SWEndor Scripting Guide - Context Functions

Version: v0.1
Created: 29 December 2019

CONTENTS

INTRODUCTION

This guide requires you to have read the Scripting Guide for writing scripts.

Context functions are the link between script code to in-game functions. These functions allow you to access parts of the game operation necessary for you build your own scenario.

Format

Context functions are called with the following format

Parameterless null function

Context function contract:
```
void Scene.FadeOut()
```
Example call in script:
```
Scene.FadeOut();
```

Parameterized null function

Context function contract:
```
void Scene.SetMaxBounds(float3 value)
```
Example call in script:
```
Scene.SetMaxBounds({10000,200,30000});
```

Parameterless value function

Context function contract:
```
int Player.GetActor()
```
Example call in script:
```
int playerID = Player.GetActor();
```

Parameterized value function

Context function contract:
```
string Actor.GetActorType(int actorID)
```
Example call in script:
```
string type = Actor.GetActorType(playerID);
```

`void` functions will return NULL, which hold no intrinsic value and cannot be operated with other values except equality / inequality checks. Other functions may return a value depending on the function's purpose.

SCRIPTING

`bool Script.TryCall(string script_name)`

Attempts to call the script function with the name `script_name`. If the script is not found, returns `false`.

`void Script.Call(string script_name)`

Transfers execution to the script function with the name `script_name`. If the script is not found, a runtime error will be thrown.

Error: `Attempted to call non-existent script '<script_name>'`

`void AddEvent(float delay, string script_name)`

Queues a script by this `script_name` to be played after a `delay` in in-game seconds. If the script does not exist, a runtime error will be thrown:

Error: `"Script event '<script_name>' does not exist!"`

SCENE

`void Scene.SetMaxBounds(float3 value)`

Sets the coordinate of the upper bound of the scenario boundary to this `value`. The scenario boundary limits the space where the player can travel in.

`void Scene.SetMinBounds(float3 value)`

Sets the coordinate of the lower bound of the scenario boundary to this `value`. The scenario boundary limits the space where the player can travel in.

`void Scene.SetMaxAIBounds(float3 value)`

Sets the coordinate of the upper bound of the AI boundary to this `value`. The AI will attempt to stay in the space within the boundary limits.

`void Scene.SetMinAIBounds(float3 value)`

Sets the coordinate of the lower bound of the AI boundary to this `value`. The AI will attempt to stay in the space within the boundary limits.

`void Scene.FadeOut()`

Initiate the fade out sequence. After fading out, the game checks if the GameWon game state is enabled. If yes, the scenario diverts to the Game Over menu, otherwise the game checks whether the GameWon game state is enabled. If yes, the scenario diverts to the Game Won menu, otherwise the game will fade in as normal.

MESSAGE

`void Message(string text, float duration, float3 color)`
`void Message(string text, float duration, float3 color, int priority)`

Displays a message `text` on the center of the screen with this `color` and for this `duration`. This message will not override any existing message with a higher `priority`. If `priority` is not defined, a value of zero is used.

CAMERA

```
void Camera.SetPlayerLook()
```

Sets the camera to use the player vision. (Follow the player craft)

```
void Camera.SetSceneLook()
```

Sets the camera to use the scene vision. Set the position of the camera and its target using Camera.SetSceneLook_LookAtX and Camera.SetSceneLook_LookFromX functions.

```
void Camera.SetDeathLook()
```

Sets the camera to use the death vision. (Circle the player craft)

```
void Camera.EnableFreeLook(bool enabled)
```

Enables / Disables free vision. When enabled, this removes player mouse control from the player craft. Instead, the player may use the mouse to rotate the camera around the craft.

```
void Camera.SetSceneLook_LookAtActor(int actorID)
void Camera.SetSceneLook_LookAtActor(int actorID, float3 offsetXYZ)
void Camera.SetSceneLook_LookAtActor(int actorID, float3 offsetXYZ, float3 offsetRelative)
```

Sets the scene camera target to the actor with the given `actorID`. If no such actor is found, do nothing.

If provided, `offsetXYZ` and/or `offsetRelative` determine world and relative offset positions from the actor body respectively.

```
void Camera.SetSceneLook_LookAtPoint(float3 point)
```

Sets the scene camera target to a fixed `point`.

```
void Camera.SetSceneLook_LookFromActor(int actorID)
void Camera.SetSceneLook_LookFromActor(int actorID, float3 offsetXYZ)
void Camera.SetSceneLook_LookFromActor(int actorID, float3 offsetXYZ, float3 offsetRelative)
```

Sets the scene camera position to the actor with the given `actorID`. If no such actor is found, do nothing.

