Trigger Reference

Documentation for version 1.0

Updated 29 June 2010

Contents

- Life
- LifeMax
- Ln (math)
- Log (math)
- Lose
- MatchNo
- MatchOver
- MoveContact
- MoveGuarded
- MoveHit
- MoveType(*,***)
- MoveReversed
- Name(*,***)
- NumEnemy
- NumExplod
- NumHelper
- NumPartner
- NumProj
- NumProjID
- NumTarget
- P1Name(*,***)
- P2BodyDist
- P2Dist
- P2Life
- P2MoveType
- P2Name(*,***)
- P2StateNo
- P2StateType
- P3Name(*,***)
- P4Name(*,***)
- PalNo
- ParentDist
- Pi (math)
- Pos
- Power
- PowerMax
- PlayerIDExist
- PrevStateNo
- ProjCancelTime
- ProjContact(*,***)
- ProjContactTime
- ProjGuarded(*,***)
- ProjGuardedTime
- ProjHit(*,***)
- ProjHitTime
- Random
- RootDist
- RoundNo
- RoundsExisted
- RoundState
- ScreenPos
- SelfAnimExist
- Sin (math)
- StateNo
- StateType
- StageVar(*,***)
- SysFVar
- SysVar
- Tan (math)
- TeamMode(*,***)
- TeamSide
- TicksPerSecond
- Time
- TimeMod(*,**,***)

About Triggers

This is an alphabetical index of function-type triggers. For details on use of function-type triggers in expressions, see the expression documentation. Unless otherwise specified, we will use P1 to represent the player who is evaluating the trigger, and P2 to represent his opponent (usually the closest opponent when in team mode).

Some triggers are nonstandard and cannot take expressions as their arguments. These are marked with a (*) in the index. Some triggers are deprecated; these are marked with a (**). Use of these triggers is not encouraged, as support for them may be removed in future versions of the engine.

Old-style triggers, marked (***), appear only in clauses of the form (trigger) (relational operator) (value). See section 8 of exp.doc for more details.

Triggers used for math are marked with (math).

For all triggers, bottom will be returned if the trigger is redirected to a nonexistent destination, or if the ID number for the redirection evaluates to bottom. This is not listed in the error conditions for each trigger.

Abs (math)

Computes the absolute value of its argument.

Format:
abs(exprn)

Arguments:
exprn
Expression to compute the absolute value of.

Return type:
Same as the type of exprn.

Error conditions:
Returns bottom if exprn evaluates to bottom.

Acos (math)

Computes the arccosine (in radians) of the specified argument.

Format:
acos(exprn)

Arguments:
exprn
Expression to compute the arccosine of (float).

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn is not in the domain of arccosine (which is [-1.0,1.0]).

Example:

```
value = acos(1)
  Sets value to the arccosine of 1, which is approximately 0.0
  (possibly with some rounding error.)
```

AILevel

Returns the difficulty level of the player's AI.

Format:
AILevel

Arguments:
none

Return type:
int

Error conditions:
none

Details:
If AI is enabled on the player, the value ranges from 1 (easiest) to 8 (most difficult). If AI is not enabled on the player, the return value is 0.

Example:

```
trigger1 = Random < AILevel * 10
  Triggers with 10% probability at AILevel 1, 20% at AILevel 2,
  etc.
```

Alive

Returns 1 if the player is still able to fight, 0 if the player has been KOed.

Format:
alive

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = Alive = 0
  Triggers if the player has been KOed.
```

Anim

Returns the current animation action number of the player.

Format:
Anim

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = Anim = 200
  Triggers if the player is currently in action 200.
```

AnimElem(*,***)

Gets the animation-time elapsed since the start of a specified element of the current animation action. Useful for synchronizing events to elements of an animation action. (reminder: first element of an action is element 1, not 0)

AnimElemTime has similar functionality to AnimElem, but can take expressions as its argument.

Format:

1. AnimElem = value1
2. AnimElem = value1, [oper] value2

Arguments:

[oper]

=, !=, <, >, <=, >=

value1 (int)

Element number to check.

value2 (int)

Value of animation-time to compare with.

Return type:

boolean int (1 or 0)

Error conditions:

Returns bottom if the specified element number is invalid for this action (e.g., it's too large or too small).

Details:

Trigger in Format 1 is true if the player's animation is at the start of the element number specified by value1. In other words, if value1 is equal to n, it is true on the first game-tick of the nth element of the animation.

Trigger in Format 2 compares the player's animation-time to t+value2, where t is time of the start of the element number specified by value1.

Notes:

AnimElem will not trigger on the first game-tick of the second or later loop of an animation with a finite looptime. For example, "AnimElem = 1" will trigger the first tick a player changes to an animation, but will not trigger on the tick that it loops. You may get it to trigger each time using "AnimElem = 1 || AnimTime = 0".

Examples:

```
trigger1 = AnimElem = 2
  True on the first game-tick that the player's animation
  is on element 2. Is equivalent to saying:
    trigger1 = AnimElem = 2, = 0

trigger1 = AnimElem = 2, = 4
  True 4 game-ticks after the start of the player's
  second animation element.

trigger1 = AnimElem = 2, >= 0
trigger1 = AnimElem = 3, < 0
  True for the whole of the second element of the player's
  animation, assuming there is a third element. If a
  third element does not exist, the second line should
  read,

    trigger1 = AnimTime <= 0
```

AnimElemNo

Returns the number of the animation element in the current action that would be displayed at the specified time. The argument to AnimElemNo represents the time to check, expressed in game ticks, relative to the present.

Format:
AnimElemNo(exprn)

Arguments:
exprn
Expression that evaluates to the time offset (int).

Return type:
int

Error conditions:
Returns bottom if you try to check a time that would fall before the start of the current action.

Notes:
If the action is currently within its looping portion, then it is assumed to have been looping forever. That is, no matter how far into the past you check, AnimElemNo will always return an element number that lies within the looping portion of the action.

Examples:

```
trigger1 = AnimElemNo(0) = 2
  True when the animation element to be displayed 0 ticks in the
  future (i.e., now) is element 2. This is equivalent to:
    trigger1 = AnimElem = 2, >= 0
    trigger1 = AnimElem = 3, < 0

trigger1 = AnimElemNo(2) = 4
  True when the animation element that will be displayed two ticks
  from now is element 4. This is equivalent to:
    trigger1 = AnimElem = 4, >= -2
    trigger1 = AnimElem = 5, < -2
```

AnimElemTime

Gets the animation-time elapsed since the start of a specified element of the current animation action. Useful for synchronizing events to elements of an animation action. (reminder: first element of an action is element 1, not 0)

Format:
AnimElemTime(exprn)

Arguments:
exprn
Expression that evaluates to the element number to check (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn evaluates to an element number that is not valid for the current action.

Notes:
AnimElemTime will not trigger on the first game-tick of the second or later loop of an animation with a finite looptime. For example, "AnimElemTime(1) = 0" will trigger the first tick a player changes to an animation, but will not trigger on the tick that it loops. You may get it to trigger each time using "AnimElemTime(1) = 0 || AnimTime = 0".

Examples:

```
trigger1 = AnimElemTime(2) = 0
  True on the first game-tick that the player's animation
  is on element 2. Is equivalent to saying:
    trigger1 = AnimElem = 2

trigger1 = AnimElemTime(2) = 4
  True 4 game-ticks after the start of the player's
  second animation element.

trigger1 = AnimElemTime(2) >= 0
trigger1 = AnimElemTime(3) < 0
```

```
  True for the whole of the second element of the player's
  animation, assuming there is a third element. If a
  third element does not exist, the second line will evaluate
  to bottom and hence trigger1 will never trigger. In this case,
  the second line should read,
    trigger1 = AnimTime <= 0
```

AnimExist

Returns 1 if the specified animation action exists for the player. The result of this trigger is undefined if the player has been placed in a custom state by a successful hit. In this situation, use SelfAnimExist.

Format:
AnimExist(exprn)

Arguments:
exprn
An expression evaluating to an animation number (int).

Return type:
boolean int (1 or 0)

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
trigger1 = !AnimExist(200)
  Triggers if the player is missing action 200.
```

AnimTime

Gives the difference between the looptime of the current animation action and the player's animation-time. Useful for knowing when the end of the animation has been reached. (Animation-time is the time in game-ticks that the player has spent within the current animation action.) The name may be confusing. Try to think of it as "time from the end of the animation". During the animation, AnimTime will always return a non-positive number.

Format:
AnimTime

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = AnimTime = 0
  Triggers when the animation-time is equal to the animation
  action's looptime, ie. the end of the action has been
  reached.
```

Asin (math)

Computes the arcsine (in radians) of the specified argument.

