TilBuci

Scripting Actions

All user interaction on TilBuci is managed by JSON-formatted texts with instructions about how to handle both input and playback flow. This document describes this format with all available instructions.

Lucas Junqueira, 2026/02

Contents

1. Scripting actions on TilBuci

You may create beautiful scenes and animations on TilBuci but they won’t be really useful if you don’t provide user interaction to them. A good interaction design is fundamental for your content to get life!

To enable these interactions, TilBuci uses a JSON-formatted text describing the actions. If you are not familiar with the JSON concept, is it a data format based on JavaScript. You don’t really need to go too deep on JSON to use it on TilBuci, but if you are interested on it, please check out json.org for further information.

The TilBuci actions are always provided as an *object* with at least two fields: *ac*, a simple string, and *param*, an array of strings (all parameters must be provided as strings, even numeric ones like “3.1416”). If an action does not require parameters, the *param* value must still be given as an empty array.

```
{
    “ac”: “actionhere”,
    “param”: [ “param1”, “param2”, “param3” ]
}
```

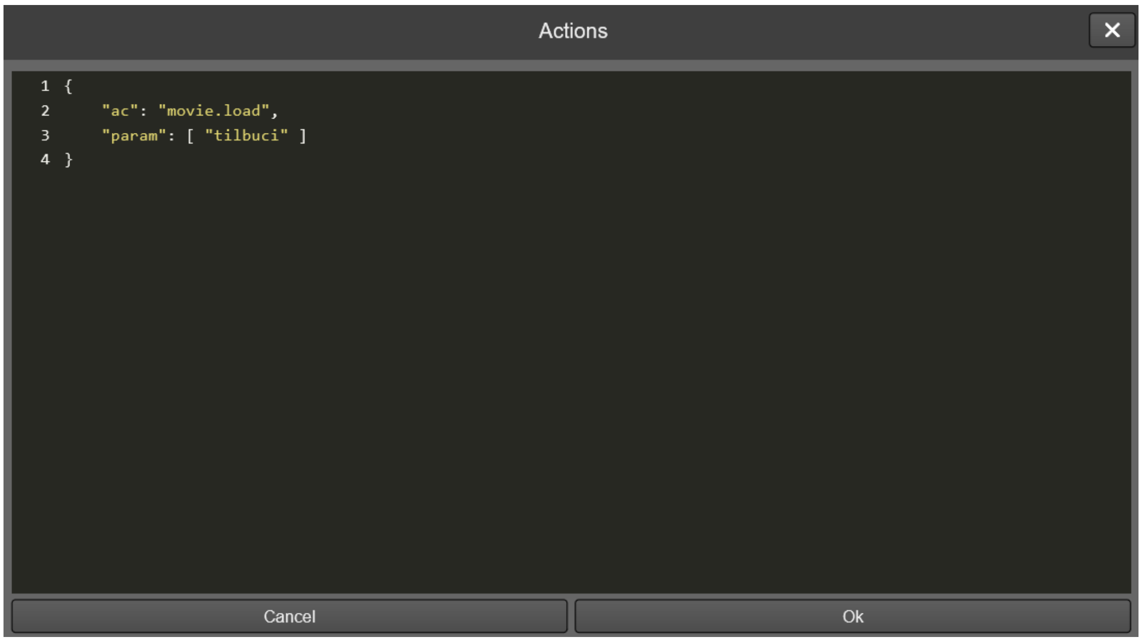
Actions may also be provided as an array of objects. You can use these arrays on every place that accepts a single action.

```
[
    { “ac”: “firstaction”, “param”: [ “param1” ] },
    { “ac”: “secondaction”, “param”: [ ]
]
```

There are some actions, like conditionals, timers and inputs, that may accept additional fields. Check out the following example:

```
{
    “ac”: “if.stringsequal”,
    “param”: [ “$okemail”, “name@email.com” ],
    “then”: {
        “ac”: “timer.set”,
        “param”: [ “displaytimer”, “1000”, “1” ],
        “tick”: [ ],
        “end”: { “ac”: “scene.load”, “param”: [ “userscene” ] }
    },
    “else”: [
        { “ac”: “timer.clearall”, “param”: [ ] },
        {
            “ac”: “input.login”,
            “param”: [ ],
            “ok”: {
                “ac”: “string.set”,
                "param": [ "okemail", "$_USERNAME" ]
            },
            "cancel": {
                "ac": "string.clear",
                "param": [ "okemail" ]
            }
        }
    ]
}
```

You can input your action script on its window at the editor. By clicking on *ok* your JSON text will be evaluated for errors (notice that this will only look for text inconsistencies, not your script logic.