If provided, `offsetXYZ` and/or `offsetRelative` determine world and relative offset positions from the actor body respectively.

```
void Camera.SetSceneLook_LookAtPoint(float3 point)
```

Sets the scene camera position to a fixed `point`.

SQUAD

```
int[] Squad.Spawn(string actorType, string squadName, string faction, int count, float spawnDelay,
bool entryByHyperspace, float3 position, float3 rotation, string formation, float formationSpacing,
float aiWaitDelay, string huntTargetType, string[] registries)
```

Spawns a squad of actors of this `actorType`, owned by this `faction`. The squad will have `count` number of members, will bear designations `squadName` followed by a number from 1 up to `count`. The squad will spawn in `spawnDelay` in-game seconds after this function call, at the specified `position` and `rotation`. If `entryByHyperspace` is enabled, the spawned craft will hyperspace in using `rotation` as the entry vector. Otherwise, the craft simply appears. All actors will be placed in `formation` with a spacing distance defined by `formationSpacing`. All actors will be in a squad with the first actor as the squad leader. All actors in the squad will register themselves in each group specified in `registries`.

After spawning in, the AI governing actors will first wait for `aiWaitDelay` seconds, then proceed to hunt the targets that matches the type given by `huntTargetType`.

This function returns an `int[]` with the actorIDs of the spawned actors.

```
bool Squad.JoinSquad(int actorID, int actor2ID)
```

The actor of this `actor2ID` will join the squad with actor `actorID`. If neither actor exists, do nothing and return `false`. Otherwise, returns `true`.

```
bool Squad.RemoveFromSquad(int actorID)
```

The actor of this `actorID` will be removed from any existing squad it is in. If the actor does not exist, do nothing and return `false`. Otherwise, returns `true`.

```
bool Squad.MakeSquadLeader(int actorID)
```

The actor of this `actorID` will be made the squad leader (hence first member) of its current squad. If the actor does not exist, do nothing and return `false`. Otherwise, returns `true`.

ACTOR

`int Actor.Spawn(string actorType, string name, string faction, string sidebarName, float spawnDelay, float3 position, float3 rotation, string[] registries)`

Spawns an actor of this `actorType`, owned by this `faction`. The actor will bear a designation `name`. If the actor appears on the sidebar, the display name `sidebarName` will be used. The actor will spawn in `spawnDelay` in-game seconds after this function call, at the specified `position` and `rotation`. The actor will register itself in each group specified in `registries`.

This function returns an `int` with the actorID of the spawned actor.

`void Actor.QueueAtSpawner(int actorID, int spawnerID)`

Removes an actor of this `actorID` from the world and place it in the spawn queue of an actor of this `spawnerID`. If neither actor exists, do nothing. Note that the spawner should have a hangar or equivalent spawner add-on if this actor is to be spawned into the world.

`string Actor.GetActorType(int actorID)`

Returns the actor type ID of an actor by this `actorID`. If the actor does not exist, return an empty string `""`.

`bool Actor.IsFighter(int actorID)`

Returns whether the actor with this `actorID` is a fighter. An actor is positively identified as a fighter if its TargetType definition includes the FIGHTER flag. If the actor does not exist, return `false`.

`bool Actor.IsLargeShip(int actorID)`

Returns whether the actor with this `actorID` is a large ship. An actor is positively identified as a large ship if its TargetType definition includes the SHIP flag. If the actor does not exist, return `false`.

`bool Actor.IsAlive(int actorID)`

Returns whether the actor with this `actorID` exists on the world and is not in the dead state. Note that the dying state still considered alive by this function.

`string Actor.GetFaction(int actorID)`

Returns the name of the faction this `actorID` belongs to. If the actor does not exist, return the neutral faction.

`void Actor.SetFaction(int actorID, string faction)`

Sets the actor's faction to `faction` belongs to.

`void Actor.AddToRegister(int actorID, string register)`

Adds this actor to a scenario register. If either the actor or register does not exist, do nothing.

`void Actor.RemoveFromRegister(int actorID, string register)`

Removes this actor to a scenario register. If either the actor or register does not exist, do nothing.