Format:
asin(exprn)

Arguments:
exprn

Expression to compute the arcsine of (float).

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn is not in the domain of arcsine (which is [-1.0,1.0]).

Example:

```
value = asin(1)
  Sets value to the arcsine of 1, which is approximately pi/2
  (possibly with some rounding error.)
```

Atan (math)

Computes the arctangent (in radians) of the specified argument.

Format:
atan(exprn)

Arguments:
exprn
Expression to compute the arctangent of (float).

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
value = atan(1)
  Sets value to the arccosine of 1, which is approximately pi/4
  (possibly with some rounding error.)
```

AuthorName(*,***)

Returns the player's author's name (specified in the .DEF file). This may also be useful for telling apart characters with the same name but different authors.

Format:
AuthorName [oper] "name"

Arguments:
[oper]
=, != (other operators not valid)

"name" (string)
Name to compare against. Must be in double quotes.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = Authorname = "Suika"
  Returns true if the character's author is named Suika.
```

BackEdgeBodyDist

BackEdgeBodyDist gives the distance from the back of the player, as determined by the end of his width bar, to the back edge of the screen.

Format:
BackEdgeBodyDist

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = BackEdgeBodyDist < 30
  Triggers if the back of the player's width bar is within 30 pixels
  of the edge of the screen in back of him.
```

BackEdgeDist

BackEdgeDist gives the distance between the x-axis of the player and the edge of the screen behind of the player.

Format:
BackEdgeDist

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = BackEdgeDist < 30
  Triggers if the x-axis of the player is within 30 pixels
  of the edge of the screen in back of him.
```

CanRecover

If the player is currently in a falling state, returns 1 if he is currently able to recover, and 0 if he is not currently able to recover. If the player is not currently falling, the output of this trigger is undefined.

Format:
CanRecover

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Ceil (math)

Implements the "ceiling" function. Returns the least integer which is greater than or equal to the specified argument.

Format:
ceil(exprn)

Arguments:
exprn
Expression to compute the ceil of.

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
1. value = ceil(5.5)
  Sets value to 6.
2. value = ceil(-2)
  Sets value to -2.
```

Command (*,***)

Triggers if the user has input the specified command.

Format:
Command [oper] "command_name"

Arguments:
[oper]
=, != (other operators not valid)

"command_name" (string)
Name of the command. Commands are defined in the player's CMD file, and are case-sensitive. If the CMD has multiple commands with the same name, then any one of those commands will work. Command names must appear within double quotes.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = Command = "fireball motion"
  True if the user inputs the command corresponding to the
  command name "fireball motion".
```

Cond (math)

This trigger takes three arguments. The first argument is a condition argument. If the condition is true (i.e., nonzero), Cond evaluates and returns the second argument. If the condition is false, Cond evaluates and returns the third argument. If the condition is bottom, then Cond returns bottom without evaluating the second or third arguments.

In all cases, any unused argument(s) are not evaluated. Therefore, Cond can be used instead of IfElse to avoid any side effects that would be caused by evaluating the unused argument (e.g., variable assignment, or performing a computation that would cause bottom to be generated).

Format:
Cond(exp_cond,exp_true,exp_false)

Arguments:
exp_cond
Expression to test.

exp_true

Expression specifying value to return if exp_cond is nonzero.

exp_false
Expression specifying value to return if exp_cond is zero.

Return type:
Type of exp_true or exp_false, whichever is returned.

Error conditions:
Returns bottom if exp_cond evaluates to bottom, or if exp_true or exp_false (whichever is actually used) evaluates to bottom.

Example:

```
value = Cond(var(3),1,2)
  Sets value to 1 if var(3) is not zero, and sets value to 2 if
  var(3) is 0.
```

Const (*)

Returns the value of one of the player's constants.

Format:
Const(param_name)

Arguments:
param_name
The name of the constant to check. Valid values are:

Return type:
Depends on specified hit parameter. See Details.

Error conditions: none

Details:
The following values of param_name return values specified in the [Data] group in the player's constants.

- data.life: Returns value of the "life" parameter. (int)
- data.power: Returns value of the "power" parameter. (int)
- data.attack: Returns value of the "attack" parameter. (int)
- data.defence: Returns value of the "defence" parameter. (int)
- data.fall.defence_mul: Returns value of the defence multiplier, calculated as 100/(f+100), where f is the "fall.defence_up" parameter. (float)
- data.liedown.time: Returns value of the "liedown.time" parameter. (int)
- data.airjuggle: Returns value of the "airjuggle" parameter. (int)
- data.sparkno: Returns value of the "sparkno" parameter. (int)
- data.guard.sparkno: Returns value of the "guard.sparkno" parameter. (int)
- data.KO.echo: Returns value of the "ko.echo" parameter. (int)
- data.IntPersistIndex: Returns value of the "IntPersistIndex" parameter. (int)
- data.FloatPersistIndex: Returns value of the "FloatPersistIndex" parameter. (int)

The following values of param_name return values specified in the [Size] group in the player's constants.

- size.xscale: Returns value of the "xscale" parameter. (float)
- size.yscale: Returns value of the "yscale" parameter. (float)
- size.ground.back: Returns value of the "ground.back" parameter. (int)
- size.ground.front: Returns value of the "ground.front" parameter. (int)
- size.air.back: Returns value of the "air.back" parameter. (int)
- size.air.front: Returns value of the "air.front" parameter. (int)
- size.height: Returns value of the "height" parameter. (int)
- size.attack.dist: Returns value of the "attack.dist" parameter. (int)
- size.proj.attack.dist: Returns value of the "proj.attack.dist" parameter. (int)
- size.proj.doscale: Returns value of the "proj.doscale" parameter. (int)
- size.head.pos.x: Returns x-component of the "head.pos" parameter. (int)
- size.head.pos.y: Returns y-component of the "head.pos" parameter. (int)
- size.mid.pos.x: Returns x-component of the "mid.pos" parameter. (int)

- size.mid.pos.y: Returns y-component of the "mid.pos" parameter. (int)
- size.shadowoffset: Returns value of the "shadowoffset" parameter. (int)
- size.draw.offset.x: Returns x-component of the "draw.offset" parameter. (int)
- size.draw.offset.y: Returns y-component of the "draw.offset" parameter. (int)

The following values of param_name return values specified in the [Velocity] group in the player's constants.

- velocity.walk.fwd.x: Returns value of the "walk.fwd" parameter. (float)
- velocity.walk.back.x: Returns value of the "walk.back" parameter. (float)
- velocity.run.fwd.x: Returns x-component of the "run.fwd" parameter. (float)
- velocity.run.fwd.y: Returns y-component of the "run.fwd" parameter. (float)
- velocity.run.back.x: Returns x-component of the "run.back" parameter. (float)
- velocity.run.back.y: Returns y-component of the "run.back" parameter. (float)
- velocity.jump.y: Returns y-component of the "jump.neu" parameter. Note: this is NOT "velocity.jump.neu.y". Only the "neu" parameters take a y-component value. (float)
- velocity.jump.neu.x: Returns x-component of the "jump.neu" parameter. (float)
- velocity.jump.back.x: Returns value of the "jump.back" paramamter. (float)
- velocity.jump.fwd.x: Returns value of the "jump.fwd" parameter. (float)
- velocity.runjump.back.x: Returns value of the "runjump.back" paramamter. (float)
- velocity.runjump.fwd.x: Returns value of the "runjump.fwd" parameter. (float)
- velocity.airjump.y: Returns y-component of the "airjump.neu" parameter. Note: this is NOT "velocity.airjump.neu.y". (float)
- velocity.airjump.neu.x: Returns x-component of the "airjump.neu" parameter. (float)
- velocity.airjump.back.x: Returns value of the "airjump.back" paramamter. (float)
- velocity.airjump.fwd.x: Returns value of the "airjump.fwd" parameter. (float)
- velocity.air.gethit.groundrecover.x: Returns x-component of the "air.gethit.groundrecover" parameter. (float)
- velocity.air.gethit.groundrecover.y: Returns y-component of the "air.gethit.groundrecover" parameter. (float)
- velocity.air.gethit.airrecover.mul.x: Returns x-component of the "air.gethit.airrecover.mul" parameter. (float)
- velocity.air.gethit.airrecover.mul.y: Returns x-component of the "air.gethit.airrecover.mul" parameter. (float)
- velocity.air.gethit.airrecover.add.x: Returns x-component of the "air.gethit.airrecover.add" parameter. (float)
- velocity.air.gethit.airrecover.add.y: Returns x-component of the "air.gethit.airrecover.add" parameter. (float)
- velocity.air.gethit.airrecover.back: Returns value of the "air.gethit.airrecover.back" parameter. (float)
- velocity.air.gethit.airrecover.fwd: Returns value of the "air.gethit.airrecover.fwd" parameter. (float)
- velocity.air.gethit.airrecover.up: Returns value of the "air.gethit.airrecover.up" parameter. (float)
- velocity.air.gethit.airrecover.down: Returns value of the "air.gethit.airrecover.down" parameter. (float)

The following values of param_name return values specified in the [Movement] group in the player's constants.

