = 5.

Now, not only is the slmod.units_in_moving_zones call executing effectively less often (though there is a portion of the total checks executing every second), but the computation has been divided up into 5 smaller function calls. This would remove a possible micro-pause that could occur for the host every five seconds had the slmod.units_in_moving_zones call been all 250 units checked against all 250 zone units every five seconds.

9.8- Flag manipulation functions

9.8.1- slmod.rand_flags_on

slmod.rand_flags_on(number *startflag*, number *endflag*, number *prob*)

Quick Description:
This function randomly sets flags between *startflag* and *endflag* to true, at a probability percentage of *prob*. If *endflag* is not specified (i.e., the function only is given only two input variables), then the function treats the second input variable as *prob* and assumes *endflag* is the same as *startflag*. Thus with only two input variables, the function operates to randomly set only *startflag* at the probability of *prob*.
Quick Notes:
Allowed values for *prob* are between 0 and 100 (inclusive).
The function only operates a single time after it is called. It can be up to 2.5 seconds after the function is called before the function does its work.
Examples:

```
slmod.rand_flags_on(15, 50)
--Randomly set flag 15 at a probability of 50%


slmod.rand_flags_on(101, 112, 35)
-- Randomly set flags 101 through 112 at a probability of 35%
```

9.8.2- slmod.rand_flag_choice

slmod.rand_flag_choice(number *startflag*, number *endflag*)

Quick Description:
When this function is called, ONE and only ONE flag between *startflag* and *endflag* will be randomly selected and set to true. In other words, a random choice is made.
Quick Notes:
endflag must be greater than *startflag*, and both must be positive integers.
The function only operates a single time after it is called. It can be up to 2.5 seconds after the function is called before the function does its work.
Example:

```
slmod.rand_flag_choice(1, 8)
```

```
-- Randomly select a single flag between 1 and 8 (inclusive) to be
-- set true
```