

# DICTIONARY

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

---

The **Dictionary** object associates keys with values. Keys are string, and their associated value can be a number or a string. It is a data storage object - it does not do any spell checking or language-specific features.

Key names in the Dictionary object are always case sensitive. This means the key "SCORE" is considered different to the key "score".

## Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the **DictionaryInstance script interface**. (JavaScript and TypeScript have built-in support for **Map** which is a similar data structure, but this allows interacting with data used in event sheets.)

## Example

Suppose the number 100 is stored with the key "score", and the string "Joe" stored with the key "name". The result storage looks like the following table:

Key	Value
name	Joe
score	100

Retrieving the key "name" with `Dictionary.Get("name")` returns "Joe", and retrieving "score" likewise returns 100. Setting "score" to 50 will change the value for the key.

This is like storing data in instance variables or event variables, but since you can use strings as keys you can store any number of values.

Dictionaries are very efficient at retrieving values. Even if you have a dictionary with thousands of keys, it is still very fast to read a value. Arrays are typically much slower to search through (e.g. using the *IndexOf* expression), since they must scan through the entire array to locate elements.

## Designing dictionaries

You can use Construct's **Dictionary Editor** Paid plans only to set the initial contents of a dictionary. You can create a new dictionary data file as a **project file** from the **Project Bar**. At runtime you can load the project file with the **AJAX** object and use the Dictionary's *Load* action to read the data file from the AJAX's *LastData* expression.

## Dictionary conditions

---

### Compare value

Compare the value stored for a key.

---

### Has key

Check if a key exists in storage.

---

### Is empty

True when there are no keys in storage.

---

### For each key

Repeat the event once for each key in storage. The *CurrentKey* and *CurrentValue* expressions return the current key and its value respectively.

---

### Compare current value

Only valid in a *For each key* event. Compare the value of the current key.

---

## Dictionary actions

---

### Add key

Add a new key to storage, with a given value. If the key already exists, its value is updated.

---

### Clear

Remove all keys from storage, making the object empty.

---

### Delete key

Remove a key and its value from storage. If the key does not exist, this has no effect.

---

### Set key

Update the value for a key which already exists. If the key does not exist, this has no effect. (Unlike *Add key*, the key will not be created.)

---

### Download

Invokes a browser download of a file containing the Dictionary's contents in JSON format.

---

### Load

Load all keys and values from JSON data previously retrieved from the Dictionary object using either the *Download* action, the *AsJSON* expression, or the AJAX object loading a project file.

*Note this does not allow loading arbitrary JSON data - the data must be in a special format for Construct. If you want to load data from a project file, create it using the New - Dictionary menu option in the Project Bar.*

## Dictionary expressions

---

### Get(key)

Return the value stored for a key, e.g. `Dictionary.Get("score")`. If the key does not exist, it returns 0.

---

### GetDefault(key, valueIfMissing)

Return the value stored for a key, but if it is missing, return a different value instead. For example `Dictionary.GetDefault("name", "guest")` will return the value of the key "name" if it exists, otherwise it will return the string "guest".

---

### KeyCount

Return the number of keys in storage.

---

### CurrentKey

### CurrentValue

In a *For each key* event, these return the key and its value (respectively) for the current key being iterated.

---

### AsJSON

Return the contents of the Dictionary object in JSON format. This can be later loaded back with the *Load* action, sent to a server via AJAX, saved to disk, and so on.