FUNCTION

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/function

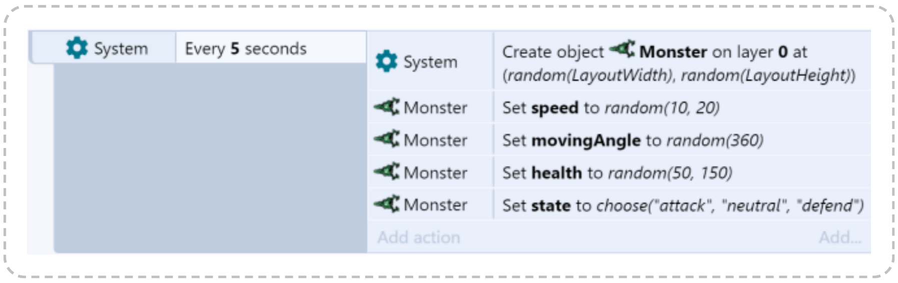
The Function plugin has been replaced with a new built-in functions feature and is now deprecated. This manual entry is provided for archival reasons only and may be removed in future.

The **Function** object can run a different event (*On function*) in an action (*Call function*). This is analogous to functions in traditional programming languages. Using functions can help you organise events and avoid having to duplicate groups of actions or events.

About functions

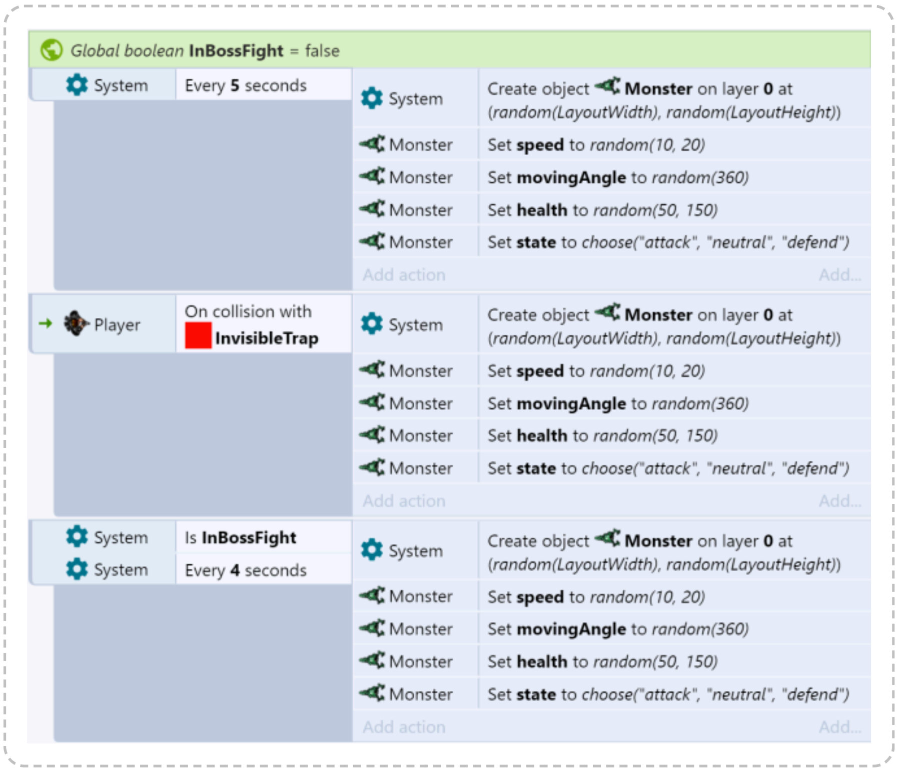
The main purpose of the Function object is using the *Call function* action. This takes the name of a function (e.g. *Call function "CreateEnemy"*). The action then triggers the corresponding *On function* event (e.g. *On function "CreateEnemy"*), running the event's actions and any sub-events, before returning to the original *Call function* action and continuing from where it was.

