LOCAL STORAGE

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

The **Local Storage** plugin can **store data locally** on the user's device. For example it can be used to store a personal best high-score. Note if you want a full-state save and load feature, see How to make savegames.

Local Storage works offline, since it stores data to the device itself. Different browsers use different storages, so data is not shared between different browsers on the same computer - each browser has its own unique storage. This is also separate to the browser cache, which is temporary storage to avoid needlessly re-downloading the same files over and over again. However the user can usually still choose to clear local storage data from their browser (with an option perhaps named something like "clear offline website data"). Non-browser export options like desktop exports and mobile apps are not affected by the user clearing any amount of data from any browser, they also use separate storage. Finally for security reasons browsers use separate storage per domain. For example all content on *construct.net* shares the same storage, but content on *facebook.com* uses a different storage and cannot access any data saved from *construct.net*.

Scripting

When using JavaScript or TypeScript coding, you can use the IStorage interface to access the same storage as Construct uses for this object (there is no dedicated script interface for the Local Storage plugin itself). Further, you can use the browser built-in storage APIs such as IndexedDB for more advanced cases.

Storage location on disk

The Local Storage plugin stores all saved data in an internal browser database. This does not produce any easily discoverable files on disk. If you want to save data to easily identifiable files on disk, such as storing save data in a file in the user's Documents folder, consider using the File System plugin instead.

Storage quotas

To prevent abuse, most browsers implement a storage quota, which is a maximum amount of data that can be saved locally. On most modern browsers this is defined as a proportion of the free storage space on the device. You can check the available quota on a device by loading Construct and checking the About dialog which shows the quota available. If the quota is exceeded, the *On error* trigger will fire when writing to storage.

Using Local Storage

Local Storage uses a very simple storage system: values are stored under named *keys*, similar to how the Dictionary object works. For example the value 100 could be stored under a key named *score*.

Local Storage is asynchronous. This means reading and writing data does not complete immediately. The actions only start the process of reading or writing a value, and the project continues to run in the interim. This ensures that slow or busy storage systems do not impact the performance of the project. When the read or write is complete, a trigger fires (*On item get* or *On item set*) which indicates either the value is available to read (with the *ItemValue* expression) or that the value was successfully written.

For example here is a flow to read the value of the key "score":

1. Use the action *Get item "score"*
2. A moment later, *On item "score" get* triggers
3. In this trigger, use the *ItemValue* expression to read the item

Here is a flow to set the new value of "score":

1. Use the action *Set item "score" to 100*
2. A moment later, *On item "score" set* triggers
3. You may not need to do anything in this trigger, but it indicates the value has been successfully written. The *Key* and *ItemValue* expressions are still set in this trigger in case you need them.

Note that you must be careful to avoid "races" when using asynchronous storage. For example the *Clear storage* action may take a moment to complete before it fires *On storage cleared*. It is possible to write more values to storage in between, while *Clear storage* is still processing. This is like "racing" the *Clear storage* and *Set item* actions: the end result depends on what order they complete in, which is unpredictable. In this case it is more or less random what happens: the written keys may be cleared, or they may not be - you cannot rely on any specific result. Therefore you should be careful to avoid this case.

Simplifying usage with Dictionary

Although it improves performance, dealing with asynchronous reads and writes can sometimes be difficult. One simple way to conveniently have synchronous storage is to store an entire Dictionary object's contents to Local Storage, by saving its *AsJSON* string. Then you can load this content from Local Storage with the *Load* action. This means only saving and loading the dictionary contents is asynchronous, and the rest of the time you can use the Dictionary object's features to synchronously access data, such as simply using its *Get* expression to immediately read a value. However you must remember to save the Dictionary again at some point before the user quits the project.

Redirecting storage

If you have a complex existing project and you decide you want to change how data is saved - such as by saving it to a file with the File System plugin, or using a cloud service - updating all usages of the Local Storage plugin can be tricky. To help with this you can enable the *In memory only* property of the object. This essentially disables the use of storage and only holds data in memory, and the data will be lost when reloading the page or app. However the data can then be accessed using the *MemoryStorageAsJSON* expression for saving elsewhere, and loaded with the *Load memory from JSON* action, to persist storage somewhere else. Note that if you use this to save to a cloud service, you may want to check the size of the saved data - usually it is not a problem to save a large amount of data locally, but it may turn out to be too much to upload and download.

Local Storage properties

In-memory only

Check to enable in-memory only storage mode. In this mode data will not be persisted if the page or app is closed, but can be stored elsewhere instead - see *Redirecting storage* above for more details.

Local Storage conditions

On any item get

Triggered after any *Get item* action completes.

On any item removed

Triggered after any *Remove item* action completes.

On any item set

Triggered after any *Set item* action completes.

On item exists

Triggered after the *Check item exists* action completes if the key checked does indeed exist. In this trigger the *ItemValue* expression is also set to the value of this key, so there is no need to use another *Get item* action to read it.

