Solstice Documentation

Release v0.1

jmd

October 07, 2015

Contents

Solstice is a scripting library and optionally a combat engine replacement plugin system for Neverwinter Nights (NWN). My main goal was to expand NWN from a platform for building adventures to also allow building new/different rulesets. The project also aims at a tight level of integration: where you could create a server with no NWN scripts, but also where you could add/replace a script without having to change a single line of NWN script.

This project is open source and licensed under the MIT License (any mentions of GPL v2 in the docs are wrong). Contributions, pull requests, feedback on API design, questions, bug reports (use the GitHub issues here) are welcome and appreciated.

Currently, this is used in production on my own server. Some things are geared towards it. You can get a sense of what I've done here. The code there is likewise MIT Licensed.

If you're curious how everything comes together take a look at the Rules module, which is the core of the system. If you want to see the details of the combat engine replacement see `examples/core_combat_engine.lua`, just be aware that that code is atypical in that it's written for the highest performance possible.

CHAPTER 1

Status

- Status: Very near beta.
- Working on getting nicer docs.
- No build instructions yet.

CHAPTER 2

Dependencies

- Linux
- Luajit 2.0
- luafilesystem
- lualogging
- nwnx_effects
- nwnx_solstice

CHAPTER 3

git

- `develop` is the branch with ongoing development.
- `master` will (hopefully) be stable releases.

CHAPTER 4

Scripting

NWN scripts are mapped to functions in the global Lua namespace. There is no concept of `void main() {...}` or `int StartingConditional() {...}`. In the former case, in Lua it's merely a function that returns no value; in the latter a function that returns a boolean. There is no concept of `OBJECT_SELF`, the object is passed explicitly.

Solstice is a more object oriented framework. e.g. rather than `SendMessageToPC(pc, "Hello")` in Solstice it would be `pc:SendMessage("Hello")`.

Examples:

A normal 'script':

```
function hello_world(obj)
  obj:SendMessage('Hello, world!')
end
```

A 'script' that can be used in a conversation conditional:

```
function is_epic_char(obj)
  return obj:GetHitDice() > 20
end
```

This has a some side effects:

- Lua function names that you're using as 'scripts' are limited to 16 characters, like script file names.
- Script editing needs to be done in external editor.
- Scripts placed into events, like in the dialog editor, will not display their contents.
- Many Lua functions can be placed into a single file.
- None of these external Lua scripts count towards resource limits and if you build your module they will be consider 'missing'.

4.1 Getting Started

In order to get Solstice up and running you'll need a few things.

4.1.1 System

A Linux server or VM with NWN setup.

4.1.2 LuaRocks

Install LuaRocks.

- Ubuntu: `sudo apt-get install luarocks`.

Install the required LuaRocks. Note that the following will require the ability to compile C files.

Install luafilesystem: `sudo luarocks install luafilesystem`

Install lualogging: `sudo luarocks install lualogging`

Install luadbi for your database (this is optional, if you don't use the `system.database`):

- MySQL: `sudo luarocks install luadbi-mysql`
- Sqlite3: `sudo luarocks install luadbi-sqlite3`
- PostgreSQL: `sudo luarocks install luadbi-postgresql`
- lua-inih: `sudo luarocks install inih`

4.1.3 NWNX

Download at install the NWNX plugins here: **TODO**

4.1.4 Solstice

Download and install Solstice or clone git repository.

4.1.5 Lua Scripts

Create a directory for your Lua scripts in your NWN install directory. All the examples use `lua`.

Create your settings.lua, preload.lua, and constant.lua in your lua script directory. See the examples provided.

4.2 Attack

This module contains ctypes and functions required for interacting with Solstice's and NWN's internal combat attack data structures.

4.2.1 ctypes

> **Warning:** The following ctypes **must** be syncronized with nwnx_solstice. If not bad, bad things will happen. Good news is it's unlikely that will every be necessary.

DamageResult
Constructor `damage_result_t`

This struct is used by the combat engine to when determining a damage roll. The length of the arrays is determined by the global DAMAGE_INDEX_NUM constant.

int32_t damages[DAMAGE_INDEX_NUM]
Normal damage.

int32_t damages_unblocked[DAMAGE_INDEX_NUM]
Unblockable damages.

int32_t immunity[DAMAGE_INDEX_NUM]
Damage immunity adjustments.

int32_t resist[DAMAGE_INDEX_NUM]
Damage resistance adjustments.

int32_t resist_remaining[DAMAGE_INDEX_NUM]
Damage resistance remaining. This is to provide feedback for DamageResistance effects with limits.

int32_t **reduction**
Damage reduction adjustment.

int32_t **reduction_remaining**
Damage reduction remaining. This is to provide feedback for DamageReduction effects with limits.

int32_t **parry**
Parry adjustment for servers using the Critical Hit Parry reduction designed by Higher Ground.

Attack
Information for an attack.

CNWSCreature* **attacker_nwn**
Internal attacker object, unused in Lua.

CNWSObject* **target_nwn**
Internal target object, unused in Lua.

CNWSCombatAttackData* **attack**
Internal NWN attack data.

int32_t **weapon**
EQUIP_TYPE_* of current weapon.

int32_t **ranged_type**
RANGED_TYPE_*

uint32_t **target_state**
Target state bitmask.

uint32_t **situational_flags**
Situational bitmask.

double **target_distance**
Distance to target.

bool **is_offhand**
Is offhand attack.

bool **is_sneak**
Is sneak attack.

bool **is_death**
Is death attack.

bool **is_killing**
Is killing blow.

int32_t **damage_total**
Total damage done.

DamageResult **dmg_result**
DamageResult ctype.

int32_t effects_to_remove[]

int32_t **effects_to_remove_len**

4.2.2 Functions

Note: When using these functions in performance critical code, you should cache them in local variables.

AddCCMessage(*info*, *type*, *objs*, *ints*, *str*)
Adds combat message to an attack.

Parameters

- **info** (*Attack*) – Attack info.

AddDamageToResult(*info*, *dmg*, *mult*)
Add damage.

Parameters

- **info** (*Attack*) – Attack info.
- **dmg** – DamageRoll
- **mult** (*int*) – Multiplier for crits, etc.

AddEffect(*info*, *attacker*, *eff*)
Adds an onhit effect to an attack.

Parameters

- **info** (*Attack*) – Attack info.
- **attacker** – *Creature*
- **eff** – *Effect*

AddVFX(*info*, *attacker*, *vfx*)

Parameters

- **info** (*Attack*) – Attack info.
- **attacker** – *Creature*
- **vfx** (*int*) – VFX_*

ClearSpecialAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

CopyDamageToNWNAttackData(*info*, *attacker*, *target*)

Parameters

- **info** (*Attack*) – Attack info.
- **attacker** – *Creature*
- **target** – *Creature*

GetAttackRoll(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsCoupDeGrace(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsCriticalHit(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsDeathAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsHit(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsRangedAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsSneakAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetIsSpecialAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetResult(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetSpecialAttack(*info*)

Parameters

- **info** (*Attack*) – Attack info.

GetType(*info*)

Parameters

- **info** (*Attack*) – Attack info.

Return type ATTACK_TYPE_*

SetAttackMod(*info*, *ab*)

Parameters

- **info** (*Attack*) – Attack info.
- **ab** (*int*) – Attack modifier.

SetAttackRoll (*info*, *roll*)

Parameters

- **info** (*Attack*) – Attack info.
- **roll** (*int*) – Attack roll.

SetConcealment (*info*, *conceal*)

Parameters

- **info** (*Attack*) – Attack info.
- **conceal** (*int*) – Concealment.

SetCriticalResult (*info*, *threat*, *result*)

Parameters

- **info** (*Attack*) – Attack info.
- **threat** (*int*) – Critical threat roll.
- **result** (*boolean*) – Is a critical hit.

SetMissedBy (*info*, *roll*)

Parameters

- **info** (*Attack*) – Attack info.
- **roll** (*int*) – Attack roll.

SetResult (*info*, *result*)

Parameters

- **info** (*Attack*) – Attack info.
- **result** (*int*) – Attack result.

SetSneakAttack (*info*, *sneak*, *death*)

Parameters

- **info** (*Attack*) – Attack info.
- **sneak** (*boolean*) – Is sneak attack.
- **death** (*boolean*) – Is death attack.

4.3 Color

Warning: It's not wise to use strings taken from NWN scripts. There are are number of text encoding issues that will cause undesirable results.

4.3.1 Functions

Encode(*r*, *g*, *b*)
Encodes RGB values.

Parameters

- **r** (*int*) – Red
- **g** (*int*) – Green
- **b** (*int*) – Blue

EncodeHex(*hex*)
Encodes a hex color string.

Parameters

- **hex** (*string*) – Same format as HTML: "#000000"

```
local C = require 'solstice.color'
assert(C.EncodeHex('#FF0000') == C.Enocde(255, 0, 0))
```

4.3.2 Constants

BLUE
RGB(102, 204, 254)

DARK_BLUE
RGB(32, 102, 254)

GRAY
RGB(153, 153, 153)

GREEN
RGB(32, 254, 32)

LIGHT_BLUE
RGB(153, 254, 254)

LIGHT_GRAY
RGB(176, 176, 176)

LIGHT_ORANGE
RGB(254, 153, 32)

LIGHT_PURPLE
RGB(204, 153, 204)

ORANGE
RGB(254, 102, 32)

PURPLE
RGB(204, 119, 254)

RED
RGB(254, 32, 32)

WHITE
RGB(254, 254, 254)

YELLOW
RGB(254, 254, 32)

END
Colored text terminator.

4.4 Dice

Roll(*dice*, *sides*, *bonus*, *times*)
Rolls arbitrary dice.

Parameters

- **dice** (*int*) – Number of dice to roll
- **sides** (*int*) – Number of sides the dice have
- **bonus** (*int*) – Bonus added to roll
- **times** (*int*) – Number of times to do roll.

DetermineBestDiceRoll(*roll1*, *roll2*)
Determines the highest maximum roll. This selects based on the maximum value of the dice roll.

Parameters

- **roll1** (`DiceRoll`) – Dice roll.
- **roll2** (`DiceRoll`) – Dice roll.

IsValid(*roll*)
Determines if roll is valid.

A roll is considered valid if dice and sides are greater than zero or the bonus is.

Parameters

- **roll** (`DiceRoll`) – Dice roll.

DoRoll(*roll*, *times*)
Do a dice roll.

Parameters

- **roll** (`DiceRoll`) – Dice roll.
- **times** (*int*) – Number of times to do roll.

DiceRollToString(*roll*)
Converts a dice roll to formatted string.

Parameters

- **roll** (`DiceRoll`) – Dice roll.

d2(*count*)
Rolls a d2

Parameters

- **count** (*int*) – Number of dice to roll.

d3(*count*)
Rolls a d3

Parameters

- **count** (*int*) – Number of dice to roll.

d4 (*count*)
Rolls a d4

Parameters

- **count** (*int*) – Number of dice to roll.

d6 (*count*)
Rolls a d6

Parameters

- **count** (*int*) – Number of dice to roll.

d8 (*count*)
Rolls a d8

Parameters

- **count** (*int*) – Number of dice to roll.

d10 (*count*)
Rolls a d10

Parameters

- **count** (*int*) – Number of dice to roll.

d12 (*count*)
Rolls a d12

Parameters

- **count** (*int*) – Number of dice to roll.

d20 (*count*)
Rolls a d20

Parameters

- **count** (*int*) – Number of dice to roll.

d100 (*count*)
Rolls a d100

Parameters

- **count** (*int*) – Number of dice to roll.

Warning: There are some non-default behaviors currently here.

4.5 Effect

Ability (*ability*, *amount*)
Creates an ability increase/decrease effect on specified ability score.

Parameters

- **ability** (*int*) – ABILITY_*
- **amount** (*int*) – If less than 0 effect will cause an ability decrease, if greater than 0 an ability increase.

Return type Invalid effect if amount is 0, or the ability is an invalid type.

ArmorClass(*amount*[, *modifier_type*])
Creates an AC increase/decrease effect.

Parameters

- **amount** (*int*) – If < 0 effect will cause a decrease by amount, it will be and increase if > 0
- **modifier_type** (*int*) – AC_* constant. (Default: AC_DODGE_BONUS)

Appear([*animation=false*])
Create a special effect to make the object "fly in".

Parameters

- **animation** (*boolean*) – Use animation (Default: false)

AreaOfEffect(*aoe*[, *enter*[, *heartbeat*[, *exit*]]])
Returns a new effect object.

Parameters

- **aoe** (*int*) – The ID of the Area of Effect
- **enter** (*string*) – The script to use when a creature enters the radius of the Area of Effect. (Default: "")
- **heartbeat** (*string*) – The script to run on each of the Area of Effect's Heartbeats. (Default: "")
- **exit** (*string*) – The script to run when a creature leaves the radius of an Area of Effect. (Default: "")

AttackBonus(*amount*[, *modifier_type*])
Create an Attack increase/decrease effect.

Parameters

- **amount** (*int*) – If < 0 effect will cause a decrease by amount, it will be and increase if > 0
- **modifier_type** (*int*) – ATTACK_TYPE_* constant. (Default: ATTACK_TYPE_MISC)

Beam(*beam*, *creator*, *bodypart*[, *miss_effect*])
Create a Beam effect.

Parameters

- **beam** (*int*) – VFX_BEAM_* Constant defining the visual type of beam to use.
- **creator** (`creature`) – The beam is emitted from this creature
- **bodypart** (*int*) – BODY_NODE_* Constant defining where on the creature the beam originates from.
- **miss_effect** (*boolean*) – If true, the beam will fire to a random vector near or past the target. (Default: false)

Blindness()
Create a Blindness effect.

BonusFeat(*feat*)
Creates a bonus feat effect.

Parameters

- **feat** (*int*) – FEAT_*

Charmed()
Create a Charm effect

Concealment(*percent*[, *miss_type*])
Creates a concealment effect.

Parameters

- **percent** (*int*) – [1,100]
- **miss_type** (*int*) – MISS_CHANCE_TYPE_* constant. (Default: MISS_CHANCE_TYPE_NORMAL)

Confused()
Creates a confusion effect.

Curse([*str*[, *dex*[, *con*[, *int*[, *wis*[, *cha*]]]]]])
Create a Curse effect.

Parameters

- **str** (*int*) – strength modifier. (Default: 1)
- **dex** (*int*) – dexterity modifier. (Default: 1)
- **con** (*int*) – constitution modifier. (Default: 1)
- **int** (*int*) – intelligence modifier. (Default: 1)
- **wis** (*int*) – wisdom modifier. (Default: 1)
- **cha** (*int*) – charisma modifier. (Default: 1)

CutsceneDominated()
Creates an effect that is guranteed to dominate a creature.

CutsceneGhost()
Creates a cutscene ghost effect

CutsceneImmobilize()
Creates a cutscene immobilize effect

CutsceneParalyze()
Creates an effect that will paralyze a creature for use in a cut-scene.

Damage(*amount*, *damage_type*[, *power*])
Creates Damage effect.

Parameters

- **amount** (*int*) – amount of damage to be dealt.
- **damage_type** (*int*) – DAMAGE_INDEX_*
- **power** (*int*) – DAMAGE_POWER_* (Default: DAMAGE_POWER_NORMAL)

DamageDecrease(*amount*[, *damage_type*[, *critical*[, *unblockable*]]])
Effect Damage Decrease

Parameters

- **amount** (*int*) – DAMAGE_BONUS_*
- **damage_type** (*int*) – DAMAGE_INDEX_* constant. (Default: DAMAGE_INDEX_MAGICAL)
- **critical** (*boolean*) – Only applicable on critical hits. (Default: false)

- **unblockable** (*boolean*) – Not modified by damage protections. (Default: false)

DamageIncrease(*amount*[, *damage_type*[, *critical*[, *unblockable*]]])
Effect Damage Increase

Parameters

- **amount** (*int*) – DAMAGE_BONUS_*
- **damage_type** (*int*) – DAMAGE_INDEX_* constant. (Default: DAMAGE_INDEX_MAGICAL)
- **critical** (*boolean*) – Only applicable on critical hits. (Default: false)
- **unblockable** (*boolean*) – Not modified by damage protections. (Default: false)

DamageRange(*start*, *stop*[, *damage_type*[, *critical*[, *unblockable*]]])
Effect Damage Increase

Parameters

- **start** (*int*) – Minimum damage.
- **stop** (*int*) – Maximum damage.
- **damage_type** (*int*) – DAMAGE_INDEX_* constant. (Default: DAMAGE_INDEX_MAGICAL)
- **critical** (*boolean*) – Only applicable on critical hits. (Default false)
- **unblockable** (*boolean*) – Not modified by damage protections. (Default false)

DamageImmunity(*damage_type*, *amount*)
Damage immunity effect.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **amount** (*int*) – 100, -1] or [1,100]

DamageReduction(*amount*, *power*[, *limit*])
Damage reduction effect.

Parameters

- **amount** (*int*) – Amount
- **power** (*int*) – Power
- **limit** (*int*) – Limit. (Default: 0)

DamageResistance(*damage_type*, *amount*[, *limit*])
Damage resistance effect.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **amount** (*int*) – Amount
- **limit** (*int*) – Limit. (Default: 0)

DamageShield(*amount*, *random*, *damage_type*[, *chance*])
Damage Shield effect.

Parameters

- **amount** (*int*) – Base damage

- **random** (*int*) – DAMAGE_BONUS_*
- **damage_type** (*int*) – DAMAGE_INDEX_*
- **chance** (*int*) – Chance of doing damage to attacker. (Default: 100)

Darkness()
Create a Darkness effect.

Dazed()
Create a Daze effect.

Deaf()
Create a Deaf effect.

Death(*spectacular*, *feedback*)
Death effect

Parameters

- **spectacular** (*boolean*) – Spectacular
- **feedback** (*boolean*) – Feedback

Disappear([*animation*])
Disappear effect.

Parameters

- **animation** (*boolean*) – Use animation. (Default: false)

DisappearAppear(*location*[, *animation*])
Disappear Appear effect.

Parameters

- **location** (*Location*) – Location.
- **animation** (*boolean*) – Use animation. (Default: false)

Disarm()
Create Disarm effect

Disease(*disease*)
Create a Disease effect.

Parameters

- **disease** (*int*) – DISEASE_*

DispelMagicAll([*caster_level*])
Create a Dispel gic All effect.

Parameters

- **caster_level** (*int*) – The highest level spell to dispel.

DispelMagicBest([*caster_level*])
Create a Dispel gic Best effect.

Parameters

- **caster_level** (*int*) – The highest level spell to dispel.

Dominated()
Create a Dominate effect.

Entangle()
Create an Entangle effect

Ethereal()
Creates a Sanctuary effect but the observers get no saving throw.

Frightened()
Create a frightened effect for use in making creatures shaken or flee.

Haste()
Create a Haste effect.

Heal(*amount*)
Creates a healing effect.

Parameters

- **amount** (*int*) – Hit points to heal.

HitPointChangeWhenDying(*hitpoint_change*)
Create a Hit Point Change When Dying effect.

Parameters

- **hitpoint_change** (*int*) – Positive or negative, but not zero.

Icon(*icon*)
Creates an icon effect

Immunity(*immunity*[, *amount*])
Create an Immunity effect.

Parameters

- **immunity** (*int*) – One of the IUNITY_TYPE_* constants.
- **amount** (*int*) – Percent immunity. (Default: 100)

Invisibility(*invisibilty_type*)
Create an Invisibility effect.

Parameters

- **invisibilty_type** (*int*) – One of the INVISIBILITY_TYPE_* constants defining the type of invisibility to use.

Knockdown()
Create a Knockdown effect

LinkEffects(*child*, *parent*)
Creates one new effect object from two seperate effect objects.

Parameters

- **child** (*Effect*) – One of the two effects to link together.
- **parent** (*Effect*) – One of the two effects to link together.

MissChance(*percent*, *misstype*)
Creates a miss chance effect.

Parameters

- **percent** (*int*) – [1,100].
- **misstype** (*int*) – MISS_CHANCE_TYPE_* constant. (Default: MISS_CHANCE_TYPE_NORMAL)

ModifyAttacks (*attacks*)
Create a modify Attacks effect that adds attacks to the target.

Parameters

- **attacks** – Maximum is 5, even with the effect stacked

MovementSpeed (*amount*)
Create a Movement Speed Increase/Decrease effect to slow target.

Parameters

- **amount** (*int*) – If < 0 effect will cause a decrease by amount, it will be and increase if > 0

NegativeLevel (*amount*, *hp_bonus*)
Create a Negative Level effect that will decrease the level of the target.

Parameters

- **amount** (*int*) – Number of levels
- **hp_bonus** (*int*) – TODO

Paralyze ()
Create a Paralyze effect.

Petrify ()
Creates an effect that will petrify a creature.

Poison (*poison*)
Create a Poison effect.

poison The type of poison to use, as defined in the POISON_* constant group.

Polymorph (*polymorph*[, *locked*])
Create a Polymorph effect that changes the target into a different type of creature.

Parameters

- **polymorph** (*int*) – POLYRPH_TYPE_*
- **locked** (*boolean*) – If true, player can't cancel polymorph. (Default: false)

RacialType (*race*)

Regenerate (*amount*, *interval*)
Create a Regenerate effect.

Parameters

- **amount** (*int*) – Amount of damage to be regenerated per time interval
- **interval** (*float*) – Length of interval in seconds

Resurrection ()
Create a Resurrection effect.

Sanctuary (*dc*)
Creates a sanctuary effect.

Parameters

- **dc** (*int*) – Must be a non-zero, positive number.

SavingThrow (*save*, *amount*[, *save_type*])
Create a Saving Throw Increase/Decrease effect to modify one Saving Throw type.

Parameters

- **save** (*int*) – The Saving Throw to affect, as defined by the SAVING_THROW_* constants group.
- **amount** (*int*) – The amount to modify the saving throws by. If > 0 an increase, if < 0 a decrease.
- **save_type** (*int*) – The type of resistance this effect applies to as defined by the SAVING_THROW_VS_* constants group. (Default: SAVING_THROW_TYPE_ALL)

SeeInvisible()
Create a See Invisible effect.