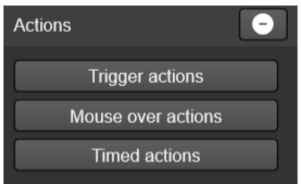
```
{
    "ac": "movie.load",
    "param": [ "tilbuci" ]
}
```

1.1. Where do I use these interaction scripts?

You can provide action scripts on many places. The usual one is on an user click/touch of an instance, but there are many more.

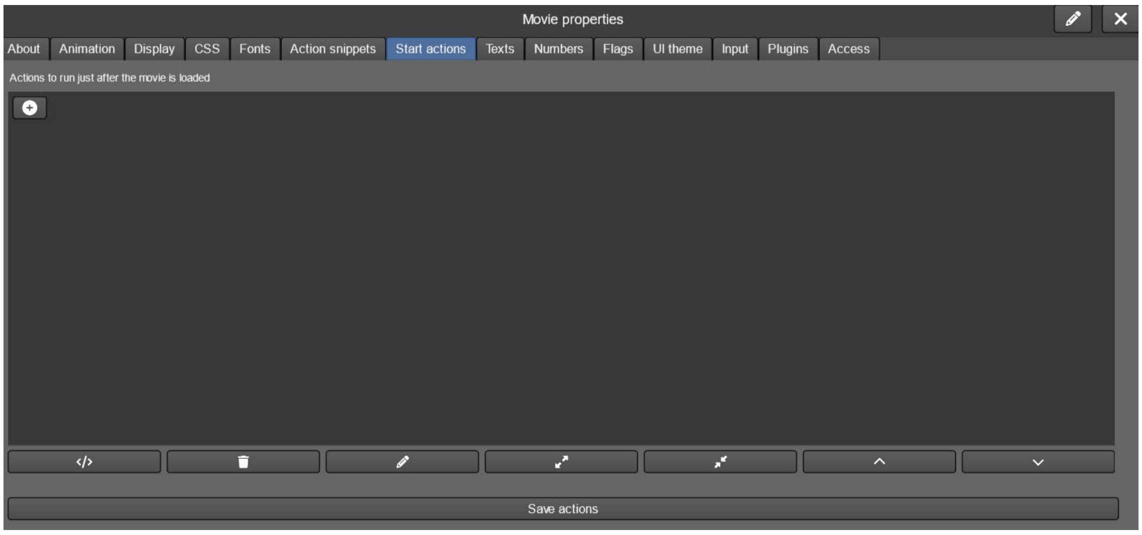
1.1.1. Instance interactions

This is the usual place to set an action script. Just select an instance on the stage and look for the *Actions* tab on the right menu. You can set actions for three different moments: trigger (when the instance is clicked/tapped), when the mouse cursor hovers it or you can use the instance playback to create timed actions.

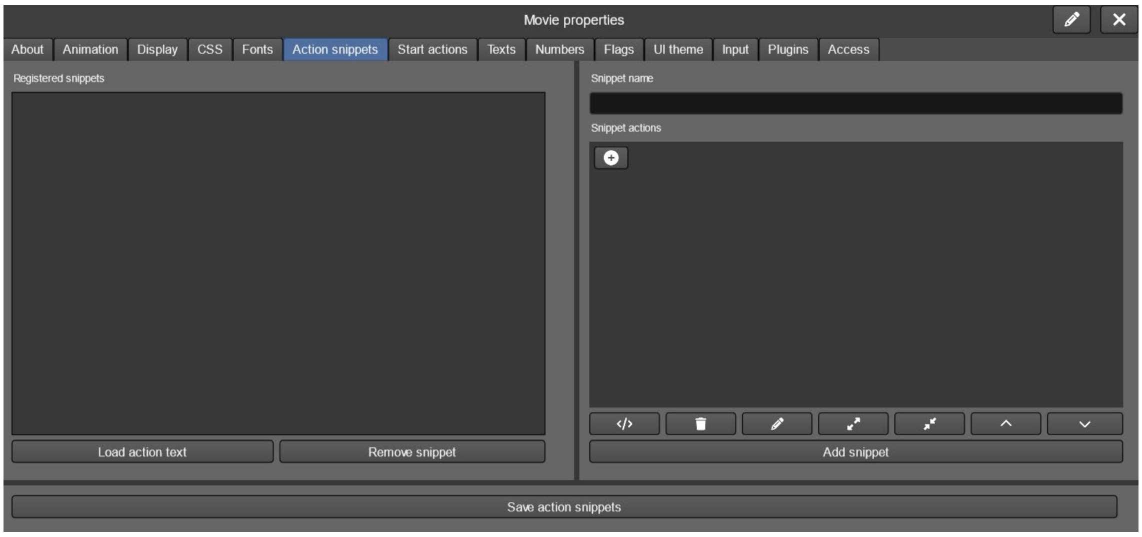


1.1.2. Movie load

You can provide actions to run just after your movie is loaded. Look for the *Movie* > *Properties* right menu and access the *Start actions* tab.

