

Module ScenEdit

Lua is a case sensitive language.

When accessing object properties directly as in 'unit.name', the property should be in lower which will match the Lua generated code. There may be a *few* special cases (e.g. mission.SISH=true which is a shortcut for scrubj/side_human) which will be documented below.

However, when accessing the properties through the module functions below, both the keyword/property and the value are case insensitive; the code will worry about matching them up.

Selector

These define the information required as part of the 'select' process for the functions. In the case of functions that 'add' things, these are also key elements to the adding process. Other properties may be included in the 'selector' such as when updating an existing table.

When selecting units, it is preferable to use the GUID as the identifier for a precise match. If not, then the side and name for a more limited search. And as a last option, just the name which search all units in the scenario. When using just the name, usually the first matching name is returned. This is okay if the names are unique. Thus including the side, it will only check the units on that side for a match.

Wrapper

These define the information that is returned from some functions. This information can be usually modified either directly (object.field) or by a wrapper Set(..) function. The particular wrapper Set(..) function is preferred as some validation is performed on the input to ensure that it is within the bounds of the field being updated.

Data type

These cover any special considerations for the data, such as longitude/latitude; degrees, DMS, N/S, E/W, etc.

Error handling

Usually when a Lua script fails, an error is thrown that ends the script at that point. While this may be okay in most cases, it is not often desired.

Whenever a Command Lua function gets an error, it will normally return a 'nil', and the error message will be available as a Lua global variables; '_errmsg_' will have the last error message, '_errfnc_' the Command Lua function that gave the error, and '_ernum_' the error code (0 is no error, and any value >0 will be an error). Thus if you get back a 'nil', you can check to see if that is due to an error or no data.

Example:

without the new error handling, the script below would probably terminate after the SE_AddMission() and the rest of script would not run.

```
local mission = ScenEdit_AddMission('USA','Marker strike','strike',{type='land'})
if mission == nil then
    if _ernum_ ~= 0 then print('Failed to add:' .. _errmsg_) else print('Something else') end
else
    print(mission)
    do some more command lua stuff...
end
```

Note: Due to how errors are now handle from SR7, if a command fails in the console window, it will show an error. If the command runs outside the console (as in an event script), it will not fail with a visible error but return a nil or or some other value. One issue with commands running in an event is that sometimes they fail with an in-game message that actually stops the rest of the script from running. Now, these event scripts will run without any in-game message showing, and the designer should check the result of the command and handle any error condition, and let the remaining script run as needed.

My intent is have all command errors behave in the same fashion in the console window; and the command errors outside a console behave without stopping the script. Which requires the designer to cater for the specific error conditions.

To emulate the expected outcome from an event, put 'Tool_EmulateNoConsole(true)' at the start of the script to be tested; it is not required in the event code as the script is already not running in a console.

Note also, that the Lua history log should also record the event script errors.

Release: 1.12

Selector

ContactSelector	Contact selector.
DamageOptions	Unit damage.
DoctrineSelector	Selects a doctrine.
DoctrineWRASelector	Selects a WRA doctrine.
EventUpdate	Event update.

NewMission	New mission options.
NewUnit	New unit selector.
ReferencePointSelector	Reference point selector.
SideOption	Side options.
SpecialAction	Special action selector.
UnitSelector	Select a unit.
UpdateUnit	Update unit sensor/mount selector.
VPContaktSelector	Contact selector.
Weapon2Magazine	Select magazine and weapon.
Weapon2Mount	Select mount and weapon.

Wrapper/Descriptor

AttackOptions	Attack options.
Contact	Contact details.
Doctrine	Doctrine options.
DoctrineWRA	WRA Doctrine options.
Group	Group details.
Mission	Mission.
ReferencePoint	Reference point information.
RefuelOptions	Unit refueling options.
Side	Side perspective.
Unit	Represents a active unit.
Zone	Exclusion/No navigation zone.

Data types

Altitude	Altitude.
Arc	Arcs.
DateTime	DateTime.
KeyStore	KeyStore.
Latitude	Latitude.
Longitude	Longitude.
Size	Size.
Stance	Stance/Posture.
TargetTypeWRA	Target type.
TimeStamp	TimeStamp.
Weather	weather conditions.

Functions

GetScenarioTitle ()	Name of the scenario.
ScenEdit_AddExplosion (explosion)	Detonates a warhead at a specified location.
ScenEdit_AddMission (SideNameOrId, MissionNameOrId, MissionType, MissionOptions)	Add new mission.
ScenEdit_AddReferencePoint (descriptor)	Add a new reference point.
ScenEdit_AddReloadsToUnit (descriptor)	Adds weapons into a mount.
ScenEdit_AddSide (table)	Add side.
ScenEdit_AddUnit (unit)	Adds a unit based on a descriptor.
ScenEdit_AddWeaponToUnitMagazine (descriptor)	Adds weapons into a magazine.
ScenEdit_AddZone (sideName, zoneType, table)	Add no-nav or exclusion zone.

ScenEdit_AssignUnitAsTarget (AUNameOrIDOrTable, MissionNameOrID)	Assigns targets to a Strike mission.
ScenEdit_AssignUnitToMission (unitname, mission, escort)	Assign a unit to a mission.
ScenEdit_AttackContact (attackerID, contactId, options)	Attack a contact ...
ScenEdit_CurrentTime ()	Get the current scenario time.
ScenEdit_DeleteMission (SideNameOrId, MissionNameOrId)	Delete mission.
ScenEdit_DeleteReferencePoint (selector)	Delete a reference point.
ScenEdit_DeleteUnit (unit)	Deletes unit.
ScenEdit_EndScenario ()	Ends the current scenario.
ScenEdit_ExecuteEventAction (EventDescriptionOrID)	Execute a Lua Event action script.
ScenEdit_ExecuteSpecialAction (eventNameOrId)	Execute a Lua Special action script ..
ScenEdit_ExportMission (SideNameOrId, MissionNameOrId)	Export mission parameters.
ScenEdit_FillMagsForLoadout (unit, loadoutid, quantity)	Fill a unit's magazine(s) with aircraft stores ...
ScenEdit_GetContact (contact)	Fetches a contact based on a selector.
ScenEdit_GetContacts (side)	Contacts from a side.
ScenEdit_GetDoctrine (selector)	Gets the doctrine of the designated object.
ScenEdit_GetDoctrineWRA (selector)	Gets the WRA doctrine.
ScenEdit_GetKeyValue (key)	Gets the value for a key from the persistent key store.
ScenEdit_GetMission (SideNameOrId, MissionNameOrId)	Get details of a mission.
ScenEdit_GetReferencePoints (selector)	Get a set of reference point(s).
ScenEdit_GetScenHasStarted ()	Has the scenario started?
ScenEdit_GetScore (side)	Get a given side's score.
ScenEdit_GetSideIsHuman (sidename)	Returns true if side is played by a human player
ScenEdit_GetSideOptions (options)	Get side options.
ScenEdit_GetSidePosture (sideA, sideB)	Gets the posture of one side towards another.
ScenEdit_GetSpecialAction (action_info)	Gets the properties of an existing special action.
ScenEdit_GetUnit (unit)	Fetches a unit based on a selector.
ScenEdit_GetWeather ()	Get the current weather conditions.
ScenEdit_HostUnitToParent (host_info)	Hosts a unit to the specified host.
ScenEdit_ImportInst (side, filename)	Imports an inst file.
ScenEdit_ImportMission (SideNameOrId, MissionNameOrId)	Import mission parameters.
ScenEdit_InputBox (string)	Open an input box with the passed prompt.
ScenEdit_KillUnit (unit)	Kill unit.
ScenEdit_MsgBox (string, style)	Show a message box with a passed string.
ScenEdit_PlayerSide ()	The player's current side.
ScenEdit_RefuelUnit (unitOptions)	Cause unit to refuel.
ScenEdit_RemoveSide (table)	Remove side from play.
ScenEdit_RemoveUnitAsTarget (AUNameOrIDOrTable, MissionNameOrID)	Removes targets from a Strike mission.
ScenEdit_RunScript (script)	Runs a script from a file.

ScenEdit_SetDoctrine (selector, doctrine)	Set the doctrine of the designated object.
ScenEdit_SetDoctrineWRA (selector, options)	Sets the WRA doctrine.
ScenEdit_SetEMCON (type, name, emcon)	Sets the EMCON of the selected object.
ScenEdit_SetKeyValue (key, value)	Sets the value for a key in the persistent key store.
ScenEdit_SetLoadout (loadoutinfo)	Sets the loadout for a unit
ScenEdit_SetMission (SideName, MissionNameOrId, MissionOptions)	Set details for a mission.
ScenEdit_SetReferencePoint (descriptor)	Update a reference point(s) with new values.
ScenEdit_SetScore (side, score, reason)	Sets a given side's score.
ScenEdit_SetSideOptions (options)	Set side options.
ScenEdit_SetSidePosture (sideAName, sideBName, posture)	Set the posture of a side towards another.
ScenEdit_SetSpecialAction (action_info)	Sets the properties of an existing special action.
ScenEdit_SetUnit (unit)	Sets the properties of a unit that already exists.
ScenEdit_SetUnitDamage (options)	Set unit damage.
ScenEdit_SetUnitSide (sidedesc)	Changes the side of a unit
ScenEdit_SetWeather (temperature, rainfall, undercloud, seastate)	Set the current weather conditions.
ScenEdit_SpecialMessage (side, message)	Displays a special message consisting of the HTML text message to side 'side'.
ScenEdit_UnitX ()	Activating Unit ..
ScenEdit_UnitY ()	Detecting Unit ...
ScenEdit_UpdateEvent (EventDescriptionOrID, options)	Sets the Lua script(s) of an event.
ScenEdit_UpdateUnit (options)	Update items on a unit.
ScenEdit_UseAttachment (attachment)	Import an attachment into the scene.
ScenEdit_UseAttachmentOnSide (attachment, sidename)	Use an attachment on a side (used for .inst files as attachments).
Tool_DumpEvents ()	Dump scenario events.
Tool_EmulateNoConsole ()	Emulates no console.
Tool_Range ()	Get range between points.
VP_GetContact (ContactGUID)	Get contact details
VP_GetSide (side)	Side information from player's perspective.
VP_GetUnit (ActiveOrContact)	Get unit details
World_GetCircleFromPoint (table)	Returns a circle around point.
World_GetElevation (location)	Returns the elevation in meters of a given position

Tables

AAR	AAR.
CargoMission	CargoMission.
Component	Component.
EMmatch	EM matches ..
Explosion	Defines the warhead to detonate.
FerryMission	FerryMission.
Fuel	Fuel.
FuelState	Status of a fuel type .
HostUnit	Requests that a unit be hosted to another
LatLon	A Position on the map.
LoadoutInfo	Information on a loadout to add/alter

Magazine	Magazine.
MineClearMission	MineClearMission.
MineMission	MineMission.
Mount	Mount.
PatrolMission	PatrolMission.
SideContact	Side's contact.
SideDescription	Information on how to change a unit's side.
SideSelector	Information to select a particular side.
SideUnit	Side's unit.
StrikeMission	StrikeMission.
SupportMission	SupportMission.
WRA	WRA settings.
WeaponLoaded	Weapon loads on mount or in magazine.
ZoneMarker	Zone marker.

Selector

ContactSelector

Contact selector.

Note that a unit and it's contact GUIDs are different for the same physical unit; the contact GUIDs are from the other side's perspective

- **side:** (*string*) The side to find the the contact on.
- **guid:** (*string*) The GUID of the contact. Interrogate ScenEdit_GetContacts(side) for the side's contacts and use the GUID from there.

