SRB2 Battle

Version 9.0 Scripting Manual

Custom Characters

General Overview

Please refer to the **example lua script** for a general overview on how to create scripts for characters with battle specific functions. It's recommended that you view this script in NotePad++ or SLADE for the easiest readability.

Updating characters from pre-7.x

- Action / special move scripts no longer follow the format of `function(mo,doaction,firering,tossflag)` and have been condensed to `function(mo,doaction)`. Attempting to call arguments #3 and #4 will return **nil**.
- `player.action2text` and `player.action2rings` no longer refer to a secondary action, but are still available as freely modifiable HUD items.
- `CBW_Battle.CheckRings` is deprecated and no longer used.
- `CBW_Battle.ActionChar` is deprecated. Special moves are now assigned through `CBW_Battle.SkinVars[skin].special`
- `CBW_Battle.AddCharPriority` is deprecated. Priority functions are now assigned through `CBW_Battle.SkinVars[skin].function_priority_ext`

SkinVars

SkinVars is a namespace which holds important information for character properties as they relate to BattleMod.

Example:

```
CBW_Battle.SkinVars["fang"] = {
	flags = SKINVARS_GUARD|SKINVARS_NOSPINSHIELD|SKINVARS_GUNSLINGER,
	weight = 105,
	shields = 1,
	special = Act.CombatRoll,
	guard_frame = 1,
	func_priority_ext = Act.CombatRoll_Priority,
	func_precollide = B.Fang_PreCollide,
	func_collide = B.Fang_Collide,
	func_postcollide = B.Fang_PostCollide,
	sprites = {
		S_FANGCHAR_AIRFIRE1,
		S_FANGCHAR_AIRFIRE2
	}
}
```

SkinVars Index

Variable	Default	Description
`flags`	`SKINVARS_GUARD`	Set of qualifiers that adjust various character properties. Can be stacked with the binary "**|**" operator. **1** `SKINVARS_GUARD` - Allows for guarding. **2** `SKINVARS_NOSPINSHIELD` - Disables shield abilities with the spin key. **4** `SKINVARS_GUNSLINGER` - Uses BattleMod's custom gunslinger functionality. **8** `SKINVARS_ROSY` - Uses BattleMod's custom piko hammer functionality.

`weight`	`100`	% multiplier used for distributing knockback during melee collisions. Must be an integer higher than 0.
`shields`	`1`	Number of reserve shields the player can carry in Battle gametypes.
`special`	`nil`	Function used for performing that character's Battle special moves. Accepts functions and **nil**.
`guard_frame`	`2`	Super transform frame to use while the player is in "guarding" state.
`func_guard_trigger`	`CBW_Battle.GuardFunc.Parry`	Action triggered when a player is attacked while guarding. Must be a function.
`func_priority`	`CBW_Battle.Priority_Fullcommon`	Core function used for handling melee attack state. Generally best left alone unless the character has especially modified core mechanics. Accepts functions and **nil**.
`func_priority_ext`	`nil`	Extra function used for handling melee attack state; can override attack states set by `func_priority`. Accepts functions and **nil**.
`func_precollide`	`nil`	Extra function used for handling custom collision behavior, which is called right before `func_collide`. Accepts functions and **nil**. You can use this function to determine what should be done to both players before doing anything, as a method to prevent node order from affecting the result of an interaction.
`func_collide`	`nil`	Extra function used for handling custom collision behavior, such as ending knuckles' glide on hit or causing tumble during fang's combat roll. Accepts functions and **nil**.