Silence()
Create a Silence effect

Skill(*skill*, *amount*)
Returns an effect to decrease a skill.

Parameters

- **skill** (*int*) – SKILL_*
- **amount** (*int*) – The amount to modify the skill by. If > 0 an increase, if < 0 a decrease.

Sleep()
Creates a sleep effect.

Slow()
Creates a slow effect.

SpellFailure(*percent*, *spell_school*)
Creates an effect that inhibits spells.

Parameters

- **percent** (*int*) – Percent chance of spell failing (1 to 100). (Default: 100)
- **spell_school** (*int*) – SPELL_SCHOOL_*. (Default: SPELL_SCHOOL_GENERAL)

SpellImmunity(*spell*)
Returns an effect of spell immunity.

Parameters

- **spell** (*int*) – SPELL_* (Default: SPELL_ALL_SPELLS)

SpellLevelAbsorption(*max_level*, *max_spells*, *school*)
Creates a Spell Level Absorption effect

Parameters

- **max_level** (*int*) – Highest spell level that can be absorbed.
- **max_spells** (*int*) – Maximum number of spells to absorb
- **school** (*int*) – SPELL_SCHOOL_*. Default: SPELL_SCHOOL_GENERAL

SpellResistance(*amount*)
Create spell resistance effect.

Parameters

- **amount** (*int*) – If > 0 an increase, if < 0 a decrease

Stunned()
Creates a Stunned effect

SummonCreature (*resref*[, *vfx*[, *delay*[, *appear*]]])
Summon Creature Effect

Parameters

- **resref** (*string*) – Identifies the creature to be summoned by resref name.
- **vfx** (*int*) – VFX_*. (Default: VFX_NONE)
- **delay** (*float*) – There can be delay between the visual effect being played, and the creature being added to the area. (Default: 0.0)
- **appear** (*boolean*) – (Default: false)

Swarm (*looping*, *resref1*[, *resref2*[, *resref3*[, *resref4*]]])
Summon swarm effect.

Parameters

- **looping** (*boolean*) – If this is `true`, for the duration of the effect when one creature created by this effect dies, the next one in the list will be created. If the last creature in the list dies, we loop back to the beginning and sCreatureTemplate1 will be created, and so on...
- **resref1** (*string*) – Blueprint of first creature to spawn
- **resref2** (*string*) – Optional blueprint for second creature to spawn.
- **resref3** (*string*) – Optional blueprint for third creature to spawn.
- **resref4** (*string*) – Optional blueprint for fourth creature to spawn.

TemporaryHitpoints (*amount*)
Create a Temporary Hitpoints effect that raises the Hitpoints of the target.

Parameters

- **amount** (*int*) – A positive integer

TimeStop ()
Create a Time Stop effect.

TrueSeeing ()
Creates a True Seeing effect.

Turned ()
Create a Turned effect.

TurnResistance (*amount*)
Create a Turn Resistance Increase/Decrease effect that can make creatures more susceptible to turning.

Parameters

- **amount** (*int*) – If > 0 an increase, if < 0 a decrease.

Ultravision ()
Creates an Ultravision effect

VisualEffect (*id*[, *miss*])
Creates a new visual effect

Parameters

- **id** (*int*) – The visual effect to be applied.
- **miss** (*boolean*) – If this is true, a random vector near or past the target will be generated, on which to play the effect. (Default: false)

Wounding (*amount*)
Creates a wounding effect

Parameters

- **amount** (*int*) – Amount of damage to do each round

Warning: There are some non-default behaviors currently here.

4.6 Itemprop

AbilityScore (*ability*, *mod*)
Create Ability bonus/penalty item property.

Parameters

- **ability** (*int*) – ABILITY_*
- **mod** (*int*) – bonus: [1, 12], Penalty [-12, -1]

ArmorClass (*value*)
Create AC item property

Parameters

- **value** (*int*) – Bonus: [1,20] Penalty [-20, -1]

Additional (*addition*)
Creates additional item property

Parameters

- **addition** (*int*) – IP_CONST_ADDITIONAL_*

ArcaneSpellFailure (*value*)
Create arcane spell failure item property

Parameters

- **value** (*int*) – [1-100]

AttackModifier (*value*)
Creatures attack modifier item property

Parameters

- **value** (*int*) – Amount attack bonus is modified.

BonusFeat (*feat*)
Item Property Bonus Feat

Parameters

- **feat** (*int*) – IP_CONST_FEAT_*

BonusLevelSpell (*class*, *level*)
Creates a "bonus spell of a specified level" itemproperty.

Parameters

- **class** (*int*) – solstice.class constant
- **level** (*int*) – [0, 9]

CastSpell (*spell*, *uses*)
Creates a "cast spell" itemproperty.

Parameters

- **spell** (*int*) – IP_CONST_CASTSPELL_*
- **uses** (*int*) – IP_CONST_CASTSPELL_NUMUSES_*

ContainerReducedWeight (*amount*)
Create a "reduced weight container" itemproperty.

Parameters

- **amount** (*int*) – IP_CONST_CONTAINERWEIGHTRED_*

DamageBonus (*damage_type*, *damage*)
Creates a damage bonus itemproperty.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **damage** (*int*) – DAMAMGE_BONUS_*

DamageRange (*damage_type*, *min*, *max*)
Creates a damage range itemproperty.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **min** (*int*) – Minimum damage.
- **max** (*int*) – Maximum damage.

DamageImmunity (*damage_type*, *amount*)
Creates a damage immunity itemproperty.

Note: If you are using CEP and CEP is set to true in your global options then you can pass values 1-100, otherwise you will have to pass the item property constants.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **amount** (*int*) – Amount.

DamagePenalty (*amount*)
Creates a damage penalty itemproperty.

Parameters

- **amount** (*int*) – [1,5]

DamageReduction (*enhancement*, *soak*)
Creates a damage reduction itemproperty.

Note: If you are using CEP then values can be passed for the soak parameter rather than IP_CONST_SOAK_*. The value must be a multiple of 5 and in the range [5, 100]

Parameters

- **enhancement** (*int*) – [1,20]

- **soak** (*int*) – Amount soaked.

DamageResistance(*damage_type*, *amount*)
Creates damage resistance item property.

If you are using CEP then values can be passed for the amount parameter rather than IP_CONST_RESIST_*. The value must be a multiple of 5 and in the range [5, 100]

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **amount** (*int*) – Resist value.

DamageVulnerability(*damage_type*, *amount*)
Creates damage vulnerability item property.

If you are using CEP and CEP is set to true in your global options then you can pass values 1-100, otherwise you will have to pass the item property constants.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **amount** (*int*) – Amount.

Darkvision()
Creates Darkvision Item Property

EnhancementModifier(*amount*)
Item Property Enhancement Bonus

Parameters

- **amount** (*int*) – If greater than 0 enhancment bonus, else penalty

ExtraDamageType(*damage_type*, *is_ranged*)
Creates an "extra damage type" item property.

Parameters

- **damage_type** (*int*) – DAMAGE_INDEX_*
- **is_ranged** (*boolean*) – ExtraRangedDamge if true, melee if false.

Freedom()
Creates a free action (freedom of movement) itemproperty.

Haste()
Creates haste item property.

HealersKit(*modifier*)
Creates a healers' kit item property.

Parameters

- **modifier** (*int*) – [1,12]

HolyAvenger()
Creates Holy Avenger item propety.

ImmunityMisc(*immumity_type*)
Creates immunity item property

Parameters

- **immumity_type** (*int*) – IMMUNITY_TYPE_*

ImprovedEvasion()
Creates Improved evasion item property.

Keen()
Creates keen item property

Light(*brightness*, *color*)
Creates a light item property.

Parameters

- **brightness** (*int*) – IP_CONST_LIGHTBRIGHTNESS_*
- **color** (*int*) – IP_CONST_LIGHTCOLOR_*

LimitUseByClass(*class*)
Creates a class use limitation item property

Parameters

- **class** (*int*) – CLASS_TYPE_*

LimitUseByRace(*race*)
Creates a race use limitation item property

Parameters

- **race** (*int*) – RACIAL_TYPE_*

MassiveCritical(*damage*)
Creates a massive criticals item property.

Parameters

- **damage** (*int*) – DAMAGE_BONUS_*

Material(*material*)
Creates material item property

Parameters

- **material** (*int*) – The material type should be [0, 77] based on iprp_matcost.2da.

Mighty(*value*)
Creates a mighty item property.

Parameters

- **value** (*int*) – [1,20]

MonsterDamage(*damage*)
Creates Monster Damage effect.

Parameters

- **damage** (*int*) – IP_CONST_MONSTERDAMAGE_*

NoDamage()
Creates no damage item property

OnHitCastSpell(*spell*, *level*)
Creates an "on hit cast spell" item property.

Parameters

- **spell** (*int*) – IP_CONST_ONHIT_CASTSPELL_*
- **level** (*int*) – Level spell is cast at.

OnHitMonster (*prop*, *special*)
Creates on monster hit item property.

> **Warning:** Item property is bugged. See NWN Lexicon.

Parameters

- **prop** (*int*) – IP_CONST_ONMONSTERHIT_*
- **special** (*int*) – Unknown

OnHitProps (*prop*, *dc*, *special*)
Creates an OnHit itemproperty.

Parameters

- **prop** (*int*) – IP_CONST_ONHIT_*
- **dc** (*int*) – IP_CONST_ONHIT_SAVEDC_*
- **special** (*int*) – Meaning varies with type. (Default: 0)

Quality (*quality*)
Creates quality item property

Parameters

- **quality** (*int*) – IP_CONST_QUALITY_*

Regeneration (*amount*)
Creates a regeneration item property.

Parameters

- **amount** (*int*) – 1, 20]

SavingThrow (*save_type*, *amount*)
Creates saving throw bonus item property

Parameters

- **save_type** (*int*) – SAVING_THROW_*
- **amount** (*int*) – 1,20] or [-20, -1]

SavingThrowVersus (*save_type*, *amount*)
Creates saving throw bonus vs item property

Parameters

- **save_type** (*int*) – SAVING_THROW_VS_*
- **amount** (*int*) – 1,20] or [-20, -1]

SkillModifier (*skill*, *amount*)
Creates skill modifier item property

Parameters

- **skill** (*int*) – solstice.skill type constant.
- **amount** (*int*) – 1, 50] or [-10, -1]

SpecialWalk ([*walk*])
Creates a special walk itemproperty.

Parameters

- **walk** (*int*) – Only 0 is a valid argument

SpellImmunityLevel (*level*)
Create an "immunity to spell level" item property.

Parameters

- **level** (*int*) – Spell level [1,9]

SpellImmunitySpecific (*spell*)
Creates an "immunity to specific spell" itemproperty.

Parameters

- **spell** (*int*) – IP_CONST_IMMUNITYSPELL_*

SpellImmunitySchool (*school*)
Creates an "immunity against spell school" itemproperty.

Parameters

- **school** (*int*) – IP_CONST_SPELLSCHOOL_*

SpellResistance (*amount*)
Creates a spell resistince item property

Parameters

- **amount** (*int*) – IP_CONST_SPELLRESISTANCEBONUS_*

ThievesTools (*modifier*)
Creates a thieves tool item property

Parameters

- **modifier** (*int*) – [1, 12]

Trap (*level*, *trap_type*)
Creates a trap item property

Parameters

- **level** (*int*) – IP_CONST_TRAPSTRENGTH_*
- **trap_type** (*int*) – IP_CONST_TRAPTYPE_*

TrueSeeing ()
Creates true seeing item property

TurnResistance (*modifier*)
Creates a turn resistance item property.

Parameters

- **modifier** (*int*) – [1, 50]

UnlimitedAmmo ([*ammo*])
Creates an unlimited ammo itemproperty.

Parameters

- **ammo** (*int*) – IP_CONST_UNLIMITEDAMMO_* (Default: IP_CONST_UNLIMITEDAMMO_BASIC)

VampiricRegeneration (*amount*)
Creates vampiric regeneration effect.

Parameters

- **amount** (*int*) – [1,20]

VisualEffect (*effect*)
Creates a visual effect item property

Parameters

- **effect** (*int*) – IP_CONST_VISUAL_*

WeightIncrease (*amount*)
Item Property Weight Increase

Parameters

- **amount** (*int*) – IP_CONST_WEIGHTINCREASE_*

WeightReduction (*amount*)
Item Property Weight Reuction

Parameters

- **amount** (*int*) – IP_CONST_REDUCEDWEIGHT_*

4.7 Game

This module provides defines an interface for game related state.

4.7.1 Constants

PLUGIN_COMBAT_ENGINE
Combat engine plugin ID.

4.7.2 2DA

Get2daColumnCount (*twoda*)
Get number of columns in 2da.

Parameters

- **twoda** – 2da file.

Get2daFloat (*twoda*, *col*, *row*)
Get float value.

Parameters

- **twoda** (*string*) – 2da file.
- **col** (*int* or *string*) – Column label or index.
- **row** (*int*) – Row index.

Get2daInt (*twoda*, *col*, *row*)
Get int value.

Parameters

- **twoda** (*string*) – 2da file.

- **col** (*int* or *string*) – Column label or index.
- **row** (*int*) – Row index.

Get2daRowCount(*twoda*)
Get number of rows in 2da.

Parameters

- **twoda** (*string*) – 2da file.

Get2daString(*twoda*, *col*, *row*)
Get string value.

Parameters

- **twoda** (*string*) – 2da file.
- **col** (*int* or *string*) – Column label or index.
- **row** (*int*) – Row index.

4.7.3 Events

EventActivateItem(*item*, *location*, *target*)
Create activate item even.

Parameters

- **item** – Item.
- **location** – Target location.
- **target** – Target object.

EventConversation()
Create conversation event.

EventSpellCastAt(*caster*, *spell*[, *is_harmful*])
Creature spell cast at event.

Parameters

- **caster** – Spell caster.
- **spell** (*int*) – SPELL_* constant.
- **is_harmful** (*bool*) – Is spell harmful to target. Default: `true`

EventUserDefined(*event*)
Create user defined event.

Parameters

- **event** – An integer id.

GetClickingObject()
Get last clicking object.

GetEnteringObject()
Get last object to enter.

GetExitingObject()
Get last object to exit.

GetItemActivated()
Gets the item activated.

GetItemActivatedTarget()
Gets item activated event target.

GetItemActivatedTargetLocation()
Gets item activated event location.

GetItemActivator()
Gets object that activated item.

GetLastPCToCancelCutscene()
Gets last PC to cancel cutscene.

GetLastPlayerDied()
Gets last player died.

GetLastPlayerDying()
Gets last player dying.

GetLastUsedBy()
Gets last object to use something.

GetPCLevellingUp()
Gets last PC that leveled up.

GetPlaceableLastClickedBy()
Get last object to click a placeable.

GetUserDefinedEventNumber()
Get user defined event number.

GetUserDefinedItemEventNumber(*obj*)
Get the current UserDefined Item Event Number.

Parameters

- **obj** (*Item*) – Item object

Return type ITEM_EVENT_* (see itemevents.2da)

SetUserDefinedItemEventNumber(*obj*, *event*)
Set the current UserDefined Item Event Number

Parameters

- **obj** (*Item*) – Item object
- **event** – ITEM_EVENT_* (see itemevents.2da)

SignalEvent(*obj*, *event*)
Signal an event.

Parameters

- **obj** (*Object*) – Object to signal.
- **event** – Event object.

4.7.4 Objects

ClearCacheData (*obj*)
Clear the effect cache.

CreateObject (*object_type*, *template*, *loc*[, *appear*[, *newtag*]])
Create an object of a specified type at a given location

Parameters

- **object_type** (*int*) – OBJECT_TYPE_*
- **template** (*string*) – The resref of the object to create from the pallet.
- **loc** (*Location*) – The location to create the object at.
- **appear** (*bool*) – If `true`, the object will play its spawn in animation. Default: `false`.
- **newtag** (*string*) – If this string is not empty, it will replace the default tag from the template. Default: “”

Return type New object or OBJECT_INVALID

ExportSingleCharacter (*player*)
Export single character.

Parameters

- **player** (*Creature*) – Object to export.

GetCanonicalID (*cre*)
Get canonical ID

Parameters

- **cre** (*Creature*) – Player character

GetModule ()
Get Module.

GetObjectByID (*id*)
Get object by ID.

Parameters

- **id** (*int*) – Object ID.

Return type An object or `OBJECT_INVALID`

GetObjectByTag (*tag*[, *nth*])
Gets an object by tag

Parameters

- **tag** (*string*) – Tag of object.
- **nth** (*int*) – Nth object. Default: 1

GetPCSpeaker ()
Gets the PC speaker.

Return type *Creature* or `OBJECT_INVALID`

GetWaypointByTag (*tag*)
Finds a waypiont by tag

Parameters

- **tag** (*string*) – Tag of waypoint.

Return type *Waypoint* or OBJECT_INVALID

ObjectsByTag(*tag*)
Iterator over objects by tag

Parameters

- **tag** (*string*) – Tag of object

ObjectsInShape(*shape*, *size*, *location*[, *line_of_sight*[, *mask*[, *origin*]]])
Iterator over objects in a shape.

Parameters

- **shape** (*int*) – SHAPE_*
- **size** (*int*) – The size of the shape. Dependent on shape or RADIUS_SIZE_*.
- **location** – Shapes location.
- **line_of_sight** (*bool*) – This can be used to ensure that spell effects don't go through walls. Default: false
- **mask** (*int*) – Object type mask. Default: OBJECT_TYPE_CREATURE.
- **origin** (*vector*) – Normally the spell-caster's position. Default: Zero vector.

PCs()
Iterator over all PCs

RemoveObjectFromCache(*obj*)
Remove object from Solstice object cache. :param obj: Any object. :type obj: *Object*

4.7.5 Plugins

The plugin in system allows registering plugins by a string identifier and optionally enforcing a particular interface.

Note: Only one plugin can be registered to a plugin interface.

RegisterPlugin(*name*[, *enforcer*])
Registers a plugin interface.

Parameters

- **name** (*string*) – Plugin interface name.
- **enforcer** (*function*) – Function that is called when a plugin attempts to load. This is to allow enforcing a particular interface.

LoadPlugin(*name*, *interface*)
Loads a plugin for a given plugin interface. If the plugin is successfully loaded the plugin system will attempt to call plugin.OnLoad if it exists.

Parameters

- **name** (*string*) – Plugin interface name.
- **interface** (*table*) – A table of functions that satisfy the plugin interface.

GetPlugin(*name*)
Gets a plugin by name.

Parameters

- **name** (*string*) – Plugin interface name.

UnloadPlugin(*name*)
Unloads a plugin for a given plugin interface. The plugin system will attempt to call `plugin.OnUnload` if it exists.

Parameters

- **name** (*string*) – Plugin interface name.

IsPluginLoaded(*name*)
Determines if a plugin is loaded.

Parameters

- **name** (*string*) – Plugin interface name.

4.7.6 Signals

OnPreExportCharacter
This event is fired before saving a character.

OnPostExportCharacter
This event is fired after saving a character.

OnObjectClearCacheData
This signal is called when an object has its data cleared from the cache. This is typically for PCs only as they are not removed from the cache, but need some data reset for when the log in again.

OnObjectRemovedFromCache
This signal is called when an object is removed from the cache. Note that PCs are never removed from the cache.

OnUpdateCombatInfo
This signal is called when combat information is updated. Only one parameters is passed: a *Creature* instance.

Note: This is only active when a combat engine has been registered!

OnUpdateEffect
This is called whenever an effect is applied or removed from a creature. Two parameters are passed: a *Creature* instance and a *Effect*. Note: there is no way to determine if the effect was applied or removed, so it's only useful in cases of updating/invalidating cached information.

4.7.7 Scripts

DumpScriptEnvironment()
Gets a string representation of the script environment.

ExecuteItemEvent(*obj*, *item*, *event*)
Executes item event. This is compatible with NWN tag based scripting. It will only work if that feature has been enabled.

Parameters

- **obj** – Object
- **item** – Item
- **event** – ITEM_EVENT_* See itemevents.2da

Return type SCRIPT_RETURN_*

ExecuteScript(*script*, *target*)
Executes a script on a specified target. This operates like the NWScript `ExecuteScriptAndReturnInt` rather than `ExecuteScript`.

Parameters

- **script** – Script to call.
- **target** – Object to run the script on.

Return type SCRIPT_RETURN_* constant.

GetItemEventName(*item*)
Gets the item event script name. This function is compatible with NWN tag based scripting.

Parameters

- **item** (*Item*) – Item that caused the event.

GetItemEventType(*obj*)
Get last item event type.

Parameters

- **obj** – Object script is being run on.

Return type ITEM_EVENT_* See itemevents.2da

LoadScript(*fname*)
Load script file.

Parameters

- **fname** (*string*) – Script file name.

LockScriptEnvironment()
Locks the script environment. After this is called no variables can be set globally in the script environment

RunScript(*script*, *target*)
Run script.

Parameters

- **script** (*string*) – Script to call.
- **target** – Object to run the script on.

SetItemEventPrefix([*prefix=""*])
Set item event prefix. This function is compatible with NWN tag based scripting.

Parameters

- **prefix** (*string*) – Prefix to add to script calls.

SetItemEventType(*obj*, *event*)
Sets item event type on object.

Parameters

- **obj** – Object script is being run on.
- **event** (*int*) – ITEM_EVENT_* See itemevents.2da

SetScriptReturnValue(*object*[, *value=SCRIPT_RETURN_CONTINUE*])
Set script return value.

Parameters

- **object** – Object script is being run on.
- **value** (*int*) – SCRIPT_RETURN_* constant.

UnlockScriptEnvironment()
Unlocks the script environment. After this is called variables can be set globally in the script environment

---

4.7.8 Time

GetDay()
Determine the current in-game calendar day.

GetHour()
Gets the current hour.

GetIsDawn()
Get if it's dawn.

Return type `bool`

GetIsDay()
Get if it's day.

Return type `bool`

GetIsDusk()
Get if it's dusk

Return type `bool`

GetIsNight()
Get if it's night

Return type `bool`

GetMillisecond()
Gets the current millisecond.

GetMinute()
Gets the current minute.

GetMonth()
Determine the current in-game calendar month.

GetSecond()
Gets the current second

GetYear()
Determine the current in-game calendar year.