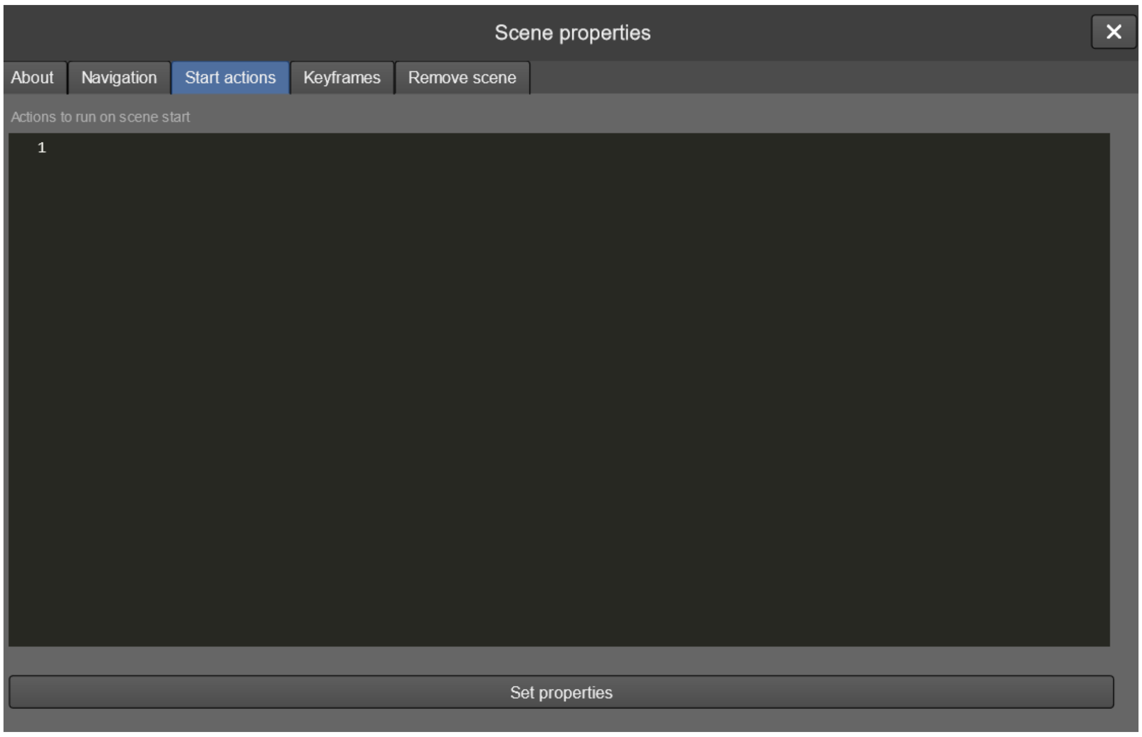


The movie properties window also gives you access to the action snippets. These are groups of actions that can be easily reused on your entire movie by calling the *run* command.