- movement.airjump.num: Returns value of the "airjump.num" parameter. (int)
- movement.airjump.height: Returns value of the "airjump.height" parameter. (int)
- movement.yaccel: Returns value of the "yaccel" parameter. (float)
- movement.stand.friction: Returns value of the "stand.friction" parameter. (float)
- movement.crouch.friction: Returns value of the "crouch.friction" parameter. (float)
- movement.stand.friction.threshold: Returns value of the "stand.friction.threshold" parameter. (float)
- movement.crouch.friction.threshold: Returns value of the "crouch.friction.threshold" parameter. (float)
- movement.jump.changeanim.threshold: Returns value of the "jump.changeanim.threshold" parameter. (float)
- movement.air.gethit.groundlevel: Returns value of the "air.gethit.groundlevel" parameter. (float)
- movement.air.gethit.groundrecover.ground.threshold: Returns value of the "air.gethit.groundrecover.ground.threshold" parameter. (float)
- movement.air.gethit.groundrecover.groundlevel: Returns value of the "air.gethit.groundrecover.groundlevel" parameter. (float)
- movement.air.gethit.airrecover.threshold: Returns value of the "air.gethit.airrecover.threshold" parameter. (float)
- movement.air.gethit.airrecover.yaccel: Returns value of the "air.gethit.airrecover.yaccel" parameter. (float)
- movement.air.gethit.trip.groundlevel: Returns value of the "air.gethit.trip.groundlevel" parameter. (float)

- movement.down.bounce.offset.x: Returns x-component of the "down.bounce.offset.x" parameter. (float)
- movement.down.bounce.offset.y: Returns y-component of the "down.bounce.offset.y" parameter. (float)
- movement.down.bounce.yaccel: Returns value of the "down.bounce.yaccel" parameter. (float)
- movement.down.bounce.groundlevel: Returns value of the "down.bounce.groundlevel" parameter. (float)
- movement.down.friction.threshold: Returns value of the "down.friction.threshold" parameter. (float)

Example:

```
trigger1 = Const(velocity.walk.fwd.x) > 4
  Triggers if the forward walking velocity is greater than 4.
```

Const240p

Converts a value from the 240p coordinate space to the player's coordinate space. The conversion ratio between coordinate spaces is the ratio of their widths.

Format:
Const240p(exprn)

Arguments:
exprn
Expression containing the value to convert. (float)

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Notes:
Non-zero position and velocity offset values in common1.cns should use one of the Const triggers to maintain consistency with characters from a different coordinate space.

Example:

```
value = Const240p(3)
  Sets value 3 if the player has a coordinate space of 320x240 (240p).
  Sets value 6 if the player has a coordinate space of 640x480 (480p).
  Sets value 12 if the player has a coordinate space of 1280x720 (720p).
```

Const480p

Converts a value from the 480p coordinate space to the player's coordinate space. The conversion ratio between coordinate spaces is the ratio of their widths.

Format:
Const480p(exprn)

Arguments:
exprn
Expression containing the value to convert. (float)

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Notes:
Non-zero position and velocity offset values in common1.cns should use one of the Const triggers to maintain consistency with characters from a different coordinate space.

Example:

```
value = Const480p(6)
  Sets value 3 if the player has a coordinate space of 320x240 (240p).
  Sets value 6 if the player has a coordinate space of 640x480 (480p).
  Sets value 12 if the player has a coordinate space of 1280x720 (720p).
```

Const720p

Converts a value from the 720p coordinate space to the player's coordinate space. The conversion ratio between coordinate spaces is the ratio of their widths.

Format:
Const720p(exprn)

Arguments:
exprn
Expression containing the value to convert. (float)

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Notes:
Non-zero position and velocity offset values in common1.cns should use one of the Const triggers to maintain consistency with characters from a different coordinate space.

Example:

```
value = Const720p(12)
  Sets value 3 if the player has a coordinate space of 320x240 (240p).
  Sets value 6 if the player has a coordinate space of 640x480 (480p).
  Sets value 12 if the player has a coordinate space of 1280x720 (720p).
```

Cos (math)

Computes the cosine of the specified argument (in radians.)

Format:
cos(exprn)

Arguments:
exprn
Expression to compute the cosine of. (float)

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
value = cos(0)
  Sets value to the cosine of 0, which is approximately 1.0
  (possibly with some rounding error.)
```

Ctrl

Returns the control flag of p1.

Format:
Ctrl

Arguments:

none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = Ctrl
  Triggers if the player has control.
```

DrawGame

Returns 1 if the player (or the player's team, in team mode) has ended the round in a draw, 0 otherwise.

Format:
Draw

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Examples:

```
trigger1 = DrawGame
  Triggers if the player (or team) ended round in a draw.
```

E (math)

Returns the value of e (2.718281828...)

Format:
e

Arguments:
none

Return type:
float

Error conditions:
none

Exp (math)

Computes the exponential of the argument (e raised to the power of the argument.) This produces slightly more accurate results than the equivalent expression e**(argument).

Format:
exp(exprn)

Arguments:
exprn
Expression to compute the exponential of (float).

Return type:
float

Error conditions:

Returns bottom if exprn evaluates to bottom.

Example:

```
value = exp(4-var(0))
  Sets value to e raised to the quantity 4-var(0).
```

Facing

Returns 1 if the player is facing to the right, and -1 if the player is facing to the left.

Format:
Facing

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
Trigger = Facing = -1
  Triggers if the player is facing toward the left of the screen.
```

Floor (math)

Implements the floor function. Returns the greatest integer less than or equal to its argument.

Format:
floor(exprn)

Arguments:
exprn
Expression to compute the floor of.

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom.

Examples:

```
1. value=floor(5.5)
  Sets value to 5.
2. value=floor(-2)
  Sets value to -2.
```

FrontEdgeBodyDist

FrontEdgeBodyDist gives the distance between the front of the player (as determined by the front edge of his width bar) and the edge of the screen.

Format:
FrontEdgeBodyDist

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = FrontEdgeBodyDist < 30
  Triggers if the front of the player is within 30 pixels
  of the edge of the screen in front of him.
```

FrontEdgeDist

FrontEdgeDist gives the distance between the x-axis of the player and the edge of the screen in front of the player.

Format:
FrontEdgeDist

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = FrontEdgeDist < 30
  Triggers if the x-axis of the player is within 30 pixels
  of the edge of the screen in front of him.
```

FVar

This trigger takes a mandatory variable number as an argument. It returns the value of the player's specified float variable.

Format:
FVar(exprn)

Arguments:
exprn
An expression evaluating to a variable number. Valid numbers at this time are 0-39.

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn evaluates to an invalid variable index.

Example:

```
trigger1 = FVar(5) = -1.23
  Triggers if the value of float variable 5 is -1.23.
```

GameHeight

Returns the height of the game space in the player's local coordinate space. The game space is defined as a spatial mapping to the visible area of the screen in which players interact. Intuitively, it can be thought of as encompassing the graphical area of the game. The dimensions of the game space is specified by the GameWidth and GameHeight parameters in mugen.cfg.

Format:

GameHeight

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = ScreenPos Y < GameHeight / 2
  Triggers if the player is above the center of the screen.
```

GameTime

Returns the total number of ticks that have elapsed in the game so far.