As another example, suppose you create an enemy with random stats in a game every 5 seconds using this event:



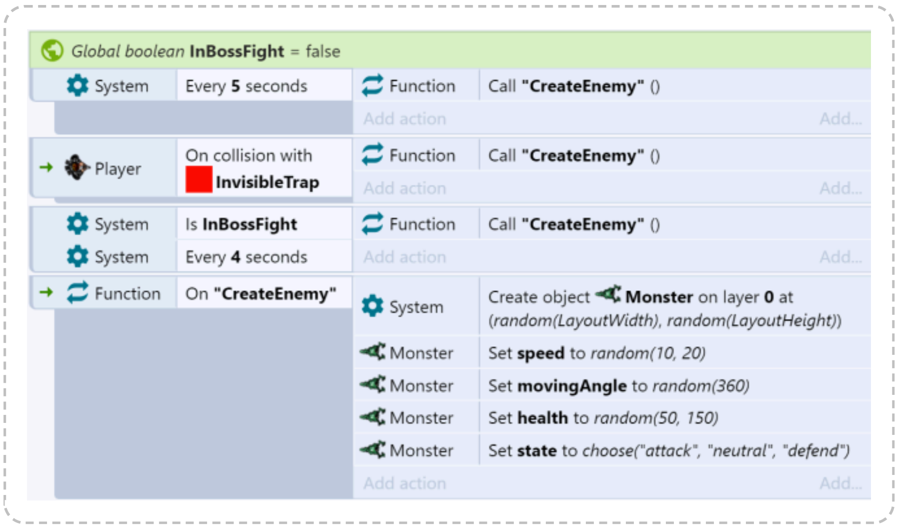
System	Every **5** seconds	System	Create object **Monster** on layer **0** at *(random(LayoutWidth), random(LayoutHeight))*
		Monster	Set **speed** to *random(10, 20)*
		Monster	Set **movingAngle** to *random(360)*
		Monster	Set **health** to *random(50, 150)*
		Monster	Set **state** to *choose("attack", "neutral", "defend")*

Suppose there are two other events where you want to create an enemy the same way: one when a player walks in to a trap, and another one every 4 seconds when in a boss fight. Without functions, you may have to copy-and-paste the actions multiple times, like this:



```
Global boolean InBossFight = false
System | Every 5 seconds
    System  | Create object Monster on layer 0 at (random(LayoutWidth), random(LayoutHeight))
    Monster | Set speed to random(10, 20)
    Monster | Set movingAngle to random(360)
    Monster | Set health to random(50, 150)
    Monster | Set state to choose("attack", "neutral", "defend")
    Add action                                                   Add...
Player | On collision with InvisibleTrap
    System  | Create object Monster on layer 0 at (random(LayoutWidth), random(LayoutHeight))
    Monster | Set speed to random(10, 20)
    Monster | Set movingAngle to random(360)
    Monster | Set health to random(50, 150)
    Monster | Set state to choose("attack", "neutral", "defend")
    Add action                                                   Add...
System | Is InBossFight
System | Every 4 seconds
    System  | Create object Monster on layer 0 at (random(LayoutWidth), random(LayoutHeight))
    Monster | Set speed to random(10, 20)
    Monster | Set movingAngle to random(360)
    Monster | Set health to random(50, 150)
    Monster | Set state to choose("attack", "neutral", "defend")
    Add action                                                   Add...
```

Notice this is becoming inconvenient. There may be times you need to repeat the actions in even more places. If you want to make a change, you then have to find every place you repeated the actions, and repeat the change. We can remove the repetition using functions. By creating a *CreateEnemy* function which has the repeated actions, we can replace all the repeated actions with a *Call function* action like this:



```
Global boolean InBossFight = false
System | Every 5 seconds
    Function | Call "CreateEnemy" ()
    Add action                                                   Add...
Player | On collision with InvisibleTrap
    Function | Call "CreateEnemy" ()
    Add action                                                   Add...
System | Is InBossFight
System | Every 4 seconds
    Function | Call "CreateEnemy" ()
    Add action                                                   Add...
Function | On "CreateEnemy"
    System  | Create object Monster on layer 0 at (random(LayoutWidth), random(LayoutHeight))
    Monster | Set speed to random(10, 20)
    Monster | Set movingAngle to random(360)
    Monster | Set health to random(50, 150)
    Monster | Set state to choose("attack", "neutral", "defend")
    Add action                                                   Add...
```

This works identically to the previous events, but is much shorter and more convenient. We can use *Call "CreateEnemy"* action anywhere we want to create an enemy, and it uses the same set of actions in the *On "CreateEnemy"* event.