HoursToSeconds(*hours*)
Converts hours to seconds

Parameters

- **hours** (*int*) – Number of hours

RoundsToSeconds(*rounds*)
Converts rounds to seconds

Parameters

- **rounds** (*int*) – Number of rounds

SetCalendar(*year*, *month*, *day*)
Set calendar

Parameters

- **year** (*int*) – Specific year to set calendar to from 1340 to 32001.
- **month** (*int*) – Specific month to set calendar from 1 to 12.
- **day** (*int*) – Specific day to set calendar to from 1 to 28.

SetTime(*hour*, *minute*, *second*, *millisecond*)
Sets the game's current time.

Parameters

- **hour** (*int*) – The new hour value, from 0 to 23.
- **minute** (*int*) – The new minute value from 0 to 1 (or 0 to a higher value if the module properties for time were changed).
- **second** (*int*) – The new second value, from 0 to 59.
- **millisecond** (*int*) – The new millisecond value, from 0 to 999.

TurnsToSeconds(*turns*)
Converts turns to seconds

Parameters

- **turns** (*int*) – Number of turns

UpdateTime()
Force update time.

4.7.9 TLK

GetTlkString(*strref*)
Get string by TLK table reference.

Parameters

- **strref** (*int*) – TLK table reference.

Return type `string`

4.8 Hook

Danger: Please note this is very advanced module. Using it incorrectly will result in segfaults. This is a very thin wrapper around the Acaos' NWNX *nx_hook_function* any documentation can probably (not) be found in the NWNX repository.

It allows you to hook nwserver functions and replace them with Lua functions directly. Note that the lua function and the original function returned by *hook* must be castable to the function pointer type passed in *HookDesc.type*, which means that the types of the parameters must be defined. Currently not all types are defined, see src/solstice/nwn/ctypes for those that are.

Internally your hook is wrapped in an anonymous function which uses `pcall`. If for whatever reason your hook fails, the original function will be called. This will mitigate most segfaults and also allows hook script files to be reloaded without requiring a server reboot.

4.8.1 Examples

```
local Hook = require 'solstice.hooks'

-- Hook CNWSCreature::GetTotalEffectBonus, print the parameters and return -2
-- for everything if a global `is_april_fools` is not false, else it calls and returns
-- the value from the original function.

local GetTotalEffectBonus_orig
local function Hook_GetTotalEffectBonus(cre, eff_switch , versus, elemental,
                                        is_crit, save, save_vs, skill,
                                        ability, is_offhand)

  print(cre, eff_switch , versus, elemental,
        is_crit, save, save_vs, skill,
        ability, is_offhand)

  if is_april_fools then
     return -2
  else
    -- If you want to inspect parameters and just call the original function you can:
    return GetTotalEffectBonus_orig(cre, eff_switch , versus, elemental,
                                    is_crit, save, save_vs, skill,
                                    ability, is_offhand)
  end
end

GetTotalEffectBonus_orig = Hook.hook {
  name = "GetTotalEffectBonus",
  address = 0x08132298,
  func = Hook_GetTotalEffectBonus,
  type = "int (*)(CNWSCreature *, uint8_t, CNWSObject *, int32_t, int32_t, uint8_t, uint8_t, uint8_t,
  length = 5
}
```

4.8.2 Tables

HookDesc
Table defining a hook.

Fields:

name [string] Name of hook.

address [int] Address of function to hook

type [string] Function pointer type.

func [function] Function to be called by hook.

length [int] Length of the hook.

4.8.3 Functions

hook(*info*)

Parameters **info** (*HookDesc*) – Table with hook data.

Return type Function pointer to the trampoline.

4.9 NWN

4.10 NWNX

The NWNX module contains submodules for interfacing with NWNX plugins. Not all plugins are currently implemented.

4.10.1 nwnx.chat

Functions

SetChatHandler(*func*)

SetCombatMessageHandler(*func*)

4.10.2 nwnx.core

Functions

HookEvent(*name*, *func*)

Danger: This is an advanced function. Only use it if you're sure you know what you're doing.

Parameters

- **name** (*string*) – Event name.
- **func** (*function*) – Event handler function. This **must** be castable to and satisfy `int (*NWNXHOOK)(uintptr_t);`

Return type boolean

4.10.3 nwnx.events

Constants

NODE_TYPE_ENTRY_NODE

NODE_TYPE_REPLY_NODE

NODE_TYPE_STARTING_NODE

LANGUAGE_CHINESE_SIMPLIFIED

LANGUAGE_CHINESE_TRADITIONAL

LANGUAGE_ENGLISH

LANGUAGE_FRENCH

LANGUAGE_GERMAN

LANGUAGE_ITALIAN

LANGUAGE_JAPANESE

LANGUAGE_KOREAN

LANGUAGE_POLISH

LANGUAGE_SPANISH

Signals

SaveCharacter

PickPocket

Attack

QuickChat

Examine

CastSpell

TogglePause

PossessFamiliar

DestroyObject

Tables

NWNXEventInfo
Event Info Table

Fields

type Event type

subtype Event subtype

target Event target or OBJECT_INVALID

item Event item or OBJECT_INVALID

pos Event location vector

Functions

BypassEvent()

GetCurrentAbsoluteNodeID()

GetCurrentNodeID()

GetCurrentNodeText(*nLangID*, *nGender*)

GetCurrentNodeType()

GetEventSignal(*event*)

Parameters

- **event** (*int*) – EVENT_TYPE_*

Return type A signal.

GetSelectedAbsoluteNodeID()

GetSelectedNodeID()

GetSelectedNodeText(*nLangID*, *nGender*)

SetCurrentNodeText(*sText*, *nLangID*, *nGender*)

SetEventReturnValue(*val*)

4.10.4 nwnx.dmactions

Constant

DM_ACTION_MESSAGE_TYPE

DM_ACTION_GIVE_XP

DM_ACTION_GIVE_LEVEL

DM_ACTION_GIVE_GOLD

DM_ACTION_CREATE_ITEM_ON_OBJECT

DM_ACTION_CREATE_ITEM_ON_AREA

DM_ACTION_HEAL_CREATURE

DM_ACTION_REST_CREATURE

DM_ACTION_RUNSCRIPT

DM_ACTION_CREATE_PLACEABLE

DM_ACTION_SPAWN_CREATURE

DM_ACTION_TOGGLE_INVULNERABILITY

DM_ACTION_TOGGLE_IMMORTALITY

Functions

SetScript(*nAction*, *sScript*)

GetID(*dm*)

Prevent(*dm*)

nGetDMAction_Param(*dm*, *second*)

GetDMAction_Param(*dm*)

GetTarget(*dm*, *second*)

GetPosition(*dm*)

GetTargetsCount(*dm*)

GetTargetsCurrent(*dm*)

4.10.5 nwnx.effects

This module is a wrapper around nwnx_effects.

Functions

GetIsEffectHandlerRegistered(*type*)

RegisterEffectHandler(*handler*, ...)

LogEffects(*obj*)

SetNativeEffectCallsUs(*truetype*)

GetCustomEffectTickRate(*effect eff*)

SetCustomEffectTickRate(*eff*, *value*)

EffectCustom(*truetype*)

GetIsItempropHandlerRegistered(*type*)

RegisterItempropHandler(*handler*, ...)

BypassNativeItemProperty()

4.10.6 nwnx.events

Constants

NODE_TYPE_ENTRY_NODE

NODE_TYPE_REPLY_NODE

NODE_TYPE_STARTING_NODE

LANGUAGE_CHINESE_SIMPLIFIED

LANGUAGE_CHINESE_TRADITIONAL

LANGUAGE_ENGLISH

LANGUAGE_FRENCH

LANGUAGE_GERMAN

LANGUAGE_ITALIAN

LANGUAGE_JAPANESE

LANGUAGE_KOREAN

LANGUAGE_POLISH

LANGUAGE_SPANISH

Signals

SaveCharacter

PickPocket

Attack

QuickChat

Examine

CastSpell

TogglePause

PossessFamiliar

DestroyObject

Tables

NWNXEventInfo
Event Info Table

Fields

type Event type

subtype Event subtype

target Event target or OBJECT_INVALID

item Event item or OBJECT_INVALID

pos Event location vector

Functions

BypassEvent()

GetCurrentAbsoluteNodeID()

GetCurrentNodeID()

GetCurrentNodeText(*nLangID*, *nGender*)

GetCurrentNodeType()

GetEventSignal(*event*)

Parameters

- **event** (*int*) – EVENT_TYPE_*

Return type A signal.

GetSelectedAbsoluteNodeID()

GetSelectedNodeID()

GetSelectedNodeText(*nLangID*, *nGender*)

SetCurrentNodeText(*sText*, *nLangID*, *nGender*)

SetEventReturnValue(*val*)

4.10.7 nwnx.haks

nwnx_haks allows a server to control the visibility of HAKs and custom TLK on player login. You can do so at various levels.

The main impetus was to allow a new player to login to a safe loading area without having to download a massive amount of haks before deciding if they feel the server was right for them.

It's also useful if you've built a world with default/CEP/Q resources and then chose later to add tilesets or your own top hak.

Example: Imagine you have a CEP world with many areas and you add some of the great tilesets greated by the community.

```
local Haks = require 'solstice.nwnx.haks'
Haks.SetFallBackTLK('cep_tlk_v26')
Haks.SetHakHidden('mytophak', 1)
Haks.SetHakHidden('mytileset', 2)
...
```

When a new player enters, the server will only require CEP 2.6 to login. You can then flag players based on which HAKs the have via some dialog and store in your DB. They will have to relog/ActivatePortal in order to have their client load the HAKs, from then on they can enter as normal.

Note: If you hide tilesets, you need to control who can enter areas that use those tilesets or it will cause the client to crash.

Functions

DumpHakList()

DumpHiddenHakList()

SetHakHidden(*hak*, *level*)

SetFallBackTLK(*tlk*)

SetEnhanceScript(*script*)

SetPlayerEnhanced(*pc*, *enhanced*)

4.10.8 nwnx.items

nwnx_items allows orrividing some aspects of items:

- Whether a creature can equip, use, unequip an item.

- The base cost and weight of item.
- The items minimum level required to equip. Note: this requires using the ILR setting on your server.

Constants

EVENT_ALL

EVENT_CAN_EQUIP
Can equip item event. Note that using `SetResult(true)()` will essentially operate like if a DM was equipping the item.

EVENT_CAN_UNEQUIP
Can unequip item event. Note only `SetResult(false)()` is respected.

EVENT_MIN_LEVEL
Minimum level required event. This is used to determine ILR.

EVENT_CAN_USE
Can use item event. Note that using `SetResult(true)()` will essentially operate like if a DM was equipping the item. SetResult(false) will highlight the item in red. However, you'll still need to override the EVENT_CAN_EQUIP event to ensure a player cannot equip the item.

EVENT_CALC_BASE_COST
Calculate base cost event. The value passed to *`SetResult()`* must be positive.

EVENT_COMPUTE_WEIGHT
Calculate item weight event. The value passed to *`SetResult()`* must be positive.

EVENT_NUM

Functions

RegisterItemEventHandler(*ev_type*, *f*)

GetDefaultILR(*item*)

SetHelmetHidden(*pc*, *val*)

SetResult(*result*)

4.10.9 nwnx.levels

nwnx_levels allows you to set the maximum level beyond 40th. Due to some limitations of the client the post 40th level process must still be done via conversation.

The LevelUp function requires certain local variables to be set:

- 'LL_CLASS': CLASS_TYPE_* + 1
- 'LL_SKILL_POINTS': Number of unspent skillpoints.
- 'LL_HP': Number of class hitpoints gained.
- 'LL_STAT': ABILITY_* + 1
- 'LL_FEAT_COUNT': Number of feats added.
- 'LL_FEAT_[Nth feat]': Feat to add. Value: FEAT_* + 1
- 'LL_SPGN[Spell Level]_USED': Number of spells gained at spell.

- 'LL_SPGN[Spell Level]_[Nth spell]': Spells removed at each spell level. Value: SPELL_* + 1
- 'LL_SPRM[Spell Level]_USED': Number of spells removes at spell
- 'LL_SPRM[Spell Level]_[Nth spell]': Spells removed at each spell level. Value: SPELL_* + 1

Notes:

- Using builtin functions to add / remove XP can cause deleveling. In Solstice you'd need to use the *direct* parameter. Because of this it requires having your own custom XP gained on kill scripts.
- All of the [Nth ...] start at 0.

Functions

GetMaxLevelLimit()

SetMaxLevelLimit(*level*)

LevelDown(*pc*)

LevelUp(*pc*)

GetMeetsLevelUpFeatRequirements(*cre*, *feat*)

4.10.10 nwnx.system

Functions

GetTMILimit()

SetTMILimit(*nLimit*)

ShutdownServer(*nForce*)

4.11 Objects

4.11.1 class AoE

class AoE

GetCreator()
Get's the creator of the AoE

GetFirstInPersistentObject(*object_mask*)
Gets the first object in an AoE. Perfer the *AoE:ObjectsInEffect()* iterator.

Parameters

- **object_mask** (*int*) – OBJECT_TYPE_* mask.

Return type First object in AOE or OBJECT_INVALID

GetNextInPersistentObject(*object_mask*)
Gets the next object in an AoE. Perfer the *AoE:ObjectsInEffect()* iterator.

Parameters

- **object_mask** (*int*) – OBJECT_TYPE_* mask.

Return type Next object in AOE or OBJECT_INVALID

GetSpellDC()
Get Spell DC.

GetSpellLevel()
Gets AoEs spell level.

ObjectsInEffect(*object_mask*)
An iterator over all objects in an AoE

Parameters

- **object_mask** (*int*) – OBJECT_TYPE_* mask.

Return type Iterator of objects satisfying the object mask.

SetSpellDC(*dc*)
Sets AoEs spell DC.

Parameters

- **dc** (*int*) – Sets spell DC.

SetSpellLevel(*level*)
Sets AoEs spell level.

Parameters

- **level** (*int*) – New caster level.

4.11.2 class Area

class Area

ClearLineOfSight(*loc1*, *loc2*)
Determines if there is a clear line of sight between two objects

Parameters

- **loc1** (*Location*) – Location A
- **loc2** (*Location*) – Location B

GetType()
Get area type.

GetPlayerCount()
Get area player count.

GetSkyBox()
Gets the sky that is displayed in the specified area.

GetTilesetResRef()
Gets the Tileset Resref for the specified area.

RecomputeStaticLighting()
Recomputes the lighting in an area based on current static lighting conditions.

GetObjectIndex(*object*)
Gets the position of specified object in the areas object list.

Parameters

- **object** (*Object*) – Object to search.

GetObjectAtIndex(*idx*)
Returns the object at specifed index of the area's object array.

Parameters

- **idx** (*int*) – Index of the object desired.

Objects()
Iterator returning all objects in a specified area.

AmbientSoundChange(*[day[, night]*)
Changes the ambient soundtracks of an area.

Parameters

- **day** (*int*) – Day track number to change to. If nil the track is unchanged
- **night** (*int*) – Night track number to change to. If nil the track is unchanged

AmbientSoundPlay()
Starts ambient sounds playing in an area.

AmbientSoundStop()
Stops ambient sounds playing in an area.

AmbientSoundSetVolume(*day*, *night*)
Changes the ambient sound volumes of an area.

Parameters

- **day** (*int*) – Day track number to change to. If nil the track is unchanged
- **night** (*int*) – Night track number to change to. If nil the track is unchanged

MusicBackgroundChange(*day*, *night*)
Changes the background music for the area specified.

Parameters

- **day** (*int*) – Day track number to change to. If nil the track is unchanged
- **night** (*int*) – Night track number to change to. If nil the track is unchanged

MusicBackgroundGetBattleTrack()
Gets the background battle track for an area.

MusicBackgroundGetTrack([*is_night*])
Gets the background track for an area.

Parameters

- **is_night** (*boolean*) – If true returns the night track. (Default: False)

MusicBackgroundPlay()
Starts the currently selected background track playing.

MusicBackgroundSetDelay(*delay*)
Changes the delay (in milliseconds) of the background music.

Parameters

- **delay** (*float*) – Time in milliseconds.

MusicBackgroundStop()
Stops the currently selected background track playing.

MusicBattleChange (*track*)
Stops the currently selected background track playing.

Parameters

- **track** (*int*) – Music track number.

MusicBattlePlay ()
Starts the currently selected battle track playing

MusicBattleStop ()
Stops the currently selected battle track playing

SetAreaTransitionBMP (*predef*[, *custom*])
Sets the graphic shown when a PC moves between two different areas in a module.

Parameters

- **predef** (*int*) – A predifined AREA_TRANSITION_* constant.
- **custom** (*string*) – File name of an area transition bitmap. (Default: "")

SetSkyBox (*skybox*)
Sets the sky that is displayed in the specified area.

Parameters

- **skybox** (*int*) – A SKYBOX_* constant (associated with skyboxes.2da)

SetWeather (*weather*)
Sets the weather in the specified area.

Parameters

- **weather** (*int*) – AREA_WEATHER_*

4.11.3 class Creature

class Creature

ActionAttack (*target*[, *passive*])
Add attack action to creature.

Parameters

- **target** (*Object*) – Target to attack.
- **passive** (*bool*) – If `true` the attack is in passive mode. Default: `false`.

ActionCastFakeSpellAtLocation (*spell*, *target*[, *path_type*])

Parameters

- **spell** (*int*) – SPELL_* constant.
- **target** (*Object*) – Object to cast fake spell at.
- **path_type** (*int*) – PROJECTILE_PATH_TYPE_*. Default: PROJECTILE_PATH_TYPE_DEFAULT

ActionCastFakeSpellAtObject (*spell*, *target*[, *path_type*])

Parameters

- **spell** (*int*) – SPELL_* constant.

- **target** (*Location*) – Location to cast spell at.
- **path_type** (*int*) – PROJECTILE_PATH_TYPE_*. Default: PROJECTILE_PATH_TYPE_DEFAULT

ActionCastSpellAtLocation(*spell*, *target*[, *metamagic*[, *cheat*[, *path_type*[, *instant*]]]])

Parameters

- **spell** (*int*) – SPELL_* constant.
- **target** (*Location*) – Location to cast spell at.
- **metamagic** (*int*) – METAMAGIC_*. Default: METAMAGIC_ANY
- **cheat** (*bool*) – If true cast spell even if target does not have the ability. Default: `false`
- **path_type** (*int*) – PROJECTILE_PATH_TYPE_*. Default: PROJECTILE_PATH_TYPE_DEFAULT
- **instant** (*bool*) – If true spell can instantaneously. Default: `false`

ActionCastSpellAtObject(*spell*, *target*[, *metamagic*[, *cheat*[, *path_type*[, *instant*]]]])

Parameters

- **spell** (*int*) – SPELL_* constant.
- **target** (*Object*) – Target
- **metamagic** (*int*) – METAMAGIC_*. Default: METAMAGIC_ANY
- **cheat** (*bool*) – If true cast spell even if target does not have the ability. Default: `false`
- **path_type** (*int*) – PROJECTILE_PATH_TYPE_*. Default: PROJECTILE_PATH_TYPE_DEFAULT
- **instant** (*bool*) – If true spell can instantaneously. Default: `false`

ActionCounterSpell(*target*)

Add counter spell action.

Parameters

- **target** (*Creature*) – Counter spell target.

ActionDoWhirlwindAttack([*feedback*[, *improved*]])

Add whirlwind attack action.

Parameters

- **feedback** (*bool*) – Send feedback. Default: `true`
- **improved** (*bool*) – Determines if effect is Improved Whirlwind Attack. Default: `false`

ActionEquipItem(*item*, *slot*)

Add equip item action.

Parameters

- **item** (*Item*) – Identified item in the creature's inventory.
- **slot** (*int*) – INVENTORY_SLOT_* constant.

ActionEquipMostDamagingMelee([*versus*[, *offhand*]])

Parameters

- **versus** (*Object*) – Object to test against. Default: `OBJECT_INVALID`

- **offhand** (*bool*) – If `true` the item is equipped in the offhand slot. Default: `false`

ActionEquipMostDamagingRanged(*[versus]*)

Parameters

- **versus** (*Object*) – Object to test against. Default: `OBJECT_INVALID`

ActionEquipMostEffectiveArmor()
Add action to equip the armor with the highest AC in the creature's inventory.

ActionExamine(*target*)

Parameters

- **target** (*Object*) – Object to examine.

ActionForceFollowObject(*target[, distance]*)
Add action to follow a creature until *Object:ClearAllActions()* is called.

Parameters

- **target** (*Object*) – Object to follow.
- **distance** (*float*) – Default: 0.0

ActionForceMoveToLocation(*target[, run[, timeout]*)

Parameters

- **target** (*Location*) – Location to move to.
- **run** (*bool*) – If `true` run to location. Default: `false`
- **timeout** (*float*) – Default: 30

ActionForceMoveToObject(*target[, run[, range[, timeout]]]*)

Parameters

- **target** (*Object*) – Object to move to.
- **run** (*bool*) – If `true` run to location. Default: `false`
- **range** (*float*) – Distance to object in meters. Default: 1.0.
- **timeout** (*float*) – Default: 30

ActionInteractObject(*target*)

Parameters

- **target** (*Placeable*) – Placeable to interact with.

ActionJumpToLocation(*loc*)

Parameters

- **loc** (*Location*) – Location to jump to.

ActionJumpToObject(*obj[, straight_line]*)

Parameters

- **obj** (*Object*) – Object to jump to.
- **straight_line** (*bool*) – If `true` creature walks in straight line to object. Default: `true`

ActionMoveAwayFromLocation(*loc[, run[, range]]*)

Parameters

- **loc** (*Location*) – Location to move away from.
- **run** (*bool*) – If `true` creature will run from location. Default: `false`
- **range** (*float*) – Distance to move in meters. Default: 40.0

ActionMoveAwayFromObject (*obj*[, *run*[, *range*]])

Parameters

- **obj** (*Object*) – Object to move away from.
- **run** (*bool*) – If `true` creature will run from object. Default: `false`
- **range** (*float*) – Distance to move in meters. Default: 40.0

ActionMoveToLocation (*loc*[, *run*])

Parameters

- **loc** (*Location*) – Location to jump to.
- **run** (*bool*) – If `true` creature will run to location. Default: `false`

ActionMoveToObject (*obj*[, *run*[, *range*]])

Parameters

- **obj** (*Object*) – Object to move to.
- **run** (*bool*) – If `true` creature will run to location. Default: `false`
- **range** (*float*) – Distance from object in meters. Default: 1.0

ActionPickUpItem (*item*)

Parameters

- **item** (*Item*) – Item to pickup.

