JSON

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

The **JSON** object can parse and read strings in JavaScript Object Notation (JSON) format, as well as writing data and converting the result back to a JSON string. A description of the JSON format is out of the scope of this manual; however there are some free tutorials you can search for on the web.

Click here to open an example of the JSON plugin.

Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IJSONInstance script interface. (JavaScript and TypeScript have built-in support for JSON, but this allows for interacting with data used in an event sheet.)

Loading a JSON file

JSON must be loaded as a string with the *Parse* action. If you have a small snippet of JSON, you can paste it directly in to the action parameter - but note in expressions a double-quote character (`"`) must be repeated twice (`""`) to avoid ending the string, which can be inconvenient. Instead it is recommended to request a JSON project file using the AJAX object. When the AJAX request completes, pass `AJAX.LastData` in to the *Parse* action. Then the data from the file can be used.

JSON paths

Construct only supports numbers and strings in expressions. To allow you to use structures like nested objects and arrays, the JSON object uses a special *path string* which identifies keys in the JSON data. The path is similar to the JavaScript syntax that would be used to access the JSON data, but note it is not actually evaluated as JavaScript code.

A path is essentially a list of nested keys separated by dots. For example consider the following JSON data:

```
{
        "foo": {
                "bar": 42
        }
}
```

The path `foo.bar` refers to the inner "bar" key, which will return the value 42.

Escaping

If a JSON key actually has a dot in it, e.g. `"my.key"`, then the dot needs to be escaped in a path string otherwise it will try to look for a key named `my` with another object key named `key` inside it. Dots preceded by a backslash (`\`) will be interpreted as part of the key name, e.g. `my\.key` will look for a key named `"my.key"`. To actually use a backslash in a key name, then use a double backslash in a path, e.g. `my\\key` will look for a key named `"my\key"`.

Types

Since Construct expressions only support numbers and strings, only number and string type properties can be directly returned in expressions. Booleans are returned as a number (0 for false and 1 for true). However using paths, loops and conditions, there are a variety of tools to identify what kind of data is available and access values held within objects and arrays (even when nested), as well as detecting special values like `null`.

Arrays

Array elements can be accessed as if their elements were numbered properties (which is actually how JavaScript specifies arrays internally). For example consider the following JSON data:

```
{
        "array": [123, 456]
}
```

Like most of Construct JSON arrays use zero-based indices, so the path `array.0` refers to the first element (123) and `array.1` refers to the second element (456), and so on.

Relative paths

The *Set path* action changes the current path, making it more convenient to access deeply nested keys. A relative path can then be used to continue from the current path. Relative paths begin with a dot, e.g. `.bar`. If a path does not begin with a dot, it is always treated as absolute (starting from the root), regardless of the current path.

For example suppose you want to access multiple keys under a common path, like `foo.bar.baz.first` and `foo.bar.baz.second`. You can first use *Set path* to set `foo.bar.baz` as the current path, and then the paths `.first` and `.second` refer to `foo.bar.baz.first` and `foo.bar.baz.second` respectively. Even with a current path set, the path `abc.def` refers to top-level keys because it does not start with a dot, so is treated as absolute.

The *For each* looping condition also sets the current path to the full path to the current key being iterated, making it convenient to retrieve data in a loop.

JSON conditions

Compare type

Test the type of a value at a given path. This can also detect the special `null` value that cannot be returned in a Construct expression, as well as identifying arrays and objects.

Compare value

Compare the value at a given path. This can only be used with number or string values.

For each

Repeats once for each key at a path in the JSON data. This can be used with either object or array types; in the case of arrays, the keys are the array indices (e.g. 0, 1, 2...) represented as a string. Inside the loop, the current path is set to the current key being iterated, so relative paths can be used to retrieve data from the current key. The *CurrentKey*, *CurrentValue* and *CurrentType* expressions return information about the current key-value pair being iterated.

Has key

Determine if a key exists at a given path.

Is boolean set

Determine if a given path contains a boolean "true" value.

On parse error

Triggered after a *Parse* action if there was invalid syntax in the JSON string resulting in an error trying to parse it.