DamageOptions

Unit damage. The component table list consists of entries for each component, identified by guid and new level. Special cases:

- if guid is 'type', then you can set a type of component to be damaged,
- if damageLevel is 'none', then the component will be repaired.

- **side:** (*string*)
- **unitname:** (*string*)
- **guid:** (*string*)
- **fires:** (*string*)
- **flood:** (*string*)
- **components:** (*{ table }*) Table of component damage setting { guid, damageLevel }

Usage:

```
components={{ '16a883a2-8e7f-4313-aae7-0af644c16337', 'none' }, { 'rudder', 'Medium' }, { 'type', typ
```

DoctrineSelector

Selects a doctrine. .. for on a side, group, mission, or unit

- **side:** (*string*) The side to select/from
- **mission:** (*string*) The name of the mission to select
- **unitname:** (*string*) The name of the unit to select
- **group:** (*string*) The name of the group to select
- **escort:** (*bool*) If a strike mission, adding 'escort=true' will update the escort doctrine

DoctrineWRASelector

Selects a WRA doctrine. .. for on a side, group, mission, or unit. *weaponid is mandatory. One of side, mission, unitname or guid is mandatory. One of contactid or target_type is mandatory*

- **side:** (*string*) The side to select/from

- **mission:** (*string*) The name of the mission to select
- **unitname:** (*string*) The name of the unit to select
- **guid:** (*string*) The unit GUID to select
- **weapon_id:** (*string*) The weapon database id
- **contact_id:** (*string*) A contact GUID (mutually exclusive with target_type)
- **target_type:** (*string*) The target type (mutually exclusive with contact_id)

EventUpdate

Event update.

- **type:** (*string*) What to update (add_condition,remove_condition,replace_condition,add_action,remove_action,replace_action)
- **description:** (*string*) The description of the Lua script (can use GUID for existing item in remove/replace)
- **script:** (*string*) The Lua script

NewMission

New mission options.

- **type:** (*string*) Mission sub-type (Applies to Patrol and Strike)
- **destination:** (*string*) Ferry mission destination
- **zone:** (*{ string }*) A table of reference points as names or GUIDs (Can apply to Patrol, Support, Mining, Cargo)

NewUnit

New unit selector.

... lists minimum fields required. Other fields from [Unit](#) may be included.

- **type:** (*string*) The type of unit (Ship, Sub, Aircraft, Facility)
- **unitname:** (*string*) The name of the unit
- **side:** (*string*) The side name or GUID to add unit to
- **dbid:** (*number*) The database id of the unit
- **latitude:** (*Latitude*) Not required if a **base** is defined
- **longitude:** (*Longitude*) Not required if a **base** is defined
- **base:** (*string*) Unit base name or GUID where the unit will be 'hosted' (applicable to AIR, SHIP, SUB)
- **loadout:** (*number*) Aircraft database loadout id (applicable to AIR)
- **altitude:** (*number*) Unit altitude (applicable to AIR)

ReferencePointSelector

Reference point selector.

To select reference point(s), specify either

- **name** and **side**, to select a reference point **name** belonging to **side** [name AND side if possible] or
- **guid**, if the unique ID of the reference point is known [preferred] or
- **area**, table of reference points (name or guid)

GUID method takes precedence over name/side if both present.

- **side:** (*string*) The side the reference point is visible to
- **name:** (*string*) The name of the reference point
- **guid:** (*string*) The unique identifier for the reference point
- **area:** (*{name or guid}*) Table of reference points (used with the Set()/Get() functions]

SideOption

Side options.

- **side:** (*string*) Side name
- **guid:** (*string*) Side guid
- **awareness:** (*string*) Side awareness
- **proficiency:** (*string*) Side proficiency

SpecialAction

Special action selector.

- **GUID:** (*string*) The GUID of the special action [READONLY]
- **ActionNameOrID:** (*string*) The name or ID of the special action
- **IsActive:** (*bool*) If the action is visible to the player
- **IsRepeatable:** (*bool*) If the player can use the action multiple times
- **NewName:** (*string*) If specified, the new name of the action
- **Description:** (*string*) If specified, the new description for the action

UnitSelector

Select a unit. ... based on either the side and name or the unique identifier (GUID).

You can use either 1. **name** and **side** 2. **guid** If both are given, then the GUID is used preferentially.

- **name:** (*string*) The name of the unit to select (for option #1)
- **side:** (*string*) The name of the unit to select (for option #1)
- **guid:** (*string*) The guid of the unit to select (for option #2)

UpdateUnit

Update unit sensor/mount selector.

... lists minimum fields required. Other fields from **Unit** may be included.

- **guid:** (*string*) The unit identifier
- **mode:** (*string*) The function to perform (*addsensor*, *deletesensor*, *addmount*, *removemount*)
- **dbid:** (*number*) The database id of the item to add [required for *add_ mode*]
- **sensorid:** (*string*) The identifier (guid) of the particular sensor to remove [required for *remove_sensor mode*]
- **mountid:** (*string*) The identifier (guid) of the particular mount to remove [required for *remove_mount mode*]
- **arc_detect:** (*{ Arcs }*) The effective arcs for the particular sensor to detect in [override defaults] (*optional*)
- **arc_track:** (*{ Arcs }*) The effective arcs for the particular sensor to track/illuminate in [override defaults] (*optional*)
- **arc_mount:** (*{ Arcs }*) The effective arcs for the particular mount [override defaults] (*optional*)

VPContactSelector

Contact selector.

Note that a unit and it's contact GUIDs are different for the same physical unit; the contact GUIDs are from the other side's perspective

- **guid:** (*string*) The GUID of the contact. Interrogate *VP_GetSide().contacts* for the side's contacts and use the GUID from there.

Weapon2Magazine

Select magazine and weapon.

A group magazine, for example, tend to have multiple magazines, with the same name. So you can specify a particular **magazine** by the GUID, or leave it out, and the function will try to fill up any **available** space with the **weapon**.

- **side:** (*string*) The side name/GUID of the unit with magazine
- **unitname:** (*string*) The name/GUID of unit with magazine
- **guid:** (*string*) GUID of the unit with magazine
- **mag_guid:** (*string*) The magazine GUID
- **wpn_dbid:** (*string*) The weapon database ID
- **number:** (*number*) Number to add
- **maxcap:** (*number*) Maximum capacity of the weapon to store
- **remove:** (*bool*) If true, this will debuct the number of weapons
- **new:** (*bool*) If true, will add the weapon if it does not exist
- **fillout:** (*bool*) If true, will fill out the weapon record to its maximum

Weapon2Mount

Select mount and weapon.

You can specify a particular **mount** by the GUID, or leave it out, and the function will try to fill up any available space with the **weapon**.

- **side:** (*string*) The side name/GUID of the unit with mount
- **unitname:** (*string*) The name/GUID of unit with mount
- **guid:** (*string*) GUID of the unit with mount
- **mount_guid:** (*string*) The mount GUID
- **wpn_dbid:** (*string*) The weapon database ID
- **number:** (*number*) Number to add
- **remove:** (*bool*) If true, this will debuct the number of weapons
- **fillout:** (*bool*) If true, will fill out the weapon record to its maximum

Wrapper/Descriptor

AttackOptions

Attack options.

- **mode:** (*string*) Targeting mode "AutoTargeted"|"0", "ManualWeaponAlloc"|"1"
- **mount:** (*number*) The attacker's mount DBID
- **weapon:** (*number*) The attacker's weapon DBID
- **qty:** (*number*) How many to allocate

Contact

Contact details.

This is from the perspective of the side being looked at. What is a contact for one side, may not be the same contact on another side. Note also the GUID of the contact is not the same as the actual unit. So depending on what functions you call, you may need to 'convert' the contact 'GUID' into the actual 'GUID' and call GetUnit() to process the actual GUID.

- **name:** (*string*) The contact name.
- **guid:** (*string*) The contact GUID. [READONLY]
- **actualunitid:** (*string*) The contact actual GUID. [READONLY]
- **latitude:** (*Latitude*) The latitude of the contact. [READONLY]
- **longitude:** (*Longitude*) The longitude of the contact. [READONLY]
- **missile_defence:** (*number*) Applicable to Facility and Ships. -1 = unknown contact [READONLY]
- **age:** (*number*) How long has contact been detected (in seconds) [READONLY]
- **type:** (*string*) Type of contact. [READONLY]
- **typed:** (*number*) Type of contact. [READONLY]
- **areaofuncertainty:** (*{ LatLon }*) Table of points defining the area of contact. [READONLY]
- **type_description:** (*string*) Contact type description. [READONLY]
- **actualunitdbid:** (*number*) Actual contact type. [READONLY]
- **classificationlevel:** (*string*) Contact classification. [READONLY]
- **potentialmatches:** (*{ EMmatch }*) Table {EMmatch} on potential EMCON emission matches. [READONLY]
- **side:** (*Side*) Contact's actual side. [READONLY]
- **fromside:** (*Side*) The side who sees this contact. [READONLY]
- **posture:** (*string*) Posture towards contact.
- **FilterOut:** (*bool*) True to filtered out contact
- **weather:** (*{ Weather }*) Table of weather parameters (temp, rainfall, underrain, seastate)
- **DropContact:** (*method*) () Drops contact from the reporting side

Doctrine

Doctrine options.

For each field, adding the suffix "*playereditable*" determines if the player can alter the setting. Not applicable to the Withdraw/Deploy options.

When setting the option, the indicated value or it's number can be used.

- **use_nuclear_weapons:** (*bool*) True if the unit should be able to employ nuclear weapons
- **engage_non_hostile_targets:** (*bool*) True if the unit should attempt hostile action against units that are not hostile
- **rtb_when_winchester:** (*bool*) (obsolete, see the new doctrine options) True if the unit should return to base when out of weapons
- **ignore_plotted_course:** (*bool*) True if the unit should ignore plotted course