Format:
GameTime

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = (GameTime % 27) = 0
  Triggers every 27th game tick.
```

GameWidth

Returns the width of the game space in the player's local coordinate space. The game space is defined as a spatial mapping to the visible area of the screen in which players interact. Intuitively, it can be thought of as encompassing the graphical area of the game. The dimensions of the game space is specified by the GameWidth and GameHeight parameters in mugen.cfg.

Format:
GameHeight

Arguments:
none

Return type:
float

Error conditions:
none

Example:

```
trigger1 = ScreenPos X >= GameWidth / 2
  Triggers if the player is to the right of the center of the screen.
```

GetHitVar(*)

When the player is in a gethit state, returns the value of the specified hit parameter.

Format:

GetHitVar(param_name)

Arguments:

param_name

The name of the hit parameter to check. Valid values are:

xveladd, yveladd, type, animtype, airtype, groundtype, damage, hitcount, fallcount, hitshaketime, hittime, slidetime, ctrltime, recovertime, xoff, yoff, zoff, xvel, yvel, yaccel, hitid, chainid, guarded, fall, fall.damage, fall.xvel, fall.yvel, fall.recover, fall.time, fall.recovertime.

Return type:

Depends on specified hit parameter. See Details.

Error conditions:

none

Details:

- xveladd: Returns the additional x-velocity that is added to the player's own when he is KOed. (float)
- yveladd: Returns the additional y-velocity that is added to the player's own when he is KOed. (float)
- type: Returns the type of the hit: 0 for none, 1 for high, 2 for low, 3 for trip (ground only).
- animtype: Returns the animation type of the hit. (0 for light, 1 for medium, 2 for hard, 3 for back, 4 for up, 5 for diag-up)
- airtype: Returns the type specified in the HitDef for an air hit.
- groundtype: Returns the type specified in the HitDef for a ground hit.
- damage: Returns the damage given by the hit. (int)
- hitcount: Returns the number of hits taken by the player in current combo. (int)
- fallcount: Returns the number of times player has hit the ground in the current combo. (int)
- hitshaketime: Returns time player is "frozen" during the hit. This number counts down by 1 for each game tick, and stops when it reaches zero. (int)
- hittime: Returns time before player regains control and returns to an idle state after being hit. This counts down by 1 per game tick, as long as hitshaketime (see above) is greater than 0. It stops counting down when the value reaches -1. (int) "GetHitVar(hittime) < 0" is equivalent to the HitOver trigger.
- slidetime: Returns time that player slides backwards (on the ground) after the hit. (int)
- ctrltime: Returns time before player regains control after guarding the hit. (int)
- recovertime: Returns time before player gets up from liedown state This number counts down to 0 for each game tick, and will count down faster if buttons are hit. (int)
- xoff: "Snap" x offset when hit (deprecated)
- yoff: "Snap" y offset when hit (deprecated)
- xvel: Fixed x-velocity imparted by hit. (float)
- yvel: Fixed y-velocity imparted by hit. (float)
- yaccel: y acceleration set by the hit. (float)
- chainid: Player-assigned chainID for last hit taken. (int)
- guarded: True if the last hit was guarded, false otherwise.
- isbound: True if the player is the subject of an attacker's TargetBind controller. Useful to prevent being stuck in thrown states. (int)
- fall: True if falling, false otherwise (int)
- fall.damage: Damage taken upon fall (int)
- fall.xvel: x velocity after bouncing off ground (float)
- fall.yvel: y velocity after bouncing off ground (float)
- fall.recover: True if player can recover, false otherwise.
- fall.recovertime: time before player can recover. (int)
- fall.kill: value of fall.kill parameter in attacker's hitdef. (int)
- fall.envshake.time: See below. (int)
- fall.envshake.freq: See below. (float)
- fall.envshake.ampl: See below. (int)
- fall.envshake.phase: Returns values set by the fall.envshake.* parameters in an attacker's hitdef. (float)

Example:

```
trigger1 = GetHitVar(yvel) < -5.5
  Triggers if the hit's specified y velocity is less than -5.5.
```

HitCount

Returns the number times the player's current attack move has hit one or more opponents. This value is valid only for a single state; after any state change, it resets to 0. To prevent it from resetting to 0, set hitcountpersist in the StateDef (see cns documentation for details). The HitCount and UniqHitCount triggers differ only when the player is hitting more than one opponent. In the case where the player is hitting two opponents with the same attack, HitCount will increase by 1 for every hit, while UniqHitCount increases by 2.

Format:
HitCount

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = HitCount > 8
  Triggers when more than 8 hits have been dealt to the
  opponent since the start of the player's attack move.
```

HitDefAttr(*,***)

Checks the attribute parameter of the player's currently-active HitDef. If the player is not currently attacking, then no parameters will match. Can be used for simple move-interrupts from weaker to stronger attacks in the CMD file.

Note: `HitDefAttr != value1, value2` is logically equivalent to `!(HitDefAttr = value1, value2)`.

Format:
HitDefAttr [oper] value1, value2

Arguments:
[oper]
=, !=

value1
A string that has at least one of the letters "S", "C" and "A" for standing, crouching and aerial attacks respectively. For example, "SA" is for standing and aerial attacks.

value2
A set of 2-character strings, separated by commas. Each 2-character string must be of the form described: The first character is either "N" for "normal", "S" for "special", or "H" for "hyper". The second character must be either "A" for "attack" (a normal hit attack) or "T" for "throw". For example, "NA, ST" is for normal attacks and special throws.

Assuming the attribute of the player's HitDef is in the form:

`arg1, arg2`

then the trigger condition is determined to be true only if arg1 is a subset of value1, AND arg2 is a subset of value2.

See the "attr" parameter of the HitDef controller in Appendix B for details.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = HitDefAttr = A, HA
  Triggers when the player is in an attack state, where
```

```
  the current HitDef has the following attributes:
      1. is an aerial attack
  and 2. is a hyper (super) attack

trigger1 = HitDefAttr = SC, NA, SA
  Triggers when the player is in an attack state, where
  the current HitDef has the following attributes:
      1. is either a standing or a crouching attack
  and 2. is either a normal attack or a special attack
```

HitFall

If the player is currently in a gethit state, returns the fall flag of the hit. The output of this trigger is undefined if the player is not in a gethit state. For an explanation of the fall flag, see the HitDef documentation.

Format:
HitFall

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = !HitFall
  Triggers if the hit did not put the player into a fall state.
```

HitOver

If the player is in a gethit state, returns 1 when the hittime has expired, and 0 otherwise. For an explanation of hittime, see the HitDef documentation.

Format:
HitOver

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = HitOver = 1
  Triggers when the player's hittime has expired.
```

HitPauseTime

Returns the time until the player's hitpause expires. The player enters a hitpause when his attack comes in contact with an opponent. The initial hitpause time is equal to the first value of the pausetime parameter in the player's HitDef. If ignorehitpause is not set, this will always return 0.

Format:
HitPauseTime

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = HitPauseTime = 0
  Triggers when the player is not paused for a hit.
```

HitShakeOver

If the player is in a gethit state, returns 1 if the hit shake (the period when he is shaking in place) has ended, and 0 otherwise.

Format:
HitShakeOver

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = HitShakeOver = 0
  Triggers if the player is still shaking from the hit.
```

HitVel

Gets the value of the velocity imparted to the player by a hit. You must specify the component that you want to check, eg. "HitVel Y" to check the vertical velocity component.

Format:
HitVel [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
none

Details:
A positive HitVel Y means that the player is moving upward on the screen. A positive HitVel X means that the player is moving backward. Note that the HitVel X trigger behaves in the opposite manner to the Vel X trigger.

Example:

```
trigger1 = HitVel X > 0.5
  True when the player's gethit x-velocity is greater than 0.5
  pixels per tick.
```

ID

Returns the ID number of the player. The ID number is unique for every player throughout the course of a match. Any helper that is created during this time will also receive its own unique ID number. This trigger may be useful for getting opponents' ID numbers, to be later used with the "playerID" redirection keyword (see exp docs). Do not confuse playerID with targetID.

Format:
ID

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
value = ID
  This sets value to the ID number of the current player.
value = EnemyNear, ID
  This sets value to the ID number of the nearest opponent.
```

IfElse (math)

This trigger takes three arguments. If the first is nonzero, IfElse returns the value of the second argument. Else, it returns the value of the third argument. All arguments are evaluated prior to execution of IfElse. In particular, any side effects caused by evaluation of the arguments (such as variable assignment, or performing a computation that generates a warning) will occur. If you wish to avoid these side effects, then use Cond.

Format:
IfElse(exp_cond,exp_true,exp_false)

Arguments:
exp_cond
Expression to test.

exp_true
Expression specifying value to return if exp_cond is nonzero.

exp_false
Expression specifying value to return if exp_cond is zero.

Return type:
Type of exp_true or exp_false, whichever is returned.

Error conditions:
Returns bottom if exp_cond evaluates to bottom, or if exp_true or exp_false (whichever is returned) evaluates to bottom.