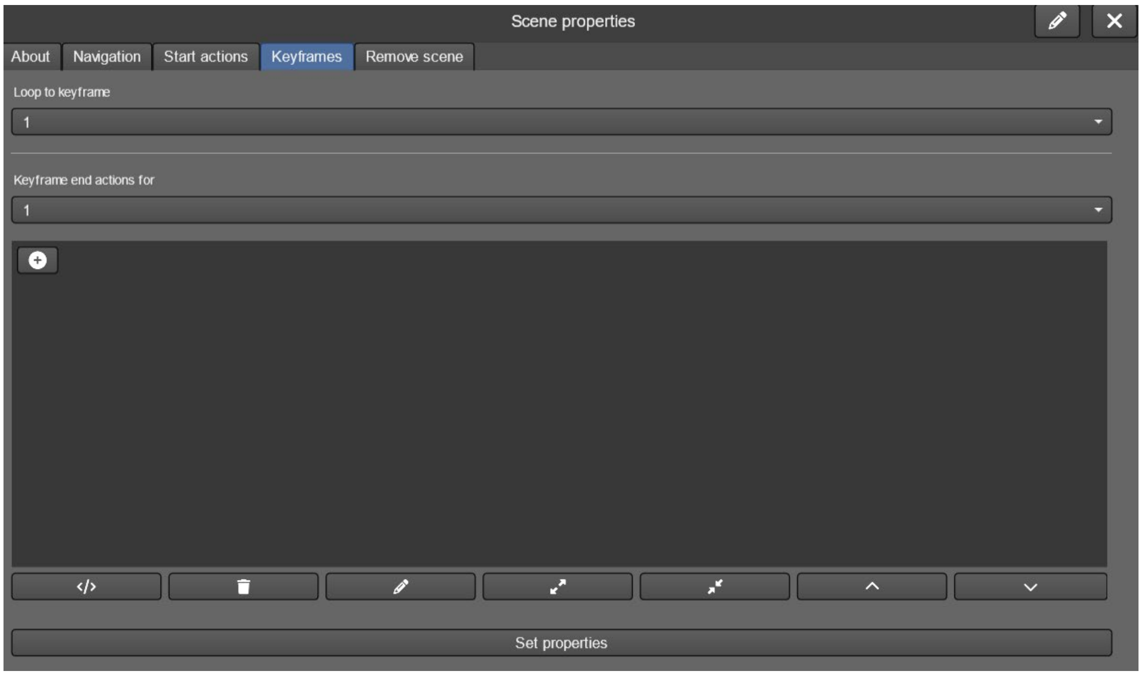
1.1.3. Scene start

You can set actions to run when a scene starts playing. Check out the *Start actions* tab from the *Scene > Properties* menu.



1.1.4. Keyframe end

Scenes can contain multiple keyframes. You may set actions to run at the end of each of them. You can set these actions either from the *Keyframes* tab of the *Scene > Properties* menu or the *Keyframes > Manage* one.

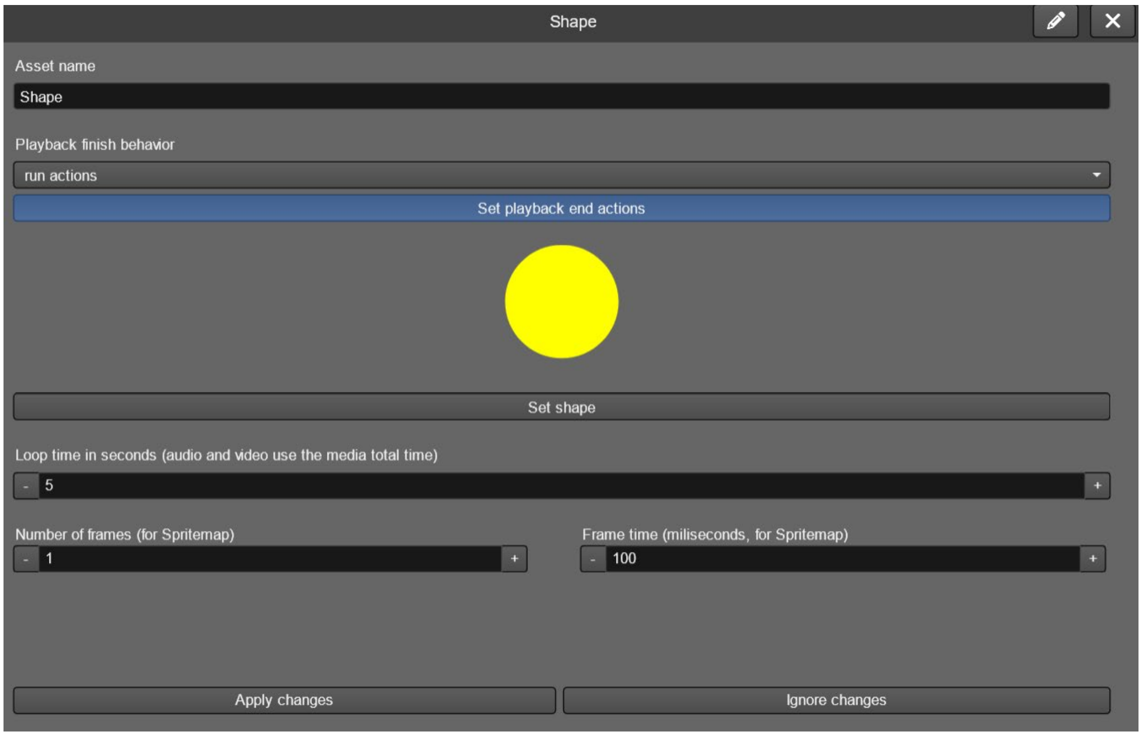


Loop to keyframe: 1

Keyframe end actions for: 1

1.1.5. Media playback end

Actions triggered when a media, like a movie, finishes playing.



Asset name: Shape

Playback finish behavior: run actions

Loop time in seconds (audio and video use the media total time): 5

Number of frames (for Spritemap): 1

Frame time (miliseconds, for Spritemap): 100

1.2. Variables

You can use variables on TilBuci actions to help design your interactions. Four variable types are available:

- Boolean
- Float
- Integer
- String

There are many actions that set the values for each of these types. The basic ones are *bool.set*, *int.set*, *float.set* and *string.set*. Every time a variable is set you can use it on any action that requires parameters of that type by using the variable name preceded by the appropriate symbol: *?* for Boolean, *#* for integer or float and *$* for string. Here are some examples:

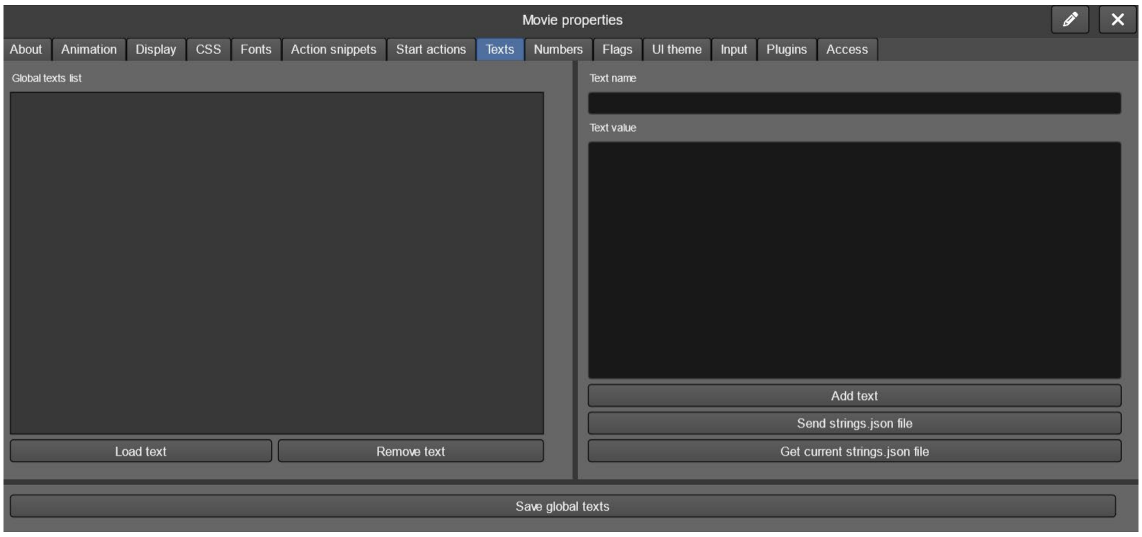
```
[
    { “ac”: “bool.set”, “param”: [ “boolvar1”, “true” ] },
    {
        “ac”: “if.bool”,
        “param”: [ “?boolvar1” ],
        “then”: { “ac”: “scene.load”, “param”: [ “tilbuci” ] },
        “else”: [ ]
    },
    { “ac”: “int.set”, “param”: [ “intvar1”, “10” ] },
    {
        “ac”: “int.sum”,
        “param”: [ “intvar2”, “#intvar1”, “20” ]
    },
    { “ac”: “float.set”, “param”: [ “floatvar1”, “10.7” ] },
    { “ac”: “float.set”, “param”: [ “floatvar2”, “10.5” ] },
    {
        “ac”: “float.min”,
        “param”: [ “floatvar3”, “#floatvar1”, “#floatvar2” ]
    },
    { “ac”: “string.set”, “param”: [ “stringvar1”, “Til” ] },
    {
        “ac”: “string.concat”,
        “param”: [ “stringvar2”, “$stringvar1”, “Buci” ]
    }
]
```

When you use the # mark TilBuci will look for a float variable first and, if not found, look for an integer one. Every time a float is used on integer operations it will be rounded.

1.3. Globals

Besides the variables you can set on your own, TilBuci comes with several global values you can use on your actions to retrieve all sorts of information from the playback. You can use them just like the variables.

Some of these globals are always available, but you can also set your own per movie. Access the left menu *Movie* > *Properties* and look for the *Texts*, *Numbers* and *Flags* tabs.



To access these globals you must use *$_TEXTS:*, *#_NUMBERS:* or *?_FLAGS:*, according to the type. *#_NUMBERS:* can be used as both integer and float values – it will be rounded for integers. Place the global name you set after the *:* on your scripts, like this example:

```
{
    “ac”: “string.set”,
    “param”: [ “stringvar1”, “$_TEXTS:my global” ]
}
```

Another group of globals that use *:* to retrieve information are the ones about instance properties. Place the instance id after the *:* to retrieve the correct value like:

```
{
    “ac”: “int.set”,
    “param”: [ “intvar1”, “#_INSTANCEWIDTH:my instance id” ]
}
```

Besides these values, plugins may also define their own globals that add to the standard ones.

1.3.1. Boolean globals

General globals

?_PLAYING	Is the movie playing right now?
?_SERVER	Online server connection active?
?_USERLOGGED	Is there an user logged?

?_HADINTERACTION	Has the visitor ever interacted?

Instance globals

?_INSTANCEPLAYING	playing content?
?_INSTANCEVISIBLE	instance visible?
?_INSTANCEFONTBOLD	text set to bold?
?_INSTANCEFONTITALIC	text set to italic?