- **engaging_ambiguous_targets:** (*string*) Ignore(0), Optimistic(1), or Pessimistic(2)
- **automatic_evasion:** (*bool*) True if the unit should automatically evade
- **maintain_standoff:** (*bool*) True if the unit should try to avoid approaching its target, only valid for ships
- **use_refuel_unrep:** (*string*) AlwaysExceptTankersRefuellingTankers(0), Never(1), AlwaysIncludingTankersRefuellingTankers(2)
- **engage_opportunity_targets:** (*bool*) True if the unit should take opportunistic shots
- **use_sams_in_anti_surface_mode:** (*bool*) True if SAMs should be used to engage surface targets
- **ignore_emcon_while_under_attack:** (*bool*) True if EMCON should be ignored and all systems should go active when engaged
- **quick_turnaround_for_aircraft:** (*string*) Yes(0), FightersAndASW(1), No(2)
- **air_operations_tempo:** (*string*) Surge(0), Sustained(1)
- **kinematic_range_for_torpedoes:** (*string*) AutomaticAndManualFire(0), ManualFireOnly(1), No(2)
- **weapon_control_status_air:** (*string*) Free(0), Tight(1), Hold(2)
- **weapon_control_status_surface:** (*string*) Free(0), Tight(1), Hold(2)
- **weapon_control_status_subsurface:** (*string*) Free(0), Tight(1), Hold(2)
- **weapon_control_status_land:** (*string*) Free(0), Tight(1), Hold(2)
- **refuel_unrep_allied:** (*string*) Yes(0), YesReceiveOnly(1), YesDeliverOnly(2), No(3)
- **fuel_state_planned:** (*string*) Bingo(0), Joker10Percent(1), Joker20Percent(2), Joker25Percent(3), Joker30Percent(4), Joker40Percent(5), Joker50Percent(6), Joker60Percent(7), Joker70Percent(8), Joker75Percent(9), Joker80Percent(10), Joker90Percent(11)
- **fuel_state_rtb:** (*string*) No(0), YesLastUnit(1), YesFirstUnit(2), YesLeaveGroup(3)
- **weapon_state_planned:** (*string*) See Weapon Doctrine table below
- **weapon_state_rtb:** (*string*) No(0), YesLastUnit(1), YesFirstUnit(2), YesLeaveGroup(3)
- **gun_strafig:** (*string*) No(0), Yes(1)
- **jettison_ordnance:** (*string*) No(0), Yes(1)
- **avoid_contact:** (*string*) No(0), YesExceptSelfDefence(1), YesAlways(2)
- **dive_on_threat:** (*string*) Yes(0), YesESMOnly(1), YesShips20nmAircraft30nm(2), No(3)
- **recharge_on_patrol:** (*string*) RechargeEmpty(0), Recharge10Percent(10), Recharge20Percent(20), Recharge30Percent(30), Recharge40Percent(40), Recharge50Percent(50), Recharge60Percent(60), Recharge70Percent(70), Recharge80Percent(80), Recharge90_Percent(90)
- **recharge_on_attack:** (*string*) RechargeEmpty(0), Recharge10Percent(10), Recharge20Percent(20), Recharge30Percent(30), Recharge40Percent(40), Recharge50Percent(50), Recharge60Percent(60), Recharge70Percent(70), Recharge80Percent(80), Recharge90_Percent(90)
- **use_aip:** (*string*) No(0), YesAttackOnly(1), YesAlways(2)
- **dipping_sonar:** (*string*) Automatically_HoverAnd150ft(0), ManualAndMissionOnly(1)
- **withdraw_on_damage:** (*string*) Ignore(0), Percent5(1), Percent25(2), Percent50(3), Percent75(4)
- **withdraw_on_fuel:** (*string*) Ignore(0), Bingo(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5)
- **withdraw_on_attack:** (*string*) Ignore(0), Exhausted(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5), LoadFullWeapons(6)
- **withdraw_on_defence:** (*string*) Ignore(0), Exhausted(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5), LoadFullWeapons(6)
- **deploy_on_damage:** (*string*) Ignore(0), Percent5(1), Percent25(2), Percent50(3), Percent75(4)
- **deploy_on_fuel:** (*string*) Ignore(0) Bingo(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5)
- **deploy_on_attack:** (*string*) Ignore(0), Exhausted(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5), LoadFullWeapons(6)
- **deploy_on_defence:** (*string*) Ignore(0), Exhausted(1), Percent25(2), Percent50(3), Percent75(4), Percent100(5), LoadFullWeapons(6)

Weapon doctrine:

Value	Number	Explanation
LoadoutSetting	0	use setting from database
Winchester	2001	Vanilla Winchester.
Winchester_ToO	2002	Same as above, but engage nearby bogies with guns after we're out of missiles. Applies to air-to-air missile loadouts only. For guns-only air-to-air loadouts and all air-to-ground loadouts the behaviour is the same as above. PREFERRED OPTION!

ShotgunBVR	3001	Disengage after firing all Beyond Visual Range (BVR, air-to-air) or Stand-Off (SO, air-to-ground) weapons. This is a risky option as your fighter aircraft may only have one medium-range air-to-air missile (AAM) left, and attempt to engage 'fresh' flights of bogies. Use with caution.
ShotgunBVR_WVR	3002	Same as above, but if easy targets or threats are nearby then shoot at them with remaining Within Visual Range (WVR, air-to-air) or SR (Short-Range, air-to-ground) weapons before disengaging.
ShotgunBVR_WVR_Guns	3003	Same as above, but also engage bogies with guns. Applies to air-to-air (A/A) loadouts only. For air-to-ground (A/G) loadouts the behaviour is the same as above.
ShotgunOneEngagementBVR	5001	Make one engagement with BVR or SO weapons. Continue fighting for as long as there are targets within easy reach and then disengage. This is a safe option as it ensures aircraft do not 'hang around' after they have expended their most potent weapons, and becoming easy targets when engaged by 'fresh' enemy units.
ShotgunOneEngagementBVR_Opportunity_WVR	5002	Same as above, but if easy targets or threats are nearby, shoot at them with remaining WVR or Short-Range weapons before disengaging. A target is considered 'easy' when within 120% of the remaining WVR or Strike weapon's maximum range. In other words, the fighter won't spend much energy chasing down a target after the Shotgun weapon state has been reached, and will leave the target area as quickly as possible. PREFERRED OPTION!
ShotgunOneEngagementBVR_Opportunity_WVR_Guns	5003	Same as above, but also engage bogies with guns. Applies to air-to-air (A/A) loadouts only. For air-to-ground (A/G) loadouts the behaviour is the same as above.
ShotgunOneEngagementBVR_And_WVR	5005	Make one engagement with BVR and WVR, or SO and Strike Weapons. Do not

		disengage when out of BVR or SO weapons, but continue the engagement with WVR weapons.
ShotgunOneEngagementBVR_And_WVR_Opportunity_Guns	5006	
ShotgunOneEngagementWVR	5011	Make one engagement with WVR or SR weapons. Continue fighting for as long as there are targets within easy reach and then disengage.
ShotgunOneEngagementWVR_Guns	5012	Same as above but also engage bogies with guns. Applies to air-to-air (A/A) loadouts only. For air-to-ground (A/G) loadouts, the behaviour is the same as above. PREFERRED OPTION!
ShotgunOneEngagementGun	5021	Make one engagement with guns. Continue fighting for as long as there are targets nearby and then disengage.
Shotgun25	4001	Disengage after 1/4 of mission-specific weapons have been expended.
Shotgun25_ToO	4002	Same as above, but if easy targets or threats are nearby then shoot at those too. Also engage bogies with guns. Applies to air-to-air (A/A) loadouts only.
Shotgun50	4011	Disengage after half of mission-specific weapons have been expended.
Shotgun50_ToO	4012	Same as above, but if easy targets or threats are nearby then shoot at those too. Also engage bogies with guns. Applies to air-to-air (A/A) loadouts only.
Shotgun75	4021	Disengage after 3/4 of mission-specific weapons have been expended.
Shotgun75_ToO	4022	Same as above, but if easy targets or threats are nearby then shoot at those too. Also engage bogies with guns. Applies to air-to-air (A/A) loadouts only.

DoctrineWRA

WRA Doctrine options.

- weapon_dbid: (*string*) Weapon number [info]
- weapon_name: (*string*) Weapon name [info]
- target_type: (*TargetTypeWRA*) Type of target [info]
- level: (*string*) The doctrine selected (at unit/mission/side) - useful Is just using GUIDs [info]
- wra: (*{ WRA }*) The WRA doctrine

Group

Group details.

- type: (*string*) Type of group. [READONLY]
- guid: (*string*) [READONLY]
- name: (*string*)
- side: (*string*) [READONLY]
- unitlist: (*{ string }*) Table of unit GUIDs in the group. [READONLY]

Mission

Mission.

- guid: (*string*) The GUID of the mission. [READONLY]
- name: (*string*) Name of mission
- isactive: (*bool*) True if mission is currently active
- side: (*string*) Mission belongs to side
- starttime: (*DateTime*) Time mission starts
- endtime: (*DateTime*) Time mission ends
- type: (*MissionClass*) Mission class(patrol,strike,etc). [READONLY]
- subtype: (*MissionSubClass*) Mission class(asw,land,etc). [READONLY]
- SISH: (*bool*) 'Scrub if side human' tick box
- unitlist: (*{ GUID }*) A table of units assigned to mission. [READONLY]
- targetlist: (*{ GUID }*) A table of targets assigned to mission. [READONLY]
- aar: (*AAR*) A table of the mission air-to-air refueling options. [READONLY]
- ferrymission: (*FerryMission*) A table of the mission specific options. [READONLY]
- mineclearmission: (*MineClearMission*) A table of the mission specific options. [READONLY]
- minemission: (*MineMission*) A table of the mission specific options. [READONLY]
- supportmission: (*SupportMission*) A table of the mission specific options. [READONLY]
- patrolmission: (*PatrolMission*) A table of the mission specific options. [READONLY]
- strikemission: (*StrikeMission*) A table of the mission specific options. [READONLY]
- cargomission: (*CargoMission*) A table of the mission specific options. [READONLY]

ReferencePoint

Reference point information.

- guid: (*string*) The unique identifier for the reference point
- side: (*string*) The side the reference point is visible to
- name: (*string*) The name of the reference point
- latitude: (*Latitude*) The latitude of the reference point
- longitude: (*Longitude*) The longitude of the reference point
- highlighted: (*bool*) True if the point should be selected
- locked: (*bool*) True if the point is locked
- bearingtype: (*bearing*) Type of bearing Fixed (0) or Rotating (1)
- relativeto: (*Unit*) The unit that reference point is relative to

RefuelOptions

Unit refueling options.

- unitSelector: (*UnitSelector*) A normal unit selector defining the unit.
- tanker: (*string*) A specific tanker defined by its name (side is assumed to be the same as unit) or GUID.
- missions: (*{ mission }*) A table of mission names or mission GUIDs.

Usage:

```
{side="United States", name="USS Test", missions={"Pitstop"}, tanker="Hose #1"}
```

Side

Side perspective.

- guid: (*string*) The GUID of the side.
- name: (*string*) The name of the side.
- units: (*{ SideUnit }*) Table of units for the designated side. [READONLY]

- **contacts:** (*{ SideContact }*) Table of current contacts for the designated side. [READONLY]
- **exclusionzones:** (*{ Zone }*) Zones for the designated side [READONLY]
- **nonavzones:** (*{ Zone }*) Zones for the designated side [READONLY]
- **awareness:** (*Awareness*) [READONLY]
- **proficiency:** (*Proficiency*) [READONLY]
- **getexclusionzone:** (*method*) (ZoneGUID | ZoneName | ZoneDescription) Returns matching *Zone* or nil
- **getnonavzone:** (*method*) (ZoneGUID | ZoneName | ZoneDescription) Returns matching *Zone* or nil

Unit

Represents a active unit.