Example:

```
value = ifelse(var(3),1,2)
  Sets value to 1 if var(3) is not zero, and sets value to 2 if
  var(3) is 0.
```

InGuardDist

Returns 1 if the player is within guarding distance of an opponent's physical or projectile attack. The guarding distance is the value of the guard.dist parameter of the opponent's HitDef. Returns 0 if out of guard distance, or the opponent is not attacking.

Format:
InGuardDist

Arguments:

none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:
none

IsHelper

This trigger takes an optional ID number as an argument. If the ID number is omitted, IsHelper returns 1 if the player is a helper character, and 0 otherwise. If the ID number is included, then IsHelper returns 1 if the player is a helper character with the specified ID number, and 0 otherwise.

Format:
1. IsHelper
2. IsHelper(exprn)

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
Returns bottom if exprn evaluates to bottom.

Examples:

```
1. trigger1 = !IsHelper
  Triggers if the player is not a helper-type character.
2. trigger1 = IsHelper(1234)
  Triggers if the player is a helper character with ID number 1234.
```

IsHomeTeam

Returns 1 if the player's team is considered the "home team". In arcade modes, the computer is always considered the home team. In versus modes, P1's side (left) is the home team.

Format:
IsHomeTeam

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:
none

Life

Returns the player's life.

Format:
Life

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = life <= 10
  Triggers if the player has 10 or less life points remaining.
```

LifeMax

Returns the maximum amount of life the player can have. This is normally the value of the "life" parameter in the [Data] group of the player variables, but may be different in situations such as team modes.

Format:
LifeMax

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = life < lifemax / 4
  Triggers if the player has less than 1/4 of his maximum life.
```

Ln (math)

Returns the natural logarithm of its argument. This produces slightly more accurate results than the otherwise equivalent expression log(e,(argument)).

Format:
ln(exprn)

Arguments:
exprn
Expression to compute the natural logarithm of (float).

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn is not positive.

Example:

```
value = ln(time)
  Sets value to the natural logarithm of the player's statetime.
```

Log (math)

Takes two arguments a and b, and returns the base-a logarithm of b.

Format:
Log(exp1,exp2)

Arguments:

exp1
: Expression giving the base of the logarithm. Must be positive.

exp2
: Expression giving the value to take the logarithm of. Must be positive.

Return type:
: float

Error conditions:
: Returns bottom if either of exp1 or exp2 evaluates to bottom, or if either of exp1 or exp2 is not positive.

Example:

```
value=log(2,64)
  Sets value to the base 2 log of 64, which is 6.0.
```

Lose

Returns 1 if the player (or the player's team, in team mode) has lost the round, 0 otherwise. Can be suffixed with "KO" or "Time" to return 1 only when the round has been lost by a KO or by time expiring, respectively.

Format:
1. Lose
2. LoseKO
3. LoseTime

Arguments:
: none

Return type:
: boolean int (1 or 0)

Error conditions:
: none

Examples:

```
1. trigger1 = Lose
  Triggers if the player (or his team) has lost the round.

2. trigger1 = !LoseKO
  Triggers if the player (or his team) has not lost the round by
  a KO. For example, this will trigger if the player's team has
  not yet lost the round, or if they have lost the round by time
  over.
```

MatchNo

Returns the current match number.

Format:
: MatchNo

Arguments:
: none

Return type:
: int

Details:
: The current match number is always 1 in versus-type modes. In Arcade and Team Arcade modes, the match number starts at 1 and increments every time a new match starts (does not increment on continue). If you finish the arcade mode and start a new game, the match number reverts to 1.

Example:
: none

MatchOver

Returns 1 if the match has ended. (For example, in the case of a best-of-three match, this will return true when one of the players or teams has won two rounds.)

Format:
MatchOver

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Details:
Currently, MatchOver does not return true until the players start their win poses (state 180). This behavior may be subject to change in future releases.

Example:

```
trigger1 = !matchover
  Triggers if the match is not over. For instance, the current round
  may not yet have ended, or it may have ended without deciding the
  match.
```

MoveContact

This trigger is valid only when the player is in an attack state. MoveContact gives a non-zero value if P2 has either been hit, or has guarded P1's attack. It gives 0 otherwise. P1 is the player, and P2 is his opponent. Typically used with the "StateNo" and "Command" triggers for detecting move-interrupts in the CMD file.

Format:
MoveContact

Arguments:
none

Return type:
int

Error conditions:
none

Details:
On attack contact, MoveContact returns 1. After contact, MoveContact's return value will increase by 1 for each game tick that P1 is not paused (P1 gets paused on contact; see pausetime parameter in HitDef controller). The values of MoveGuarded, MoveHit and MoveReversed increment in the same fashion.

Note 1: the values of MoveContact, MoveGuarded, MoveHit and MoveReversed are set simultaneously. For example, if one HitDef in a move hits successfully, MoveHit will return non-zero. If a following HitDef in the same move is guarded, MoveGuarded will return non-zero, and the other three triggers will return 0.

Note 2: the values of the four Move* triggers reset to 0 and stop incrementing after a state transition. See "movehitpersist" parameter for StateDefs (CNS docs) for how to override this behavior.

Examples:

```
trigger1 = MoveContact
  True if P1's attack did not miss P2.

trigger1 = MoveContact = 1
  True from the time P1's attack came in contact with P2, until just after P1's pausetime wears off.
```

MoveGuarded

This trigger is valid only when the player is in an attack state. MoveGuarded gives a non-zero value if P2 is guarding, or has guarded, P1's attack. It gives 0 otherwise. P1 is the player, and P2 is his opponent. Typically used with the "StateNo" and "Command" triggers for detecting move-interrupts in the CMD file.

Format:
MoveGuarded

Arguments:
none

Return type:
int

Error conditions:
none

Details:
See Details section for MoveContact trigger.

Example:

```
trigger1 = MoveGuarded
  True if P1's attack was guarded by P2.
```

MoveHit

This trigger is valid only when the player is in an attack state. MoveHit gives a non-zero value if P2 has been hit by P1's attack. It gives 0 otherwise. Typically used with the "StateNo" and "Command" triggers for detecting move-interrupts in the CMD file.

Format:
MoveHit

Arguments:
none

Return type:
int

Error conditions:
none

Details:
See Details section for MoveContact trigger.

Example:

```
trigger1 = MoveHit
  True if P1's attack connected successfully with P2.
```

MoveType(*,***)

MoveType gives the player's move-type. Refer to the section on StateDef in the CNS documentation for more details on MoveType. Useful for "move interrupts" in the CMD file.

Format:
MoveType [oper] move_type

Arguments:
[oper]
=, != (other operators not valid)

move_type (char)
A, I, H

Attack, Idle and GetHit move-types respectively.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = movetype != H
  Triggers if the player is not currently in a gethit-type state.
```

MoveReversed

This trigger is valid only when the player is in an attack state. MoveReversed gives a non-zero value if P1's attack has been reversed by P2. It gives 0 otherwise.

Format:
MoveReversed

Arguments:
none

Return type:
int

Error conditions:
none

Details:
See Details section for MoveContact trigger.

Example:

```
trigger1 = MoveReversed
  True if P1's attack was reversed by P2.
```

Name(*,***)

Returns the player's name (the internal name specified in the .DEF file, which may not be the same as the displayed name).

Format:
Name [oper] "name"

Arguments:
[oper]
=, != (other operators not valid)

"name" (string)
Name to compare against. Must be in double quotes.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = Name = "Kumquat"
  Returns true if the player is named Kumquat.
```

NumEnemy

NumEnemy returns the number of opponents that exist. Neutral players and normal helpers are not considered opponents.

Format:
NumEnemy

Arguments:
none

Return type:
int

Error conditions:
none

Examples:

```
trigger1 = NumEnemy = 2
trigger1 = enemynear(1), name = "Squash"
  Triggers if there are 2 opponents, and the second-closest one is
  named Squash.
```

NumExplod

This trigger takes an ID number as an optional argument. If the ID number is omitted, NumExplod returns the number of explods owned by the player. If the ID number is included, then NumExplod returns the number of explods with that ID number that are owned by the player. The ID number must be greater than -1. An ID number of -1 or less will give the same behavior as if the ID number is omitted.

Format:
1. NumExplod
2. NumExplod(exprn)

Arguments:
exprn
Expression evaluating to an ID number (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom.

Examples:

```
1. trigger1 = NumExplod >= 4
  Triggers if the player currently owns 4 or more explods.
2. trigger1 = NumExplod(1234) >= 4
  Triggers if the player currently owns 4 or more explods with ID
  1234.
```

NumHelper

This trigger takes an ID number as an optional argument. If the ID number is omitted, then NumHelper returns the total number of helpers currently owned by the player. If the ID number is included, then NumHelper returns the total number of helpers with that ID number owned by the player. The ID number must be greater than 0. If the ID number is 0 or less, then all helpers are counted.