`float3 Actor.GetLocalPosition(int actorID)`

Returns the local position vector of the actor with this `actorID`. If the actor does not exist, return an empty `float3 {0,0,0}`.

`float3 Actor.GetLocalRotation(int actorID)`

Returns the local rotation vector of the actor with this `actorID`. Rotation is given in degrees. If the actor does not exist, return an empty `float3 {0,0,0}`.

`float3 Actor.GetLocalDirection(int actorID)`

Returns the local direction vector of the actor with this `actorID`. If the actor does not exist, return an empty `float3 {0,0,0}`.

`float3 Actor.GetGlobalPosition(int actorID)`

Returns the world position vector of the actor with this `actorID`. If the actor does not exist, return an empty `float3 {0,0,0}`.

`float3 Actor.GetGlobalRotation(int actorID)`

Returns the world rotation vector of the actor with this `actorID`. Rotation is given in degrees. If the actor does not exist, return an empty `float3 {0,0,0}`.

`float3 Actor.GetGlobalDirection(int actorID)`

Returns the world direction vector of the actor with this `actorID`. If the actor does not exist, return an empty `float3 {0,0,0}`.

`void Actor.SetLocalPosition(int actorID, float3 value)`

Sets the local position vector of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetLocalRotation(int actorID, float3 value)`

Sets the local rotation vector of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetLocalDirection(int actorID, float3 value)`

Sets the local direction vector of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.LookAtPoint(int actorID, float3 point)`

Sets the rotation vector of the actor with this `actorID` so that it faces the position with the coordinate `point`. The roll (z-rotation) of the actor will be zeroed. If the actor does not exist, do nothing.

`int[] Actor.GetChildren(int actorID)`

Returns an `int[]` with the actorIDs of the children of the actor with this `actorID`. If the actor does not exist, or if the children of the actor has not yet been spawned, return an `int[]` with zero members. Note that only children that have been spawned will count. As actors spawn their children after they are spawned in, it is recommended to use this function at least a few frames after requesting their spawn.

`int[] Actor.GetChildrenByType(int actorID, string type)`

Returns an `int[]` with the actorIDs of the children of the actor with this `actorID`, whose type ID matches `type`. If the actor does not exist, or if the children of the actor has not yet been spawned, return an `int[]` with zero members. Note that only children that have been spawned will count. As actors spawn their children after they are spawned in, it is recommended to use this function at least a few frames after requesting their spawn.

`float Actor.GetHP(int actorID)`

Returns the HP (shield + hull ratings) of the actor with this `actorID`. If the actor does not exist, return `0`.

`float Actor.GetShd(int actorID)`

Returns the shield rating of the actor with this `actorID`. If the actor does not exist, return `0`.

`float Actor.GetHull(int actorID)`

Returns the hull rating of the actor with this `actorID`. If the actor does not exist, return `0`.

`float Actor.GetMaxHP(int actorID)`

Returns the maximum HP (shield + hull ratings) of the actor with this `actorID`. If the actor does not exist, return `0`.

`float Actor.GetMaxShd(int actorID)`

Returns the maximum shield rating of the actor with this `actorID`. If the actor does not exist, return `0`.

`float Actor.GetMaxHull(int actorID)`

Returns the maximum hull rating of the actor with this `actorID`. If the actor does not exist, return `0`.

`void Actor.SetHP(int actorID, float value)`

Sets the HP (shield + hull ratings) of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetShd(int actorID, float value)`

Sets the shield rating of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetHull(int actorID, float value)`

Sets the hull rating of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetMaxHP(int actorID, float value)`

Sets the maximum HP (shield + hull ratings) of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetMaxShd(int actorID, float value)`

Sets the maximum shield rating of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.SetMaxHull(int actorID, float value)`

Sets the maximum hull rating of the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`float Actor.GetArmor(int actorID, string damageType)`

Returns the damage multiplier on the `damageType` to the actor with this `actorID`. If the actor does not exist, return `0`.

As of version 0.1, the accepted `damageType` values are: `COLLISION`, `LASER`, `MISSILE`, `TORPEDO`.

`void Actor.SetArmor(int actorID, string damageType, float value)`

Sets the damage multiplier on the `damageType` to the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

As of version 0.1, the accepted `damageType` values are: `COLLISION`, `LASER`, `MISSILE`, `TORPEDO`.