On parse success

Triggered after a *Parse* action if the JSON syntax was valid and parsing completed successfully.

JSON actions

Note that when setting values, nonexistent keys are created as necessary. For example if the JSON file is empty but you set the number 5 at the path `foo.bar`, the `foo` and `bar` keys are automatically created, resulting in the data `{ "foo": { "bar": 5 } }`.

Delete key

Delete the key at a path so it is no longer present in the JSON data.

> *Note this action cannot remove elements from an array. Use the array modifying actions instead.*

Parse

Parse a string of JSON data and load it in to the object so it can be accessed. If the data is valid and is parsed successfully, *On parse success* is triggered; otherwise if the data is invalid and parsing fails, *On parse error* is triggered.

Pop value

Remove an element at the start or end of an array located at a path. If the path does not specify an array, this does nothing.

Push value

Add an element with the given value at the start or end of an array located at a path. If the path does not specify an array, this does nothing.

Insert value

Inserts an element with the given value into an array located at a path, increasing the size of the array by 1. The element is inserted at a specified index, if any elements existed at or after that index they pushed forward by 1.

Remove values

Removes a specified number of elements from an array located at a path, reducing the size of the array. Elements are removed starting at a specified index, if there are less elements after the array than requested to be removed then only the available number will be removed.

Set array

Create an array with a given number of elements at a path. If an array already exists at the given path, it is resized to the given number of elements. In both cases, any new elements are initialised to 0.

Set boolean

Set a true or false value at a path.

Set JSON

Parse a string of JSON data, and set the value at a path to the resulting JSON. This is useful to merge data from different sources in to the same JSON object.

Set null

Set the special `null` value at a path.

Set object

Set an empty object at a path. If there is already an object at the given path, it is replaced with an empty object.

Set path

Set the current path. This allows relative keys to continue from this path.

Set value

Set a number or string value at a path.

Toggle boolean

Toggle a boolean value at a path. If the value at the path is not a Boolean, do nothing.

Add to

Add a value to the numerical value at a path. If the value at the path is not numerical, do nothing.

Subtract from

Subtract a value from the numerical value at a path. If the value at the path is not numerical, do nothing.

JSON expressions

ArraySize

Return the length of an array at a path. If there is not an array at the given path, returns -1.

Back

Front

Return the element at the start (front) or end (back) of an array at a given path. If there is not an array at the given path, returns -1.

CurrentKey

In a *For each* loop, a string of the current key name. If an array is being looped, the current key is a string of the current index, e.g. "0", "1"...

CurrentType

In a *For each* loop, a string representing the type of the current value, which can be one of `"null"`, `"array"`, `"object"`, `"boolean"`, `"number"` or `"string"`.

CurrentValue

In a *For each* loop, the current value. This only returns numbers or strings, or booleans as a number (0 for false and 1 for true). All other types will return 0.

Get

Get the value at a given path. The path can be relative to the current path or the current key in a *For each* loop. This only returns numbers or strings, or booleans as a number (0 for false and 1 for true). All other types will return 0.

Type

Get a string representing the type of a value at a given path, which can be one of `"null"`, `"array"`, `"object"`, `"boolean"`, `"number"` or `"string"`. The path can be relative to the current path or the current key in a *For each* loop.

Path

Return the current path.

ToBeautifiedString

ToCompactString

Return the current JSON data either as a formatted string with line breaks and indentation ("beautified"), or as a minimal string excluding any line breaks or indentation ("compact"). Beautified strings are easier to read, but compact strings are more efficient for storing, sending over the Internet, and loading. Both beautified and compact strings always represent identical data, and there are a range of third-party tools available that can convert between beautified and compact representation.

GetAsBeautifiedString

GetAsCompactString

Return the JSON data at a specified location either as a formatted string with line breaks and indentation ("beautified"), or as a minimal string excluding any line breaks or indentation ("compact"). These expressions are conceptually similar to "ToBeautifiedString" and "ToCompactString" respectively, but for a specific part of the current data instead of everything. In that way they are the opposite half of the "Set JSON" action, which allow you to set a value from a JSON string at a given location.