1.3.2. Number globals

General globals

#_KEYFRAME	Current keyframe number (starts on 0).
#_AREABIG	Movie “big” dimension.
#_AREASMALL	Movie “small” dimension.
#_CONTENTX	Movie display actual X position
#_CONTENTY	Movie display actual Y position
#_CONTENTWIDTH	Movie display actual width
#_CONTENTHEIGHT	Movie display actual height

Instance globals

#_INSTANCEX	x position
#_INSTANCEY	y position
#_INSTANCEWIDTH	width
#_INSTANCEHEIGHT	height
#_INSTANCEALPHA	alpha value (0 to 1.0)
#_INSTANCEVOLUME	sound volume
#_INSTANCEORDER	order index (starts on 0)
#_INSTANCECOLORALPHA	color overlay alpha (0 to 1.0)
#_INSTANCEROTATION	rotation (0 to 359, experimental)
#_INSTANCEFONTSIZE	text font size
#_INSTANCEFONTLEADING	text leading

1.3.3. Text globals

General globals

$_MOVIETITLE	Current movie title
$_MOVIEID	Current movie id
$_SCENETITLE	Current scene title
$_SCENEID	Current scene id
$_ORIENTATION	Player orientation (horizontal or vertical)
$_RENDER	Current display render mode (webgl or dom)
$_RUNTIME	The current runtime (see below for datails)
$_URLMOVIE	Url to accesse current movie directly
$_URLSCENE	Url to access current movie+scene directly
$_USERNAME	Logged user name (e-mail)
$_VERSION	Current TilBuci player version
$_WSSERVER	URL for the current webservice requests
$_SESSION	Current player session ID
$_YEAR	Current year as YYYY
$_MONTH	Current month as MM
$_DAY	Current day as DD
$_HOUR	Current hour as HH (24h)

$_MINUTE	Current minute as MM
$_SECOND	Current seconds as SS
$_DATE	Current date as YYYY-MM-DD
$_TIME	Current time as HH:MM:SS (24h)

The Tilbuci content can be played on different runtimes. If necessary, the *$_RUNTIME* global can be used to find out the current one. The possible returns are:

website	Running from an exported website
pwa	Running as a pwa application
desktop	Running from a desktop application (Windows, Linux or macOS)
mobile	Running from a mobile app (Android or iOS/iPadOS)
publish	Running from a publish service like itch.io
embed	Running form a TilBuci player embed on an OpenFL project
tilbuci	Running from the TilBuci installation (default)

Instance globals

$_INSTANCECOLOR	Overlay color (0x###### hex format)
$_INSTANCETEXT	Current text set to instance
$_INSTANCEFONT	Text font name
$_INSTANCEFONTCOLOR	Text font color (0x###### hex format)

1.3.4. Input values

You can create input elements using actions. Every input has an unique name that can be used to retrieve its current value, just inform the given name after the : like "$_INPUT:name". The possible values are:

$_INPUT	text input content
$_TAREA	text area content
#_NUMERIC	numeric stepper value
?_TOGGLE	toggle input value

1.3.5. Form values

While using the form contraption, you can use these values to retrieve the data provided by the visitor using the element name, like *$_FORM: name*

All information is retrieved as string, even from numeric or toggle inputs. You can use TilBuci's convert actions if needed.

1.4. Strings.json file

Another way to work with strings is the usage of a *strings.json* file. You may upload this file containing pairs of string variable names and values to simplify your creation. At this file, the string variables must be grouped so you can easily change their value from an user interaction (like changing language settings) – you can use any name you want for the groups. A simple *strings.json* file content will look like this:

```
{
    “group1”: {
        “string1”: “site name: ”,
        “string2”: “site: ”
    },
    “group2”: {
        “string1”: “nome da página: ”,
        “string2”: “página: ”
    },

}
```

Before using the *strings.json* content you must set the values group with the *string.setgroup* action, like this:

```
[
    { “ac”: “string.setgroup”, “param”: [ “group1” ] },
    {
        “ac”: “string.concat”,
        “param”: [ “stringvar1”, “$string1”, “TilBuci” ]
    }
]
```

The script above will set the value of *$stringvar1* to “site name: TilBuci”. However, if you set another group, the results are different:

```
[
    { “ac”: “string.setgroup”, “param”: [ “group2” ] },
    {
        “ac”: “string.concat”,
        “param”: [ “stringvar1”, “$string1”, “TilBuci” ]
    }
]
```

In that case, the *$stringvar1* value will be “nome da página: TilBuci”.

1.5. Instance text and replace

When you set a string variable (by using an action or with the *strings.json* file) you can also retrieve its value on instance text. When you create a paragraph image, just use the variable name (and nothing more) instead of directly input the text. It will be automatically replaced by the current value of the variable when it shows up.

1.5.1. Content replace at runtime

Besides setting a single variable to an instance, you may also set values to be automatically replaced at runtime in both instance display and action operations. There are two actions to set these replacements.