ActionPlayAnimation (*animation*[, *speed*[, *dur*]])
Causes creature to play an animation.

Parameters

- **animation** (*int*) – ANIMATION_* constant.
- **speed** (*float*) – Speed of the animation. Default: 1.0
- **dur** (*float*) – Duration of animation. Not applicable to fire and forget animations. Default: 0.0.

ActionPutDownItem (*item*)

Parameters

- **item** (*Item*) – Item to put down.

ActionRandomWalk ()
The action subject will generate a random location near its current location and pathfind to it. ActionRandomwalk never ends, which means it is necessary to call ClearAllActions in order to allow a creature to perform any other action once ActionRandomWalk has been called.

ActionRest ([*check_sight*])
The creature will rest if not in combat and no enemies are nearby.

Parameters

- **check_sight** (*bool*) – If `true` allow creature to rest if enemies are nearby. Default: `false`.

ActionSit (*chair*)

Parameters

- **chair** (*Placeable*) – Object to sit on.

ActionTouchAttackMelee (*target*[, *feedback*])
Add melee touch attack action.

Parameters

- **target** – Object to perform attack on.
- **feedback** (*bool*) – If `true` feedback will be displayed in the combat log. Default: `true`

Return type 0 for miss, 1 for hit, 2 for critical hit.

ActionTouchAttackRanged (*target*[, *feedback*])
Add melee touch attack action.

Parameters

- **target** – Object to perform attack on.
- **feedback** (*bool*) – If `true` feedback will be displayed in the combat log. Default: `true`

Return type 0 for miss, 1 for hit, 2 for critical hit.

ActionUnequipItem (*item*)

Parameters

- **item** (*Item*) – Item to unequip.

ActionUseFeat (*feat*, *target*)

Parameters

- **feat** (*int*) – FEAT_*
- **target** – Target
- **target** – *Object*

ActionUseItem (*item*, *target*, *area*, *loc*, *prop*)

ActionUseSkill (*skill*, *target*, *subskill*, *item*)

Parameters

- **skill** (*int*) – SKILL_* constant.
- **target** (*Object*) – Object to target.
- **subskill** (*int*) – SUBSKILL_* constant.
- **item** (*Item*) – Item used in conjunction with skill.

ActionUseTalentAtLocation (*talent*, *loc*)

Parameters

- **talent** (`Talent`) – Talent to use.
- **loc** (*Location*) – Location to use talent.

ActionUseTalentOnObject (*talent*, *target*)

Parameters

- **talent** (`Talent`) – Talent to use.
- **target** (*Object*) – Target to use talent on.

ActivatePortal (*ip*, *password*, *waypoint*, *seemless*)
Activates a portal between servers.

Parameters

- **ip** (*string*) – DNS name or IP address (and optional port) of new server.
- **password** (*string*) – Password for login to the destination server. Default: ""
- **waypoint** (*string*) – If set, arriving PCs will jump to this waypoint after appearing at the start location. Default: ""
- **seemless** (*bool*) – If true, the transition will be made 'seamless', and the PC will not get a dialog box on transfer. Default: `false`

AddHenchman (*master*)
Adds a henchman NPC to a PC.

Parameters

- **master** (*Creature*) – NPCs new master.

AddJournalQuestEntry (*plot*, *state*, *entire_party*, *all_pc*, *allow_override*)
Add an entry to a player's Journal. (Create the entry in the Journal Editor first).

Parameters

- **plot** (*string*) – The tag of the Journal category (case sensitive).
- **state** (*int*) – The ID of the Journal entry.
- **entire_party** (*bool*) – If true, the entry is added to the journal of the entire party. Default: `true`
- **all_pc** (*bool*) – If true, the entry will show up in the journal of all PCs in the module. Default: `false`
- **allow_override** (*bool*) – If true, override restriction that nState must be > current Journal Entry. Default: `false`

AddKnownFeat (*feat*[, *level*])
Add known feat to creature

Parameters

- **feat** (*int*) – FEAT_*
- **level** (*int*) – If level is specified feat will be add at that level. Default: 0

AddKnownSpell (*sp_class*, *sp_id*, *sp_level*)
Add known spell to creature.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_id** (*int*) – SPELL_*
- **sp_level** (*int*) – Spell level.

AddToParty (*leader*)
Add PC to party

Parameters

- **leader** (*Creature*) – Faction leader

AdjustAlignment(*alignment*, *amount*[, *entire_party*])
Adjust creature's alignment.

Parameters

- **alignment** (*int*) – ALIGNMENT_* constant.
- **amount** (*int*) – Amount to adjust
- **entire_party** (*bool*) – If true entire faction's alignment will be adjusted. Default: `false`

AdjustReputation(*target*, *amount*)
Adjust reputation

Parameters

- **target** – Target
- **amount** (*int*) – Amount to adjust

BlackScreen()
Sets the screen to black.

BootPC()
Abruptly kicks a player off a multi-player server.

ChangeToStandardFaction()
Changes creature to standard faction

Classes()
TODO: What does this return?

ClearPersonalReputation(*target*)
Clears personal reputation

Parameters

- **target** – Target

DayToNight([*transition_time*])
Changes the current Day/Night cycle for this player to night

Parameters

- **transition_time** (*float*) – Time it takes to become night. Default: 0

DecrementRemainingFeatUses(*feat*)
Decrement remaining feat uses.

Parameters

- **feat** (*int*) – FEAT_*

DecrementRemainingSpellUses(*spell*)
Decrements the remaining uses of a spell.

Parameters

- **spell** (*int*) – SPELL_*

Equips(*creature*)
Iterator of a creature's equipped items.

Parameters

- **creature** (*bool*) – If true include creature items. Default: `false`

ErrorMessage (*message*, ...)
Send error message on server channel.

Parameters

- **message** (*string*) – Format string, see string.format
- **...** – Arguments to format string

ExploreArea (*area*[, *explored*])
Reveals the entire map of an area to a player.

Parameters

- **area** (*Area*) – Area to explorer.
- **explored** (*bool*) – `true` (explored) or `false` (hidden). Whether the map should be completely explored or hidden. Default: `true`

FactionMembers ([*pc_only*])
Faction Member Iterator.

Parameters

- **pc_only** (*bool*) – If true NPCs will be ignored. Default: `true`

FadeFromBlack ([*speed*])
Fades screen from black

Parameters

- **speed** (*int*) – FADE_SPEED_* constant. Default: FADE_SPEED_MEDIUM.

FadeToBlack ([*speed*])
Fades screen to black

Parameters

- **speed** (*int*) – FADE_SPEED_* constant. Default: FADE_SPEED_MEDIUM.

ForceEquip (*equips*)
Forces creature to equip items

Parameters

- **equips** (*table*) – A table with items indexed by INVENTORY_SLOT_* constants.

ForceUnequip (*item*)
Forces creature to unequip an item

Parameters

- **item** (*Item*) – The item in question.

GetAbilityIncreaseByLevel (*level*)
Gets ability score that was raised at a particular level.

Parameters

- **level** (*int*) – Level in question.

GetAbilityModifier (*ability*[, *base*])
Get the ability score of a specific type for a creature.

Parameters

- **ability** (*int*) – ABILITY_*.
- **base** (*bool*) – If `true` will return the base ability modifier without bonuses (e.g. ability bonuses granted from equipped items). (Default: `false`)

Return type Returns the ability modifier of type ability for self (otherwise -1).

GetAbilityScore (*ability*[, *base*])
Get the ability score of a specific type for a creature.

Parameters

- **ability** (*int*) – ABILITY_*.
- **base** (*bool*) – If `true` will return the base ability score without bonuses (e.g. ability bonuses granted from equipped items). (Default: `false`)

Return type Returns the ability score of type ability for self (otherwise -1).

GetDexMod ([*armor_check*])
Gets a creatures dexterity modifier.

Parameters

- **armor_check** (*bool*) – If true uses armor check penalty. (Default: `false`)

GetAILevel ()
Gets creature's AI level.

GetActionMode (*mode*)
Check if a creature is using a given action mode

Parameters

- **mode** (*int*) – ACTION_MODE_*

GetAge ()
Gets creature's age.

GetAlignmentGoodEvil ()
Determines the disposition of a creature.

Return type ALIGNMENT_* constant.

GetAlignmentLawChaos ()
Determines the disposition of a creature.

Return type ALIGNMENT_* constant.

GetAnimalCompanionName ()
Gets a creature's animal companion name.

Return type `string`

GetAnimalCompanionType ()
Get a creature's familiar creature type.

Return type Animal companion constant.

GetAppearanceType ()
Gets creature's appearance type

GetArcaneSpellFailure ()
Get creature's arcane spell failure.

GetAssociate(*assoc_type*[, *nth*])
Returns an object's associate.

Parameters

- **assoc_type** (*int*) – solstice.associate type constant.
- **nth** (*int*) – Which associate to return. Default: 1

GetAssociateType()
Returns the associate type of the specified creature

Return type associate type constant.

GetAttackTarget()
Get creature's attack target

GetAttemptedAttackTarget()
Get creature's attempted attack target

GetAttemptedSpellTarget()
Get creature's attempted spell target

GetBICFileName()

GetBodyPart(*part*)
Gets creature's body part

Parameters

- **part** (*int*) –

GetBonusSpellSlots(*sp_class*, *sp_level*)
Get bonus spell slots.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.

GetChallengeRating()
Get creature's challenge rating

GetClassByLevel(*level*)
Determines class that was chosen at a particular level.

Parameters

- **level** (*int*) – Level to get class at.

Return type CLASS_TYPE_* constant or CLASS_TYPE_INVALID on error.

GetClassByPosition(*position*)
Get class type by position

Parameters

- **position** (*int*) – Valid values: 0, 1, or 2

Return type CLASS_TYPE_* or CLASS_TYPE_INVALID.

GetClericDomain(*domain*)
Determines a cleric's domain.

Parameters

- **domain** (*int*) – Cleric's first or second domain.

GetCombatMode()
Gets creature's active combat mode.

Return type COMBAT_MODE_* constant.

GetConversation()
Gets creature's conversation resref

GetCutsceneCameraMoveRate()
Get cutscene camera movement rate.

GetCutsceneMode()
Get a creature's cutscene mode

GetDamageFlags()
Get creature's damage flags.

GetDeity()
Gets creature's deity.

GetDetectMode()

GetFactionEqual(*target*)
Get if factions are equal.

Parameters

- **target** – Target

GetFamiliarName()
Gets the creature's familiar creature name.

GetFamiliarType()
Gets the creature's familiar creature type.

Return type FAMILIAR_*

GetFavoredEnemenyMask()
Determine Creatures Favored Enemey Bit Mask.

GetGender()
Gets creature's gender.

GetGoingToBeAttackedBy()
Get creatures attacker.

GetGoodEvilValue()
Determines a creature's good/evil rating.

GetHasFeat(*feat*, *has_uses*, *check_successors*)
Determine if creature has a feat

Parameters

- **feat** (*int*) – FEAT_*
- **has_uses** (*bool*) – Check the feat is usable. Default: `false`
- **check_successors** (*bool*) – Check feat successors. Default: `false`

GetHasFeatEffect(*feat*)
Determines if creature has a feat effect.

Parameters

- **feat** (*int*) – FEAT_*

GetHasSkill(*skill*)
Determines if a creature has a skill

Parameters

- **skill** (*int*) – SKILL_*.

GetHasSpell(*spell*)
Determines whether a creature has a spell available.

Parameters

- **spell** (*int*) – SPELL_*

GetHasTalent(*talent*)
Determines whether a creature has a specific talent.

Parameters

- **talent** – The talent which will be checked for on the given creature.

GetHasTrainingVs(*vs*)
Determine if creature training vs.

Parameters

- **vs** (*Creature*) – Target.

GetHenchman(*nth*)
Gets the nth henchman of a PC.

Parameters

- **nth** (*int*) – Henchman index.

GetHighestFeat(*feat*)
Determines the highest known feat. This function checks all feat successors.

Parameters

- **feat** (*int*) – FEAT_*

Return type `bool` indicating whether creature has feat and the highest feat.

GetHighestFeatInRange(*low_feat*, *high_feat*)
Returns the highest feat in a range of feats.

Parameters

- **low_feat** (*int*) – FEAT_*
- **high_feat** (*int*) – FEAT_*

Return type FEAT_* or -1 on error.

GetHighestLevelClass()
Determines creatures highest class level

Return type CLASS_TYPE_*, level

GetHitDice(*use_neg_levels*)
Calculate a creature's hit dice.

Parameters

- **use_neg_levels** (*bool*) – If true negative levels factored in to total hit dice. Default: `false`

GetInventorySlotFromItem(*item*)
Determine inventory slot from item

Parameters

- **item** (*Item*) – Item

Return type INVENTORY_SLOT_* or -1

GetIsAI()
Determine if creature is an AI.

GetIsBlind()
Determines if a creature is blind.

GetIsBoss()
Determine boss creature. TODO: This should be removed.

GetIsDM()
Determines if Creature is a DM

GetIsDMPossessed()
Gets if creature is possessed by DM.

GetIsEncounterCreature()
Get if creature was spawned by encounter.

GetIsEnemy(*target*)
Determine if target is an enemy

Parameters

- **target** – Target

GetIsFavoredEnemy(*vs*)
Determine if creature is favored enemy.

GetIsFlanked(*vs*)
Determines if a creature is flanked.

Parameters

- **vs** (*Creature*) – Attacker

GetIsFlatfooted()
Determines if a creature is flatfooted.

GetIsFriend(*target*)
Determine if target is a friend

Parameters

- **target** – Target

GetIsHeard(*target*)
Determines if an object can hear another object.

Parameters

- **target** – The object that may be heard.

GetIsImmune(*immunity*, *versus*)
Get if creature has immunity.

Parameters

- **immunity** (*int*) – IMMUNITY_TYPE_*

- **versus** (*Object*) – Versus object.

Return type bool

GetIsInCombat()
Determines if creature is in combat.

GetIsInConversation()
Determines whether an object is in conversation.

GetIsInvisible(*vs*)
Determines if target is invisible.

Parameters

- **vs** (*Object*) – Creature to test again.

GetIsNeutral(*target*)
Determine if target is a neutral

Parameters

- **target** – Target

GetIsPC()
Determine if creature is a PC.

GetIsPCDying()
TODO: ???

GetIsPolymorphed()
Get if creature is polymorphed

GetIsPossessedFamiliar()
Retrieves the controller status of a familiar.

GetIsReactionTypeFriendly(*target*)
Determine reaction type if friendly

Parameters

- **target** (*Object*) – Target

GetIsReactionTypeHostile(*target*)
Determine reaction type if hostile

Parameters

- **target** (*Object*) – Target

GetIsReactionTypeNeutral(*target*)
Determine reaction type if neutral.

Parameters

- **target** (*Object*) – Target

GetIsResting()
Check whether a creature is resting.

GetIsSeen(*target*)
Determines whether an object sees another object.

Parameters

- **target** (*Object*) – Object to determine if it is seen.

GetIsSkillSuccessful (*skill*, *dc*, *vs*, *feedback*, *auto*, *delay*, *take*, *bonus*)
Determines if skill check is successful

Parameters

- **skill** (*int*) – SKILL_*
- **dc** – Difficulty Class
- **vs** – Versus a target
- **feedback** – If true sends feedback to participants.
- **auto** – If true a roll of 20 is automatic success, 1 an automatic failure
- **delay** – Delay in seconds.
- **take** – Replaces dice roll.
- **bonus** – And bonus.

GetIsWeaponEffective (*vs*, *is_offhand*)
Determines if weapon is effect versus a target.

Parameters

- **vs** – Attack target.
- **is_offhand** – true if the attack is an offhand attack.

GetItemInSlot (*slot*)
Gets an equipped item in creature's inventory.

Parameters

- **slot** – INVENTORY_SLOT_*

GetKnownFeat (*index*)
Gets known feat at index

Parameters

- **index** (*int*) – Index of feat

GetKnownFeatByLevel (*level*, *idx*)
Gets known feat by level at index

Parameters

- **level** – Level in question.
- **idx** – Index of feat.

GetKnownSpell (*sp_class*, *sp_level*, *sp_idx*)
Gets known spell.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_idx** – Index of the spell.

Return type SPELL_* or -1 on error.

GetKnowsFeat (*feat*)
Determines if a creature knows a feat. Feats acquired from gear/effects do not count.

Parameters

- **feat** (*int*) – FEAT_*

GetKnowsSpell(*sp_class*, *sp_id*)
Determines if creature knows a spell.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_id** – SPELL_*

GetLastAssociateCommand()
Get the last command issued to a given associate.

Return type COMMAND_*

GetLastAttackMode()
Get's last attack mode used by creature.

GetLastAttackType()
Get's last attack type used by creature.

GetLastPerceived()
Determines the last perceived creature in an OnPerception event.

GetLastPerceptionHeard()
Determines if the last perceived object was heard.

GetLastPerceptionInaudible()
Determines whether the last perceived object is no longer heard.

GetLastPerceptionSeen()
Determines if the last perceived object was seen.

GetLastPerceptionVanished()
Determines the last perceived creature has vanished.

GetLastTrapDetected()
Gets last trap detected by creature.

GetLastWeaponUsed()
Gets last weapon used by creature.

GetLawChaosValue()
Determines a creature's law/chaos value.

GetLevelByClass(*class*)
Get number of levels a creature by class

Parameters

- **class** (*int*) – CLASS_TYPE_* type constant.

GetLevelByPosition(*position*)
Get number of levels a creature by position

Parameters

- **position** (*int*) – Valid values: 0, 1, or 2

GetLevelStats(*level*)

GetMaster()
Determines who controls a creature.

GetMaxAttackRange(*target*)
Determines creatures maximum attack range.

Parameters

- **target** (*Object*) – Target to attack

GetMaxHitPoints()
Get creature's maximum hit points. See `rules.GetMaxHitPoints()`

GetMaxHitPointsByLevel(*level*)
Get max hit points by level

Parameters

- **level** (*int*) – The level in question.

GetMaxSpellSlots(*sp_class*, *sp_level*)
Gets max spell slots.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.

GetMemorizedSpell(*sp_class*, *sp_level*, *sp_idx*)
Determines if a spell is memorized

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_idx** (*int*) – Index of the spell.

GetPCBodyBag()
TODO: Expose this??

GetPCBodyBagID()
TODO: Expose this??

GetPCFileName()
Gets PC characters bic file.

GetPCIPAddress()
Retrieves the IP address of a PC.

GetPCPlayerName()
Retrieves the login name of the player of a PC.

GetPCPublicCDKey(*single_player*)
Retrieves the public version of the PC's CD key.

Parameters

- **single_player** (*bool*) – If set to true, the player's public CD key will be returned when the player is playing in single player mode. Otherwise returns an empty string in single player mode. Default: `false`

GetPhenoType()
Get creature's phenotype

GetPositionByClass(*class*)
Determines class position by class type.

Parameters

- **class** (*int*) – CLASS_TYPE_*

Return type 0, 1, 2, or -1 on error.

GetRacialType()
Gets creature's race.

GetReflexAdjustedDamage(*damage*, *dc*, *savetype*, *versus*)
Determines reflex saved damage adjustment.

Parameters

- **damage** (*int*) – Total damage.
- **dc** (*int*) – Difficulty class
- **savetype** (*int*) – Saving throw type constant.
- **versus** (*Creature*) – Creature to roll against.

GetRelativeWeaponSize(*weap*)
Determines a weapons weapon size relative to a creature.

Parameters

- **weap** (*Item*) – The weapon in question.

GetRemainingFeatUses(*feat*, *has*)
Get remaining feat uses

Parameters

- **feat** (*int*) – FEAT_*
- **has** (*bool*) – If true function assumes that creature has the feat in question. Default: `false`.

GetRemainingSpellSlots(*sp_class*, *sp_level*)
Determines remaining spell slots at level.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.

GetReputation(*target*)
Gets reputation of creature.

Parameters

- **target** – Target

GetSavingThrowBonus(*save*)
Gets creatures saving throw bonus

Parameters

- **save** – SAVING_THROW_*

GetSize()
Gets creature's size

GetSkillCheckResult(*skill*, *dc*, *vs*, *feedback*, *auto*, *delay*, *take*, *bonus*)
Determines a skill check.

Parameters

- **int skill** (*int*) – SKILL_*
- **dc** (*int*) – Difficulty Class
- **vs** – Versus a target
- **feedback** (*bool*) – If `true` sends feedback to participants.
- **auto** (*bool*) – If true a roll of 20 is automatic success, 1 an automatic failure
- **delay** (*float*) – Delay in seconds.
- **take** (*int*) – Replaces dice roll.
- **bonus** (*int*) – And bonus.

GetSkillIncreaseByLevel (*level*, *skill*)
Gets the amount a skill was increased at a level.

Parameters

- **level** (*int*) – Level to check
- **skill** (*int*) – SKILL_*

Return type -1 on error.

GetSkillPoints ()
Returns a creatures unused skillpoints.

GetSkillRank (*skill*[, *vs*[, *base*]])
Gets creature's skill rank.

Parameters

- **skill** (*int*) – SKILL_*
- **vs** – Versus. Default: OBJECT_INVALID
- **base** (*bool*) – If true returns base skill rank. Default: `false`

GetStandardFactionReputation (*faction*)
Get standard faction reputation

Parameters

- **faction** (*int*) – STANDARD_FACTION_* constant.

GetStartingPackage ()
Gets creature's starting package.

GetSubrace ()
Gets creature's subrace

GetTail ()
Gets creature's tail.

GetTalentBest (*category*, *cr_max*)
Determines the best talent of a creature from a group of talents.

Parameters

- **category** (*int*) – TALENT_CATEGORY_*
- **cr_max** (*int*) – The maximum Challenge Rating of the talent.