> *Note ItemValue is not set if binary data was stored.*

On item get

Triggered after a *Get item* action completes for a given key. The *ItemValue* expression is set to the value of the key, except for when reading binary data.

On item missing

Triggered after the *Check item exists* action completes if the key checked does not exist.

On item removed

Triggered after the *Remove item* action completes for a given key.

On item set

Triggered after the *Set item* action completes for a given key. This indicates the data is now in storage.

Compare key

Compare the current value of the *Key* expression, which is the name of the current key in a trigger. This can be useful in the *On any item...* triggers.

Compare value

Compare the current value of the *ItemValue* expression, which is set to the item value when getting an item or in *On item exists*.

Is persistent

True if the browser has granted the current domain persistent storage permission. See the *Request persistent storage* action for more details.

On all key names loaded

Triggered after the *Get all key names* action completes. In this trigger the *KeyCount* and *KeyAt* expressions give the list of all the key names.

On error

Triggered at any time while using Local Storage if an error occurs, such as if a write failed, or the maximum storage quota was exceeded. The *ErrorMessage* expression is set to the error message if available.

On storage cleared

Triggered after the *Clear storage* action completes and storage is now empty.

Is processing gets

True if any *Get item* actions are still processing, i.e. any *On item get* trigger is yet to fire for a *Get item* action.

Is processing sets

True if any *Set item* actions are still processing, i.e. any *On item set* trigger is yet to fire for a *Set item* action.

On all gets complete

Triggered when all outstanding *Get item* actions are completed, i.e. when *Is processing gets* first becomes false. In other words if 10 *Get item* actions are all used at the same time, *On all gets complete* triggers when all 10 items have been read and fired their *On item get* triggers.

On all sets complete

Triggered when all outstanding *Set item* actions are completed, i.e. when *Is processing sets* first becomes false. In other words if 10 *Set item* actions are all used at the same time, *On all sets complete* triggers when all 10 items have been written and fired their *On item set* triggers.

Local Storage actions

Check item exists

Check if a key exists in storage. This triggers either *On item exists* if the key exists, or *On item missing* if the key does not exist. If the item exists, the *ItemValue* expression is set to the key value in the *On item exists* trigger, so there is no need to use a subsequent *Get item* action to read the value.

> *Note ItemValue is not set if binary data was stored.*

Get item

Get binary item

Read the value of a key in storage. This triggers *On item get* when the value has been read. When reading binary data, the data will be written to the chosen Binary Data object; otherwise the *ItemValue* expression is set to the value of the key.

Remove item

Remove (delete) a key from storage. This triggers *On item removed* when the key has been removed.

Set item

Set binary item

Set the value of a key in storage. This triggers *On item set* when the value has been written. When setting binary data, the contents of a Binary Data object are written; otherwise the text or number provided is used.

Clear storage

Remove (delete) all items from storage, reverting it back to the empty state. This triggers *On storage cleared* when completed.

Get all key names

Retrieve a list of all the key names that currently exist in storage. This triggers *On all key names loaded* when the list has been loaded, where the *KeyCount* and *KeyAt* expressions can be used to access the list.

Request persistent storage

Request that storage be made persistent for the current domain. This typically only applies to web browsers, as all other export options already use persistent storage. In the context of a web browser, all storage is preserved on a "best effort" basis, but may be erased in various situations such as if too much storage space is being used, if the site has not been visited for too long, and so on. If persistent storage permission is granted, then the browser will avoid automatically deleting the storage wherever possible; it may also avoid deleting the storage even if the user chooses to clear storage in browser settings, unless they specifically acknowledge they want to delete the site's data. Requesting persistent storage may show a visible prompt to the user asking them to accept permission, and this action may only be allowed to be used in a user input trigger (such as mouse click or touch). The action is asynchronous, meaning it can be used with *Wait for previous actions to complete*, after which the permission will have been granted or refused. The *Is persistent* condition reflects the current persistent permission state.

Load memory from JSON

Only applicable when the property *In-memory only* is enabled. Load the entire state of the storage - including all keys and values - from a string of JSON data previously returned by the *MemoryStorageAsJSON* expression. For more details on usage, see *Redirecting storage* above.

Local Storage expressions

ItemValue

The value for a key that has been read or written in a Local Storage trigger, such as *On item get* or *On item exists*. This can be either a string or a number. This returns 0 if used outside of a Local Storage trigger, or if binary data was stored instead of text or a number.

Key

The name of the key that was modified in any Local Storage trigger, such as *On item get*, *On item set* or *On any item set*. This returns an empty string if used outside of a Local Storage trigger.

ErrorMessage

In *On error*, the text of the error message if any is available.

KeyAt(index)

In *On all key names loaded*, the name of the key at the given zero-based index in the list.

KeyCount

In *On all key names loaded*, the number of key names in the list.

MemoryStorageAsJSON

Only applicable when the property *In-memory only* is enabled. Returns the entire state of the storage - including all keys and values - as a string of JSON data. This can then be loaded again later with the *Load memory from JSON* action. For more details on usage, see *Redirecting storage* above.