- **type:** (*string*) The type of object, may be 'Facility', 'Ship', 'Submarine', or 'Aircraft'. [READONLY]
- **name:** (*string*) The unit's name.
- **side:** (*string*) The unit's side. [READONLY]
- **guid:** (*string*) The unit's unique ID. [READONLY]
- **subtype:** (*string*) The unit's subtype (if applicable). [READONLY]
- **base:** (*Unit*) The unit's assigned base.
- **latitude:** (*Latitude*) The latitude of the unit.
- **longitude:** (*Longitude*) The longitude of the unit .
- **DBID:** (*number*) The database ID of the unit [READONLY]
- **altitude:** (*number*) The altitude of the unit in meters.
- **speed:** (*number*) The unit's current speed.
- **throttle:** (*Throttle*) The unit's current throttle setting.
- **autodetectable:** (*bool*) True if the unit is automatically detected.
- **holdposition:** (*bool*) True if the unit should hold.
- **holdfire:** (*{ table }*) Doctrine WCS setting for {air,surface,subsurface,land}. [READONLY]
- **heading:** (*number*) The unit's heading .
- **proficiency:** (*string*) The unit proficiency, "Novice"|0, "Cadet"|1,"Regular"|2, "Veteran"|3, "Ace"|4.
- **newname:** (*string*) If changing existing unit, the unit's new name .
- **course:** (*{ LatLon }*) The unit's course, as a table of lat/lon pairs
- **fuel:** (*{ Fuel }*) A table of fuel types used by unit.
- **mission:** (*Mission*) The unit's assigned mission. Can be changed by setting to the Mission name or guid (calls ScenEdit_AssignUnitToMission)
- **group:** (*Group*) The unit's group (if applicable). Can be changed assigning an existing or new name. It will try to create a group if new (experimental)
- **airbornetime:** (*number*) how long aircraft has been flying. [READONLY]
- **loadoutdbid:** (*number*) current aircraft loadout DBID. [READONLY]
- **unitstate:** (*string*) Message on unit status. [READONLY]
- **fuelstate:** (*string*) Message on unit fuel status. [READONLY]
- **weaponstate:** (*string*) Message on unit weapon status. [READONLY]
- **manualSpeed:** (*string* or *number*) Desired speed or 'OFF' to turn off manual mode
- **manualAltitude:** (*bool*) To turn on/off manual mode
- **damage:** (*{ table }*) Table {dp,flood,fires,startDp} of start and current DP, flood and fire level. [READONLY]
- **magazines:** (*{ Magazine }*) A table of magazines (with weapon loads) in the unit or group. Can be updated by [ScenEdit_AddWeaponToUnitMagazine](#) [READONLY]
- **mounts:** (*{ Mount }*) A table of mounts (with weapon loads) in the unit or group. Can be updated by [ScenEdit_AddReloadsToUnit](#) [READONLY]
- **components:** (*{ Component }*) A table of components on the unit. [READONLY]
- **ascontact:** (*{ table }*) A table {side,guid,name} of this unit seen from the other sides (as contacts). [READONLY]
- **weather:** (*{ Weather }*) Table of weather parameters (temp, rainfall, underrain, seastate)
- **delete:** (*method*) () Immediately removes unit object
- **filterOnComponent:** (*method*) (*type*) Filters unit on *type* of component and returns a *Component* table.
- **rangetotarget:** (*method*) ('contactid') Calculate flat distance to a contact location
- **areaTriggersFired:** (*{ table }*) Table of active 'in area' triggers that have fired for unit
- **OODA:** (*{ table }*) Table contain unit's "observe, orient, decide, act" values {evasion, targeting, detection}

ScenEdit only

- **refuel:** (*bool*) Trigger the unit to attempt an UNREP
- **RTB:** (*bool*) Trigger the unit to return to base
- **moveto:** (*bool*) Set a desired alt/depth instead of jumping to actual
- **manualSpeed:** (*string* or *number*) Desired speed or 'OFF' to turn off manual mode

- **manualAltitude:** (*bool*) To turn on/off manual mode

Zone

Exclusion/No navigation zone.

- **guid:** (*string*) The GUID of the zone. [READONLY]
- **description:** (*string*) The description of the zone.
- **isactive:** (*bool*) Zone is currently active.
- **area:** (*{ ZoneMarker }*) A set of reference points marking the zone. [READONLY]
- **affects:** (*{ unitTypes }*) List of unit types (ship, submarine, aircraft, facility)
- **locked:** (*bool*) Zone is locked.
- **markas:** (*Posture*) Posture of violator of exclusion zone.

Data types

Altitude

Altitude. ... is the height or depth of a unit. The altitude is displayed in meters when accessing the data. It can be set using either meters or feet by adding M or FT after it. The default is M if just a number is used.

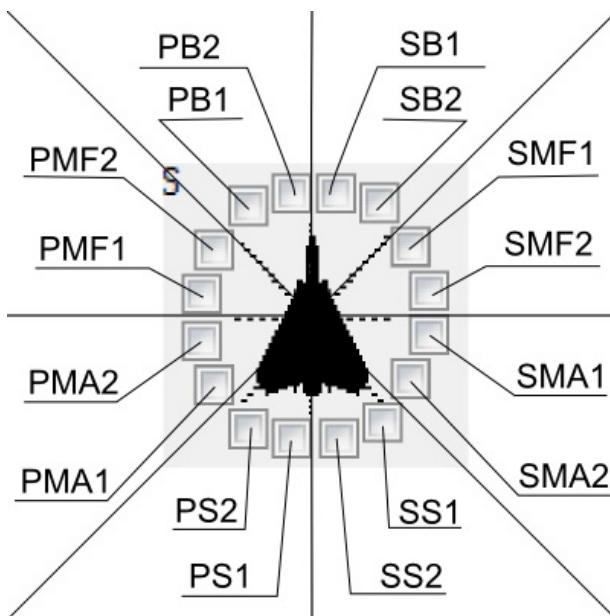
Usage:

```
{altitude='100 FT'} or {altitude='100 M'} or {altitude='100'}
```

Arc

Arcs. ... for sensor and mounts.

Arc codes:



DateTime

DateTime. ... is displayed and changed as per LOCALE e.g. US would be 'MM/DD/YYYY HH:MM:SS AM/PM' eg '8/13/2002 12:14 PM'

KeyStore

KeyStore. The simulation allows for user data to be stored within the save file. This is done by associating **keys** with **values**. Key/value pairs added to the persistent store is retained when the game is saved and resumed. Keys and values are both represented as non-nil strings.

- **key:** (*string*) The key to associate the value with
- **value:** (*string*) The value to keep

Latitude

Latitude. ... is degrees N or S of the equator as 'S 60.20.10' or as +/- as -60.336. The data in the tables is held as a signed number.

Longitude

Longitude. ... is degrees E or W of Greenwich line as 'W 60.20.10' or as +/- as -60.336. The data in the tables is held as a signed number.

Size

Size. ... various size attributes (eg flightsize in mission).

When setting the value, either the number or the description (in quotes) can be used.

Flight size:

0	None*
1	SingleAircraft
2	TwoAircraft
3	ThreeAircraft
4	FourAircraft
6	SixAircraft

Flight quantity:

0	None
1	Flight_x1
2	Flight_x2
3	Flight_x3
4	Flight_x4
6	Flight_x6
8	Flight_x8
12	Flight_x12
All	All

Group size:

0	None*
1	SingleVessel
2	TwoVessel
3	ThreeVessel
4	FourVessel
6	SixVessel

*This can also be set by using a value of 'all'.

Stance

Stance/Posture. ... how one side sees another.

When setting the value, either the number or the description (in quotes) can be used. Stance codes:

0	Neutral
1	Friendly

2	Unfriendly
3	Hostile
4	Unknown

TargetTypeWRA

Target type.

None	1001
Decoy	1002
Air_Contact_Unknown_Type	1999
Aircraft_Unspecified	2000
Aircraft_5th_Generation	2001
Aircraft_4th_Generation	2002
Aircraft_3rd_Generation	2003
Aircraft_Less_Capable	2004
Aircraft_High_Perf_Bombers	2011
Aircraft_Medium_Perf_Bombers	2012
Aircraft_Low_Perf_Bombers	2013
Aircraft_High_Perf_Recon_EW	2021
Aircraft_Medium_Perf_Recon_EW	2022
Aircraft_Low_Perf_Recon_EW	2023
Aircraft_AEW	2031
Helicopter_Unspecified	2100
Guided_Weapon_Unspecified	2200
Guided_Weapon_Supersonic_Sea_Skimming	2201
Guided_Weapon_Subsonic_Sea_Skimming	2202
Guided_Weapon_Supersonic	2203
Guided_Weapon_Subsonic	2204
Guided_Weapon_Ballistic	2211
Satellite_Unspecified	2300
Surface_Contact_Unknown_Type	2999
Ship_Unspecified	3000
Ship_Carrier_0_25000_tons	3001
Ship_Carrier_25001_45000_tons	3002
Ship_Carrier_45001_95000_tons	3003
Ship_Carrier_95000_tons	3004
Ship_Surface_Combatant_0_500_tons	3101
Ship_Surface_Combatant_501_1500_tons	3102
Ship_Surface_Combatant_1501_5000_tons	3103
Ship_Surface_Combatant_5001_10000_tons	3104

Ship_Surface_Combatant_10001_25000_tons	3105
Ship_Surface_Combatant_25001_45000_tons	3106
Ship_Surface_Combatant_45001_95000_tons	3107
Ship_Surface_Combatant_95000_tons	3108
Ship_Amphibious_0_500_tons	3201
Ship_Amphibious_501_1500_tons	3202
Ship_Amphibious_1501_5000_tons	3203
Ship_Amphibious_5001_10000_tons	3204
Ship_Amphibious_10001_25000_tons	3205
Ship_Amphibious_25001_45000_tons	3206
Ship_Amphibious_45001_95000_tons	3207
Ship_Amphibious_95000_tons	3208
Ship_Auxiliary_0_500_tons	3301
Ship_Auxiliary_501_1500_tons	3302
Ship_Auxiliary_1501_5000_tons	3303
Ship_Auxiliary_5001_10000_tons	3304
Ship_Auxiliary_10001_25000_tons	3305
Ship_Auxiliary_25001_45000_tons	3306
Ship_Auxiliary_45001_95000_tons	3307
Ship_Auxiliary_95000_tons	3308
Ship_Merchant_Civilian_0_500_tons	3401
Ship_Merchant_Civilian_501_1500_tons	3402
Ship_Merchant_Civilian_1501_5000_tons	3403
Ship_Merchant_Civilian_5001_10000_tons	3404
Ship_Merchant_Civilian_10001_25000_tons	3405
Ship_Merchant_Civilian_25001_45000_tons	3406
Ship_Merchant_Civilian_45001_95000_tons	3407
Ship_Merchant_Civilian_95000_tons	3408
Submarine_Surfaced	3501
Subsurface_Contact_Unknown_Type	3999
Submarine_Unspecified	4000
Land_Contact_Unknown_Type	4999
Land_Structure_Soft_Unspecified	5000
Land_Structure_Soft_Building_Surface	5001
Land_Structure_Soft_Building_Reveted	5002
Land_Structure_Soft_Structure_Open	5005
Land_Structure_Soft_Structure_Reveted	5006
Land_Structure_Soft_Aerostat_Moring	5011
Land_Structure_Hardened_Unspecified	5100

Land_Structure_Hardened_Building_Surface	5101
Land_Structure_Hardened_Building_Reveted	5102
Land_Structure_Hardened_Building_Bunker	5103
Land_Structure_Hardened_Building_Underground	5104
Land_Structure_Hardened_Structure_Open	5105
Land_Structure_Hardened_Structure_Reveted	5106
Runway_Facility_Unspecified	5200
Runway	5201
Runway_Grade_Taxiway	5202
Runway_Access_Point	5203
Radar_Unspecified	5300
Mobile_Target_Soft_Unspecified	5400
Mobile_Target_Soft_Mobile_Vehicle	5401
Mobile_Target_Soft_Mobile_Personnel	5402
Mobile_Target_Hardened_Unspecified	5500
Mobile_Target_Hardened_Mobile_Vehicle	5501
Underwater_Structure	5601
Air_Base_Single_Unit_Airfield	5801

TimeStamp

TimeStamp. ... is a representation of time defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970

Weather

weather conditions.

- temp: (*number*) Temperature (average for GetWeather(), or time-of-day for unit/contact)
- rainfall: (*number*) Rainfall rate
- undercloud: (*number*) Fraction under cloud
- seastate: (*number*) Sea state

Functions

GetScenarioTitle ()

Name of the scenario.

Returns:

(*string*) The title of the scenario

ScenEdit_AddExplosion (explosion)

Detonates a warhead at a specified location.

Parameters:

- explosion: (*Explosion*) Describes the explosion.