GetTalentRandom(*category*)
Retrieves a random talent from a group of talents that a creature possesses.

Parameters

- **category** (*int*) – TALENT_CATEGORY_*

GetTargetState(*target*)
Get target state bit mask.

Parameters

- **target** (*Creature*) – Creature target.

GetTotalFeatUses(*feat*)
Get total feat uses.

Parameters

- **feat** (*int*) – FEAT_*

GetTotalKnownFeats()
Get total known feats.

GetTotalKnownFeatsByLevel(*level*)
Get total known feats by level.

Parameters

- **level** (*int*) – The level to check.

Return type -1 on error.

GetTotalKnownSpells(*sp_class*, *sp_level*)
Determines total known spells at level.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.

GetTotalNegativeLevels()
Gets total negative levels

GetTrainingVsMask()

GetTurnResistanceHD()
Determines turn resistance hit dice.

GetWeaponFromAttackType(*atype*)

Parameters

- **atype** (*int*) – ATTACK_TYPE_*

Return type An item or `OBJECT_INVALID`

GetWings()
Gets creature's wings

GetWizardSpecialization()
Gets a creature's wizard specialization.

GetXP()
Gets a creatures XP.

GiveGold (*amount*, *feedback*, *source*)
Gives gold to creature

Parameters

- **amount** (*int*) – Amount of gold to give.
- **feedback** (*bool*) – Sends feedback to creature. Default: `true`
- **source** (*Object*) – Source object. Default: OBJECT_INVALID

IncrementRemainingFeatUses (*feat*)
Increment remaining feat uses.

Parameters

- **feat** (*int*) – FEAT_*

JumpSafeToLocation (*loc*)

Parameters

- **loc** (*Location*) – Location to jump to.

JumpSafeToObject (*obj*)

Parameters

- **obj** (*Object*) – Object to jump to.

JumpSafeToWaypoint (*way*)

Parameters

- **way** (*Waypoint*) – Waypoint to jump to.

LevelUpHenchman ([*class*[, *ready_spells*[, *package*]]])
Levels up a creature using the default settings.

Parameters

- **class** (*int*) – CLASS_TYPE_* Default: CLASS_TYPE_INVALID
- **ready_spells** (*bool*) – Determines if all memorizable spell slots will be filled without requiring rest. Default: `false`
- **package** (*int*) – PACKAGE_* Default: PACKAGE_INVALID

LockCameraDirection ([*locked*])
Locks a creatures camera direction.

Parameters

- **locked** (*bool*) – (Default: false)

LockCameraDistance ([*locked*])
Locks a creatures camera distance.

Parameters

- **locked** (*bool*) – (Default: false)

LockCameraPitch ([*locked*])
Locks a creatures camera pitch.

Parameters

- **locked** (*bool*) – (Default: false)

ModifyAbilityScore (*ability*, *value*)
Modifies the ability score of a specific type for a creature.

Parameters

- **ability** (*int*) – ABILITY_*.
- **value** (*int*) – Amount to modify ability score

ModifySkillRank (*skill*, *amount*, *level*)
Modifies skill rank.

Parameters

- **skill** (*int*) – SKILL_*
- **amount** (*int*) – Amount to modify skill rank.
- **level** (*int*) – If a level is specified the modification will occur at that level.

ModifyXP (*amount*, *direct*)
Modifies a creatures XP.

Parameters

- **amount** (*int*) – Amount of XP to give or take.
- **direct** (*bool*) – If true the xp amount is set directly with no feedback to player. Default: `false`

NightToDay ([*transition_time*])
Changes the current Day/Night cycle for this player to daylight

Parameters

- **transition_time** (*float*) – Time it takes for the daylight to fade in Default: 0

NotifyAssociateActionToggle (*mode*)
Notifies creature's associates of combat mode change

Parameters

- **mode** (*int*) – COMBAT_MODE_* constant.

PlayVoiceChat (*id*)

Parameters

- **id** (*int*) – VOICE_CHAT_* constant.

PopUpDeathGUIPanel (*respawn_enabled*, *wait_enabled*, *help_strref*, *help_str*)
Displays a customizable death panel.

Parameters

- **respawn_enabled** (*bool*) – If `true`, the "Respawn" button will be enabled. Default: `true`
- **wait_enabled** (*bool*) – If `true`, the "Wait For Help" button will be enabled. Default: `true`
- **help_strref** (*int*) – String reference to display for help. Default: 0
- **help_str** (*string*) – String to display for help which appears in the top of the panel. Default: ""

PopUpGUIPanel (*gui_panel*)
Displays a GUI panel to a player.

Parameters

- **gui_panel** (*int*) – GUI_PANEL_* constant.

RecalculateDexModifier()
Recalculates a creatures dexterity modifier.

ReequipItemInSlot(*slot*)
Forces the item in an inventory slot to be reequiped.

Parameters

- **slot** (*int*) – INVENTORY_SLOT_*

RemoveFromParty()
Remove PC from party.

RemoveHenchman(*master*)
Removes the henchmen from the employ of a PC.

Parameters

- **master** (*Creature*) – Henchman's master

RemoveJournalQuestEntry(*plot*, *entire_party*, *all_pc*)
Removes a journal quest entry from a PCs journal.

Parameters

- **plot** (*string*) – The tag for the quest as used in the toolset's Journal Editor.
- **entire_party** (*bool*) – If this is true, the entry will be removed from the journal of everyone in the party. Default: `true`
- **all_pc** (*bool*) – If this is true, the entry will be removed from the journal of everyone in the world. Default: `false`

RemoveKnownFeat(*feat*)
Remove feat from creature.

Parameters

- **feat** (*int*) – FEAT_*

RemoveKnownSpell(*sp_class*, *sp_level*, *sp_id*)
Remove known spell from creature

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_id** (*int*) – SPELL_*

RemoveSummonedAssociate(*master*)
Removes an associate NPC from the service of a PC.

Parameters

- **master** (*Creature*) – Creature's master.

ReplaceKnownSpell(*sp_class*, *sp_id*, *sp_new*)
Remove known spell from creature

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.

- **sp_id** (*int*) – SPELL_*
- **sp_new** (*int*) – SPELL_*

RestoreBaseAttackBonus()
Restores a creature's base number of attacks.

RestoreCameraFacing()
Restore creatures camera orientation.

SendChatMessage(*channel*, *from*, *message*)
Sends a chat message

Parameters

- **channel** (*int*) – Channel the message to send message on.
- **from** (*Object*) – Sender.
- **message** (*string*) – Text to send.

SendMessage(*message*, ...)
Sends a message to the PC.

Parameters

- **message** (*string*) – Format string, see string.format
- **...** – Arguments to format string

SendMessageByStrRef(*strref*)
Sends a message to the PC by StrRef.

Parameters

- **strref** (*int*) – StrRef of the message to send

SendServerMessage(*message*, ...)
Simple wrapper around `SendChatMessage()` that sends a server message to a player.

Parameters

- **message** (*string*) – Format string, see string.format
- **...** – Arguments to format string

SetAILevel(*ai_level*)
Sets creature's AI level.

Parameters

- **ai_level** (*int*) – AI_LEVEL_* constant.

SetActionMode(*mode*, *status*)
Sets the status of an action mode on a creature

Parameters

- **mode** (*int*) – ACTION_MODE_*
- **status** (*int*) – New value.

SetAge(*age*)
Set creature's age.

Parameters

- **age** (*int*) – New age.

SetAppearanceType (*type*)
Sets creature's appearance type

Parameters

- **type** (*int*) – Appearance type.

SetAssociateListenPatterns ()
Prepares an associate (henchman, summoned, familiar) to be commanded.

SetBaseAttackBonus (*amount*)
Sets a creature's base number of attacks.

Parameters

- **amount** (*int*) – Amount of attacks.

SetBodyPart (*part*, *model_number*)
Sets creature's body part

Parameters

- **part** – CREATURE_PART_* constant.
- **model_number** – CREATURE_MODEL_TYPE_* constant.

SetCameraFacing (*direction*[, *distance*[, *pitch*[, *transition_type*]]])
Set creatures camera orientation.

Parameters

- **direction** (*float*) – direction to face.
- **distance** (*float*) – Camera distance. Default: -1.0
- **pitch** (*float*) – Camera pitch. Default: -1.0
- **transition_type** (*int*) – CAMERA_TRANSITION_TYPE_* constant. Default: CAMERA_TRANSITION_TYPE_SNAP

SetCameraHeight (*height*)
Set camera height

Parameters

- **height** (*int*) – New height.

SetCameraMode (*mode*)
Set Camera mode

Parameters

- **mode** (*int*) – New mode

SetClericDomain (*domain*, *newdomain*)
Sets a cleric's domain.

Parameters

- **domain** (*int*) – Cleric's first or second domain
- **newdomain** (*int*) – See domains.2da

SetCombatMode (*mode*, *change*)
Sets creature's combat mode

Parameters

- **mode** (*int*) – COMBAT_MODE_* constant.
- **change** (*bool*) – If false the combat mode is already active.

SetCutsceneCameraMoveRate (*rate*)
Sets camera movement rate.

Parameters

- **rate** (*float*) – New movement rate

SetCutsceneMode (*in_cutscene*, *leftclick_enabled*)
Sets cutscene move

Parameters

- **in_cutscene** (*bool*) – Default: `false`
- **leftclick_enabled** (*bool*) – Default: `false`

SetDeity (*deity*)
Sets creature's deity

Parameters

- **deity** (*string*) – New deity

SetGender (*gender*)
Sets creature's gender

Parameters

- **gender** (*int*) – New gender

SetIsTemporaryEnemy (*target*, *decays*, *duration*)
Set creature as a temporary enemy

Parameters

- **target** (*Object*) – Target
- **decays** (*bool*) – If true reactions will return after duration. Default: `false`
- **duration** (*float*) – Time in seconds. Default: 180.0

SetIsTemporaryFriend (*target*, *decays*, *duration*)
Set creature as a temporary friend

Parameters

- **target** (*Object*) – Target
- **decays** (*bool*) – If true reactions will return after duration. Default: `false`
- **duration** (*float*) – Time in seconds. Default: 180.0

SetIsTemporaryNeutral (*target*, *decays*, *duration*)
Set creature as a temporary neutral

Parameters

- **target** (*Object*) – Target
- **decays** (*bool*) – If true reactions will return after duration. Default: `false`
- **duration** (*float*) – Time in seconds. Default: 180.0

SetKnownFeat (*index*, *feat*)
Set known feat on creature

Parameters

- **index** (*int*) – Feat index to set
- **feat** (*int*) – FEAT_*

SetKnownFeatByLevel (*level*, *index*, *feat*)
Set known feat by level

Parameters

- **level** (*int*) – Level to set the feat on.
- **index** (*int*) – Feat index
- **feat** (*int*) – FEAT_*

SetKnownSpell (*sp_class*, *sp_level*, *sp_idx*, *sp_id*)
Sets a known spell on creature

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_idx** (*int*) – Index of the spell to change.
- **sp_id** (*int*) – SPELL_*

SetLootable (*lootable*)
Sets creature lootable

Parameters

- **lootable** (*int*) – New lootable value

SetMaxHitPointsByLevel (*level*, *hp*)
Set max hitpoints by level.

Parameters

- **level** (*int*) – The level in question.
- **hp** (*int*) – Amount of hitpoints.

SetMemorizedSpell (*sp_class*, *sp_level*, *sp_idx*, *sp_spell*, *sp_meta*, *sp_flags*)
Sets a memorized spell on creature

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_idx** (*int*) – Index of the spell to change.
- **sp_spell** (*int*) – SPELL_*
- **sp_meta** (*int*) – METAMAGIC_*
- **sp_flags** (*int*) – Spell flags.

SetMovementRate (*rate*)
Set creatures movement rate.

Parameters

- **rate** (*int*) – MOVE_RATE_*

SetPCBodyBag(*bodybag*)
TODO: Expose???

SetPCBodyBagID(*bodybagid*)
TODO: Expose???

SetPCDislike(*target*)
Sets that a player dislikes a creature (or object).

Parameters

- **target** (*Creature*) – The creature that dislikes the PC (and the PC dislike it).

SetPCLike(*target*)
Causes a creature to like a PC.

Parameters

- **target** (*Creature*) – Target to alter the feelings of.

SetPCLootable(*lootable*)
TODO: Expose???

SetPanelButtonFlash(*button*, *enable_flash*)
Make a panel button in the player's client start or stop flashing.

Parameters

- **button** (*int*) – PANEL_BUTTON_* constant.
- **enable_flash** (*bool*) – `true` to flash, `false` to stop flashing

SetPhenoType(*phenotype*)
Set creature's phenotype

Parameters

- **phenotype** (*int*) – Phenotype constant.

SetRemainingSpellSlots(*sp_class*, *sp_level*, *sp_slots*)
Sets a remaining spell slots on creature.

Parameters

- **sp_class** (*int*) – CLASS_TYPE_*.
- **sp_level** (*int*) – Spell level.
- **sp_slots** (*int*) – Number of slots.

SetSavingThrowBonus(*save*, *bonus*)
Sets creatures saving throw bonus

Parameters

- **save** (*int*) – SAVING_THROW_* constant.
- **bonus** (*int*) – New saving throw bonus

SetSkillPoints(*amount*)
Sets a creatures skillpoints available.

Parameters

- **amount** (*int*) – New amount

SetSkillRank(*skill*, *amount*)
Sets a creatures skill rank

Parameters

- **skill** (*int*) – SKILL_*
- **amount** (*int*) – New skill rank

SetStandardFactionReputation (*faction*, *rep*)
Set standard faction reputation

Parameters

- **faction** (*int*) – STANDARD_FACTION_* constant.
- **rep** (*int*) – Reputation. 0-100 inclusive.

SetSubrace (*subrace*)
Set creature's subrace

Parameters

- **subrace** (*string*) – New subrace

SetTail (*tail*)
Sets creature's tail

Parameters

- **tail** (*int*) – Tail type constant.

SetWings (*wings*)
Sets creature's wings

Parameters

- **wings** (*int*) – Wing type constant.

SetWizardSpecialization (*specialization*)
Set a wizard's specialization.

Parameters

- **specialization** (*int*) – see schools.2da

SetXP (*amount*, *direct*)
Sets a creatures XP

Parameters

- **amount** (*int*) – Amount to set XP to.
- **direct** (*bool*) – If true the xp amount is set directly with no feedback to player. Default: `false`.

SpeakOneLinerConversation (*resref*[, *target*])

Parameters

- **resref** (*string*) – Dialog resref.
- **target** (*int*) – Must be specified if there are creature specific tokens in the string. Default: OBJECT_TYPE_INVALID

StopFade ()
Stops a screen fade

StoreCameraFacing ()
Stores camera orientation.

SuccessMessage (*message*, ...)
Send success message on server channel.

Parameters

- **message** (*string*) – Format string, see string.format
- **...** – Arguments to format string

SummonAnimalCompanion ()
Summons creature's animal companion

SummonFamiliar ()
Summons creature's familiar

SurrenderToEnemies ()
Causes all creatures in a 10 meter (1 tile) radius to stop actions. Improves the creature's reputation with nearby enemies for 3 minutes. Only works for NPCs.

TakeGold (*amount*, *feedback*, *source*)
Takes gold to creature

Parameters

- **amount** (*int*) – Amount of gold to take.
- **feedback** (*bool*) – Sends feedback to creature. Default: `true`
- **source** (*Object*) – Source object. Default: `OBJECT_INVALID`

UnpossessFamiliar ()
Unpossesses a familiar from its controller.

4.11.4 class Door

class Door

GetIsActionPossible (*action*)
Determines whether an action can be used on a door.

Parameters

- **action** (*int*) – DOOR_ACTION_*

DoAction (*action*)
Does specific action to target door.

Parameters

- **action** (*int*) – DOOR_ACTION_*

4.11.5 class Encounter

class Encounter

GetActive ()
Gets whether an encounter has spawned as is active.

GetDifficulty ()
Get the difficulty level of the encounter.

Return type ENCOUNTER_DIFFICULTY_*

GetNumberSpawned()
Get number of creatures spawned

GetSpawnsCurrent()
Get the number of times that the encounter has spawned so far.

GetSpawnsMax()
Get the maximum number of times that an encounter will spawn.

GetSpawnPointCount()
Gets the number of spawn points

GetSpawnPointByIndex(*idx*)
Gets a spawn point location.

Parameters

- **idx** (*int*) – Index in the spawn poing list.

Return type *Location*

SetActive(*value*)
Sets an encounter to active or inactive.

Parameters

- **value** (*boolean*) – new value

SetDifficulty(*value*)
Sets the difficulty level of an encounter.

Parameters

- **value** (*int*) – ENCOUNTER_DIFFICULTY_*

SetSpawnsMax(*value*)
Sets the maximum number of times that an encounter can spawn.

Parameters

- **value** (*int*) – The new maximum spawn value.

SetSpawnsCurrent(*value*)
Sets the number of times that an encounter has spawned.

Parameters

- **value** (*int*) – The new number of times the encounter has spawned.

4.11.6 class Effect

class Effect

ToString()
Converts an effect to a formatted string.

GetCreator()

Return type *Object*

GetDuration()
Gets the duration of an effect.

Return type The duration specified when applied for the effect. The value of this is undefined for effects which are not of DURATION_TYPE_TEMPORARY.

GetDurationRemaining()
Gets the remaing duration of an effect

Return type The remaining duration of the specified effect. The value of this is undefined for effects which are not of DURATION_TYPE_TEMPORARY.

GetDurationType()
Get duration type

Return type DURATION_TYPE_*

GetFloat(*index*)
Get effect float value.

Parameters

- **index** (*int*) – Index

GetId()
Gets the specifed effects Id

GetIsValid()
Determines whether an effect is valid.

GetInt(*index*)
Get effect integer value at the index specified.

Parameters

- **index** (*int*) – Index

GetObject(*index*)
Get effect object value.

Parameters

- **index** (*int*) – Index

GetSpellId()
Gets Spell Id associated with effect

Return type SPELL_* constant.

GetString(*index*)
Gets a string on an effect.

Parameters

- **index** (*int*) – Index to store the string. [0, 5]

GetSubType()
Get the subtype of the effect.

Return type SUBTYPE_* constant.

GetType()
Gets effects internal 'true' type.

SetAllInts(*val*)
Set all integers to a specified value

SetCreator (*object*)
Sets the effects creator

Parameters

- **object** (*Object*) – New effect creator.

SetDuration (*dur*)

SetDurationType (*dur*)

SetFloat (*index*, *float*)
Set effect float

Parameters

- **index** (*int*) – Index. [0, 3]
- **float** (*float*) – Float

SetInt (*index*, *value*)
Sets the internal effect integer at the specified index to the value specified. Source: nwnx_structs by Acaos

SetNumIntegers (*num*)
Set number of integers stored on an effect. Calling this on an effect will erase any integers already stored on the effect.

Parameters

- **num** (*int*) – Number of integers.

SetObject (*index*, *object*)
Set effect object

Parameters

- **index** (*int*) – Index. [0, 3]
- **object** (*Object*) – Object

SetSpellId (*spellid*)
Sets the effect's spell id as specified, which will later be returned with Effect:GetSpellId().

Parameters

- **spellid** (*int*) – SPELL_* constant.

SetString (*index*, *str*)
Sets a string on an effect.

Parameters

- **index** (*int*) – Index to store the string. [0, 5]
- **str** (*string*) – String to store.

SetSubType (*value*)
Set the subtype of the effect.

Parameters

- **value** (*int*) – SUBTYPE_*

SetType (*value*)
Sets effects type.

Parameters

- **value** (*int*) – EFFECT_TYPE_*

SetExposed(*val*)
Set exposed.

Parameters

- **val** (*boolean*) – Value

SetIconShown(*val*)
Set icon shown.

Parameters

- **val** (*boolean*) – Value

4.11.7 class Item

class Item

GetEntireAppearance()
Encodes an items appearance. Source: nwnx_funcs by Acaos

Return type A string encoding the appearance

GetItemAppearance(*appearance_type*, *index*)
Returns the appearance of an item

Parameters

- **appearance_type** (*int*) – ITEM_APPR_TYPE_*
- **index** (*int*) – ITEM_APPR_WEAPON_* or ITEM_APPR_ARMOR_*

RestoreAppearance(*appearance*)
Restores an items appearance.

Parameters

- **appearance** (*string*) – An encoding from *Item:GetEntireAppearance()*

SetAppearance(*index*, *value*)
Set item appearance

Parameters

- **index** (*int*) – index
- **value** (*int*) – value

SetColor(*index*, *value*)
Set item color

Parameters

- **index** (*int*) – index
- **value** (*int*) – value

GetACValue()
Get the armor class of an item.

ComputeArmorClass()
Compute armor class.

GetBaseArmorACBonus()
Gets Armor's Base AC bonus.

Note: Note this is currently hardcoded to the typical vanilla NWN values.

Return type -1 if item is not armor.

Copy(*[target, copy_vars]]*)
Duplicates an item.

Parameters

- **target** (*Object*) – Create the item within this object's inventory. (Default: OBJECT_INVALID)
- **copy_vars** (*boolean*) – If true, local variables on item are copied. (Default: false)

CopyAndModify(*modtype*, *index*, *value*, *copy_vars*)
Copies an item, making a single modification to it

Parameters

- **modtype** – Type of modification to make.
- **index** (*int*) – Index of the modification to make.
- **value** (*int*) – New value of the modified index
- **copy_vars** (*boolean*) – If true, local variables on item are copied. (Default: false)

GetBaseType()
Get the base item type.

Return type BASE_ITEM_INVALID if invalid item.

SetBaseType(*value*)
Sets an items base type

Parameters

- **value** (*int*) – BASE_ITEM_*

GetGoldValue()
Determines the value of an item in gold pieces.

SetGoldValue(*value*)
Sets an items gold piece value when IDed Source: nwnx_funcs by Acaos

Parameters

- **value** (*int*) – New gold value.

GetStackSize()
Get item's stack size.

SetStackSize(*value*)
Set item's stack size.

Parameters

- **value** (*int*) – New stack size.

GetPossesor()
Get item possessor.

