EXPRESSIONS

View online: https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/expressions

In events, **expressions** are used to calculate sums or retrieve information from objects, such as a Sprite's X co-ordinate. Expressions are entered in to the Parameters dialog when adding or editing a condition or action which has parameters. The Expressions dictionary is also shown at the same time and provides a dictionary of all the system and object expressions available in a project.

Some examples of expressions, which can range from a simple number to a complex calculation, are given below:

- `0`
- `random(360)`
- `Sprite.X`
- `(Player1.X + Player2.X) / 2`
- `Sprite.8Direction.Speed`
- `Sprite.X + cos(Sprite.Angle) * Sprite.Speed * dt`

Numbers

Numbers are simply entered as digits with an optional fractional part separated by a dot, e.g. `5` or `-1.2`. Fractional numbers may begin with a dot, e.g. `.5`.

Text (strings)

Text is also known as *strings* in software development, and Construct also sometimes uses this naming convention. Text in expressions should be surrounded by double-quotes, e.g. `"Hello"`

The double-quotes are not included as part of the text, so setting a text object to show the expression `"Hello"` will make it show **Hello**, without any double-quotes. To include a double-quote in a string, use two double-quotes next to each other (""), e.g. `"He said ""hi"" to me"` will return **He said "hi" to me**.

> *Using quotes for strings only applies to expressions. Don't use them in other places like in property values in the Properties Bar.*

You can use `&` to build strings out of mixed text and numbers, e.g. `"Your score is: " & score`

To add a line break to a string use the system expression **newline**, e.g. `"Hello" & newline & "world"`

The **StringSub** system expression is also useful for building strings. You can use it like this: `StringSub("Hi {0}, your score is {1}!", "Sam", 100)` which will produce the string **Hi Sam, your score is 100!**. The first string can use *placeholders* like `{0}`, `{1}`, `{2}` etc. and these are replaced by the following parameters. You can have as many placeholders as you like and they can appear anywhere in the first string, so long as the appropriate number of parameters follow on.

Operators

You can use the following operators in expressions:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder after division)
^	Raise to power, e.g. 5 ^ 2 = 25
&	Build strings, e.g. `"Your score is: " & score`
=, <>, <, <=, >, >=	Comparison operators, e.g. `score < 10`. Return 1 if comparison is true or 0 if false.
?:	Conditional operator, in the form `condition ? result_if_true : result_if_false`. Allows testing conditions in expressions. The condition counts as true if it is non-zero, and false if it is zero. E.g. `score < 0 ? "Game over!" : "Keep going!"`
&, |	When used on numbers, `&` is logical AND and `|` is logical OR. (Note if either side is a string, `&` instead does string concatenation.) These are useful combined with the comparison operators, e.g. `score < 0 | health < 0`, which returns 1 if either condition is true, else 0 for false.

Note a common mistake is to write comparison expressions like `value = 1 | 2` with the intent to match `value` to either 1 or 2. However this doesn't work as it is actually evaluated as `(value = 1) | 2`, which always evaluates as true. Similarly `value = (1 | 2)` won't work as `1 | 2` evaluates to true, so it only tests if `value` is true. The correct way to test this is using `value = 1 | value = 2`.

Object expressions

Objects have their own expressions to retrieve information about the object. These are written in the form `Sprite.X` (the object name, a dot, then the expression name). The Expressions dictionary lists all the available expressions in the project, and they are further documented in the reference section of the manual.

The expression `Self` can be used as a short-cut to refer to the current object. For example, in an action for the *Player* object, `Self.X` refers to `Player.X`.

You can add a zero-based object index to get expressions from different object instances. For example `Sprite(0).X` gets the first Sprite instance's X position, and `Sprite(1).X` gets the second instance's X position. For more information see index IDs (IIDs) in instances. You can

also pass another expression for the index. Negative numbers start from the opposite end, so `Sprite(-1).X` gets the last Sprite's X position.

Behavior expressions

If an object has a behavior with its own expressions, they are written in the form `Object.Behavior.Expression`, e.g. `Sprite.8Direction.Speed`.

System expressions

The built-in system expressions are listed in the reference. These include some basic mathematical functions like `sqrt` (square root).