Usage:

```
ScenEdit_AddExplosion ({warheadid=253, lat=unit.latitude, lon=unit.longitude, altitude=unit
```

ScenEdit_AddMission (SideNameOrId, MissionNameOrId, MissionType, MissionOptions)

Add new mission.

Parameters:

- SideNameOrId: (*string*) The mission side
- MissionNameOrId: (*string*) The mission name
- MissionType: (*string*) The mission type (Strike, Ferry, Patrol, etc)
- MissionOptions: (*NewMission*) The mission specific options as a table

Returns:

(*Mission*) A mission descriptor of the added mission or nil otherwise.

Usage:

```
local mission = ScenEdit_AddMission('USA', 'Marker strike', 'strike', {type='land'})
```

ScenEdit_AddReferencePoint (descriptor)

Add a new reference point.

This function adds a new reference point as defined by a descriptor. The descriptor **must** contain at least a name, side, latitude and longitude.

Parameters:

- descriptor: (*ReferencePoint*) The reference point details to create

Returns:

(*ReferencePoint*) The reference point wrapper for the new reference point.

Usage:

```
ScenEdit_AddReferencePoint({side="United States", name="Downed Pilot", lat=0.1, lon=4, high
```

ScenEdit_AddReloadsToUnit (descriptor)

Adds weapons into a mount.

Parameters:

- descriptor: (*Weapon2Mount*) Describes the weapon and mount to update

Returns:

(*number*) Number of items added to the magazine

Usage:

```
ScenEdit_AddReloadsToUnit({unitname='Mech Inf #1', w_dbid=773, number=1, w_max=10})
```

ScenEdit_AddSide (table)

Add side.

...

Parameters:

- table: (*table*) The side

Returns:

(*Side*)

ScenEdit_AddUnit (unit)

Adds a unit based on a descriptor.

Parameters:

- **unit:** (*NewUnit*) The unit details to add

Returns:

(*Unit*) A complete descriptor for the added unit

Usage:

-

```
ScenEdit_AddUnit({type = 'Aircraft', name = 'F-15C Eagle', loadoutid = 16934,
heading = 0, dbid = 3500, side = 'NATO', Latitude="N46.00.00",Longitude="E25.00.00",
altitude="5000 ft",autodetectable="false",holdfire="true",proficiency=4})
```

-

```
ScenEdit_AddUnit({type = 'Air', unitname = 'F-15C Eagle', loadoutid = 16934, dbid = 3500,
```

ScenEdit_AddWeaponToUnitMagazine (descriptor)

Adds weapons into a magazine.

Parameters:

- **descriptor:** (*Weapon2Magazine*) Describes the weapon and magazine to update

Returns:

(*number*) Number of items added to the magazine

Usage:

```
ScenEdit_AddWeaponToUnitMagazine({unitname='Ammo', w_dbid=773, number=1, w_max=10})
```

ScenEdit_AddZone (sideName, zoneType, table)

Add no-nav or exclusion zone.

...

Parameters:

- **sideName:** (*string*) Side name/GUID
- **zoneType:** (*string*) Type of zone to add: 0 = no-nav, 1 = exclusion
- **table:** (*{ table }*) Description, Isactive, Area { of RPs }, Affects { of UnitTypes }, MarkAs (exc only), Locked (non only)

Returns:

(*{ Zone }*)

ScenEdit_AssignUnitAsTarget (AUnitNameOrIDOrTable, MissionNameOrID)

Assigns targets to a Strike mission.

'UnitX' can be used as a unit descriptor. Contacts can also be assigned. Refer to the VP_ functions for details

Parameters:

- **AUnitNameOrIDOrTable:** (*string* or *table*) The name/GUID of the unit, or a table of name/GUID to add to target list
- **MissionNameOrID:** (*string*) The mission to update

Returns:

(*{ GUID }*) A table of targets added

Usage:

-

```
ScenEdit_AssignUnitAsTarget({'target1', 'target2'}, 'Land strike')
```

•

```
ScenEdit_AssignUnitAsTarget('UnitX', 'Land strike')
```

ScenEdit_AssignUnitToMission (unitname, mission, escort)

Assign a unit to a mission.

The function takes a unit and mission descriptor, and assigns the unit to the mission if it exists. Produces a pop up error (not catchable) if the unit or mission does not exist. The 'UnitX' can be used as the unit descriptor

Parameters:

- unitname: (*string*) The name/GUID of the unit to assign
- mission: (*string*) The mission name/GUID to use
- escort: (*bool*) [Default=False] If the mission is a strike one, then assign unit to the 'Escort' for the strike

Returns:

(*boolean*) True/False for Successful/Failure

Usage:

```
ScenEdit_AssignUnitToMission("Bar #1", "Strike")
```

ScenEdit_AttackContact (attackerID, contactId, options)

Attack a contact ... as an auto-target or manual target with weapon allocation

Parameters:

- attackerID: (*string*) The unit attacking as GUID or Name
- contactId: (*string*) The contact being attacked as GUID or Name (GUID is better as the name can change as its classification changes)
- options: (*{ AttackOptions }*) Contains type of attack and weapon allocation if required)

Returns:

(*bool*) True if attack successful assigned

Usage:

```
ScenEdit_AttackContact(attackerID, contactId ,{mode='1', mount=438, weapon=1413, qty=10}) -
```

ScenEdit_CurrentTime ()

Get the current scenario time.

Returns:

(*TimeStamp*) The UTC Unix timestamp of the current time in-game.

Usage:

```
local now = ScenEdit_CurrentTime()
local elapsed = now - timeFromLastTriggered
if elapsed > 60*5 then
  -- been more than 5 minutes, set the lastTriggered time to now
  timeFromLastTriggered = now
end
```

ScenEdit_DeleteMission (SideNameOrId, MissionNameOrId)

Delete mission. .. unassigns any units attached to it.

Parameters:

- SideNameOrId: (*string*) The mission side
- MissionNameOrId: (*string*) The mission name

Returns:

(*bool*) True if mission has been removed.

Usage:

```
local mission = ScenEdit_AddMission('USA', 'Marker strike', 'strike', {type='land'})
```

ScenEdit_DeleteReferencePoint (selector)

Delete a reference point.

Given a reference point selector, this function will remove it.

Parameters:

- selector: (*ReferencePointSelector*) The reference point to delete.

Returns:

(*bool*) True if deleted

ScenEdit_DeleteUnit (unit)

Deletes unit. .. and no event triggers.

Parameters:

- unit:

Usage:

```
ScenEdit_DeleteUnit({side="United States", unitname="USS Abcd"})
```

ScenEdit_EndScenario ()

Ends the current scenario.

Usage:

```
ScenEdit_EndScenario()
```

ScenEdit_ExecuteEventAction (EventDescriptionOrID)

Execute a Lua Event action script. .. but does not show results of execution of the action

Parameters:

- EventDescriptionOrID: (*string*) The description or ID of the event action

Returns:

(*string*) "Ok" on execution or nothing.

ScenEdit_ExecuteSpecialAction (eventNameOrId)

Execute a Lua Special action script .. but does not show results of execution of the action

Parameters:

- eventNameOrId: (*string*) The name or ID of the event action

Returns:

(*string*) "Ok" on execution or nothing.

ScenEdit_ExportMission (SideNameOrId, MissionNameOrId)

Export mission parameters. ... as a XML file in folder Command_base/Defaults. [Experimental as this should really be treated like an attachment so can be imported with Scenario]

Parameters:

- SideNameOrId: (*string*) The mission side
- MissionNameOrId: (*string*) The mission name

Returns:

(*{ guid }*) Mission GUID exported.

Usage:

```
local mission = ScenEdit_ExportMission('USA', 'Marker strike')
```

ScenEdit_FillMagsForLoadout (unit, loadoutid, quantity)

Fill a unit's magazine(s) with aircraft stores ... for a specified loadout ID. (options)

Parameters:

- unit: (*UnitSelector*) The unit to select (name or GUID)
- loadoutid: (*number*) ID of the desired loadout
- quantity: (*number*) How many 'packs' of the specified loadout to populate for

Returns:

List of reports for successful (or not) additions

Usage:

```
ScenEdit_FillMagsForLoadout('{unit='RAF Lakenheath', loadoutid=45162, quantity=12}')
```

ScenEdit_GetContact (contact)

Fetches a contact based on a selector.

This function is mostly identical to [ScenEdit_GetUnit](#) except that it references contacts on a side,

Parameters:

- contact: (*ContactSelector*) The contact to select. Must be defined by a side and contact GUID for that side.

Returns:

(*Contact*) A contact descriptor if it or nil otherwise.

Usage:

```
ScenEdit_GetContact({side="United States", guid="c4114322-900c-428d-a3e3-0af701e81a7a"})
```

ScenEdit_GetContacts (side)

Contacts from a side.

Parameters:

- side: (*string*) The name/guid of the side's contacts.

Returns:

(*{ Contacts }*) Table of contact details

Usage:

```
local con = ScenEdit_GetContacts('south korea')
```


ScenEdit_GetDoctrine (selector)

Gets the doctrine of the designated object.

This function looks up the doctrine of the object selected by selector, and throws an exception if the unit does not exist.

Parameters:

- selector: (*DoctrineSelector*) The selector for the object to look up

Returns:

(*Doctrine*) The doctrine of the selected object

Usage:

```
ScenEdit_GetDoctrine({side="Soviet Union"}).use_nuclear_weapons
```

ScenEdit_GetDoctrineWRA (selector)

Gets the WRA doctrine.

Returns the WRA setting In a table For the selected side/mission/unit based On the target type. Nothing returned Or empty values means that the weapon Is Not For the target type.

Parameters:

- selector: (*DoctrineWRASelector*) The selector for the object to look up

Returns:

(*DoctrineWRA*) The WRA doctrine of the selected object

Usage:

-

```
ScenEdit_GetDoctrineWRA({guid = 'a1a52edf-3541-4b55-bea4-58d4e1ab11dc', contact_id='Boeing'})
```

-

```
ScenEdit_GetDoctrineWRA({side='sidea', target_type='Surface_Contact_Unknown_Type', weapon_type='Surface_Contact_Unknown_Type'})
```

ScenEdit_GetKeyValue (key)

Gets the value for a key from the persistent key store.

This function retrieves a value put into the store by [ScenEdit_SetKeyValue](#). The keys must be identical.

Parameters:

- key: (*string*) The key to fetch the associated value of

Returns:

(*string*) The value associated with the key. "" if none exists.

Usage:

```
ScenEdit_SetKeyValue("A", "2")
ScenEdit_GetKeyValue("A") -- returns "2"
```

ScenEdit_GetMission (SideNameOrId, MissionNameOrId)

Get details of a mission.

Parameters:

- SideNameOrId: (*string*) The mission side

- **MissionNameOrId:** (*string*) The mission name

Returns:

(*Mission*) A mission descriptor if the mission exists or nil otherwise.

Usage:

```
local mission = ScenEdit_GetMission('USA', 'CV CAP Left')
```

ScenEdit_GetReferencePoints (selector)

Get a set of reference point(s).

Given a reference point selector, the function will return a table of the reference point descriptors.

Parameters:

- **selector:** (*ReferencePointSelector*)

Returns:

(*{ReferencePoint}*) The table of reference point descriptors for the selector

Usage:

```
local points = ScenEdit_GetReferencePoints({side="United States", area={"rp-100", "rp-101", "
```

ScenEdit_GetScenHasStarted ()

Has the scenario started?

Returns:

(*bool*) True if the scenario has started

Usage:

```
local state = ScenEdit_GetScenHasStarted()
```

ScenEdit_GetScore (side)

Get a given side's score.