`void Actor.SetArmorAll(int actorID, float value)`

Sets the damage multiplier on all damage types to the actor with this `actorID` to this `value`. If the actor does not exist, do nothing.

`void Actor.RestoreArmor(int actorID)`

Restores the damage multipliers to the actor with this `actorID` back to its actor type definitions. If the actor does not exist, do nothing.

`var Actor.SetProperty(int actorID, string propertyName)`
`void Actor.SetProperty(int actorID, string propertyName, var value)`

Gets / Sets a property of the actor with this `actorID`.

As of version 0.1, the following is a list of supported properties:

Property	Type	Description
Regen		
`Regen.NoRegen`	`bool`	If set to `true`, the actor will be immune to regeneration, although it can still apply regeneration to its parent / children / siblings.
`Regen.Self`	`float`	The rate in which an actor replenishes its shield rating per in-game second.
`Regen.Child`	`float`	The rate in which an actor replenishes each of its children's shield rating per in-game second.
`Regen.Parent`	`float`	The rate in which an actor replenishes its parent's shield rating per in-game second.
`Regen.Sibling`	`float`	The rate in which an actor replenishes each of its sibling's shield rating per in-game second.
AI		
`AI.CanEvade`	`bool`	Determines whether the AI governing this actor is allowed to evade upon getting hit.
`AI.CanRetaliate`	`bool`	Determines whether the AI governing this actor is allowed to retaliate upon getting hit.

`AI.HuntWeight`	`float`	Determines the weight for AI hunt calculations. The greater the number, the more probable the actor will be selected as a hunt target.
Movement		
`Movement.ApplyZBalance`	`bool`	Determines whether an actor will self-correct its z-rotation towards zero.
`Movement.MinSpeed`	`float`	Determines the actor's minimum speed
`Movement.MaxSpeed`	`float`	Determines the actor's maximum speed
`Movement.Speed`	`float`	Determines the actor's current speed
`Movement.MaxSpeedChangeRate`	`float`	Determines the maximum rate where the actor can change its speed.
`Movement.MaxTurnRate`	`float`	Determines the maximum rate of turn for the actor
Health		
`Health.HP`	`float`	Determines the actors current HP (shield + hull ratings)
`Health.Shd`	`float`	Determines the actors current shield rating
`Health.Hull`	`float`	Determines the actors current hull rating
`Health.MaxHP`	`float`	Determines the actors maximum HP (shield + hull ratings)
`Health.MaxShd`	`float`	Determines the actors maximum shield rating
`Health.MaxHull`	`float`	Determines the actors maximum hull rating
Spawner		
`Spawner.Enabled`	`float`	Determines whether the actor spawner is enabled. To achieve full functionality of the spawner functions, a hangar-type actor must be assigned as a child of this actor
`Spawner.SpawnTypes`	`string[]`	Determines the possible of actor types to spawn. Each iteration is selected randomly.
`Spawner.SpawnsRemaining`	`int`	Determines the number of spawns remaining. This refers to the number of spawn sets. If the actor's hangar spawns 4 actors at a time, the effective number of spawned actors is 4 times the number of spawned sets.
Transform		
`Transform.Scale`	`float`	Determines whether an actor will self-correct its z-rotation towards zero.
`Transform.Position`	`float3`	Determines the actor's local position
`Transform.Rotation`	`float3`	Determines the actor's local rotation (in degrees)
`Transform.Direction`	`float3`	Determines the actor's local direction

Misc		
InCombat	bool	Determines whether the actor is considered a combat object. If set to `false`, it is ignored in AI hunt calculations and aggressive tracking calculations.
Name	string	Determines the name of this actor, if shown. If set to an empty string, the name (not id) of the actor type will be used.
SideBarName	string	Determines the display name of this actor on the side bar, if shown. If set to an empty string, the name of the actor type will be used.

AI

```
bool AI.QueueFirst(int actorID, string actionType, …)
bool AI.QueueNext(int actorID, string actionType, …)
bool AI.QueueLast(int actorID, string actionType, …)
```

Queues an action to the AI of an actor of this `actorID`.

QueueFirst queues the action before the current action, replacing it.
QueueNext queues the action immediately after the current action.
QueueLast queues the action after the last queued action.

If the actor does not exist or the `actionType` is not well defined, return `false`. An error may be thrown if a valid `actionType` is used with malformed input (e.g. incorrect number / types of parameters).