Format:
1. NumHelper
2. NumHelper(exprn)

Arguments:
exprn
Expression evaluating to an ID number (int).

Return type:

int

Error conditions:
Returns bottom if exprn evaluates to bottom.

Examples:

```
1. trigger1 = NumHelper < 2
  Triggers if the player now has less than 2 helpers.
2. trigger1 = NumHelper(1234) < 2
  Triggers if the player now has less than 2 helpers with ID 1234.
```

NumPartner

NumPartner returns the number of partners that exist. Neutral players and normal helpers are not considered partners.

Format:
NumPartner

Arguments:
none

Return type:
int

Error conditions:
none

Examples:

```
trigger1 = NumPartner = 1
trigger1 = partner, life < 200
  Triggers if the player has a partner with less than 200 life
```

NumProj

Returns the total number of projectiles currently owned by the player.

Format:
NumProj

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = NumProj = 0
  Triggers if the player has no currently active projectiles.
```

NumProjID

This trigger takes an ID number as a required argument. It returns the number of projectiles currently owned by the player and having the specified ID number.

Format:
NumProjID(exprn)

Arguments:

exprn
Expression evaluating to an ID number.

Return type:
int

Error conditions:
If a negative ID is specified, then the ID defaults to 0. Returns bottom if exprn evaluates to bottom.

Example:

```
trigger1 = NumProjID(1234) = 1
  Triggers if there the player currently owns exactly 1 projectile
  with the ID number 1234.
```

NumTarget

This trigger takes an ID number as an optional argument. If the ID number is omitted, NumTarget returns the current number of targets for the player. If the ID number is included, then NumTarget returns the number of targets for the player which have that target ID number. The ID number must be greater than -1. An ID number of -1 or less will give the same behavior as if the ID number is omitted.

Format:
1. NumTarget
2. NumTarget(exprn)

Arguments:
exprn
Expression evaluating to an ID number (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom.

Examples:

```
1. trigger1 = NumExplod >= 4
  Triggers if the player currently owns 4 or more explods.
2. trigger1 = NumExplod(1234) >= 4
  Triggers if the player currently owns 4 or more explods with ID
  1234.
```

P1Name(*,***)

This is an alias for the Name trigger. See "Name".

P2BodyDist

Returns the distance of P2 from P1, where P1 is the player, and P2 is his opponent. P2BodyDist is useful in the CMD for cases where P1 has an attack that is different when performed close to P2.

Format:
P2BodyDist [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
none

Details:

For comparing the Y-distance, P2BodyDist gives the difference in the heights of the players' Y-axes. A negative value means that P2 is above P1.

For comparing the X-distance, P2BodyDist gives the X-distance of P2's front from P1's front. So, if the players are standing right next to each other, then P2BodyDist is 0. Remember that you can set the width of the player in "front.width", etc. under [Size] in the player variables.

See also P2Dist.

Example:

```
trigger1 = P2BodyDist X < 30
  Triggers if the front of P2 is within 30 pixels of the front of
  P1.
```

P2Dist

Returns the distance of P2 from P1, where P1 is the player, and P2 is his opponent.

Format:

P2Dist [component]

Arguments:

[component]

X, Y

Return type:

float

Error conditions:

none

Details:

For comparing the Y-distance, P2Dist gives the difference in the heights of the players' Y-axes. A negative value means that P2 is above P1.

For comparing the X-distance, P2Dist gives the X-distance of P2's axis from P1's axis. A positive value indicates P2 is in front of P1. See also P2BodyDist.

Example:

```
trigger1 = P2Dist Y <= -12
  True if P2 is at least 12 pixels higher up than P1.
```

P2Life

Same as Life, except that this returns the opponent's life.

P2MoveType

Same as MoveType, except that this returns the opponent's movetype.

P2Name(*,***)

Same as P1Name, except that this returns the name of the primary opponent (the opponent in versus mode, or the first opponent in team mode).

If there is no primary opponent, then p2name = "name" returns 0 no matter what name is specified. Similarly, p2name != "name" will return 1 no matter what name is specified.

P2StateNo

Same as StateNo, except that this returns the opponent's state number.

Error conditions:
Returns bottom if p2 does not exist. (For instance, if the round has been won.)

P2StateType

Same as StateType, except that this returns the opponent's state type.

Error conditions:
Returns bottom if p2 does not exist. (For instance, if the round has been won.)

P3Name(*,***)

Same as P1Name, except that this returns the name of the player's teammate, if present.

If there is no teammate, then p3name = "name" returns 0 no matter what name is specified. Similarly, p3name != "name" will return 1 no matter what name is specified.

P4Name(*,***)

Same as P1Name, except that this returns the name of the secondary opponent, if present.

If there is no secondary opponent, then p4name = "name" returns 0 no matter what name is specified. Similarly, p4name != "name" will return 1 no matter what name is specified.

PalNo

Returns the palette number of the player (i.e., the color scheme chosen for the character during character select.)

Format:
PalNo

Arguments:
none

Return type:
int

Error conditions:
none

Details:
The palette ordering is specified in the character's def file.
If omitted, the default ordering is:

A	B	C	X	Y	Z	A2	B2	C2	X2	Y2	Z2
1	2	3	4	5	6	7	8	9	10	11	12

Example:

```
trigger1 = PalNo = 5
  Returns true if the current palette number is 5.
```

ParentDist

This trigger is only valid for helper-type characters. ParentDist returns the distance from the helper to its parent. It works similarly to P2Dist.

Format:
ParentDist [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
Returns bottom if the player does not have a parent (e.g., if the parent was destroyed or KO'd).

Details:
For comparing the Y-distance, ParentDist gives the difference in the heights of the players' Y-axes. A negative value means that the parent is above its child. For comparing the X-distance, ParentDist gives the X-distance of the parent's axis from the child's axis. A positive value indicates the parent is in front of the child.

Example:

```
trigger1 = ParentDist X != 0
  Triggers if the parent is not at the exact same x-position as the
  helper character.
```

Pi (math)

This trigger returns the numeric value of pi (3.141593...)

Format:
pi

Arguments:
none

Return type:
float

Error conditions:
none

Pos

Gets the value of the player's position. You must specify the component that you want to check, eg. "Pos Y" to check the Y-position.

Format:
Pos [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
none

Details:
For "Pos X", the value is relative to the center of the screen (value 0). Negative is left, positive is right.

For "Pos Y", the value is relative to the floor. Negative is higher up, positive is below the floor.

Example:

```
trigger1 = Pos Y >= 0
  True when the player is below the floor.
```

Power

Returns the amount of power the player has.

Format:
Power

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = power >= 1000
  True if player has at least 1000 power (level 1).
```

PowerMax

Returns the maximum amount of power the player can have. This is normally 3000 (level 3).

Format:
PowerMax

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = power < powermax / 2
  True if player has less than half his maximum power.
```

PlayerIDExist

Returns 1 if a player with the specified ID number exists, 0 otherwise. This ID number is obtained using the "ID" trigger (see ID). Do not confuse PlayerID with TargetID.

Format:
PlayerIDExist(ID_number)

Arguments:
ID_number
An expression that evaluates to the ID number to check for (int)

Return type:
boolean int (1 or 0)

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
trigger1 = PlayerIDExist(var(4))
  Triggers if a player with an ID number equal to the value of
  var(4) exists.
```

PrevStateNo

Returns the number of the state that the player was last in. The results of this trigger are not guaranteed to be accurate.