Parameters:

- **side:** (*string*) The name of the side

Returns:

(*num*) The side's score

Usage:

```
ScenEdit_GetScore("PlayerSide")
```

ScenEdit_GetSideIsHuman (sidename)

Returns true if side is played by a human player

Parameters:

- **sidename:** (*string*) The name of the side to check if is human controlled

Returns:

True iff the side specified is human

ScenEdit_GetSideOptions (options)

Get side options. ... for components on unit. (options)

Parameters:

- options: (*SideSelector*)

Returns:

(*{ SideOption }*) The side options

Usage:

```
ScenEdit_GetSideOptions({side='SideA'})
```

ScenEdit_GetSidePosture (sideA, sideB)

Gets the posture of one side towards another.

Parameters:

- sideA: (*string*) The name of the first side
- sideB: (*string*) The name of the second side

Returns:

The posture of sideA towards sideB, one of 'N', 'F', 'H', or 'A'.

ScenEdit_GetSpecialAction (action_info)

Gets the properties of an existing special action.

Parameters:

- action_info: (*SpecialAction*) The special action to retrieve

ScenEdit_GetUnit (unit)

Fetches a unit based on a selector.

This function is mostly identical to [ScenEdit_SetUnit](#) except that if no unit is selected by the selector portion of `unit`, then the function returns nil instead of producing an exception.

Parameters:

- unit: (*UnitSelector*) The unit to select.

Returns:

(*Unit*) A complete unit descriptor if the unit exists or nil otherwise.

Usage:

```
ScenEdit_GetUnit({side="United States", unitname="USS Test"})
```

ScenEdit_GetWeather ()

Get the current weather conditions.

Returns:

(*table*) Table of weather parameters [temp (average), rainfall, undercloud, seastate]

Usage:

```
ScenEdit_GetWeather()
```

ScenEdit_HostUnitToParent (host_info)

Hosts a unit to the specified host. The unit will be moved from any location, including flying or hosted elsewhere.

Parameters:

- `host_info`: (*HostUnit*) The information about the hosting request

ScenEdit_ImportInst (side, filename)

Imports an inst file.

Parameters:

- `side`: (*string*) The side to import the inst file as
- `filename`: (*string*) The filename of the inst file

ScenEdit_ImportMission (SideNameOrId, MissionNameOrId)

Import mission parameters. ... from a XML file in folder Command_base/Defaults. [Experimental as this should really be treated like an attachment so can be imported with Scenario]

Parameters:

- `SideNameOrId`: (*string*) The mission side
- `MissionNameOrId`: (*string*) The mission name

Returns:

(*guid*) Mission GUID imported.

Usage:

```
local mission = ScenEdit_ExportMission('USA', 'Marker strike')
```

ScenEdit_InputBox (string)

Open an input box with the passed prompt.

Parameters:

- `string`: (*string*) The string to display to the user

Returns:

Data entered into box

ScenEdit_KillUnit (unit)

Kill unit. ... and triggers event. (unit)

Parameters:

- `unit`:

Returns:

(*bool*) True if successful

Usage:

```
ScenEdit_KillUnit({side='SideA', unitname='ship'})
```

ScenEdit_MsgBox (string, style)

Show a message box with a passed string.

Style numbers 1 = OK and Cancel buttons. 2 = Abort, Retry, and Ignore buttons. 3 = Yes, No, and Cancel buttons 4 = Yes and No buttons. 5 = Retry and Cancel buttons.

Parameters:

- `string`: (*string*) The string to display to the user
- `style`: (*num*) The style of the message box

Returns:

button number pressed.

ScenEdit_PlayerSide ()

The player's current side.

Returns:

(*string*) The name of the current side

Usage:

```
local side = ScenEdit_PlayerSide()
```

ScenEdit_RefuelUnit (unitOptions)

Cause unit to refuel.

The unit should follow it's AAR configuration. You can force it use a specific tanker or ones from a set of missions.

Parameters:

- unitOptions: (*RefuelOptions*) The unit and refueling options.

Returns:

(*String*) If successful, then empty string. Else message showing why it failed to

Usage:

-

```
ScenEdit_RefuelUnit({side="United States", unitname="USS Test"})
```

-

```
ScenEdit_RefuelUnit({side="United States", unitname="USS Test", tanker="Hose #1"})
```

-

```
ScenEdit_RefuelUnit({side="United States", unitname="USS Test", missions={"Pitstop"}})
```

ScenEdit_RemoveSide (table)

Remove side from play. This removes ALL units and contacts.

Parameters:

- table: (*{ table }*) The side

Returns:

(*Side*)

ScenEdit_RemoveUnitAsTarget (AUnitNameOrIDOrTable, MissionNameOrID)

Removes targets from a Strike mission.

The 'UnitX' can be used as a unit descriptor

Parameters:

- AUnitNameOrIDOrTable: (*string* or *table*) The name/GUID of the unit, or a table of name/GUID to remove from target list
- MissionNameOrID: (*string*) The mission

Returns:

(*{ GUID }*) A table of targets removed

Usage:

```
ScenEdit_RemoveUnitAsTarget({'target1', 'target2'}, 'Land strike')
```

ScenEdit_RunScript (script)

Runs a script from a file. The file **script** must be inside the [Command base directory]/Lua directory, or else the game will not be able to load it. You can make the file point to files within a sub-directory of this, as in 'library/cklib.lua' The file to find will be of the form [Command base directory]/Lua/[**script**] A file can also be loaded indirectly from an attachment **ScenEdit_UseAttachment**

Parameters:

- **script**: The file containing the script to run.

Usage:

```
ScenEdit_RunScript('mylibrary.lua')
```

ScenEdit_SetDoctrine (selector, doctrine)

Set the doctrine of the designated object.

This function uses selector to find the thing to modify, then modifies the doctrine of that object based on the given object. Can be used to affect doctrine for Side, Mission, Unit/Group

Parameters:

- **selector**: (*DoctrineSelector*) The selector for the object to modify.
- **doctrine**: (*Doctrine*) A table of doctrines to update

Usage:

-

```
ScenEdit_SetDoctrine({side="Soviet Union"}, {kinematic_range_for_torpedoes = "AutomaticA
```

-

```
ScenEdit_SetDoctrine({side="Soviet Union", mission="ASW PATROL"}, {kinematic_range_for_t
```

-

```
ScenEdit_SetDoctrine({side="Soviet Union", unitname="Bear #2"}, {use_nuclear_weapons= "y
```

ScenEdit_SetDoctrineWRA (selector, options)

Sets the WRA doctrine.

Returns the WRA setting like GetDoctrineWRA

The values below can be used in the option settings 'inherit' = reverts to the side level setting 'system' = reverts to the database level setting (does not apply to 'firing_range') 'max' = use the appropriate maximum setting 'none' = not to be used

Parameters:

- **selector**: (*DoctrineWRASelector*) The selector for the object to update
- **options**: (*{ WRA }*) Table of settings {qtysalvo, shootersalvo, firingrange, selfdefence}. The order IS IMPORTANT as no keys used.

Returns:

(*DoctrineWRA*) The WRA doctrine of the selected object

Usage:

-

```
ScenEdit_SetDoctrineWRA({guid = 'a1a52edf-3541-4b55-bea4-58d4e1ab11dc', target_type='Surf
```

•

```
ScenEdit_SetDoctrineWRA({guid = 'a1a52edf-3541-4b55-bea4-58d4e1ab11dc', target_type='Surf
```

ScenEdit_SetEMCON (type, name, emcon)

Sets the EMCON of the selected object. Select the object by specifying the type and the object's name.

Type is the type of object to set the EMCON on. It can be one of 4 values:

- **Side** - Set an entire side's EMCON (e.g. United States using active radar)
- **Mission** - Set the EMCON for a mission (e.g. Minesweepers active sonar)
- **Group** - Set the EMCON for an entire group (e.g. Package #20 active radar)
- **Unit** - Set the EMCON for a single group (e.g. Hornet #14 passive radar)

emcon is a compound structure. The string follows the following grammar, with each clause separated by a semicolon (;)

- **Inherit** indicates that the output EMCON should be at least the parent's EMCON. Inherit must come first.
- A transmitter "set" statement. Each is of the form **type=status**, where type can be any one of **Radar**, **Sonar**, and **OECD**, and status can be any one of **Passive** or **Active**.

Parameters:

- **type**: (*string*) The type of the thing to set the EMCON on.
- **name**: (*string*) The name or GUID of the object to select.
- **emcon**: (*string*) The new EMCON for the object.

Usage:

•

```
ScenEdit_SetEMCON('Side', 'NATO', 'Radar=Active;Sonar=Passive')
```

•

```
ScenEdit_SetEMCON('Mission', 'ASW Patrol #1', 'Inherit;Sonar=Active')
```

•

```
ScenEdit_SetEMCON('Unit', 'Camel 2', 'OECD=Active')
```

ScenEdit_SetKeyValue (key, value)

Sets the value for a key in the persistent key store.

This function allows you to add values, associated with keys, to a persistent store **KeyStore** that is retained when the game is saved and resumed. Keys and values are both represented as non-nil strings. The value is retrieved by **ScenEdit_GetKeyValue**.

Parameters:

- **key**: (*string*) The key to associate with
- **value**: (*string*) The value to associate

Usage:

```
ScenEdit_SetKeyValue("A", "B")
ScenEdit_GetKeyValue("A") -- returns "B"
```

ScenEdit_SetLoadout (loadoutinfo)

Sets the loadout for a unit

Parameters:

- **loadoutinfo:** (*LoadoutInfo*) The loadout information to apply

Returns:

(*bool*) True

ScenEdit_SetMission (SideName, MissionNameOrId, MissionOptions)

Set details for a mission.

Parameters:

- **SideName:** (*string*) The mission side
- **MissionNameOrId:** (*string*) The mission name
- **MissionOptions:** (*Mission*) The mission options as a table.

Returns:

(*Mission*) A mission descriptor if the mission exists or nil otherwise.

Usage:

```
local mission = ScenEdit_SetMission('USA', 'CV CAP Left', {TankerUsage=1, OnStation=2})
```

ScenEdit_SetReferencePoint (descriptor)

Update a reference point(s) with new values.

Given a valid *ReferencePointSelector* as part of the descriptor, the function will update the values contained in the descriptor. Values may be omitted from the descriptor if they are intended to remain unmodified. The 'area' selector is useful for changing some common attribute, like locking or highlighting, in bulk.

Additional key=value options are;

NEWNAME='string' to rename a reference point

TOGGLEHIGHLIGHTED = True to flip the reference point(s) highlight

CLEAR = True to remove the 'relative to' of the reference point(s)

Parameters:

- **descriptor:** (*ReferencePoint*) A valid selector with other values to update.

Returns:

(*ReferencePoint*) The reference point descriptor for the reference point or first one in the area.

Usage:

-

```
ScenEdit_SetReferencePoint({side="United States", name="Downed Pilot", lat=0.5})
```

-

```
ScenEdit_SetReferencePoint({side="United States", name="Downed Pilot", lat=0.5, lon="N50
```

-

```
ScenEdit_SetReferencePoint({side="United States", area={"rp-100", "rp-101", "rp-102", "rp-10
```

ScenEdit_SetScore (side, score, reason)

Sets a given side's score.