The rest of the parameters depend on what action type is used. The following table lists the possible combinations:

Action Type	Parameters (#Optional parameters)	Description
`"idle"`	`none`	Brings the actor to Idle. Usually this action generates a Hunt action and completes instantly. If there is nothing to hunt, then simply wait.

`"hunt"`	`#string targetType`	The actor calculates its next target.
		Accepted `targetType` values:
		`LASER,`
		`MUNITION,`
		`FLOATING,`
		`FIGHTER,`
		`SHIP,`
		`STRUCTURE,`
		`ADDON,`
		`SHIELDGENERATOR,`
		`ANY`
		
		Default `targetType` value is `"ANY"`.

`"selfdestruct"`	`none`	Sets the actor to DEAD.
`"delete"`	`none`	Removes the actor from the world. The dead state is skipped.
`"lock"`	`none`	Applies a lock on the actor. This lock can be unlocked using `AI.UnlockOne`
`"wait"`	`#float duration`	Sets the actor to wait for a specified number of in-game seconds. Default `duration` value is `5`.
`"evade"`	`#float duration`	Sets the actor to take evasion action for a specified number of in-game seconds. Default `duration` value is `2.5`.

"move"	**float3 destination,** **float speed,** **#float close_distance,** **#bool can_interrupt**	The actor will move towards a specific **destination**, maintaining a certain **speed**, until the actor is within **close_distance** distance from the point. Default **close_distance** value is -1. (Determined by actor type) Default **can_interrupt** value is true.
"forcedmove"	**float3 destination,** **float speed,** **#float close_distance,** **#float duration**	The actor will move towards a specific **destination**, maintaining a certain **speed**, until the actor is within **close_distance** distance from the point or when **duration** is reached. Default **close_distance** value is -1. (Determined by actor type) Default **duration** value is 999999.
"rotate"	**float3 destination,** **float speed,** **#float close_angle,** **#bool can_interrupt**	The actor will rotate towards a specific **destination**, maintaining a certain **speed**, until the actor is within **close_angle** angle from the point. Default **close_abgle** value is 0.1 degrees. Default **can_interrupt** value is true.
"hyperspacein"	**float3 destination**	The actor travels by hyperspace into this **destination**.
"hyperspaceout"	**none**	The actor begins hyperspace sequence exiting this battlefield.
"attackactor"	**int actorid,** **#float follow_distance,** **#float close_distance,** **#bool can_interrupt,** **#bool hunt_interval**	The actor will attack an actor of this **actorid**, maintaining a certain **follow_distance**, for up to **hunt_interval** in-game seconds. It attempts to evade if the distance is less than **close_distance.** Default **follow_distance** value is -1. (Determined by actor type) Default **close_distance** value is -1. (Determined by actor type) Default **can_interrupt** value is true. Default **hunt_interval** value is 15.
"followactor"	**int actorid,** **#float follow_distance,** **#bool can_interrupt**	The actor will follow an actor of this **actorid**, maintaining a certain **follow_distance**. Default **follow_distance** value is 500. Default **can_interrupt** value is true.
"setgamestateb"	**string state_name,** **bool state**	Sets the game state of this **state_name** to a **state** value.

```
void AI.UnlockOne(int actorID)
```

Removes an AI lock from the actor of this `actorID`. If the actor does not exist, return `false`.

```
string AI.ClearQueue(int actorID)
string AI.ForceClearQueue(int actorID)
```

Clears the AI queue for the actor of this `actorID`. If the actor does not exist, return `false`.

ClearQueue will only clear actions until the first non-interruptible action.
ForceClearQueue will empty the queue regardless.

GAME

```
float GetGameTime()
```

Returns the current game time.

```
float GetLastFrameTime()
```

Returns the duration of the last frame. This is related to FPS. (Typically a frame lasts 0.033 seconds in 30 FPS)

```
string GetDifficulty()
```

Returns the selected difficulty of the scenario.

```
string GetPlayerActorType()
```

Returns the selected actor type of the scenario.

```
string GetPlayerName()
```

Returns the player name, defined by the scenario.

```
int GetStageNumber()
```

Returns the current stage number.

```
void SetStageNumber(int value)
```

Sets the current stage number to this `value`.

```
bool GetGameStateB(string state_name)
bool GetGameStateB(string state_name, bool defaultValue)
```

Gets the value of a boolean game state with this `state_name`.