AddItemProperty(*dur_type*, *ip*, *duration*)
Add an itemproperty to an item

Parameters

- **dur_type** (*int*) – DURATION_TYPE_*
- **ip** (*Itemprop*) – Itemproperty to add.
- **duration** (*float*) – Duration Duration in seconds in added temporarily. (Default: 0.0)

GetHasItemProperty(*ip_type*)
Check whether an item has a given property.

Parameters

- **ip_type** (*int*) – ITEM_PROPERTY_*

ItemProperties()
Iterator over items properties

RemoveItemProperty(*ip*)
Removes an item property

Parameters

- **ip** (*Itemprop*) – Item property to remove.

GetDroppable()
Determines if an item can be dropped.

SetDroppable(*flag*)
Set droppable flag.

Parameters

- **flag** (*boolean*) – New value.

GetIdentified()
Determines whether an object has been identified.

GetInfiniteFlag()
Gets if there is an infinite quantity of any item in a store.

SetIdentified([*is_ided*])
Sets an item identified

Parameters

- **is_ided** (*boolean*) – (Default: `false`)

SetInfiniteFlag([*infinite*])
Sets and items infinite quantity flag.

Parameters

- **infinite** (*boolean*) – (Defaut: `false`)

GetCursedFlag()
Get item cursed flag.

SetCursedFlag(*flag*)
Set item cursed flag.

Parameters

- **flag** (*boolean*) – New flag.

GetWeight()
Gets item weight.

SetWeight(*weight*)
Sets item's weight.

Parameters

- **weight** (*int*) – New weight.

4.11.8 class Itemprop

Itemprop inherits from *Effect*

class Itemprop

ToString()
Convert itemprop effect to formatted string. Overrides *Effect:ToString()*

GetCostTable()
Returns the cost table number of the itemproperty. See the 2DA files for value definitions.

GetCostTableValue()
Returns the cost table value of an itemproperty. See the 2DA files for value definitions.

GetParam1()
Returns the Param1 number of the item property.

GetParam1Value()
Returns the Param1 value of the item property.

GetPropertySubType()
Returns the subtype of the itemproperty

GetPropertyType()
Returns the subtype of the itemproperty

SetValues(*type*, *subtype*, *cost*, *cost_val*, *param1*, *param1_val*, *chance*)
Sets the item property effect values directly.

4.11.9 class Location

class Location

Constants

INVALID
Invalid location. Aliased globally as LOCATION_INVALID.

Functions

Location.**Create**(*position*, *orientation*, *area*)
Create a new location

Parameters

- **position** (*Vector*) – Location's position

- **orientation** (*Vector*) – Location's orientation
- **area** (*Area*) – Location's area

Return type *Location* instance.

Location.**FromString**(*str*)
Convert string to location.

Parameters

- **str** (*string*) – String representation of a location. Format: "area_tag (x, y, z) orientation"

Return type *Location* instance.

Methods

Location:**ApplyEffect**(*durtype*, *eff*, *duration*)
Applies an effect to a location.

Parameters

- **durtype** (*int*) – DURATION_TYPE_*
- **eff** (*Effect*) – Effect to apply.
- **duration** (*float*) – Duration in seconds. If not passed the visual will be applied as DURATION_TYPE_INSTANT.

Location:**ApplyVisual**(*vfx*, *duration*)
Applies a visual effect to a location

Parameters

- **vfx** (*int*) – VFX_*
- **duration** (*float*) – Duration in seconds. If not passed the visual will be applied as DURATION_TYPE_INSTANT.

Location:**GetNearestObject**(*mask*, *nth*)
Gets nearest object to location

Parameters

- **mask** (*int*) – OBJECT_TYPE_*
- **nth** (*int*) – Which object to find.

Location:**GetNearestCreature**(*type1*, *value1*, *nth*, ...)
Gets nearest creature to location.

Parameters

- **type1** (*int*) – First criteria type
- **value1** (*int*) – First crieria value
- **nth** (*int*) – Nth nearest.
- **type2** (*int*) – Second criteria type. (Default: -1)
- **value2** (*int*) – Second criteria value. (Default: -1)
- **type3** (*int*) – Third criteria type. (Default: -1)
- **value3** (*int*) – Third criteria value. (Default: -1)

Location:**ToString**()
Convert location to string

Location:**Trap**(*type*, *size*, *tag*, *faction*, *on_disarm*, *on_trigger*)
Create square trap at location.

Parameters

- **type** (*int*) – TRAP_BASE_TYPE_*
- **size** (*float*) – (Default 2.0)
- **tag** (*string*) – Trap tag. (Default: "")
- **faction** (*int*) – Trap faction. (Default: STANDARD_FACTION_HOSTILE)
- **on_disarm** (*string*) – OnDisarm script. (Default: "")
- **on_trigger** (*string*) – OnTriggered script. (Default: "")

Location:**SetTileMainLightColor**(*color1*, *color2*)
Sets the main light colors for a tile.

Parameters

- **color1** (*int*) – AREA_TILE_SOURCE_LIGHT_COLOR_*
- **color2** (*int*) – AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**SetTileSourceLightColor**(*color1*, *color2*)
Sets the source light color for a tile.

Parameters

- **color1** (*int*) – AREA_TILE_SOURCE_LIGHT_COLOR_*
- **color2** (*int*) – AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**GetTileMainLight1Color**()
Determines the color of the first main light of a tile.

Return type AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**GetTileMainLight2Color**()
Determines the color of the second main light of a tile.

Return type AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**GetTileSourceLight1Color**()
Determines the color of the first source light of a tile.

Return type AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**GetTileSourceLight2Color**()
Determines the color of the second source light of a tile.

Return type AREA_TILE_SOURCE_LIGHT_COLOR_*

Location:**GetArea**()
Get area from location.

Location:**GetDistanceBetween**(*to*)
Gets distance between two locations.

Parameters

- **to** (*Location*) – The location to get the distance from.

Location:**GetFacing**()
Gets orientation of a location

Location:**GetPosition**()
Gets position vector of a location

4.11.10 class Module

class Module

Areas()

GetName()

GetStartingLocation()

GetGameDifficulty()

GetMaxHenchmen()

SetMaxHenchmen(*max*)

SetModuleXPScale(*scale*)

4.11.11 class Object

class Object

ActionCloseDoor(*door*)
An action that causes an object to close a door.

Parameters

- **door** (*Door*) – Door to close

ActionGiveItem(*item*, *target*)
Gives a specified item to a target creature.

Parameters

- **item** (*Item*) – Item to give.
- **target** (*Object*) – Receiver

ActionLockObject(*target*)
An action that will cause a creature to lock a door or other unlocked object.

Parameters

- **target** (*Door* or *Placeable*) – Door or placeable object that will be the target of the lock attempt.

ActionOpenDoor(*door*)
An action that will cause a creature to open a door.

Parameters

- **door** (*Door*) – Door to close

ActionPauseConversation()
Pause the current conversation.

ActionResumeConversation()
Resume a conversation after it has been paused.

ActionSpeakString(*message*[, *volume*])
Causes an object to speak.

Parameters

- **message** (*string*) – String to be spoken.
- **volume** (*int*) – VOLUME_* constant. Default: VOLUME_TALK

ActionSpeakStringByStrRef(*strref*, *volume*)
Causes the creature to speak a translated string.

Parameters

- **strref** (*int*) – Reference of the string in the talk table.
- **volume** (*int*) – VOLUME_* constant. Default: VOLUME_TALK

ActionStartConversation(*target*[, *dialog*[, *private*[, *hello*]]])
Action to start a conversation with a PC

Parameters

- **target** (*Object*) – An object to converse with.
- **dialog** (*string*) – The resource reference (filename) of a conversation. Default: “”
- **private** (*bool*) – Specify whether the conversation is audible to everyone or only to the PC. Default: `false`
- **hello** (*bool*) – Determines if initial greeting is played. Default: `true`

ActionTakeItem(*item*, *target*)
Takes an item from an object.

Parameters

- **item** (*Item*) – The item to take.
- **target** (*Object*) – The object from which to take the item.

ActionUnlockObject(*target*)
Causes a creature to unlock a door or other locked object.

Parameters

- **target** (*Door* or *Placeable*) – Door or placeable object that will be the target of the unlock attempt.

ActionWait(*time*)
Adds a wait action to an objects queue.

Parameters

- **time** (*float*) – Time in seconds to wait.

ApplyEffect(*dur_type*, *effect*, *duration*)
Applies an effect to an object.

Parameters

- **dur_type** (*int*) – DURATION_TYPE_* constant.
- **effect** (*Effect*) – Effect to apply.

- **duration** (*float*) – Time in seconds for effect to last. Default: 0.0

ApplyVisual(*vfx*, *duration*)
Applies visual effect to object.

Parameters

- **vfx** (*int*) – VFX_* constant.
- **duration** (*float*) – Duration in seconds. If not passed effect will be of duration type DURATION_TYPE_INSTANT

AssignCommand(*f*)
Assigns a command to an object.

Note: No longer really necessary as all actions are explicitly assigned.

Parameters

- **f** (*function*) – A closure.

BeginConversation(*target*, *conversation*)
Begin conversation.

Parameters

- **target** – Object to converse with
- **conversation** (*string*) – Dialog resref. Default: ""

ChangeFaction(*faction*)
Changes objects faction.

Parameters

- **faction** (*Object*) – Object of desired faction.

CheckType(...)
Checks object type.

Parameters

- **...** – Any number of OBJECT_TYPE_* constants

ClearAllActions(*[clear_combat]*)
Removes all actions from an action queue.

Parameters

- **clear_combat** (*bool*) – combat along with all other actions. Default: `false`.

Copy(*location*, *owner*, *tag*)
Copies an object

Parameters

- **location** – Location to copy the object to.
- **owner** – Owner of the object
- **tag** (*string*) – Tag of new object. Default: ""

CountItem(*id[, resref]*)
Get number of items that an object carries.

Parameters

- **id** (*string*) – Tag or Resref.
- **resref** (*bool*) – If `true` count by resref. Default: `false`

DecrementLocalInt (*name*, *val*)
Decrements local integer variable.

Parameters

- **name** (*string*) – Variable name.
- **val** (*int*) – Amount to decrement. Default: 1

Return type New value.

DelayCommand (*delay*, *action*)
Delays a command.

Parameters

- **delay** (*float*) – Time in seconds.
- **action** (*function*) – A closure.

DeleteLocalBool (*name*)
Boolean wrapper around DeleteLocalInt Int/Bool values are stored under the same name

Parameters

- **name** (*string*) – Variable name.

DeleteLocalFloat (*name*)
Delete a local float variable

Parameters

- **name** (*string*) – Variable to delete

DeleteLocalInt (*name*)
Delete a local int variable

Parameters

- **name** (*string*) – Variable name.

DeleteLocalLocation (*name*)
Delete a local location variable

Parameters

- **name** (*string*) – Variable to delete

DeleteLocalObject (*name*)
Delete a local object variable

Parameters

- **name** (*string*) – Variable to delete

DeleteLocalString (*name*)
Delete a local string variable

Parameters

- **name** (*string*) – Variable to delete

Destroy ([*delay*])
Destroy an object.

Parameters

- **delay** (*float*) – Delay (in seconds) before object is destroyed. Default: 0.0

DoCommand(*action*)
Inserts action into action queue.

Parameters

- **action** (*function*) – A closure.

DoDamage(*amount*)

Effects([*direct*])
An iterator that iterates over applied effects.

Parameters

- **direct** (*bool*) – If `true`, the actual effect will be returned. Modifying it will modify the applied effect. If `false`, a copy of the effect will be returned.

FortitudeSave(*dc*, *save_type*, *vs*)
Do fortitude save.

Parameters

- **dc** (*int*) – Difficult class
- **save_type** (*int*) –
- **vs** (*Object*) – Save versus object

GetAllVars([*match*[, *type*]])
Get all variables. TODO: Document variable type constant.

Parameters

- **match** (*string*) – A string pattern for testing variable names. See string.find
- **type** (*int*) – Get variables of a particular type.

GetArea()
Get area object is in.

GetCasterLevel()
Determines caster level.

GetColor(*channel*)

GetCommandable()
Get is object commandable.

GetCurrentAction()
Returns the currently executing Action.

GetCurrentHitPoints()
Gets an object's current hitpoints

GetDescription([*original*[, *identified*]])
Get object's description.

Parameters

- **original** (*bool*) – If true get original description. Default: `false`
- **identified** (*bool*) – If true get identified description. Default: `true`

GetDistanceToObject(*obj*)
Get distance between two objects.

Parameters

- **obj** (*Object*) – Object to determine distance to.

GetEffectAtIndex(*idx*)
Get an effect.

Note: This returns effects directly. Modifying them will modify the applied effect.

Parameters

- **idx** (*int*) – Index is zero based.

Return type *Effect* or `nil`

GetEffectCount()
Get the number effects applied to an object;

GetFacing()
Get direction object is facing.

GetFactionAverageGoodEvilAlignment()

GetFactionAverageLawChaosAlignment()

GetFactionAverageLevel()

GetFactionAverageReputation(*target*)

GetFactionAverageXP()

GetFactionBestAC([*visible*])
Get faction member with best AC.

Parameters

- **visible** (*bool*) – If `true` member must be visible. Default: `false`.

GetFactionGold()
Get factions gold.

GetFactionId()
Gets an objects faction ID.

GetFactionLeader()

GetFactionLeastDamagedMember(*visible*)

GetFactionMostDamagedMember(*visible*)

GetFactionMostFrequentClass()

GetFactionStrongestMember(*visible*)

GetFactionWeakestMember(*visible*)

GetFactionWorstAC(*visible*)

GetFortitudeSavingThrow()
Get fortitude saving throw.

GetGold()
Get object's gold.

GetHardness()
Determine object's 'hardness'.

GetHasEffectById(*id*)
Get if an object has an effect by ID.

Parameters

- **id** (*int*) – Effect ID.

GetHasInventory()
Determines if object has an inventory.

GetHasSpellEffect(*spell*)
Get if object has an effect applied by a spell.

Parameters

- **spell** (*int*) – SPELL_* constant.

GetIsDead()
Determine if dead.

GetIsImmune(*immunity*)
Determines if object is immune to an effect.

Parameters

- **immunity** (*int*) – IMMUNITY_TYPE_* constant.

Return type Always `false`.

GetIsInvulnerable()
Determine if object is invulnerable.

GetIsListening()
Get if object is listening.

GetIsOpen()
Get is object open.

GetIsTimerActive(*name*)
Determine if named timer is still active.

Parameters

- **name** (*string*) – Timer name.

Return type `bool`

GetIsTrapped()
Check whether an object is trapped.

Return type `bool`

GetIsValid()
Determines if an object is valid.

GetItemPossessedBy(*tag*[, *is_resref*])
Determine if object has an item.

Parameters

- **tag** (*string*) – Object tag to search for
- **is_resref** (*bool*) – If true search by resref rather than tag. Default: `false`

GetKeyRequired()
Determine if a key is required.

GetKeyRequiredFeedback()
Feedback for when missing key.

GetKiller()
Gets the object's killer.

Return type *Object*

GetLastAttacker()
Determine who last attacked a creature, door or placeable object.

GetLastDamager()
Get the object which last damaged a creature or placeable object.

GetLastHostileActor()
Gets the last living, non plot creature that performed a hostile act against the object.

GetLastOpenedBy()
Gets the last object to open an object.

GetLocalBool(*name*)
A wrapper around GetLocalInt returning `false` if the result is 0, `true` otherwise.

Parameters

- **name** (*string*) – Variable name

GetLocalFloat(*name*)
Get local float variable

Parameters

- **name** (*string*) – Variable name

GetLocalInt(*name*)
Get a local int variable

Parameters

- **name** (*string*) – Variable name

GetLocalLocation(*name*)
Get local location variable

Parameters

- **name** (*string*) – Variable name

GetLocalObject(*name*)
Get local object variable

Parameters

- **name** (*string*) – Variable name

GetLocalString(*name*)
Get local string

Parameters

- **name** (*string*) – Variable name.

GetLocalVarByIndex(*index*)
Get local variable by index. TODO: Return type???

Parameters

- **index** – Positive integer.

GetLocalVarCount()
Get local variable count.

GetLocation()
Get object's location.

Return type *Location*

GetLockDC()
Get lock's DC.

GetLockKeyTag()
Get lock's key tag.

Return type string

GetLockable()
Get if object is lockable.

GetLocked()
Get if object is locked.

GetMaxHitPoints()
Get object's max hitpoints.

GetName(*original*)
Get object's name.

Parameters

- **original** (*bool*) – If true returns object's original name, if false returns overridden name. Default: false.

GetNearestCreature(*type1*, *value1*, *nth*, ...)
Gets nearest creature by criteria types and values.

GetNearestObject(*obj_type*, *nth*)
Get nearest object.

Parameters

- **obj_type** (*int*) – OBJECT_TYPE_* constant. Default: OBJECT_TYPE_ALL
- **nth** (*int*) – Which object to return. Default: 1

GetNearestObjectByTag(*tag*, *nth*)
Get nearest object by tag.

Parameters

- **tag** (*string*) – Tag of object
- **nth** (*int*) – Which object to return. Default: 1

GetNearestTrap(*is_detected*)
Get nearest trap.

Parameters

- **is_detected** (*bool*) – If true return only detected traps. Default: false.

GetPlotFlag()
Get plot flag.

GetPortraitId()
Get portrait ID.

GetPortraitResRef()
Get portrait resref.

Return type `string`

GetPosition()
Get object's position.

Return type *Vector*

GetReflexSavingThrow()
Get reflex saving throw.

GetResRef()
Returns the Resref of an object.

Return type `string`

GetSpellCastAtCaster()
Gets the caster of the last spell.

GetSpellCastAtHarmful()
Determine if the last spell cast at object is harmful.

GetSpellCastAtId()
Determine spell id of the last spell cast at object.

GetSpellCastClass()
Determine class of the last spell cast at object.

GetSpellCastItem()
Get item of that last spell cast.

GetSpellId()
Get spell id of that last spell cast.

GetSpellResistance()
Get spell resistance.

GetSpellSaveDC(*spell*)
Determine spell save DC.

Parameters

- **spell** (*int*) – SPELL_* constant.

GetSpellTargetLocation()
Get spell target location.

Return type *Location*

GetSpellTargetObject()
Get last spell target.

Return type *Object*

GetTag()
Determine the tag associated with an object.

Return type `string`

GetTransitionTarget()
Gets transition target.

GetTrapBaseType()

GetTrapCreator()
Get traps creator

GetTrapDetectable()
Get if trap is detectable.

GetTrapDetectDC()
Get the DC required to detect trap.

GetTrapDetectedBy(*creature*)
Get if trap was detected by creature

GetTrapDisarmable()
Get if trap is disarmable

GetTrapDisarmDC()
Get DC required to disarm trap

GetTrapFlagged()
Get if trap is flagged

GetTrapKeyTag()
Get trap's key tag

GetTrapOneShot()
Get if trap is oneshot

GetType()
Get an object's type.

Return type OBJECT_TYPE_* or OBJECT_TYPE_NONE on error.

GetUnlockDC()
Get lock unlock DC.

GetWillSavingThrow()
Get will saving throw.

GiveItem(*resreff[, stack_size[, new_tag[, only_once]]]*)
Create a specific item in an objects inventory.

Parameters

- **resref** (*string*) – The blueprint ResRef string of the item to be created or tag.
- **stack_size** (*int*) – The number of items to be created. Default: 1
- **new_tag** (*string*) – If this string is empty (""), it be set to the default tag from the template. Default: ""
- **only_once** (*bool*) – If `true`, function will not give item if object already possess one. Default `false`

HasItem(*tag*)
Determines if an object has an item by tag.

Parameters

- **tag** (*string*) – Tag to search for.

IncrementLocalInt(*name*, *val*)
Increment local integer variable

Parameters

- **name** (*string*) – Variable name
- **val** (*int*) – Amount to increment. Default: 1

Return type New local variable value.

Items()
Iterator over items in an object's inventory.

LineOfSight(*target*)
Get is target in line of sight

Parameters

- **target** (*Object*) – Target to check.

ModifyCurrentHitPoints(*amount*)
Modifies an object's current hitpoints.

Parameters

- **amount** (*int*) – Amount to modify.

OpenInventory(*target*)
Open Inventory of specified target

Parameters

- **target** – Creature to view the inventory of.

PlaySound(*sound*)
Causes object to play a sound

Parameters

- **sound** (*string*) – Sound to play

PlaySoundByStrRef(*strref*[, *as_action*])
Causes object to play a sound.

Parameters

- **strref** (*int*) – Sound to play
- **as_action** (*bool*) – Determines if this is an action that can be stacked on the action queue. Default: `true`

ReflexSave(*dc*, *save_type*, *vs*)
Do reflex save.

Parameters

- **dc** (*int*) – Difficult class
- **save_type** (*int*) –
- **vs** (*Object*) – Save versus object

RemoveEffect(*effect*)
Removes an effect from object

Parameters

- **effect** (*Effect*) – Effect to remove.

RemoveEffectByID(*id*)
Removes an effect from object by ID

Parameters

- **id** (*int*) – Effect id to remove.

RemoveEffectsByType (*type*)
Remove effect by type.

Parameters

- **type** (*int*) – EFFECT_TYPE_*

ResistSpell (*vs*)
Attempt to resist a spell.

Parameters

- **vs** (*Object*) – Attacking caster

SetColor (*channel*, *value*)

SetCommandable (*commandable*)
Set is object commandable.

Parameters

- **commandable** (*bool*) – New value. Default: `false`.

SetCurrentHitPoints (*hp*)
Sets an object's current hitpoints.

Parameters

- **hp** (*int*) – A number between 1 and 10000

SetDescription (*description*, *identified*)
Set object's description.

Parameters

- **description** (*string*) – New description.
- **identified** (*bool*) – If `true` sets identified description. Default: `true`

SetFacing (*direction*)
Set direction object is facing in.

SetFacingPoint (*target*)
Set the point the object is facing.

Parameters

- **target** (*Vector*) – Vector position.

SetFactionId (*faction*)
Sets an objects faction ID.

Parameters

- **faction** (*int*) – New faction ID.

SetFortitudeSavingThrow (*val*)
Set fortitude saving throw.

Parameters

- **val** (*int*) – New value

SetHardness (*hardness*)
Sets an object's hardness.