Format:
StateNo

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = PrevStateNo = [200,650]
  Returns true if the player's last state number is between 200 and 650,
  inclusive.
```

ProjCancelTime

This trigger takes an required nonnegative ID number as an argument. If the player's last projectile to make any kind of contact was cancelled by an opponent's projectile and had the specified ID number, then ProjCancelTime returns the number of ticks since that contact occurred. If the specified ID number is 0, then the projectile ID is not checked. If no projectile meets all the above conditions, then ProjCancelTime returns -1.

Format:
ProjCancelTime(exprn)

Arguments:
exprn
Expression evaluating to a nonnegative ID number (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom. If a negative ID is specified, then the ID defaults to zero.

Examples:

```
1. trigger1 = ProjCancelTime(1234) = 1
  Triggers if a projectile with ID 1234 was just cancelled by an
  opponent's projectile.
2. trigger1 = ProjCancelTime(0) != -1 && ProjCancelTime(0) < 15
  Triggers if any of the player's projectiles were cancelled
  within last 15 ticks.
```

ProjContact(*,***)

This trigger takes an optional ID number as a suffix. If the ID number is omitted, ProjContact returns true if any of the player's projectiles either successfully hit the opponent or were guarded by the opponent. When the ID number is specified, ProjContact returns true only if any of the player's projectiles with the specified ID number either successfully hit the opponent or was guarded.

Format:

1. ProjContact[ID] = value
2. ProjContact[ID] = value, [oper] value2

Arguments:

[ID]
Optional ID number.

value (boolean)
Value to compare against. 0 for false, 1 for true.

[oper]
=, !=, <, >, <=, >=

value2
Time value to compare against.

Return type:
boolean int

Error conditions:
none

Details:
ProjContact will trigger once for each hit of the projectile, so a multi-hit projectile can trigger multiple times.

The first form of ProjContact shown above is only valid for one tick after contact, unlike MoveContact.

For the second form, ProjContact returns true if the projectile made contact n ticks ago, where n is a nonnegative number satisfying the relation "n [oper] value2".

Specifying an ID number of 0 gives the same behavior as if the ID number is omitted (check all projectiles).

Examples:

```
1. trigger1 = ProjContact1234 = 1
  Triggers if a projectile with ID 1234 just made contact with the
  opponent.
2. trigger1 = ProjContact456 = 0, < 15
  Triggers if no projectile with ID 456 made contact in the last 15
  ticks.
```

ProjContactTime

This trigger takes an required nonnegative ID number as an argument. If the player's last projectile to make any kind of contact, made contact with the opponent and had the specified ID number, then ProjContactTime returns the number of ticks since that contact occurred. If the specified ID number is 0, then the projectile ID is not checked. If no projectile meets all the above conditions, then ProjContactTime returns -1.

Format:
ProjContactTime(exprn)

Arguments:

exprn
Expression evaluating to a nonnegative ID number (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom. If a negative ID is specified, then the ID defaults to zero.

Examples:

```
1. trigger1 = ProjContactTime(1234) = 1
  Triggers if a projectile with ID 1234 just made contact with
  the opponent.
2. trigger1 = ProjContactTime(0) != -1 && ProjContactTime(0) < 15
  Triggers if any of the player's projectiles made successful
  contact with the opponent within the last 15 ticks.
```

ProjGuarded(*,***)

This trigger takes an optional ID number as a suffix. If the ID number is omitted, ProjGuarded returns true if any of the player's projectiles were guarded by the opponent. When the ID number is specified, ProjGuarded returns true only if one of the player's projectiles with the specified ID number was guarded by the opponent.

Format:

1. ProjGuarded[ID] = value
2. ProjGuarded[ID] = value, [oper] value2

Arguments:

[ID]
: Optional ID number.

value (boolean)
: Value to compare against. 0 for false, 1 for true.

[oper]
: =, !=, <, >, <=, >=

value2
: Time value to compare against.

Return type:
: boolean int (1 or 0)

Error conditions:
: none

Details:
: ProjGuarded will trigger once for each hit of the projectile, so a multi-hit projectile can trigger multiple times.

: The first form of ProjGuarded shown above is only valid for one tick after hit, unlike MoveGuarded.

: For the second form, ProjGuarded returns true if the projectile was guarded n ticks ago, where n is a nonnegative number satisfying the relation "n [oper] value2".

: Specifying an ID number of 0 gives the same behavior as if the ID number is omitted (check all projectiles).

Examples:

```
1. trigger1 = ProjGuarded1234 = 1
  Triggers if the opponent just blocked a projectile with ID 1234.
2. trigger1 = ProjGuarded = 1, < 15
  Triggers if the opponent blocked any projectile in the last 15
  ticks.
```

ProjGuardedTime

This trigger takes an required nonnegative ID number as an argument. If the player's last projectile to make any kind of contact was guarded by the opponent and had the specified ID number, then ProjGuardedTime returns the number of ticks since that contact occurred. If the specified ID number is 0, then the projectile ID is not checked. If no projectile meets all the above conditions, then ProjGuardedTime returns -1.

Format:
: ProjCancelTime(exprn)

Arguments:

exprn
: Expression evaluating to a nonnegative ID number (int).

Return type:
: int

Error conditions:
: Returns bottom if exprn evaluates to bottom. If a negative ID is specified, then the ID defaults to zero.

Examples:

```
1. trigger1 = ProjGuardedTime(1234) = 1
  Triggers if a projectile with ID 1234 was just guarded by the
  opponent.
2. trigger1 = ProjGuardedTime(0) != -1 && ProjGuardedTime(0) < 15
  Triggers if any of the player's projectiles was guarded by the
  opponent within the last 15 ticks.
```

ProjHit(*,***)

This trigger takes an optional positive ID number as a suffix. If the ID number is omitted, ProjHit returns true if any of the player's projectiles successfully hit the opponent. When the ID number is specified, ProjHit returns true only if one of the player's projectiles with the specified ID number successfully hit the opponent.

Format:

1. ProjHit[ID] = value
2. ProjHit[ID] = value, [oper] value2

Arguments:

[ID] (int)
: Optional ID number.

value (boolean)
: Value to compare against. 0 for false, 1 for true.

[oper]
: =, !=, <, >, <=, >=

value2
: Time value to compare against.

Return type:
: boolean int

Error conditions:
: none

Details:
: ProjHit will trigger once for each hit of the projectile, so a multi-hit projectile can trigger multiple times.

  The first form of ProjHit shown above is only valid for one tick after hit, unlike MoveHit.

  For the second form, ProjHit returns true if the projectile hit n ticks ago, where n is a nonnegative number satisfying the relation "n [oper] value2".

  Specifying an ID number of 0 gives the same behavior as if the ID number is omitted (check all projectiles).

Examples:

```
1. trigger1 = ProjHit1234 = 1
  Triggers if a projectile with ID 1234 just made successful
  contact with the opponent.
2. trigger1 = ProjHit1234 = 1, < 15
  Triggers if any of the player's projectiles made successful
  contact with the opponent within the last 15 ticks.
```

ProjHitTime

This trigger takes an required nonnegative ID number as an argument. If the player's last projectile to make any kind of contact successfully hit the opponent and had the specified ID number, then ProjHit returns the number of ticks since that contact occurred. If the specified ID number is 0, then the projectile ID is not checked. If no projectile meets all the above conditions, then ProjHitTime returns -1.

Format:
: ProjHitTime(exprn)

Arguments:
: exprn

Expression evaluating to a nonnegative ID number (int).

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom. If a negative ID is specified, then the ID defaults to zero.

Examples:

```
1. trigger1 = ProjHitTime(1234) = 1
  Triggers if a projectile with ID 1234 just made successful
  contact with the opponent.
2. trigger1 = ProjHitTime(0) != -1 && ProjHitTime(0) < 15
  Triggers if any of the player's projectiles made successful
  contact with the opponent within the last 15 ticks.
```

Random

Returns a random number between 0 and 999, inclusive.

Format:
Random

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = Random <= 249
  Triggers if the random number returned is less than or equal to
  249. (This occurs with 25% probability.)
```

RootDist

This trigger is only valid for helper-type characters. RootDist returns the distance from the helper to its root. The root is the main player character who owns the helper: for instance, if you select Kumquat to play with, and Kumquat spawns a helper named Kiwi, who in turn spawns a helper named Penguin, then Penguin's root is Kumquat, and Penguin is a descendant of Kumquat. RootDist works similarly to P2Dist.

Format:
RootDist [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
Returns bottom if the player has no root.

Details:
For comparing the Y-distance, RootDist gives the difference in the heights of the players' Y-axes. A negative value means that the root is above its descendant.

For comparing the X-distance, ParentDist gives the X-distance of the root's axis from the descendant's axis. A positive value indicates the root is in front of its descendant.

Example:

```
trigger1 = RootDist X != 0
  Triggers if the root is not at the exact same x-position as the
  helper character.
```

RoundNo

Returns the current round number.

Format:
RoundNo

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = RoundNo = 3
  Triggers if this is the third round of the match.
```

RoundsExisted

Returns the number of rounds the player has existed for. On the first round, returns 0. This is useful for a Turns mode intro.

Format:
RoundsExisted

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = RoundsExisted = 0
trigger1 = TeamMode = Turns
trigger1 = RoundNo > 0
  Triggers if the player has just entered a Turns mode team match
  after the first round. You can use this example with a
  ChangeState controller to switch to an intro state by
  overriding the Initialize state (state 5900).
```

RoundState

Returns the current round state number.