If defined, `defaultValue` is return if the game state is not defined (no value assigned).

```
float GetGameStateF(string state_name)
float GetGameStateF(string state_name, float defaultValue)
```

Gets the value of a floating-point game state with this `state_name`.

If defined, `defaultValue` is return if the game state is not defined (no value assigned).

```
string GetGameStateB(string state_name)
string GetGameStateB(string state_name, string defaultValue)
```

Gets the value of a string game state with this `state_name`.

If defined, `defaultValue` is return if the game state is not defined (no value assigned).

```
void SetGameStateB(string state_name, bool value)
```

Sets the value of a boolean game state with this `state_name` to `value`.

```
void SetGameStateF(string state_name, float value)
```

Sets the value of a floating-point game state with this `state_name` to `value`.

```
void SetGameStateS(string state_name, string value)
```

Sets the value of a string game state with this `state_name` to `value`.

```
int GetRegisterCount(string register_name)
```

Gets the number of actors attached to the game register with this `register_name`.

As of v0.1, the supported registers are:

CriticalAllies (allies shown in cyan in the side-bar)
CriticalEnemies (enemies shown in red in the side-bar)

```
float GetTimeSinceLostWing()
float GetTimeSinceLostShip()
float GetTimeSinceLostStructure()
```

Returns the number of in-game seconds since the player's faction lost an actor of a specific target type.

GetTimeSinceLostWing tracks actors with the target type FIGHTER.
GetTimeSinceLostShip tracks actors with the target type SHIP.
GetTimeSinceLostStructure tracks actors with the target type STRUCTURE.

PLAYER

bool Player.AssignPlayer(int actorID)

Assigns the player to an actor of this `actorID`. If the actor does not exist, return `false`.

---

int Player.GetActor()

Returns the `actorID` of the player actor. If the actor does not exist, return `-1`.

---

void Player.RequestSpawn()

Sets the state to allow spawners (hangars, player spawners) to spawn the player. Note that the spawner may not spawn the player immediately.

---

void Player.SetMovementEnabled(bool enabled)

Sets whether the player movement controls are enabled.

---

void Player.SetAI(bool enabled)

Sets whether the player is controlled by AI. Overrides movement controls.

---

void Player.SetLives(int lives)

Sets the number of player lives.

---

void Player.DecreaseLives()

Decrements the player lives by 1.

SCORE

void Score.SetScorePerLife(float score)

Sets the increment to the score requirement for a new +1 life every time the score requirement is reached.

---

void Score.SetScoreForNextLife(float score)

Sets the score requirement for a new +1 life. When this score is reached, the player receives +1 life, and this value is incremented by the value set in `Score.SetScorePerLife`.

---

void Score.ResetScore(float score)

Resets the score records to default. This resets the current score, kill count, hit count, death count and kill records to 0.

FACTION

```
void Faction.Add(string name, float3 color)
```

Creates a new faction with an identifier `name`. Units belonging to this faction will be represented by this `color`.

```
float3 Faction.GetColor(string faction)
```

Returns the color of `faction`. If the faction does not exist, a runtime error will be thrown:

> Error: `At least one of the factions is not defined.`

```
void Faction.SetColor(string faction1, float3 color)
```

Sets the color of `faction` to `color`. If the faction does not exist, a runtime error will be thrown:

> Error: `At least one of the factions is not defined.`

```
void Faction.MakeAlly(string faction1, string faction2)
```

Sets two factions by the name of `faction1` and `faction2` to be allied to each other. The relationship is mutual. If at least one of the factions does not exist, a runtime error will be thrown:

> Error: `At least one of the factions is not defined.`

```
void Faction.MakeEnemy(string faction1, string faction2)
```

Sets two factions by the name of `faction1` and `faction2` to be enemies of each other. The relationship is mutual. If at least one of the factions does not exist, a runtime error will be thrown:

> Error: `At least one of the factions is not defined.`

```
int Faction.GetWingCount(string faction)
int Faction.GetShipCount(string faction)
int Faction.GetStructureCount(string faction)
```

Gets the number of actors of a certain target type belonging to the faction.

GetWingCount counts the number of actors with the target type FIGHTER.
GetShipCount counts the number of actors with the target type SHIP.
GetStructureCount counts the number of actors with the target type STRUCTURE.