It is often useful to split many parts of your events in to functions like this, so they can be conveniently re-used across event sheets.

Parameters

When calling a function, you can also pass *parameters*. These are simply numbers or strings that are made available to the function. For example, the *CreateEnemy* function from the previous example could be modified to take two parameters: the X and the Y co-ordinates at which to create the enemy. This helps functions to be made more general purpose by using extra information from the action calling the function.

To add a parameter to a function call, click the **Add parameter** link that appears in the Parameters dialog when editing the *Call function* action. This is a special link that only appears for this action in the Function object. Inside an *On function* event, you can then use the *Param* expression with the zero-based index of the parameter to retrieve the corresponding value.

Advanced function features

Like in programming languages, the Function object supports the following:

- Functions calling other functions
- Functions calling themselves (recursion)
- Returning values from functions
- Calling functions from expressions (which also returns the return value)

Note that functions calling other functions or recursing create a new "stack" of local variables. In other words, like in programming languages, local variables are unique at each level of function call. This does not apply to static local variables or global variables.

Also note the Function object logs to the browser console if it is used incorrectly, such as calling a non-existent function or accessing a parameter that was not passed. This can help identify problems using functions in large projects.

Returning values from functions

Functions can also return a result. For example, a *factorial* function could calculate the mathematical result and return it. In an *On function* event, the return value can be set using the *Set return value* action.

If the event was called using the *Call function* action, the returned value is afterwards available using the *ReturnValue* expression. Functions can also be called directly from an expression using the *Call* expression; in this case the return value is automatically returned as the result of the *Call* expression.

JavaScript integration

It is strongly recommended to use the Addon SDK to integrate JavaScript code with Construct. However it is possible to trigger a function in the Function object from JavaScript code using the following function:

```
if (self.c2_callFunction)
    self.c2_callFunction("name", ["param1", "param2"]);
```

(The name still refers to C2 for legacy reasons.) Note if the Function object is not included in a project, the `c2_callFunction` function will not exist, so the if check is necessary before using it. The function with the given "name" is triggered synchronously. Parameters are optional and can be omitted, but must be provided as an array in the second argument, and parameters may only be string or number values (any other types will return as 0 in Construct). The `c2_callFunction` method also returns the return value set in Construct (if any), and also can only return a string or number.

Function conditions

Compare parameter

Compare the value of one of the parameters to a function call. This condition should only be used in an *On function* event, since outside of function calls there are no parameters set.

On function

Triggered when the corresponding *Call function* action is used.

Function actions

Call expression

This is an alternative to the *Call function* action. It simply provides a parameter to enter an expression, and the result is ignored. You can use this to call a function via the *Function.Call(...)* expression, which may be more convenient if using a very large number of parameters.

Call function

Trigger the corresponding *On function* events. Additional parameters can be passed that are accessed by the *Param* expression.

Set return value

In a function event, set the value to be returned to the caller. This is either returned by the *Call* expression or accessed later using the *ReturnValue* expression.

Function expressions

Call

Call a function directly from an expression. The expression returns the return value that was set in the function, or 0 if no return value was set. Additional parameters can optionally be added after the name of the function, e.g. `Function.Call("CreateEnemy", 123, 456)`.

Param

Retrieve a parameter passed to a function call by its zero-based index. For example, `Function.Param(0)` returns the value of the first parameter.

ParamCount

Return the number of parameters passed to a function call.

ReturnValue

Return the value set using the *Set return value* action from the last function call. If *Set return value* is not used in a function, it returns 0.