Format:
RoundState

Arguments:
none

Return type:
int

Error conditions:

none

Details:

Return values: 0: Pre-intro - screen fades in 1: Intro 2: Fight - players do battle 3: Pre-over - just a round is won or lost 4: Over - win poses

Example:

```
trigger1 = RoundState = 2
  Triggers if the actual fighting portion of the round is in
  progress.
```

ScreenPos

Gets the value of the player's absolute (screen-relative) position. You must specify the component that you want to check, eg. "Pos Y" to check the Y-position.

Format:

Pos [component]

Arguments:

[component]

X, Y

Return type:

float

Error conditions:

none

Details:

For "ScreenPos X", the value is relative to the left of the screen (value 0). Negative is left, positive is right.

For "ScreenPos Y", the value is relative to the top of the screen. Negative is above the screen, positive is downward.

Example:

```
trigger1 = ScreenPos Y >= 0 && ScreenPos Y < GameHeight
  True when the player's is in the screen's vertical extent.
```

SelfAnimExist

Like AnimExist, except that this only checks P1's animation data. If P1 has been given P2's animation data by a hit, SelfAnimExist will not check P2's animation data to determine whether or not a given action exists.

Sin (math)

Computes the sine of the specified argument (in radians.)

Format:

sin(exprn)

Arguments:

exprn

Expression to compute the sine of. (float)

Return type:

float

Error conditions:

Returns bottom if exprn evaluates to bottom.

Example:

```
value = sin(pi/2)
  Sets value to the sine of pi/2, which is approximately 1.0
  (possibly with some rounding error.)
```

StateNo

Returns the player's current state number. Useful for "move interrupts" in the CMD file.

Format:
StateNo

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = stateno = [200,650]
  Returns true if the state number is between 200 and 650,
  inclusive.
```

StateType

StateType gives the player's state-type. Refer to the section on StateDef in the CNS documentation for more details on StateType. Useful for "move interrupts" in the CMD file.

Format:
StateType [oper] state_type

Arguments:
[oper]
=, != (other operators not valid)

state_type (char)
S, C, A
Stand, Crouch and Air state-types.

Return type:
boolean int (1 or 0)

Error conditions:
none

Example:

```
trigger1 = StateType != A
  Triggers if the player is not in an air-type state.
```

StageVar(*,***)

Returns information about the stage. A limited number of parameters are supported.

Format:
StageVar(param_name) [oper] "string"

Arguments:
param_name
The name of the stage parameter to check. Valid values are:
info.name, info.displayname, info.authorname.

[oper]
=, != (other operators not valid)

"string" (string)
String to compare against. Must be in double quotes.

Return type:
boolean int (1 or 0)

Error conditions:
none

Details:
- info.author: Compares the value of the "author" parameter in the [Info] group. (boolean)
- info.displayname: Compares the value of the "displayname" parameter in the [Info] group. (boolean)
- info.name: Compares the value of the "name" parameter in the [Info] group. (boolean)

Versions:
1.0 and newer

Example:

```
trigger1 = StageVar(info.author) = "Suika"
  Returns true if the stage author is named "Suika".
```

SysFVar

This trigger takes a mandatory variable number as an argument. It returns the value of the player's specified system float variable. This trigger should NOT be used under normal circumstances. System variables are reserved for bookkeeping in common1.cns.

Format:
FVar(exprn)

Arguments:
exprn
An expression evaluating to a variable number. Valid numbers at this time are 0-4.

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn evaluates to an invalid variable index.

Example:

```
trigger1 = SysFVar(0) = -1.23
  Triggers if the value of system float variable 0 is -1.23.
```

SysVar

This trigger takes a mandatory variable number as an argument. It returns the value of the player's specified system int variable. This trigger is NOT to be used under normal circumstances. System variables are reserved for bookkeeping in common1.cns.

Format:
Var(exprn)

Arguments:
exprn
An expression evaluating to a variable number. Valid numbers at this time are 0-4.

Return type:
int

Error conditions:
Returns bottom if exprn evaluates to bottom, or if exprn evaluates to an invalid variable index.

Example:

```
trigger1 = SysVar(0) = -34
  Triggers if the value of system variable 0 is -34.
```

Tan (math)

Computes the tangent of the specified argument (in radians.)

Format:
tan(exprn)

Arguments:
exprn
Expression to compute the tangent of. (float)

Return type:
float

Error conditions:
Returns bottom if exprn evaluates to bottom.

Example:

```
value = tan(pi/4)
  Sets value to the tangent of pi/4, which is approximately 1.0
  (possibly with some rounding error.)
```

TeamMode(*,***)

TeamMode gives the current mode of play for the player's team.

Format:
TeamMode [oper] mode

Arguments:
[oper]
=, != (other operators not valid)

mode (string)
single - single player
simul - 2 players simultaneously
turns - turns battle

Return type:
boolean int (1 or 0)

Error conditions:
none

Notes:
In survival mode, TeamMode = turns on the enemy side.

Example:

```
trigger1 = TeamMode = Single
  Triggers if the player is playing in single play.
```

TeamSide

Returns the team side the player is on. 1 represents P1 side (left), 2 for P2 side (right).

Format:
TeamSide

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = TeamSide = 2
  Triggers if player is on the P2 side team.
```

TicksPerSecond

Returns the number of ticks per second. Useful for time calculations.

Format:
TicksPerSecond

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = Time > 10 * TicksPerSecond
  Triggers after 10 seconds, regardless of game speed.
```

Time

Returns the state-time of the player (that is, the number of ticks that the player has been in the current state so far).

Format:
Time

Arguments:
none

Return type:
int

Error conditions:
none

Example:

```
trigger1 = Time = 2
  Triggers when the player's state-time is 2.
```

TimeMod(*,**,***)

Returns the remainder when the state-time of the player is divided by the specified value.

The % operator subsumes the functionality of TimeMod, so it is recommended that you use % instead.

Format:

TimeMod [oper] divisor, value1

Arguments:

[oper]

=, !=, <, >, <=, >=

divisor (int)

Positive number to use as a divisor.

value1 (int)

Value to compare remainder against.

Return type:

boolean int (1 or 0)

Error conditions:

Returns bottom if the divisor is 0.

Example:

```
trigger1 = TimeMod = 4, 3
  Triggers when the state-time is 3, 7, 11, 15, ...
```

UniqHitCount

Returns the total number of hits the player's current attack move has done. This value is valid only for a single state; after any state change, it resets to 0. To prevent it from resetting to 0, set hitcountpersist in the StateDef (see cns documentation for details). The HitCount and UniqHitCount triggers differ only when the player is hitting more than one opponent. In the case where the player is hitting two opponents with the same attack, HitCount will increase by 1 for every hit, while UniqHitCount increases by 2.

Format:

UniqHitCount

Arguments:

none

Return type:

int

Error conditions:

none

Example:

```
trigger1 = UniqHitCount = [4,6]
  Triggers when 4, 5 or 6 hits have been dealt since
  the start of the player's attack move.
```

Var

This trigger takes a mandatory variable number as an argument. It returns the value of the player's specified int variable.

Format:

Var(exprn)

Arguments:

exprn

An expression evaluating to a variable number. Valid numbers at this time are 0-59.

Return type:

int

Error conditions:

Returns bottom if exprn evaluates to bottom, or if exprn evaluates to an invalid variable index.

Example:

```
trigger1 = Var(0) = -34
  Triggers if the value of variable 0 is -34.
```

Vel

Gets the value of the player's velocity. You must specify the component that you want to check, eg. "Vel Y" to check the Y-velocity.

Format:
Vel [component]

Arguments:
[component]
X, Y

Return type:
float

Error conditions:
none

Details:
For Vel X, a positive value indicates that the player is moving forward. (This behavior is the opposite of HitVel X's behavior.) For Vel Y, a positive value indicates that the player is moving downward.

Example:

```
trigger1 = Vel Y >= 0
  True when the player is not moving upward.
```

Win

Returns true if the player (or the player's team, in team mode) has won the round, false otherwise. Can be suffixed with "KO", "Time", or "Perfect" to return true only when the round has been won by a KO, by time expiring, or with no life lost, respectively.

Format: 1. Win 2. WinKO 3. WinTime 4. WinPerfect

Arguments:
none

Return type:
boolean int (1 or 0)

Error conditions:
none

Examples:

```
1. trigger1 = Win
  Triggers if the player (or his team) has won the round.
2. trigger1 = !WinKO
  Triggers if the player (or his team) has not won the
  round by a KO. For example, this will trigger if the
  player's team has not yet won the round, or if they have
  won the round by time over.
```