Parameters

- **hardness** (*int*) – New hardness value.

SetIsDestroyable (*[destroyable[, raiseable[, selectable]]]*)

Parameters

- **destroyable** (*bool*) – Default: `false`
- **raiseable** (*bool*) – Default: `false`
- **selectable** (*bool*) – Default: `true`

SetKeyRequired (*key_required*)
Set lock requires key

Parameters

- **key_required** (*bool*) – If `true` key is required, if `false` not.

SetKeyRequiredFeedback (*message*)
Set feedback message

Parameters

- **message** (*string*) – Message sent when creature does not have the key

SetKeyTag (*tag*)
Set lock's key tag

Parameters

- **tag** (*string*) – New key tag.

SetLastHostileActor (*actor*)
Sets the last hostile actor.

Parameters

- **actor** (*Object*) – New last hostile actor.

SetListenPattern (*pattern*, *number*)
Set listening patterns.

Parameters

- **pattern** (*string*) – Pattern to listen for.
- **number** (*int*) – Number. Default: 0

SetListening (*val*)
Set object to listen or not.

Parameters

- **val** (*bool*) – New value.

SetLocalBool (*name*, *val*)
A wrapper around SetLocalInt.

Parameters

- **name** (*string*) – Variable name
- **val** (*bool*) – Value

SetLocalFloat (*name*, *val*)
Set local float variable

Parameters

- **name** (*string*) – Variable name
- **val** (*float*) – Value

SetLocalInt (*name*, *val*)
Set local int variable

Parameters

- **name** (*string*) – Variable name
- **val** (*int*) – Value

SetLocalLocation (*name*, *val*)
Set local location variable

Parameters

- **name** (*string*) – Variable name
- **val** (*Location*) – Value

SetLocalObject (*name*, *val*)
Set local object variable

Parameters

- **name** (*string*) – Variable name
- **val** (*Object*) – Value

SetLocalString (*name*, *val*)
Set local string variable

Parameters

- **name** (*string*) – Variable name
- **val** (*string*) – Value

SetLockDC (*dc*)
Set lock's lock DC.

Parameters

- **dc** (*int*) – New DC.

SetLockLockable (*lockable*)
Set lock to be lockable.

Parameters

- **lockable** (*bool*) – if `true` lock can be locked, if `false` it can't.

SetLocked (*locked*)
Set object locked

Parameters

- **locked** (*bool*) – If `true` object will be locked, if `false` unlocked.

SetMaxHitPoints (*hp*)
Sets an object's max hitpoints.

Parameters

- **hp** (*int*) – New maximum hitpoints.

SetName ([*name*])
Set object's name.

Parameters

- **name** (*string*) – New name.

SetPlotFlag (*flag*)
Set object's plot flag.

Parameters

- **flag** (*bool*) – If `true` object is plot. Default: `false`.

SetPortraitId (*id*)
Set portrait ID.

Parameters

- **id** (*int*) – Portrait ID

SetPortraitResRef (*resref*)
Set Portrait resref.

Parameters

- **resref** (*string*) – Portrait resref

SetReflexSavingThrow (*val*)
Set reflex saving throw

Parameters

- **val** (*int*) – New value.

SetTag (*tag*)
Changes an objects tag.

Parameters

- **tag** (*string*) – New tag.

SetTimer (*var_name*, *duration*[, *on_expire*])
Sets a timer on an object

Parameters

- **var_name** (*string*) – Variable name to hold the timer
- **duration** (*float*) – Duration in seconds before timer expires.
- **on_expire** (*function*) – Function which takes to arguments then object the timer was set on and the variable.

SetTrapDetectedBy (*object*[, *is_detected*])
Set whether an object has detected the trap

Parameters

- **object** (*Creature*) – the detector
- **is_detected** (*bool*) – Default: `false`

SetTrapKeyTag(*tag*)
Set the trap's key tag

Parameters

- **tag** (*string*) – New key tag.

SetUnlockDC(*dc*)
Set lock's unlock DC.

Parameters

- **dc** (*int*) – New DC.

SetWillSavingThrow(*val*)
Set will saving throw.

Parameters

- **val** (*int*) – New value.

SpeakString(*text*, *volume*)
Forces an object to immediately speak.

Parameters

- **text** (*string*) – Text to be spoken.
- **volume** (*int*) – VOLUME_* constant. Default: VOLUME_TALK

SpeakStringByStrRef(*strref*, *volume*)
Causes an object to instantly speak a translated string.

Parameters

- **strref** (*int*) – TLK string reference to speak.
- **volume** (*int*) – VOLUME_* constant. Default: VOLUME_TALK

TakeItem(*id*[, *count*[, *resref*]])
Take an item from an object.

Note: This function handles stack size reduction. It also checks if the object posses enough of them before taking any.

Parameters

- **id** (*string*) – Tag or Resref.
- **count** (*int*) – How many of the items to take. Default: 1
- **resref** (*bool*) – If `true` take by resref. Default: `false`.

Return type Amount taken.

Trap(*type*, *faction*, *on_disarm*, *on_trigger*)
Traps an object.

Parameters

- **type** (*int*) – TRAP_BASE_TYPE_*
- **faction** (*int*) – STANDARD_FACTION_*
- **on_disarm** (*string*) – OnDisarmed script. Default: ""

- **on_trigger** (*string*) – OnTriggered script. Default: “”

WillSave(*dc*, *save_type*, *vs*)
Do will save :param int dc: Difficult class :param int save_type: :param vs: Save versus object. :type vs: *Object*

4.11.12 class Placeable

class Placeable

DoAction(*action*)

GetIsActionPossible(*action*)

GetIsStatic()

GetIllumination()

GetSittingCreature()

GetUseable()

SetAppearance(*value*)

SetIllumination(*illuminate*)

SetUseable(*useable*)

4.11.13 class Trigger

class Trigger

4.11.14 class Sound

class Sound

Play()
Play sound.

Stop()
Stop sound

SetVolume(*volume*)
Set volume.

Parameters

- **volume** (*int*) – New volume.

SetPosition(*position*)
Set sounds position.

Parameters

- **position** (*Vector*) – Position.

4.11.15 class Store

class Store

GetGold()
Get store's gold

GetIdentifyCost()
Get store's identify price

GetMaxBuyPrice()
Get store's max buy price

Open(*pc*, *up*, *down*)
Open store

Parameters

- **pc** (*Creature*) – PC to open the store for.
- **up** (*int*) – Bonus markup
- **down** (*int*) – Bonus markdown

SetGold(*gold*)
Set amount of gold a store has.

SetIdentifyCost(*val*)
Set the price to identify items.

SetMaxBuyPrice(*val*)
Set the max buy price.

4.11.16 class Trap

4.11.17 class Vector

class Vector

FromAngle(*angle*)
Converts angle to vector

FromString(*str*)
Converts a string to a Vector. Format: "<x>, <y>, <z>"

Normalize()
Normalizes vector

Magnitude()
Calculates vector's magnitude

MagnitudeSquared()
Calculates vector's magnitude squared

LineOfSight(*target*)
Checks if target is in line of sight.

Arguments

Parameters

- **target** (*Object*) – Any object

Subtract(*other*)
Subtract vectors.

Parameters

- **other** (*Vector*) – Vector to subtract.

ToAngle()
Converts vector to angle

ToString()
Converts Vector to string

4.11.18 class Waypoint

class Waypoint

SetMapPinEnabled([*enabled*])
Set's a map pin's status

Parameters

- **enabled** – Enable/Disable map pin. (Default: `false`)

4.12 Rules

The rules modules, imported globally as `Rules`, allows anyone who wishes to modify default behaviors to do so in a way that will be compatible with all other users. The only limitation is that the function signatures must remain the same.

4.12.1 Sections

Abilities

DebugAbilities(*cre*)
Returns a string with some abiltiy related information.

Parameters

- **cre** (*Creature*) – Creature instance

Return type string

GetAbilityName(*ability*)
Gets the name of an ability.

Parameters

- **ability** (*int*) – ABILITY_*

Return type string

GetAbilityEffectLimits([*cre*[, *ability*]])
Get the limits of ability effects. Both parameters are optional, the are there merely to facilitate customizing effect limits by ability or creature, supposing someone wanted to do that.

Parameters

- **cre** (*Creature*) – Creature instance
- **ability** (*int*) – ABILITY_*

Returns -12, 12

Return type int

GetAbilityEffectModifier(*cre*[, *ability*])

Get ability modification from effects. The return value is not clamped or modified by *GetAbilityEffectLimits()*.

Warning: This currently does not provide default behavior. Everything stacks: items, spells, etc.

Parameters

- **cre** (*Creature*) – Creature instance
- **ability** (*int*) – ABILITY_*

Return type If the `ability` parameter is not passed an array of all ability effect modifiers of length `ABILITY_NUM` is returned. Note this array is static and should not be modified or stored by callers. If the `ability` parameter is passed only that ability effect modifier is returned.

Armor Class

DebugArmorClass(*cre*)

Generates a string with armor class related information.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `string`

GetACVersus(*cre*, *vs*, *touch*, *is_ranged*, *attack*, *state*)

Parameters

- **cre** (*Creature*) – Creature instance.
- **vs** (*Object*) – Object instance.
- **touch** (*bool*) – Versus touch attack.
- **is_ranged** (*bool*) – Versus ranged attack.
- **attack** – Attack info.
- **state** (*int*) – Combat state.

Return type `int`

GetArmorCheckPenalty(*cre*)

Determines armor check penalty.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `int`

GetArmorClassModifierLimits(*cre*)

Limits of armor class modifiers.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type -20, 20

Attack Bonus

DebugAttackBonus(*cre*)
Generates a string with attack bonus related information.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `string`

GetAttackBonusVs(*cre*, *atype*[, *target*])

Parameters

- **cre** (*Creature*) – Creature instance.
- **atype** (*int*) – ATTACK_TYPE_*
- **target** (*Object*) – Object instance.

Return type Total attack bonus.

GetBaseAttackBonus(*cre*[, *pre_epic=false*])
Determines base attack bonus.

Parameters

- **cre** (*Creature*) – Creature instance.
- **pre_epic** (*boolean*) – If `true` only calculate pre-epic BAB.

GetEffectAttackModifier(*cre*[, *atype*[, *target*]])
Determines the attack bonus from effects.

Parameters

- **cre** (*Creature*) – Creature instance.
- **atype** (*int*) – ATTACK_TYPE_*
- **target** (*Object*) – Object instance.

Return type If `atype` is passed to the function the unclamped attack bonus is returned, if not an `int32_t` array of all ATTACK_TYPE_* bonuses is returned.

GetEffectAttackLimits(*cre*)
Determines the minimum and maximum attack bonus can be modified by effects.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type -20, 20

GetRangedAttackMod(*cre*, *target*, *distance*)

Parameters

- **cre** (*Creature*) – Creature instance.
- **target** (*Object*) – Object instance.

- **distance** (*float*) – Distance to `target`

Return type `int`

Classes

CanUseClassAbilities (*cre*, *class*)
Determine if creature can use class abilites.

Parameters

- **cre** (*Creature*) – Creature instance.
- **class** (*int*) – CLASS_TYPE_*

GetClassName (*class*)
Get class name.

Arguments

Parameters

- **class** (*int*) – CLASS_TYPE_*

Return type string

GetHitPointsGainedOnLevelUp (*class*, *pc*)
Get number of hitpoints class gains on level up.

Parameters

- **class** (*int*) – CLASS_TYPE_*
- **pc** (*Creature*) – Creature instance.

GetLevelBonusFeats (*cre*, *class*, *level*)
Get bonus feats for level.

Parameters

- **cre** (*Creature*) – Creature instance.
- **class** (*int*) – CLASS_TYPE_*
- **level** (*int*) – Class level.

GetSkillPointsGainedOnLevelUp (*class*, *pc*)
Get number of skillpoints class gains on level up.

Parameters

- **class** (*int*) – CLASS_TYPE_*
- **pc** (*Creature*) – Creature instance.

SetCanUseClassAbilitiesOverride (*class*, *func*)
Registers a class ability handler.

Parameters

- **class** (*int*) – CLASS_TYPE_*
- **func** (*function*) – A function that takes a creature and optionally a CLASS_TYPE_* argument and returns a boolean indicating whether the creature can use the abilities for the class and the creatures class level. You **must** return both or an assertion will fail.

Example

```
local function monk(cre, class)
   local level = cre:GetLevelByClass(class)
   if level == 0 then return false, 0 end

   if not cre:GetIsPolymorphed() then
      local chest = cre:GetItemInSlot(INVENTORY_SLOT_CHEST)
      if chest:GetIsValid() and chest:ComputeArmorClass() > 0 then
         return false, level
      end

      local shield = cre:GetItemInSlot(INVENTORY_SLOT_LEFTHAND)
      if shield:GetIsValid() and
         (shield:GetBaseType() == BASE_ITEM_SMALLSHIELD
          or shield:GetBaseType() == BASE_ITEM_LARGESHIELD
          or shield:GetBaseType() == BASE_ITEM_TOWERSHIELD)
      then
         return false, level
      end
   end

   return true, level
end

Rules.SetCanUseClassAbilitiesOverride(CLASS_TYPE_MONK, monk)
```

Combat

Interfaces

CombatEngine

Fields

DoPreAttack [function] Function to do pre-attack initialization, taking attacker and target object instances. Note that since DoMeleeAttack, and DoRangedAttack have no parameters, the very least you need to do is store those in local variables for later use.

DoMeleeAttack [function] Function to do a melee attack.

DoRangedAttack [function] Function to do a ranged attack.

UpdateCombatInformation [function] Update combat information function, taking a Creature object instance. This is optional can be used to do any other book keeping you might need.

Functions

GetCombatEngine()

Get current combat engine.

Return type *CombatEngine*

RegisterCombatEngine(*engine*)

Register a combat engine.

Parameters

- **engine** (*CombatEngine*) – Combat engine.

SetCombatEngineActive (*active*)
Set combat engine active. This is implicitly called by RegisterCombatEngine.

Parameters

- **active** (*boolean*) – Turn combat engine on or off.

Combat Modifiers

GetCombatModifier (*type*, *modifier*, *cre*)

Parameters

- **type** (*int*) – COMBAT_MOD_*
- **modifier** (*int*) – ATTACK_MODIFIER_*
- **cre** (*Creature*) – Creature.

RegisterComabtModifier (*type*, *func*)

Parameters

- **type** (*int*) – COMBAT_MOD_*
- **func** (*function*) – A function taking two parameters: an ATTACK_MODIFIER_* constant and a *Creature* instance.

Concealment

GetConcealment (*cre*, *vs*, *is_ranged*)
Determine concealment.

Parameters

- **cre** (*Creature*) – Creature instance.
- **vs** (*Creature*) – Creature instance.
- **is_ranged** (*boolean*) – Check versus ranged attack.

Constants

ConvertSaveToItempropConstant (*const*)

Parameters

- **const** (*int*) – SAVING_THROW_*

ConvertSaveVsToItempropConstant (*const*)

Parameters

- **const** (*int*) – SAVING_THROW_VS_*

ConvertImmunityToIPConstant (*const*)

Parameters

- **const** (*int*) – IMMUNITY_TYPE_*

GetConstantTable ()

Return type The global constant table.

RegisterConstants(*tda*, *column_label*[, *extract*[, *value_label*[, *value_type*]]])
Register constant loader.

Parameters

- **tda** (*string*) – 2da name (without .2da)
- **column_label** (*string*) – Label of the 2da column that contains constant names.
- **extract** (*string*) – A lua `string.match` pattern for extracting a constant name.
- **value_label** (*string*) – Label of the 2da column that contains the constants value. If not passed constant value will be the 2da row number.
- **value_type** (*string*) – Constant type. Only used when `value_label` is passed. Legal values: "int", "string", "float"

RegisterConstant(*name*, *value*)
Register constant in global constant table.

Parameters

- **name** (*string*) – Constant's name.
- **value** – Consants's value. Can be any Lua object.

Damage

ConvertDamageToItempropConstant(*const*)

ConvertDamageIndexToItempropConstant(*const*)

ConvertItempropConstantToDamageIndex(*const*)

GetDamageColor(*index*)

Parameters

- **index** (*int*) – DAMAGE_INDEX_*

GetDamageName(*index*)

Parameters

- **index** (*int*) – DAMAGE_INDEX_*

GetDamageVisual(*dmg*)

Parameters

- **dmg** (*int*) – DAMAGE_INDEX_*

UnpackItempropDamageRoll(*ip*)

UnpackItempropMonsterRoll(*ip*)

Damage Reduction

DebugDamageImmunity(*cre*)
Generates a debug string with damage immunity related values.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `string`

DebugDamageResistance(*cre*)
Generates a debug string with damage resistance related values.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `string`

DebugDamageReduction(*cre*)
Generates a debug string with damage reduction related values.

Parameters

- **cre** (*Creature*) – Creature instance.

Return type `string`

DoDamageImmunity(*obj*, *amt*, *dmgidx*)
Determines the amount damage is modified by damage immunity.

Parameters

- **obj** (*Object*) – Object instance.
- **amt** (*int*) – Damage amount.
- **dmgidx** (*int*) – DAMAGE_INDEX_*

Return type Modified damage amount and the amount damage was modified.

DoDamageReduction(*obj*, *amt*, *eff*, *power*)
Determines the amount damage is modified by damage reduction. If the parameter `eff` is a damage absorbtion limit, it will be removed from `obj`.

Parameters

- **obj** (*Object*) – Object instance.
- **amt** (*int*) – Damage amount.
- **eff** (*Effect*) – Effect to use to modify the damage amount. Generally this should be the value returned from *GetBestDamageReductionEffect()*.
- **power** (*int*) – Damage power.

Return type Modified damage amount, the amount damage was modified, and a `bool` indicating whether the effect was removed.

DoDamageResistance(*obj*, *amt*, *eff*, *dmgidx*)
Determines the amount damage is modified by damage resistance. If the parameter `eff` is a damage absorbtion limit, it will be removed from `obj`.

Parameters

- **obj** (*Object*) – Object instance.
- **amt** (*int*) – Damage amount.
- **eff** (*Effect*) – Effect to use to modify the damage amount. Generally this should be the value returned from *GetBestDamageResistEffect()*.
- **dmgidx** (*int*) – DAMAGE_INDEX_*

Return type Modified damage amount, the amount damage was modified, and a `bool` indicating whether the effect was removed.

GetBaseDamageImmunity(*cre*, *dmgidx*)
Get base damage immunity.

Parameters

- **cre** (*Creature*) – Creature instance.
- **dmgidx** (*int*) – DAMAGE_INDEX_*

GetBaseDamageReduction(*cre*)
Get base damage reduction.

Note: This function doesn't have a user supplied override. To modify its behavior simply replace the function.

Parameters

- **cre** (*Creature*) – Creature instance.

GetBaseDamageResistance(*cre*, *dmgidx*)
Get base damage resistance.

Parameters

- **cre** (*Creature*) – Creature instance.
- **dmgidx** (*int*) – DAMAGE_INDEX_*

GetBestDamageReductionEffect(*obj*, *power*[, *start*])
Determines the best damage reduction effect currently applied to `obj`. The effect with the highest reduction at any power level greater than `power` is selected. If multiple effects have the same reduction the effect with the highest soak absorbtion limit is selected.

Parameters

- **obj** (*Object*) – Object instance.
- **power** (*int*) – Damage power.
- **start** (*int*) – Hint for where to start looking in `obj`'s effect list. This is uesful only for creature objects.

Return type *Effect*

GetBestDamageResistEffect(*obj*, *dmgidx*[, *start*])
Determines the best damage resistance effect currently applied to `obj`. The effect with the highest resistance to `dmgidx` is selected. If multiple effects have the same resistance the effect with the highest damage absorbtion limit is selected.

Parameters

- **obj** (*Object*) – Object instance.
- **dmgidx** (*int*) – DAMAGE_INDEX_*
- **start** (*int*) – Hint for where to start looking in `obj`'s effect list. This is uesful only for creature objects.

Return type *Effect*

GetEffectDamageImmunity(*obj*[, *dmgidx*])
Get damage immunity from effects. The values returned by this function are not clamped by *GetEffectDamageImmunityLimits()*

Parameters

- **obj** (*Object*) – Object instance.
- **dmgidx** (*int*) – DAMAGE_INDEX_*

Return type If `dmgidx` is provided an `int` is returned, otherwise an array of all damage immunity effects is returned.

GetEffectDamageImmunityLimits (*obj*)

Parameters

- **obj** (*Object*) – Object instance.

Return type -100, 100

SetBaseDamageImmunityOverride (*func*, ...)
Sets a damage immunity override function.

Example

```
local function rdd(cre)
   local res = 0
   if cre:GetLevelByClass(CLASS_TYPE_DRAGON_DISCIPLE) >= 10 then
      res = 100
   end
   return res
end

Rules.SetBaseDamageImmunityOverride(rdd, DAMAGE_INDEX_FIRE)
```

Parameters

- **func** (*function*) – (*Creature*) -> `int`
- `...` – DAMAGE_INDEX_* constants.

SetBaseDamageResistanceOverride (*func*, ...)
Sets a damage resistance override function.

Parameters

- **func** (*function*) – (*Creature*) -> `int`
- `...` – DAMAGE_INDEX_* constants.

Feats

GetFeatIsFirstLevelOnly (*feat*)
Determine is first level feat only.

Parameters

- **feat** (*int*) – FEAT_*

GetFeatName (*feat*)
Get feat name.

Parameters

- **feat** (*int*) – FEAT_*

GetFeatSuccessors (*feat*)
Get array of feats successors.

Parameters

- **feat** (*int*) – FEAT_*

Return type Array of FEAT_* constants.

GetIsClassBonusFeat (*feat*, *class*)
Determine if feat is class bonus feat.

Parameters

- **feat** (*int*) – FEAT_*
- **class** (*int*) – CLASS_TYPE_*

GetIsClassGeneralFeat (*feat*, *class*)
Determine if feat is class general feat.

Parameters

- **feat** (*int*) – FEAT_*
- **class** (*int*) – CLASS_TYPE_*

GetIsClassGrantedFeat (*feat*, *class*)
Determine if feat is class granted feat.