If the faction does not exist, a runtime error will be thrown:

> Error: `The faction is not defined.`

```
int Faction.GetWingLimit(string faction)
int Faction.GetShipLimit (string faction)
int Faction.GetStructureLimit (string faction)
```

Gets the limit of actors of a certain target type belonging to the faction. The limit counts down whenever a new actor of matching type is spawned. When the limit reaches zero, spawner will be blocked from spawning more actors of the matching type. The player craft is not affected by this limitation.

If the limit is set to `-1`, the limit is ignored. Actors spawned using either `Squad.Spawn` or `Actor.Spawn` ignores this limit.

GetWingCount gets the limit of actors with the target type FIGHTER.
GetShipCount gets the limit of actors with the target type SHIP.
GetStructureCount gets the limit of actors with the target type STRUCTURE.

If the faction does not exist, a runtime error will be thrown:

> Error: `The faction is not defined.`

```
void Faction.SetWingLimit(string faction, int limit)
void Faction.SetShipLimit (string faction, int limit)
void Faction.SetStructureLimit (string faction, int limit)
```

Sets the limit of actors of a certain target type belonging to the faction. The limit counts down whenever a new actor of matching type is spawned. When the limit reaches zero, spawners will be blocked from spawning more actors of the matching type. The player craft is not affected by this limitation.

If the limit is set to `-1`, the limit is ignored. Actors spawned using either `Squad.Spawn` or `Actor.Spawn` ignores this limit.

SetWingCount sets the limit of actors with the target type FIGHTER.
SetShipCount sets the limit of actors with the target type SHIP.
SetStructureCount sets the limit of actors with the target type STRUCTURE.

If the faction does not exist, a runtime error will be thrown:

> Error: `The faction is not defined.`

```
int Faction.GetWingSpawnLimit(string faction)
int Faction.GetShipSpawnLimit (string faction)
int Faction.GetStructureSpawnLimit (string faction)
```

Gets the spawn limit of actors of a certain target type belonging to the faction. If the number of actors of a certain type present equals or exceeds this value, spawners will be blocked from spawning more actors of the matching type. The player craft is not affected by this limitation.

If the limit is set to `-1`, the limit is ignored. Actors spawned using either `Squad.Spawn` or `Actor.Spawn` ignores this limit.

GetWingSpawnCount gets the limit of actors with the target type FIGHTER.
GetShipSpawnCount gets the limit of actors with the target type SHIP.
GetStructureSpawnCount gets the limit of actors with the target type STRUCTURE.

If the faction does not exist, a runtime error will be thrown:

> Error: `The faction is not defined.`

```
void Faction.SetWingSpawnLimit(string faction, int limit)
void Faction.SetShipSpawnLimit (string faction, int limit)
void Faction.SetStructureSpawnLimit (string faction, int limit)
```

Sets the spawn limit of actors of a certain target type belonging to the faction. If the number of actors of a certain type present equals or exceeds this value, spawners will be blocked from spawning more actors of the matching type. The player craft is not affected by this limitation.

If the limit is set to `-1`, the limit is ignored. Actors spawned using either `Squad.Spawn` or `Actor.Spawn` ignores this limit.

SetWingSpawnCount sets the limit of actors with the target type FIGHTER.
SetShipSpawnCount sets the limit of actors with the target type SHIP.
SetStructureSpawnCount sets the limit of actors with the target type STRUCTURE.

If the faction does not exist, a runtime error will be thrown:

> Error: `The faction is not defined.`

AUDIO

```
int Audio.GetMood()
```

Used for dynamic music only. Gets the current audio `mood`.

```
void Audio.SetMood(int mood)
```

Used for dynamic music only. Sets the audio mood to `mood`. Negative values can be used to trigger interrupts.

```
void Audio.SetMusic(string piece_name)
void Audio.SetMusic(string piece_name, bool loop)
void Audio.SetMusic(string piece_name, bool loop, int position_ms)
void Audio.SetMusic(string piece_name, bool loop, int position_ms, int end_ms)
```

Sets the current music to the piece `piece_name`. This will stop any music that is currently playing.

If defined, `loop` determines whether the music piece will loop, `position_ms` determines the start position (in milliseconds) of the piece, and `end_ms` determines the end position of the piece.

```
void Audio.SetMusicLoop(string piece_name)
void Audio.SetMusicLoop(string piece_name, int position_ms)
```

Sets the loop music to the piece `piece_name`. If the current music is not looped, the loop music will begin after the current music has finished playing.