`func_postcollide`	`nil`	Extra function used for handling custom collision behavior, which is called right after collision bounce physics are applied. Accepts functions and **nil**.
`func_exhaust`	`nil`	Extra function used for handling custom exhaustion mechanics. Return `true` to prevent exhaustion from being replenished on floor contact. Accepts functions and **nil**.
`sprites`	`{}`	Table. Used for defining extra sprites for use with certain special moves (e.g. `sprites[1] = S_TAILS_SWIPE`)

Action script variable index

Variable	Description
`mo`	Argument #1 in the special move function. Refers to the player object performing the special move. It is recommended that the `mo.player` is stored as **local** `player` for ease of access.
`doaction`	Argument #2 in the special move function. **0** if the player has not pressed the special move key; **1** if the player is able and has pressed the key; **2** if the player is currently holding the key down; **-1** has attempted and is unable to perform a command (e.g. actions are not allowed, action is on cooldown, etc.).
`player.actiontext`	HUD text which appears next to a red ring icon.
`player.actionrings`	Amount of rings to consume when a special move is performed. Will display in the HUD if above 0.
`player.action2text`	If a string is defined, this will display below the first string, next to a flag icon.
`player.action2rings`	"Ring amount" displayed next to `player.action2text`, if

	applicable. Unused by default, but still available as a HUD item.
`player.actionstate`	Integer. Used to determine the stage of a character's attack. **0** is default, and will be set to such if the player takes damage or is parried by another player.
`player.actiontime`	Integer; does nothing by default. Recommended usage is to increment up/down per frame depending on the player's attack type and action state.
`player.actionsuper`	If **true**, `player.actionstate` will not be interrupted by parries and damage received. Resets to **false** each frame, just before action script is run.
`player.actioncooldown`	Number of tics that the player's special moves are on cooldown for.

Action Functions Index

Function	Description
`CBW_Battle.CanDoAction (player)`	Returns **true** if the player is allowed to perform actions. (Does not factor in cooldown.)
`CBW_Battle.PayRings (player,spendrings,sound)`	Spends the player's rings. `spendrings` defaults to `player.actionrings`. If `sound` is **false**, function will not play the "spend rings" sound effect. If the player does not have enough rings, the function adds the number of unpaid rings to `player.actiondebt`.
`CBW_Battle.ApplyCooldown (player,cooldown,applydebt)`	Applies cooldown to the player's action. `cooldown` defaults to **0**. `applydebt` defaults to **true**. If **applydebt** is true *and* the player has an outstanding ring debt, the cooldown is increased to either `TICRATE*2` or `cooldown+cooldown*debt/10`, whichever is larger.

Additional Variables and Functions

Note: This is not an extensive list of all the variables and functions that can be referenced, but it includes the ones that may be most useful to modders and scripters.

Variable/Function	Description

`player.iseggrobo`	**Boolean**. Determines if the player is in the eggrobo state for Tag Eggrobo.
`player.eggrobo_transforming`	**Boolean**. Indicates that the player is currently transforming into an eggrobo.
`player.isjettysyn`	**Boolean**. Determines if the player is in the Revenge Jettysyn state for Survival gametypes.
`player.gotcrystal`	**Boolean**. Determines if the player is holding the diamond in Diamond In the Rough.
`player.gotflagdebuff`	**Boolean**. Determines if the player is currently slowed down by holding a flag/diamond.
`player.exhaustmeter`	Depletion meter that limits how long characters like Knuckles, Tails, Metal Sonic can remain in the air. A value of `FRACUNIT` indicates full meter, and is reset to such whenever the player touches the ground.
`player.lockaim`	**Boolean**. Forces the player's aim and angle in place for the next tic. Resets to **false** afterward.
`player.lockmove`	**Boolean.** Forces forward and side movement inputs to 0 for the next tic. Resets to **false** afterward.
`player.canguard`	**Boolean.** Allows guarding. This variable is reset just before the player's special move function is performed.
`player.guard`	**Integer**. Indicates guard state. State of 0 is neutral; 1 indicates the player is currently guarding; 2 indicates the player is in their guard triggered animation; -1 indicates the player is in their "guard failed" endlag state.
`player.guardtics`	**Integer.** Continuously subtracts downward to reach the next guard state.
`player.airdodge`	**Integer.** Counts upwards until the air dodge is over. 0 indicates that an air dodge is ready to use. -1 indicates that the air dodge has completed, but the player has still not touched the ground yet.
`CBW_Battle.DoPlayerFlinch (player, time, angle, thrust, force)`	Sets the player into skid-recoil/no-control state. `time`: Time in tics to disallow player movement. `angle`: Angle of skid thrust. `thrust`: Recoil thrust amount while player is on the ground. `force`: If **true**, an InstaThrust is applied to the player using the angle and thrust values provided.