Parameters:

- **side:** (*string*) The name/GUID of the side
- **score:** (*num*) The new score for the side
- **reason:** (*string*) The reason for the score to change

Returns:

(*num*) The new score for the side

Usage:

```
ScenEdit_GetScore("PlayerSide", 20)
```

ScenEdit_SetSideOptions (options)

Set side options. ... AWARENESS and PROFICIENCY. (options)

Parameters:

- options: (*SideOptions*) The side items to be changed.

Returns:

(*{ SideOption }*) The side options

Usage:

```
ScenEdit_SetSideOptions('{side='SideA',awareness='OMNI',PROFICIENCY='ace'})
```

ScenEdit_SetSidePosture (sideAName, sideBName, posture)

Set the posture of a side towards another.

This will set side A's posture towards side B to the specified posture. This is the same as **Stance**, but only the first character of the name is used as shown in the table

Posture codes:

F	Friendly
H	Hostile
N	Neutral
U	Unfriendly

Parameters:

- sideAName: (*string*) Side A's name or GUID
- sideBName: (*string*) Side B's name or GUID
- posture: (*string*) The posture of side A towards side B

Returns:

(*boolean*) True/False for Successful/Failure

Usage:

```
ScenEdit_SetSidePosture("LuaSideA", "LuaSideB", "H")
```

ScenEdit_SetSpecialAction (action_info)

Sets the properties of an existing special action.

Parameters:

- action_info: (*SpecialAction*) The special action to modify

ScenEdit_SetUnit (unit)

Sets the properties of a unit that already exists.

Parameters:

- unit: (*Unit*) The unit to edit. Must be at least a selector.

Returns:

(*Unit*) A complete descriptor for the added unit

Usage:

•

```
ScenEdit_SetUnit({side="United States", unitname="USS Test", lat = 5})
```

•

```
ScenEdit_SetUnit({side="United States", unitname="USS Test", lat = 5})
```

•

```
ScenEdit_SetUnit({side="United States", unitname="USS Test", lat = 5, lon = "N50.20.10"})
```

•

```
ScenEdit_SetUnit({side="United States", unitname="USS Test", newname="USS Barack Obama"})
```

•

```
ScenEdit_SetUnit({side="United States", unitname="USS Test", heading=0, HoldPosition=1, ...})
```

ScenEdit_SetUnitDamage (options)

Set unit damage. ... for components on unit. (options)

Parameters:

- options: (*DamageOptions*)

Returns:

(*Component*) The unit's components object

Usage:

```
ScenEdit_SetUnitDamage({side='SideA', unitname='Ship', fires=1, components={ {'rudder', 'Med' ...
```

ScenEdit_SetUnitSide (sidedesc)

Changes the side of a unit

Parameters:

- sidedesc: (*SideDescription*) The description of how to change the unit's side

ScenEdit_SetWeather (temperature, rainfall, undercloud, seastate)

Set the current weather conditions.

This function takes four numbers that describe the desired weather conditions. These conditions are applied globally.

Parameters:

- temperature: (*number*) The current baseline temperature (in deg C). Varies by latitude.
- rainfall: (*number*) The rainfall rate, 0-50.
- undercloud: (*number*) The amount of sky that is covered in cloud, 0.0-1.0
- seastate: (*number*) The current sea state 0-9.

Returns:

(*boolean*) True/False for Successful/Failure

Usage:

```
ScenEdit_SetWeather(math.random(0,25), math.random(0,50), math.random(0,10)/10.0, math.rand
```

ScenEdit_SpecialMessage (side, message)

Displays a special message consisting of the HTML text **message** to side **side**.

Parameters:

- **side**: (*string*) The side name/guid to display the message on
- **message**: (*string*) The HTML text to display to the player

ScenEdit_UnitX ()

Activating Unit .. that triggered the current Event. Otherwise, a nil is returned.

Note that UnitX() can also be used as a shortcut for ScenEdit_UnitX()

Returns:

(*Unit*) The triggering unit

Usage:

-

```
ScenEdit_SetUnitDamage({side=UnitX().side, unitname=UnitX().name, fires=1, components={
```

-

```
local unit = ScenEdit_UnitX()
```

ScenEdit_UnitY ()

Detecting Unit ... from a Unit Detected event trigger. Otherwise, a nil is returned.

Note that UnitY() can also be used as a shortcut for ScenEdit_UnitY()

Returns:

(*table*) The detecting unit and sensors used as { unit = {unit object}, sensor = {[1] = {name, type}, [2] = {name, type}, etc} }

Usage:

-

```
ScenEdit_SetUnitDamage({side=UnitY().side, unitname=UnitY().name, fires=1, components={
```

-

```
local by = ScenEdit_UnitY()
print('Y:'); print( by)
print('Detected by: ');print( by.unit.name .. ' of type ' .. by.unit.type .. ' from ' ..
print('Sensor: ');print( by.sensor[1].name .. ' of type ' .. by.sensor[1].type);
```

ScenEdit_UpdateEvent (EventDescriptionOrID, options)

Sets the Lua script(s) of an event.

Parameters:

- **EventDescriptionOrID**: (*string*) The event name/GUID to update
- **options**: (*EventUpdate*) The event options

Returns:

(*table* or *bool*) Table of event option (new or previous values)

ScenEdit_UpdateUnit (options)

Update items on a unit.

Parameters:

- options: (*UpdateUnit*) The unit sensor/? details to update

Returns:

(*Unit*) The updated unit

Usage:

-

```
ScenEdit_UpdateUnit({guid='2cd64757-1b66-4609-ad56-df41bee652e5',mode='add_sensor',dbid=...
```

-

```
ScenEdit_UpdateUnit({guid='2cd64757-1b66-4609-ad56-df41bee652e5',mode='remove_sensor',dbid=...
```

ScenEdit_UseAttachment (attachment)

Import an attachment into the scene.

Parameters:

- attachment: (*string*) Either the name of the attachment or the GUID of the attachment

ScenEdit_UseAttachmentOnSide (attachment, sidename)

Use an attachment on a side (used for .inst files as attachments).

Parameters:

- attachment: (*string*) Either the name of the attachment or the GUID of the attachment
- sidename: (*string*) The name of the side to import the attachment into

Tool_DumpEvents ()

Dump scenario events. ... useful for checking. Also writes a file to the scenario folder()

Returns:

(*xml*) Dump of events with trigger/condition/action

Tool_EmulateNoConsole ()

Emulates no console. ... useful running event code in the console and seeing how it behaves as an 'event'(mode)

Returns:

(*boolean*) If interactive

Tool_Range ()

Get range between points.

The points can be a GUID of a unit/contact or a latitude/longitude point.(fromHere, toHere)

Returns:

(*number*) The horizontal distance in NM

Usage:

-

```
Tool_Range('8269b881-20ce-4f2e-baa0-6823e46d55a4', '004aa55d-d553-428d-a727-26853737c8f4')
```

•

```
Tool_Range( {latitude='33.1991547589118', longitude='138.376876749942'}, '8269b881-20ce-4
```

VP_GetContact (ContactGUID)

Get contact details This will get the information about a contact unit

Parameters:

- ContactGUID: (*VPContactSelector*) The contact selector to interrogate

Returns:

(*Contact*) The information associated with the contact

Usage:

```
local vp = VP_GetSide({name = "NATO"})
local cp = vp.contacts --List Of contacts
local guid = cp[12].objectid -- a specific contact
local contact = VP_GetContact({guid=guid}) -- details of contact as distinct to unit detail
```

VP_GetSide (side)

Side information from player's perspective.

Gets a side object from the perspective of the player.

Parameters:

- side: (*SideSelector*) The side to get information about.

Returns:

(*Side*) Information about the side selected, from the perspective of the player.

Usage:

```
local a = VP_GetSide({Side = 'sidea'}) -- a side object
local z = a.nonavzones --List of no-nav zones for the side if you need to find it
local n = a : getnonavzone(z[1].guid) -- required zone object for a particular zone
n.isactive = false -- turn it off
```

VP_GetUnit (ActiveOrContact)

Get unit details This will get the information about an active unit or a contact unit

Parameters:

- ActiveOrContact: (*UnitSelector*) The unit selector to interrogate

Returns:

(*Unit*) The information associated with the unit

World_GetCircleFromPoint (table)

Returns a circle around point.

Parameters:

- table: (*Point*)

Returns:

Table of points

World_GetElevation (location)

Returns the elevation in meters of a given position

Parameters:

- **location:** (*Point*) The position to find the elevation of

Returns:

The elevation of the point in meters

Tables

AAR

AAR. .. refueling options; these are updated by ScenEdit_SetMission()

Fields:

- **use_refuel_unrep:** (*string*) This is same as the one from Doctrine setting
- **tankerUsage:** (*string* or *number*) Automatic(0), Mission(1)
- **launchMissionWithoutTankersInPlace:** (*bool*)
- **tankerMissionList:** (*{ mission name or guid }*) Table of missions to use as source of refuellers
- **tankerMinNumber_total:** (*number*)
- **tankerMinNumber_airborne:** (*number*)
- **tankerMinNumber_station:** (*number*)
- **maxReceiversInQueuePerTanker_airborne:** (*number*)
- **fuelQtyToStartLookingForTanker_airborne:** (*number*) Percentage of fuel (0-100)
- **tankerMaxDistance_airborne:** (*string* or *number*) Use 'internal' or set a range. The code will match the lowest available setting

CargoMission

CargoMission. .. options; these are updated by ScenEdit_SetMission()

Fields:

- **oneThirdRule:** (*bool*) True if activated
- **transitThrottleAircraft:** (*string*)
- **transitAltitudeAircraft:** (*string*)
- **stationThrottleAircraft:** (*string*)
- **stationAltitudeAircraft:** (*string*)
- **transitThrottleSubmarine:** (*string*)
- **transitDepthSubmarine:** (*string*)
- **stationThrottleSubmarine:** (*string*)
- **stationDepthSubmarine:** (*string*)
- **transitThrottleShip:** (*string*)
- **stationThrottleShip:** (*string*)
- **useFlightSize:** (*bool*) True if minimum size required
- **useGroupSize:** (*bool*) True if minimum size required
- **zone:** (*{ name or guid }*) Table of reference point names and/or GUIDs

Component

Component.

This identifies the component/item(s) that are present in a unit. See [Unit:filterOnComponent](#) on how to filter this table

Fields:

- **comp_guid:** (*string*) GUID
- **comp_dbid:** (*string*) Database ID
- **comp_name:** (*string*) Name
- **comp_type:** (*string*) Type of component (list ?? sensor, rudder, etc)
- **comp_status:** (*string*) Status
- **comp_damage:** (*string*) Damage Severity

EMmatch

EM matches .. details

Fields:

- dbid: *(number)* Databse id.
- name: *(string)* Matching id name.
- category: *(string)* Not applicable to weapons (Missile,Torpedo)
- type: *(string)* Not applicable to Facility
- subtype: *(number)* Type of item within the contact type (subtype is "Fighter" within scope of Aircraft)
- missile_defence: *(number)* Applicable to Facility and Ships

Explosion

Defines the warhead to detonate.

Fields:

- warheadid: *(number)* The ID of the warhead to detonate
- lat: *(Latitude)* The latitude of the detonation
- lon: *(Longitude)* The longitude of the detonation
- altitude: *(Altitude)* The altitude of the detonation

Usage:

```
{warheadid=253, lat=unit.latitude, lon=unit.longitude, altitude=unit.altitude}
```

FerryMission

FerryMission. .. options; these are updated by ScenEdit_SetMission()

Fields:

- ferryBehavior: *(string)* Values OneWay(0), Cycle(1), Random(2)
- ferryThrottleAircraft: *(string)*
- ferryAltitudeAircraft: *(string)*
- ferryTerrainFollowingAircraft: *(bool)*
- flightSize: *(Size)*
- minAircraftReq: *(string)*
- useFlightSize: *(bool)*

Fuel

Fuel.