If defined, `position_ms` determines the start position (in milliseconds) of the piece.

```
void Audio.SetDynMusic(string piece_name)
```

Sets the dynamic music to the piece `piece_name`. This piece must be identified as a dynamic music piece. This will stop any music that is currently playing.

```
void Audio.StopMusic()
```

Stops the currently playing music.

```
void Audio.PauseMusic()
```

Pauses the currently playing music.

```
void Audio.ResumeMusic()
```

Resumes the previous paused music.

```
void Audio.SetSound(string sound_name)
void Audio.SetSound(string sound_name, float volume)
void Audio.SetSound(string sound_name, float volume, bool loop)
```

Plays the sound `sound_name`. Multiple instances of the same sound can be played concurrently with this function.

If defined, `volume` determines the sound volume (1.0 being 100%), and `loop` determines the sound will loop.

```
void Audio.SetSoundSingle(string sound_name)
void Audio.SetSoundSingle(string sound_name, bool interrupt)
void Audio.SetSoundSingle(string sound_name, bool interrupt, float volume)
void Audio.SetSoundSingle(string sound_name, bool interrupt, float volume, bool loop)
```

Plays the sound `sound_name`. Other instances of the same sound will be stopped.

If defined, `interrupt` determines whether the music piece will interrupt, `volume` determines the sound volume (1.0 being 100%), and `loop` determines the sound will loop.

```
void Audio.StopSound(string sound_name)
```

Stops playing the sound `sound_name`.

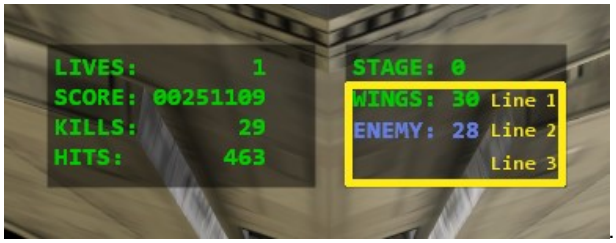
```
void Audio.StopAllSounds ()
```

Stops all sounds.

UI

```
void UI.SetLine1Color(float3 color)
void UI.SetLine2Color(float3 color)
void UI.SetLine3Color(float3 color)
```

Sets the color of line X on the programmable UI display (below Stage number).



```
void UI.SetLine1Text(string text)
void UI.SetLine2Text(string text)
void UI.SetLine3Text(string text)
```

Sets the text of line X on the programmable UI display (below Stage number). The text should be within 15 characters.

MATH

```
float GetDistance(float3 point1, float3 point2)
```

Returns the distance between `point1` and `point2`.

```
float GetActorDistance(int actorID1, int actorID2)
float GetActorDistance(int actorID1, int actorID2, float limit)
```

Returns the distance between two actors `actorID1` and `actorID2`. If `limit` is defined, returns `limit` if the distance is greater than it.

```
int Int(float value)
```

Returns the integer value of `value`.

```
float Max(float value1, float value2)
```

Returns the greater value between `value1` and `value2`.

```
float Max(float value1, float value2)
```

Returns the smaller value between `value1` and `value2`.

```
string FormatAsTime (float value)
```

Returns a string by the format <minute>:<second> (00:00) that represents `value` in seconds. For example, 96 seconds would be represented by “01:36”. Fractional seconds are omitted.

MISCELLANEOUS

```
bool IsNull(var value)
```

Returns `true` if the value is a `null` value. (`null` is returned by `void` functions, or if a variable has not been assigned any value)

```
float Random()
```

Returns a random value between 0 and 1.

```
int Random(int max)
```

Returns a random integer between 0 (inclusive) and `max` (exclusive).

```
int Random(int min, int max)
```

Returns a random integer between `min` (inclusive) and `max` (exclusive).

```
var GerArrayElement(var[] array, int index)
```

Returns the value at position `index` of the `array`. If the array is not a valid array, a runtime error is thrown:

Error: `Attempted to apply GetArrayElement on a non-array object.`

The table below indicates the valid variable types that can be used:

Array Type (var[])	Element Type (var)	Effect
`bool[]`	`bool`	Returns `array[index]`
`int[]`	`int`	
`float[]`	**`float`**	
`string[]`	`string`	
`string`	`string`	Returns a string with the character at position `index` of the `array`