Parameters

- **feat** (*int*) – FEAT_*
- **class** (*int*) – CLASS_TYPE_*

GetMaximumFeatUses (*feat*[, *cre*])
Determines a creatures maximum feat uses.

Parameters

- **feat** (*int*) – FEAT_*
- **cre** (*Creature*) – Creature instance.

GetMasterFeatName (*master*)
Get Master Feat Name

Parameters

- **master** (*int*) – Master feat.

SetMaximumFeatUsesOverride (*func*, ...)
Register a function to determine maximum feat uses.

Parameters

- **func** (*function*) – A function taking two arguments, a Creature instance and and a FEAT_* constant and returns an integer. **Note that returning 100 is equivalent to infinite uses.**
- **...** – FEAT_* constants.

Example

```
-- Let the Champion of Torm have a couple more uses of Divine Wrath
Rules.RegisterFeatUses(
 function(feat, cre)
    local uses = 1
    local level = cre:GetLevelByClass(CLASS_TYPE_DIVINE_CHAMPION)
    if level >= 30 then
       uses = 3
```

```
        elseif level >= 20 then
           uses = 2
        end
        return uses
      end,
      FEAT_DIVINE_WRATH)
```

SetUseFeatOverride (*func*, ...)
Registers a function to be called when a feat is used.

Note: The feat use handler will be called immediately, as such it has limited applicability to feats that require an action.

Parameters

- **func** (*function*) – A function taking four arguments, FEAT_* constant, the user, a target, and a position. To bypass the engines UseFeat function return `true`.
- `...` – FEAT_* constants.

Example

```
local function feat_handler(feat, user, target, position)
  if target:GetIsValid() and target:GetIsPC() then
    target:SendMessage("Hello there.  This is %s", user:GetName())
  end

  -- The game engine doesn't need to handle this.
  return true
end

Rules.SetUseFeatOverride(feat_handler, FEAT_HELLO_THERE)
```

Hitpoints

GetMaxHitPoints (*cre*)
Determine Maximum Hitpoints.

Parameters

- **cre** (*Creature*) – Creature.

Return type `int`

Immunities

Note: Immunities versus alignements, races, etc have not been implemented.

DebugEffectImmunities (*cre*)
Generate a debug string with effect immunity info.

Parameters

- **vs** (*Creature*) – Creature.

Return type `string`

GetEffectImmunity(*cre*, *imm*, *vs*)
Determines total effect immunity. This is the maximum of creatures innate immunity and their innate immunity plus immunity effect modifiers. The result is not clamped by *GetEffectImmunityLimits()*.

Note: This function is not limited to default NWN behavior. It was modified to facilitate a percentate immunity to an IMMUNITY_TYPE_*. However, this doesn't modify the default behavior of item properties or *effect.Immunity()* so it stills work as expected.

Parameters

- **cre** (*Creature*) – Creature.
- **imm** (*int*) – IMMUNITY_TYPE_* constant.
- **vs** (*Creature*) – Creature.

Return type int

GetEffectImmunityLimits(*cre*)

Parameters

- **cre** (*Creature*) – Creature.

Return type 0, 100

GetEffectImmunityModifier(*cre*, *imm*, *vs*)
Determines the amount the modifier from effects.

Parameters

- **cre** (*Creature*) – Creature.
- **imm** (*int*) – IMMUNITY_TYPE_* constant.
- **vs** (*Creature*) – Creature.

GetInnateImmunity(*cre*, *imm*)
Get innate immunity.

Parameters

- **cre** (*Creature*) – Creature.
- **imm** (*int*) – IMMUNITY_TYPE_* constant.

Return type int

SetInnateImmunityOverride(*func*, ...)

Parameters

- **func** – Function taking a creature parameter and returning a percent immunity.
- **...** – List of IMMUNITY_TYPE_* constants.

Levels

GetGainsStatOnLevelUp(*level*)
Determine if an ability score is gained on level up.

Parameters

- **level** – Class level.

Return type `boolean`

GainsFeatAtLevel (*level*)
Determine if a feat is gained on level up.

Parameters

- **level** – Class level.

Return type `boolean`

GetXPLevelRequirement (*level*)
Determine XP requirements for level.

Parameters

- **level** – Class level.

Return type `int`

Modes

CombatMode
Table defining a combat mode.

Fields

use [`function` or `true`] Determines if combat mode is usable. If this field is `true` the mode is always usable if the feat used to apply it is usable.

modifier [`function`] Determines what the attack modifier is for a particular ATTACK_MODIFIER_* type. The function must accept two parameters an ATTACK_MODIFIER_* and a *Creature* instance. Returning `nil` indicates the ATTACK_MODIFIER_* is not applicable to the given mode.

GetCanUseMode (*mode*, *cre*)

Parameters

- **mode** (*int*) – COMBAT_MODE_*
- **cre** (*Creature*) – Creature.

Return type `boolean`

GetModeModifier (*mode*, *modifier*, *cre*)

Parameters

- **mode** (*int*) – COMBAT_MODE_*
- **modifier** (*int*) – ATTACK_MODIFIER_*
- **cre** (*Creature*) – Creature.

Return type Dependent on modifier type.

RegisterMode (*mode*, ...)

Parameters

- **mode** (*CombatMode*) – Combat mode interface.
- **...** – COMBAT_MODE_* constant(s).

Races

GetRaceAbilityBonus (*race*, *ability*)
Determine race's ability bonus.

Parameters

- **race** (*int*) – RACIAL_TYPE_*
- **ability** (*int*) – ABILITY_*

Saves

GetSaveEffectLimits (*cre*, *save*, *save_vs*)
Get save effect limits.

Parameters

- **cre** (*Creature*) – Creature.
- **save** (*int*) – SAVING_THROW_* constant.
- **save_vs** (*int*) – SAVING_THROW_VS_* constant.

Return type -20, 20

GetSaveEffectModifer (*cre*, *save*, *save_vs*)
Get save effect bonus unclamped.

Parameters

- **cre** (*Creature*) – Creature.

Situations

GetSituationModifier (*situ*, *modifier*, *cre*)

Parameters

- **situ** (*int*) – SITUATION_*
- **modifier** (*int*) – ATTACK_MODIFIER_*
- **cre** (*Creature*) – Creature.

RegisterSituation (*situation*, *func*)

Parameters

- **situ** (*int*) – SITUATION_*
- **func** (*function*) – A function taking two parameters: an ATTACK_MODIFIER_* and a *Creature* instance. The return type is dependent on the attack modifier type.

Skills

CanUseSkill (*skill*, *cre*)
Determines if a creature can use a skill.

Parameters

- **skill** (*int*) – SKILL_*

- **cre** (*Creature*) – Creature.

Return type boolean

GetIsClassSkill(*skill*, *class*)
Determines if a skill is a class skill.

Parameters

- **skill** (*int*) – SKILL_*
- **class** (*int*) – CLASS_TYPE_*

Return type boolean

GetSkillAbility(*skill*)
Get skill's associated ability.

Parameters

- **skill** (*int*) – SKILL_*

Return type ABILITY_* or -1

GetSkillAllCanUse(*skill*)
Check if skill requires training.

Parameters

- **skill** (*int*) – SKILL_*

Return type boolean

GetSkillArmorCheckPenalty(*cre*, *skill*)
Determine penalty from armor/shield.

Parameters

- **cre** (*Creature*) – Creature.
- **skill** (*int*) – SKILL_*

Return type int

GetSkillEffectLimits([*cre*[, *skill*]])
Get the limits of skill effects. Both parameters are optional, the are there merely to facilitate customizing effect limits by skill or creature, supposing someone wanted to do that.

Parameters

- **cre** (*Creature*) – Creature.
- **skill** (*int*) – SKILL_*

Return type -50, 50

GetSkillEffectModifier(*cre*[, *skill*])
Get skill modification from effects. The return value is not clamped or modified by *GetSkillEffectLimits()*.

Parameters

- **cre** (*Creature*) – Creature.
- **skill** (*int*) – SKILL_*

Return type If the `skill` parameter is not passed an array of all skill effect modifiers of length `SKILL_NUM` is returned. Note this array is static and should not be modified or stored by callers. If the `skill` parameter is passed only that skill effect modifier is returned.

GetSkillFeatBonus(*cre*, *skill*)
Get Skill Bonuses from feats.

Parameters

- **cre** (*Creature*) – Creature.
- **skill** (*int*) – SKILL_*

Return type `int`

GetSkillHasArmorCheckPenalty(*skill*)
Check if skill has armor check penalty.

Parameters

- **skill** (*int*) – SKILL_*

Return type `boolean`

GetSkillIsUntrained(*skill*)
Check if skill requires training.

Parameters

- **skill** (*int*) – SKILL_*

Return type `boolean`

GetSkillName(*skill*)
Get Skill name.

Parameters

- **skill** (*int*) – SKILL_*

Return type `string`

Special Attacks

SpecialAttack
Table interface for special attacks. All fields are optional.

Fields:

ab [`function` or `int`] Determines attack bonus modifier. If this field is a integer that value is returned for every special attack. If it is a function it must satisfy the same function signature as *GetSpecialAttackModifier()*

damage [`function` or `DamageRoll`] Determines damage modifier. If the value is a `DamageRoll` that value is returned for every special attack. If it is a function it must satisfy the same function signature as *GetSpecialAttackDamage()*

impact [`function`] Determines if a special attack is successful and optionally any effect(s) to be applied. The function, if any, must satisfy the same function signature as *GetSpecialAttackImpact()*. Its `boolean` return value indicates whether a special attack was successful or not, `false` or `nil` indicates the target has resisted the attack.

If no function is set, the special attack is always successful.

use [`function`] Determines if a special attack is useable. The function will be called with the following parameters: special attack type, attacker, target and it must return `true` or `false`. Note: the function is responsible for providing any feedback to the player.

GetSpecialAttackDamage (*special_attack*, *info*, *attacker*, *target*)
Determine special attack damage. Should only every be called from a combat engine.

Parameters

- **special_attack** (*int*) – SPECIAL_ATTACK_*
- **info** – Attack ctype from combat engine.
- **attacker** (*Creature*) – Attacking creature.
- **target** (*Creature*) – Attacked creature.

Return type `DamageRoll`

GetSpecialAttackImpact (*special_attack*, *info*, *attacker*, *target*)
Determine special attack effect. Should only every be called from a combat engine.

Note: The boolean return value indicates whether the special attack was successful.

Parameters

- **special_attack** (*int*) – FEAT_* or SPECIAL_ATTACK_*
- **info** – Attack ctype from combat engine.
- **attacker** (*Creature*) – Attacking creature.
- **target** (*Creature*) – Attacked creature.

Return type `boolean` and optionally an *Effect* or an array of *Effect*.

GetSpecialAttackModifier (*special_attack*, *info*, *attacker*, *target*)
Determine special attack bonus modifier. Should only every be called from a combat engine.

Parameters

- **special_attack** (*int*) – SPECIAL_ATTACK_*
- **info** – Attack ctype from combat engine.
- **attacker** (*Creature*) – Attacking creature.
- **target** (*Creature*) – Attacked creature.

Return type `int`

RegisterSpecialAttack (*special_attack*, *...*)
Register special attack handlers.

The vararg parameter(s) can be any usable feat, it is not limited to hard-coded special attacks. When a special attack is registered, a use feat event handler is also registered; it will handle adding the special attack action, will override any other uses of the feat, and any feedback messages like *Special Attack Resisted* floating strings.

Parameters

- **special_attack** – See the *SpecialAttack* interface.
- `...` – FEAT_* or SPECIAL_ATTACK_* constants.

Example

```
    local Eff = require 'solstice.effect'
    local Attack = require 'solstice.attack'
    local GetAttackRoll = Attack.GetAttackRoll

    local function kd_use(id, attacker, target)
      if Rules.GetIsRangedWeapon(attacker:GetItemInSlot(INVENTORY_SLOT_RIGHTHAND)) then
        if attacker:GetIsPC() then
          -- Normally for these hardcoded feats a localized string would be sent,
          -- but this is just an example.
          attacker:SendMessage("You can not use Knockdown with ranged weapons.")
        end
        return false
      end
      return true
    end

    local function kd_impact(id, info, attacker, target)
      local size_bonus = id == SPECIAL_ATTACK_KNOCKDOWN_IMPROVED and 1 or 0
      if target:GetSize() > attacker:GetSize() + size_bonus then return false end

      if GetAttackRoll(info) > target:GetSkillRank(SKILL_DISCIPLINE) then
         local eff = Eff.Knockdown()
         eff:SetDurationType(DURATION_TYPE_TEMPORARY)
         eff:SetDuration(6)
         return true, eff
      end

      return false
    end

    Rules.RegisterSpecialAttack({ use = kd_use, impact = kd_impact, ab = -4},
                                SPECIAL_ATTACK_KNOCKDOWN_IMPROVED,
                                SPECIAL_ATTACK_KNOCKDOWN)
```

Weapons

AttackTypeToEquipType (*atype*)

Parameters

- **atype** (*int*) – ATTACK_TYPE_*

Return type EQUIP_TYPE_*

BaseitemToWeapon (*base*)

EquipTypeToAttackType (*atype*)

Parameters

- **atype** (*int*) – EQUIP_TYPE_*

Return type ATTACK_TYPE_*

GetCreatureDamageBonus (*cre*, *item*)

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

GetDualWieldPenalty(*cre*)
Get dual wielding penalty.

Parameters

- **cre** (*Creature*) – Creature.

GetIsMonkWeapon(*item*, *cre*)

Parm item Item instance.

Parameters

- **cre** (*Creature*) – Creature.

GetIsRangedWeapon(*item*)

Parm item Item instance.

GetIsWeaponFinessable(*item*, *cre*)

Parm item Item instance.

Parameters

- **cre** (*Creature*) – Creature.

GetIsWeaponLight(*item*, *cre*)

Parameters

- **cre** (*Creature*) – Creature.

GetIsWeaponSimple(*item*, *cre*)

Parameters

- **cre** (*Creature*) – Creature.

GetOffhandAttacks(*cre*)
Determine number of offhand attacks.

Parameters

- **cre** (*Creature*) – Creature.

GetOnhandAttacks(*cre*)
Determine number of onhand attacks.

Parameters

- **cre** (*Creature*) – Creature.

GetUnarmedDamageBonus(*cre*)
Determine unarmed damage bonus.

Parameters

- **cre** (*Creature*) – Creature.

GetWeaponAttackAbility(*cre*, *item*)

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

Return type ABILITY_*

GetWeaponAttackBonus(*cre*, *weap*)

Parameters

- **cre** (*Creature*) – Creature.

GetWeaponBaseDamageType(*item*)
Determine weapons base damage type.

Note: This does not support multiple weapon damage types and most likely never will.

Parm item Item instance.

GetWeaponBaseDamage(*item*, *cre*)
Determine weapons base damage roll.

Parm item Item instance.

Parameters

- **cre** (*Creature*) – Creature.

GetWeaponDamageAbility(*cre*, *item*)

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

Return type ABILITY_*

GetWeaponIteration(*cre*, *item*)

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

GetWeaponFeat(*masterfeat*, *basetype*)

GetWeaponPower(*cre*, *item*)
Determine weapons damage power.

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

GetWeaponType(*item*)

Parm item Item instance.

GetWeaponCritRange(*cre*, *item*)
Determine weapons critical hit range.

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

GetWeaponCritMultiplier(*cre*, *item*)
Determine weapons critical hit multiplier.

Parameters

- **cre** (*Creature*) – Creature.

Parm item Item instance.

InventorySlotToAttackType (*atype*)

Parameters

- **atype** (*int*) – Inventory slot constant.

Return type ATTACK_TYPE_*

InitializeNumberOfAttacks (*cre*)
Initialize combat rounds attack counts.

Parameters

- **cre** (*Creature*) – Creature.

SetWeaponAttackAbilityOverride (*ability*, *func*)

Parameters

- **ability** (*int*) – ABILITY_*

SetWeaponDamageAbilityOverride (*ability*, *func*)

Parameters

- **ability** (*int*) – ABILITY_*

SetWeaponFeat (*masterfeat*, *basetype*, *feat*)

4.12.2 Examples

Replacing a function.

```
-- In some file that you load from your preload file.

-- Suppose you think it more appropriate to change Epic Toughness feats to grant 40hp per feat.
-- We'll leave aside whether this is a good or bad idea.

local function GetMaxHitPoints(cre)
  local res = 0
  local not_pc = cre:GetIsAI()
  local level = cre:GetHitDice()
  if cre:GetHasFeat(FEAT_TOUGHNESS) then
    res = res + level
  end

  res = res + math.max(0, cre:GetAbilityModifier(ABILITY_CONSTITUTION) * level)

  local pm = cre:GetLevelByClass(CLASS_TYPE_PALE_MASTER)
  local pmhp = 0
  if pm >= 5 then
    if pm >= 25 then
      pmhp = 18 + (math.floor(pm / 5) * 20)
    elseif pm >= 15 then
      pmhp = 18 + (math.floor(pm / 5) * 10)
    elseif pm >= 10 then
```

```
      pmhp = 18 + math.floor((pm - 10) / 5)
    else
      pmhp = pm * 3
    end
  end
  res = res + pmhp

  local epictough = cre:GetHighestFeatInRange(FEAT_EPIC_TOUGHNESS_1, FEAT_EPIC_TOUGHNESS_10)
  if epictough ~= -1 then
    -- Changed 20 -> 40.
    local et = 40 * (epictough - FEAT_EPIC_TOUGHNESS_1 + 1)
    res = res + et
  end

  -- Some of the underlying engine object is exposed here.
  -- It isn't necessary to understand this unless you have a reason
  -- to change it... and you probably shouldn't.
  if not_pc then
    res = res + cre.obj.obj.obj_hp_max
  else
    local base = 0
    for i = 1, cre:GetHitDice() do
      base = base + cre:GetMaxHitPointsByLevel(i)
    end
    res = res + base
    cre.obj.obj.obj_hp_max = res
  end

  if res <= 0 then res = 1 end

  return res
end

-- Replace the global function and now anywhere in your code
-- or in the api that calls this function will call your
-- modified version.
Rules.GetMaxHitPoints = GetMaxHitPoints
```

Using overrides.

A number of functions provide explicit means of overriding behaviors. In those cases it's not necessary to replace any functions. To see examples of those checkout the sections of the rules module.

4.13 System

4.13.1 Database

ConnectDatabase(*driver_name*, *dbname*, *dbuser*, *dbpassword*, *dbhost*, *dbport*)

Connect to a database.

Parameters

- **driver_name** (*string*) – 'MySQL', 'PostgreSQL', or 'SQLite3' depending on which database you use.

- **dbname** (*string*) – Name.
- **dbuser** (*string*) – User.
- **dbpassword** (*string*) – Password.
- **dbhost** (*string*) – Host.
- **dbport** (*int*) – Port.

GetDatabase()
Get the active database connection.

4.13.2 Logging

SetLogger(*logger*)
Sets the default logging function.

Parameters

- **logger** – See LuaLogging.

GetLogger()
Get current logger.

FileLogger(*filename*, *date_time*)
Create a file logger.

Parameters

- **filename** (*string*) – File name.
- **date_time** (*string*) – Date/time format see os.date

4.13.3 Lua

CollectGarbage()
Run Lua garbage collector.

Return type The amount in KB freed.

LogGlobalTable()
Log the global Lua table

4.14 Why Lua?

There are also other projects similar to this: nwnx_jvm, nwnx_ruby, nwnx_lua. So you might wonder why make another?

There are a few reasons I felt Lua and particularly LuaJIT was better suited for this particular project:

- Lua is very easy to embed.
- Lua is a much more powerful language and also has a great deal of gamedev industry support. It's used by World of Warcraft, CryEngine, Dark Souls, even in the Baldur's Gate series, and many others. So someone looking to develop skills for a career in gamedev will be able to learn a directly applicable programming language, while availing themselves of the accessibility of the NWN Toolset.

- Due to the dynamic nature of Lua, everything can be replaced. If you don't like the way some library function works: replace it. There is the caveat that some functions forward to NWScript built-in functions. In those cases you have a couple options: Redo them in Lua or redo them in nwnx_solstice and expose a C(++) function that does what you want. Depending on what you want those will have varying levels of difficulty.
- Building and distributing libraries is much simpler, since you have a mechanism for creating modules. Namespacing is also much more pleasant since you don't have to rely on resref prefixes.
- LuaJIT is one of the fastest implementations of any dynamic language.
- LuaJIT's Foreign Function Interface (FFI) library makes it very easy to expose C data structures, so the huge amount of reverse engineering work done for NWNX can be easily reused. It's also quite easy to import other C libraries.
- Access to a number of Lua libraries. If you have reason to use a database, a socket, or file related operations, you can do those directly and not rely on NWNX extensions and shunting all data through LocalStrings.
- Access to more data structures: arrays, tables, and the ability to implement your own.
- Constants don't have to be predefined, they can be loaded at runtime. You can do this however you choose: Read from 2DAs, parse NWScript, or another document.
- Advanced NWNX users can directly hook nwserver functions with Lua functions.

In the fairness of disclosure there are some things about Lua that aren't so pleasant:

- Lua is extremely different from NWScript. It's a dynamically typed language, so many errors that would have been caught at compile time will only be found at runtime. If you make a typo or a syntax error restarting the server might be necessary.
- As above: testing would require a Linux server, either online or in a VM.
- Debugging is pretty much limited to printing/logging.
- Features like nwnx_resman's live reloading are not fully supported.
- Lua indexes arrays starting at 1. Some functions that communicate with C(++) have indexes starting at 0. This is sometimes confusing.
- You need to be aware of the global exports from *solstice.util* because you can clobber these with your own scripts, which will introduce varying levels of breakage.

4.15 Known Issues

- Most effects versus races, alignments, etc are not currently supported.
- The more you change, the more out of alignment the character sheet will be. There are currently no hooks attempting to remedy that. It is possible in the future, however.

4.16 To Do

- Document data structures shared by nwnx_solstice and Solstice.
- Expand 'Getting Started' to cover building LuaJIT and NWNX.
- Add more examples.
- Documentation is in the process of being converted form docstrings to ReStructured Text and Sphinx

CHAPTER 5

Indices and tables

- genindex
- search

a

c

d

e

g

i

n

r

h

n

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y