`CBW_Battle.DoPlayerTumble (player, time, angle, thrust, force)`	Sets the player into tumble state, leaving them wide open to attack. `time`: Time in tics to disallow player movement. `angle`: Angle of skid thrust. `thrust`: Recoil thrust amount while player is on the ground. `force`: If **true**, an InstaThrust is applied to the player using the angle and thrust values provided.
`CBW_Battle.MyTeam (player1,player2)`	Returns **true** if the players or objects are allies. Returns **false** if they're opponents. Returns **nil** if one of the arguments is undefined.
`CBW_Battle.PreRoundWait()`	Returns **true** if the game is currently in the pre-round waiting / skin select period.
`CBW_Battle.DrawCVSprite (player,n)`	Sets the player mobj state to the value `sprites[n]` as defined by the player character's SkinVars. Returns **true** if sprite was successfully set; if argument 2 is **nil**, the function will return **false**.

Creating and Specifying Custom Gametypes

Battle gametypes use the `CBW_Battle.Gametypes` namespace to specify certain properties per mode. **Nil** values are generally treated as **false**.

Variable	Description
`CBW_Battle.Gametypes.SuddenDeath[GT_*]`	If **true** and `GTR_OVERTIME` is enabled, players will increase in scale during overtime.
`CBW_Battle.Gametypes.Battle[GT_*]`	Gametype uses the “Battle” format. Player collision is always enabled in this format against enemy players.
`CBW_Battle.Gametypes.CP[GT_*]`	Gametype uses the “Control Point” format.
`CBW_Battle.Gametypes.Arena[GT_*]`	Gametype uses the “Arena” format.
`CBW_Battle.Gametypes.Diamond[GT_*]`	Gametype uses the “Diamond in the Rough” format.

The following functions can also be used to return **true** or **false** whether the current gametype has the respective properties:

```
CBW_Battle.BattleGametype()
CBW_Battle.CPGametype()
CBW_Battle.ArenaGametype()
CBW_Battle.DiamondGametype()
```

Custom Bashable Objects

Bashable objects are specified by running the following function in the object's "MobjSpawn" hook:
`CBW_Battle.CreateBashable(mo,weight,friction,smooth,sentient)`

The following function must then be placed in the object's "MobjThinker" hook:
`B.BashableThinker(mo)`

`CreateBashable` Arguments

Arguments	Default	Description
`mo`	<n/a>	The object to specify properties for.
`weight`	80	% modifier for determining knockback distribution in collisions. Must be an integer greater than 0.
`friction`	2	Factor to slow object when sliding from knockback. A higher factor will slow down the object further; a factor of 0 will cause the object to be in perpetual motion without slowing down.
`smooth`	false	Accepts **boolean**. If true, object will have ball-like physics, rolling down surfaces.
`sentient`	false	Accepts **boolean**. If true, object will respond to damage received and players will earn points for attacking it. Designed for use with the Tails Doll training object.

Projectile Features

`MF_MISSILE` objects have a handful of special properties which should be set in a `MobjSpawn` hook. (Note that parrying a projectile is not the same as blocking one. Blocking requires a high enough defense value)

Variable	Description
`mo.hit_sound`	Sound effect to play when damaging an enemy
`mo.blockable`	Determines how much defense is required to block the projectile. If this value is not set, the projectile is unblockable like usual.

`mo.block_stun`	Amount of stun to apply to the defender when the projectile is blocked.
`mo.block_sound`	Sound effect to play when the projectile is blocked.
`mo.block_hthrust`	Amount of horizontal thrust to apply to the defender when the projectile is blocked.
`mo.block_vthrust`	Amount of vertical thrust to apply to the defender when the projectile is blocked.
`mo.donotwaterslow`	Prevents the projectile from being slowed underwater.