The various types of fuel(s), and their state, carried by the unit. Use `ScenEditSetUnit()` to set the fuel rather than the unit.fuel... `ScenEditSetUnit({...,fuel={{'GasFuel',1500},{2001,8000}}..})` It is easier and less prone to error; you can use the fuel name or the fuel number code

Fuel Types:

1001	'NoFuel'
2001	AviationFuel
3001	DieselFuel
3002	OilFuel
3003	GasFuel
4001	Battery
4002	AirIndepedent
5001	RocketFuel
5002	TorpedoFuel
5003	WeaponCoast

Fields:

- fueltype: (*{ FuelState }*) The state of the type(s) of fuel in the unit.

Usage:

```
local fuel = u.fuel
fuel[3001].current = 400
u.fuel = fuel
```

FuelState

Status of a fuel type.

Fields:

- current: (*number*) The current fuel level of the type
- max: (*number*) How much can be stored for the type
- name: (*string*) Name of the fuel

Usage:

```
local fuel = u.fuel
fuel[3001].current = 400
u.fuel = fuel
```

HostUnit

Requests that a unit be hosted to another

Fields:

- HostedUnitNameOrID: (*string*) The name or GUID of the unit to put into the host
- SelectedHostNameOrID: (*string*) The name or GUID of the host to put the unit into

LatLon

A Position on the map. ... is referred to with latitude (north/south) and longitude (east/west) coordinates. These can be represented in two forms, degrees minutes seconds, and decimal degrees. The Command Lua API supports both forms.

A set of latitude & longitude to define a point. Within the simulation, the values are recorded in decimal degrees.

Fields:

- latitude: (*number*)
- longitude: (*number*)

LoadoutInfo

Information on a loadout to add/alter

Fields:

- UnitName: (*string*) The name/GUID of the unit to change the loadout on
- LoadoutID: (*number*) The ID of the new loadout; 0 = use the current loadout
- TimeToReady_Minutes: (*number*) How many minutes until the loadout is ready
- IgnoreMagazines: (*bool*) If the new loadout should rely on the magazines having the right weapons ready
- ExcludeOptionalWeapons: (*bool*) Exclude optional weapons from loadout (*optional*)

Magazine

Magazine.

Note that when dealing with a magazine in a unit, it may consist of one or more actual magazine 'blocks'. This is what is being referred to here, rather than the ONE magazine group for the unit/group.

Fields:

- `mag_capacity`: (*string*) Capacity|Storage
- `mag_dbid`: (*string*) Database ID
- `mag_guid`: (*string*) GUID
- `mag_name`: (*string*) Name
- `mag_weapons`: (*{ WeaponLoaded }*) Table of weapon loads in magazine

MineClearMission

MineClearMission. ... options; these are updated by `ScenEdit_SetMission()`

Fields:

- `oneThirdRule`: (*bool*)
- `transitThrottleAircraft`: (*string*)
- `transitAltitudeAircraft`: (*string*)
- `transitTerrainFollowingAircraft`: (*bool*)
- `stationThrottleAircraft`: (*string*)
- `stationAltitudeAircraft`: (*string*)
- `stationTerrainFollowingAircraft`: (*bool*)
- `transitThrottleSubmarine`: (*string*)
- `transitDepthSubmarine`: (*string*)
- `stationThrottleSubmarine`: (*string*)
- `stationDepthSubmarine`: (*string*)
- `transitThrottleShip`: (*string*)
- `stationThrottleShip`: (*string*)
- `flightSize`: (*Size*)
- `minAircraftReq`: (*string*)
- `useFlightSize`: (*bool*)
- `groupSize`: (*Size*)
- `useGroupSize`: (*bool*)
- `zone`: (*{ name or guid }*) Table of reference point names and/or guids

MineMission

MineMission. ... options; these are updated by `ScenEdit_SetMission()`

Fields:

- `oneThirdRule`: (*bool*)
- `transitThrottleAircraft`: (*string*)
- `transitAltitudeAircraft`: (*string*)
- `transitTerrainFollowingAircraft`: (*bool*)
- `stationThrottleAircraft`: (*string*)
- `stationAltitudeAircraft`: (*string*)
- `stationTerrainFollowingAircraft`: (*bool*)
- `transitThrottleSubmarine`: (*string*)
- `transitDepthSubmarine`: (*string*)
- `stationThrottleSubmarine`: (*string*)
- `stationDepthSubmarine`: (*string*)
- `transitThrottleShip`: (*string*)
- `stationThrottleShip`: (*string*)
- `flightSize`: (*Size*)
- `minaircraftreq`: (*string*)
- `useFlightSize`: (*bool*)
- `groupSize`: (*Size*)
- `useGroupSize`: (*bool*)
- `zone`: (*{ name or guid }*) Table of reference point names and/or guids
- `armingdelay`: (*time*) In format of 'days:hours:minutes:seconds' e.g. 1 day, 4 hours, 30 minutes would be '1:4:30:0'

Mount

Mount.

A mount is similar to a magazine, but it refers to the actual **loads** on the **weapon**, rather than a storage area.

Fields:

- `mount_dbid`: (*string*) Database ID
- `mount_guid`: (*string*) GUID

- mount_name: (*string*) Name
- mount_weapons: (*{ WeaponLoaded }*) Table of weapon loads on mount

PatrolMission

PatrolMission. ... options; these are updated by ScenEdit_SetMission()

Fields:

- oneThirdRule: (*bool*) True if activated
- checkOPA: (*bool*) True if can investigate outside zones
- checkWVR: (*bool*) True if can investigate within weapon range
- activeEMCON: (*bool*) True if active EMCON in patrol zone
- transitThrottleAircraft: (*string*)
- transitAltitudeAircraft: (*string*)
- transitTerrainFollowingAircraft: (*bool*) True if terrain following
- stationThrottleAircraft: (*string*)
- stationAltitudeAircraft: (*string*)
- stationTerrainFollowingAircraft: (*bool*) True if terrain following
- attackThrottleAircraft: (*string*)
- attackAltitudeAircraft: (*string*)
- attackTerrainFollowingAircraft: (*bool*) True if terrain following
- attackDistanceAircraft: (*string*)
- transitThrottleSubmarine: (*string*)
- transitDepthSubmarine: (*string*)
- stationThrottleSubmarine: (*string*)
- stationDepthSubmarine: (*string*)
- attackThrottleSubmarine: (*string*)
- attackDepthSubmarine: (*string*)
- attackDistanceSubmarine: (*string*)
- transitThrottleShip: (*string*)
- stationThrottleShip: (*string*)
- attackThrottleShip: (*string*)
- attackDistanceShip: (*string*)
- flightSize: (*Size*)
- minAircraftReq: (*string*)
- useFlightSize: (*bool*) True if min size required
- groupSize: (*Size*)
- useGroupSize: (*bool*) True if min size required
- prosecutionZone: (*{ name or guid }*) Table of reference point names and/or GUIDs
- patrolZone: (*{ name or guid }*) Table of reference point names and/or GUIDs

SideContact

Side's contact.

Fields:

- guid: (*string*) The GUID of the contact. Note that this is not the GUID of the unit, you must use VP_GetUnit to resolve it.
- name: (*string*) The name of the contact.

SideDescription

Information on how to change a unit's side. ORDER IS IMPORTANT

Fields:

- side: (*string*) The original side
- name: (*string*) The name/GUID of the unit
- newside: (*string*) The new side

SideSelector

Information to select a particular side.

Fields:

- GUID: (*string*) The GUID of the side to select. Preferred.
- Name: (*string*) The name of the side to select.

SideUnit

Side's unit.

Fields:

- **guid:** (*string*) The GUID of the unit. This is the actual guid.
- **name:** (*string*) The name of the unit.

StrikeMission

StrikeMission. .. options; these are updated by ScenEdit_SetMission(). Note that these are split between the escorts and strikers

Fields:

- **escortFlightSizeShooter:** (*Size*)
- **escortMinShooter:** (*number*)
- **escortMaxShooter:** (*number*)
- **escortResponseRadius:** (*number*)
- **escortUseFlightSize:** (*bool*) True if minimum size required
- **escortFlightSizeNonshooter:** (*Size*)
- **escortMinNonshooter:** (*number*)
- **escortMaxNonshooter:** (*number*)
- **escortGroupSize:** (*Size*)
- **escortUseGroupSize:** (*bool*) True if minimum size required
- **strikeOneTimeOnly:** (*bool*) True if activated
- **strikeMinimumTrigger:** (*string*)
- **strikeMax:** (*number*)
- **strikeFlightSize:** (*Size*)
- **strikeMinAircraftReq:** (*number*)
- **strikeUseFlightSize:** (*bool*) True if minimum size required
- **strikeGroupSize:** (*Size*)
- **strikeUseGroupSize:** (*bool*) True if minimum size required
- **strikeAutoPlanner:** (*bool*) True if activated
- **strikePreplan:** (*bool*) True if pre-planned target list only
- **strikeRadarUsage:** (*number*) Radar usage
- **strikeMinDist:** (*number*) Strike minimum distance
- **strikeMaxDist:** (*number*) Strike maximum distance

SupportMission

SupportMission. .. options; these are updated by ScenEdit_SetMission()

Fields:

- **oneThirdRule:** (*bool*)
- **transitThrottleAircraft:** (*string*)
- **transitAltitudeAircraft:** (*string*)
- **transitTerrainFollowingAircraft:** (*bool*)
- **stationThrottleAircraft:** (*string*)
- **stationAltitudeAircraft:** (*string*)
- **stationTerrainFollowingAircraft:** (*bool*)
- **transitThrottleSubmarine:** (*string*)
- **transitDepthSubmarine:** (*string*)
- **stationThrottleSubmarine:** (*string*)
- **stationDepthSubmarine:** (*string*)
- **transitThrottleShip:** (*string*)
- **stationThrottleShip:** (*string*)
- **flightSize:** (*Size*)
- **minAircraftReq:** (*string*)
- **useFlightSize:** (*bool*)
- **groupSize:** (*Size*)
- **useGroupSize:** (*bool*)
- **zone:** (*{ name or guid }*) Table of reference point names and/or guids
- **loopType:** (*string*) Values of ContinuousLoop(0) or SingleLoop(1)
- **onStation:** (*string*)
- **oneTimeOnly:** (*bool*)

- activeEMCON: (*bool*)
 - tankerOneTime: (*bool*)
 - tankerMaxReceivers: (*string*)
-

WRA

WRA settings. ... absence of fields implies that it is NOT used

Fields:

- qty_salvo: (*string*) Weapons per salvo ('Max' or a number)
 - shooter_salvo: (*string*) Shooters per salvo ('Max' or a number)
 - firing_range: (*string*) Firing range ('Max' or a number)
 - self_defence: (*string*) Self-defence range ('Max' or a number)
-

WeaponLoaded

Weapon loads on mount or in magazine.

Fields:

- wpn_guid: (*string*) GUID
 - wpn_current: (*string*) Current loads available
 - wpn_maxcap: (*string*) Maximum loads
 - wpn_default: (*string*) Default loads (to fill out)
 - wpn_dbid: (*string*) Database ID
 - wpn_name: (*string*) Name
-

ZoneMarker

Zone marker.

Fields:

- guid: (*string*) The GUID of the Reference Point.
- longitude: (*Longitude*)
- latitude: (*Latitude*)

generated by *LDoc 1.4.3*

Last updated 2017-05-21 00:27:01