First, *replace.setstring*: using this action you will set string parts that will always replaced by something else, on paragraph and html instances and also on usual string operations.

```
[
    { “ac”: “replace.setstring”, “param”: [ “[MAIL]”, “$_USERNAME” ] },
    { “ac”: “string.set”, “param”: [ “stringvar1”, “my email is [MAIL]” ] }
]
```

If there is an user logged with the e-mail “doggo@tilbuci.com.br”, the *$stringvar1* variable will receive the value “my email is doggo@tlbuci.com.br”. Also, if you set an html instance content to a file with “<p>the e-mail address is [MAIL]<p>”, it will be shown as “the e-mail address is doggo@tlbuci.com.br” as well.

The second automatic replace you can set affects file names used by audio, html, picture and video instances. If you call *repace.setfile*, every time TilBuci attempts to load a file, it will look for the needle text on the name and replace it. This can be very useful to load different assets based on user language choices. Check out this example: if you are using a picture named “en/title.png” on your movie, after you call the following action, TilBuci will load “pt/title.png” instead.

```
{ “ac”: “replace.setfile”, “param”: [ “en/”, “pt/” ] }
```

2. Available actions

Here we present a list of currently available actions on TilBuci (*ac* field at the action object). Please note that plugins may add additional commands to this list. Some of these actions require parameters, but even if none is needed you still must add the *param* field to the object. Additional optional fields can be used on some actions, like *then*, *else*, *ok*, *cancel*, *tick*, *end*, *success* and *error*.

2.1. Boolean conditions

Conditional statements based on boolean values.

if.bool

Checks if the boolean value is true.

params
1. the boolean variable to check

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.boolset

Checks if a boolean variable exists.

params
1. the variable name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

2.2. Boolean values

Management of boolean variables.

bool.clear

Removes a boolean variable.

params
1. the variable name

bool.clearall

Removes all boolean variables.

bool.set

Sets a boolean variable.

params
1. the variable name
2. the variable value

bool.setinverse

Inverts the value of a boolean variable.

params
1. the variable name

2.3. Data

These actions handle user persistent data management. This can be done both locally on the browser storage of remotely using your server. The user must be logged in to access remote data. Logged user can still access local data. TilBuci uses a dark default theme for the load state UI, but you can adjust the colors according to your content at the *UI colors* tab of the *Movie properties* window.

data.event

Registers an event on TilBuci database at the “events” table. You may use events to record visitor interactions and keep track of your movies’ access. While the Google Analytics plugin offers a more sophisticated approach, the *data.event* action is easier to use and keeps the data in your own server.

All events must receive a name, and you can add as many information you want providing additional string parameters to the action call. All events are recorded with the current movie, scene and visitor information. Access the Visitor button on the left menu and go to the Event tab to download the recorded information for analysis.

If TilBuci finds an error while sending the event to the server, such as an Internet connection failure, the data is recorded locally and sent together with the next event sending action. TilBuci holds up to 100 unsent events for future sending. When exporting the events spreadsheet from the “Visitors” left menu on TilBuci editor, both the received date/time and the original one are included.

params
1. the event name

data.eventclear

Clears the unsent events pool for the current movie.

data.liststates

Shows the load state window so the user can choose the one to load from the last 3 saved states on server. When loaded, the current variable values are replaced by the saved ones and the scene playing while the state was saved is loaded. All interface text must be set as text globals at the *Text* tab of the *Movie properties* window – if not set, default English strings are used. The text names you must set to avoid using default values are shown on table below. Global texts can be changed at runtime by using the *string.setglobal* action.

Global	**Description**	**Default value**
titlestates	Window title	Select a state to load
waistates	States load wait message	Please wait while a list of available saves states is loaded.
selectstate	Selects state message	Select a state to load.
nostates	No save states found message	Sorry, no saved states found to load.
dateformat	Date/time format string	Y-m-d h:i a (use the PHP format described here: https://www.php.net/manual/en/datetime.format.php)

additional optional object fields
- success: actions to run after successful data load
- error: actions to run after error on data load

data.load

Loads custom data saved at the server. The loaded values replaces the variables used on *data.save* action.

params
1. data save name

additional optional object fields
- success: actions to run after successful data load
- error: actions to run after error on data load

data.loadlocal

Loads custom data saved at the browser storage. The loaded values replaces the variables used on *data.savelocal* action.

params
1. data save name

additional optional object fields
- success: actions to run after successful data load
- error: actions to run after error on data load

data.loadquickstate

Loads the quick state saved on the server. When loaded, the current variable values are replaced by the saved ones and the scene playing while the state was saved is loaded.

additional optional object fields

- success: actions to run after successful data load
- error: actions to run after error on data load

data.loadstatelocal

Loads the state saved on browser storage. When loaded, the current variable values are replaced by the saved ones and the scene playing while the state was saved is loaded.

additional optional object fields

- success: actions to run after successful data load
- error: actions to run after error on data load

data.save

Saves custom data at the server. At least two parameters are required, but you can add as many as you want to be saved under the same name. Each saved value must be the name of an already set variable, preceded by its type, like in the following table.

Variable type	Prefix	Example
boolean	B:	B:boolvar1
float	F:	F:floatvar1
integer	I:	I:intvar1
string	S:	S:stringvar1

params

1. data save name
2. first variable to save

additional optional object fields

- success: actions to run after successful data save
- error: actions to run after error on data save

data.savelocal

Saves custom data at the browser storage. At least two parameters are required, but you can add as many as you want to be saved under the same name. Each saved value must be the name of an already set variable, preceded by its type, like in the following table.

Variable type	Prefix	Example
boolean	B:	B:boolvar1
float	F:	F:floatvar1
integer	I:	I:intvar1
string	S:	S:stringvar1

params

1. data save name
2. first variable to save

additional optional object fields

- success: actions to run after successful data save
- error: actions to run after error on data save

data.savequickstate

Quickly saves all current variable values as well the current scene playing on the server. Saving a quick save overrides the previously saved one for current user/movie.

additional optional object fields

- success: actions to run after successful data save
- error: actions to run after error on data save

data.savestate

Saves all current variable values as well the current scene playing on the server. The server holds up to 3 save states for each user on each movie, besides the *quick save*.

params

1. state title (for reference while loading)

additional optional object fields

- success: actions to run after successful data save
- error: actions to run after error on data save

data.savestatelocal

Saves all current variable values as well the current scene playing on the browser storage. The browser can hold only one state per movie.

additional optional object fields

- success: actions to run after successful data save
- error: actions to run after error on data save

2.4. Float conditions

Conditional statements based on float values.

if.floatsdifferent

Checks if the two float values are different.

params

1. first value
2. second value

additional optional object fields

- then: actions to run if the result is true
- else: actions to run if the result is false

if.floatsequal

Checks if the two float values are equal.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.floatset

Checks if a float variable exists.

params
1. the variable name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

if.floatgreater

Checks if the first value is greater than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.floatgreaterequal

Checks if the first value is greater or equal than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.floatlower

Checks if the first value is lower than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.floatlowerequal

Checks if the first value is lower or equal than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

2.5. Float values

Management of float variables.

float.abs

Returns the absolue value of a float.

params
1. the variable name to receive the result
2. the value

float.clear

Removes a float variable.

params
1. the variable name

float.clearall

Removes all float variables.

float.divide

Divides two or more float values. At least 3 parameters are required but you can provide as many as you want and their values are sequentially divided.

params
1. the variable name to receive the result
2. the first value
3. the second value

float.max

Returns the maximum between two float values.

params
1. the variable name to receive the result
2. the first value
3. the second value

float.min

Returns the minimum between two float values.

params
1. the variable name to receive the result
2. the first value
3. the second value

float.multiply

Multiplies two or more float values. At least 3 parameters are required but you can provide as many as you want and their values are values are sequentially multiplied.

params

1. the variable name to receive the result
2. the first value
3. the second value

float.random

Generates a random float value.

params
1. the variable name to receive the rando value
2. minimum value
3. maximum value

float.set

Sets a float variable.

params
1. the variable name
2. the variable value

float.subtract

Subtracts two or more float values. At least 3 parameters are required but you can provide as many as you want and their values are subtracted from the result.

params
1. the variable name to receive the result
2. the first value
3. the second value

float.sum

Sums two or more float values. At least 3 parameters are required but you can provide as many as you want and their values are add to the result.

params

1. the variable name to receive the result
2. the first value
3. the second value

float.toint

Converts a float value to an int one (rounds).

params
1. the int variable name to receive the result
2. the float value

float.tostring

Converts a float value to a string.

params
1. the string variable name to receive the result
2. the float value

2.6. Input

These actions asks for user input to use on the movie. TilBuci uses a dark default theme for input UI, but you can adjust the colors according to your content at the *UI theme* tab of the *Movie properties* window.

Besides the preferred input boxes you may add input elements right into your design using actions. These inputs appear over any graphics on your scene and must also be removed using actions (they do not automatically disappear when a new scene is loaded). The size/place given is relative to your content. These elements also respect the *UI theme* settings of your movie.

input.email

Shows a text input window the ckecks for a valid e-mail address on confirmation. You may also add buttons to include common e-mail service domains like “@gmail.com” – just add additional string parameters.

params
1. string variable name to get the resulting e-mail address
2. text to display on input window
3. common email service domain, like “@hotmail.com” (optional)

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.float

Shows a dialog window asking for a float input.

params
1. float variable name to get the resulting input
2. text to display on input window
3. numeric stepper step value
4. minimum input value
5. maximum input value

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.int

Shows a dialog window asking for an int input.

params
1. int variable name to get the resulting input
2. text to display on input window
3. numeric stepper step value
4. minimum input value
5. maximum input value

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.list

Presents a list to the user asking for a selection. You must provide at least 4 parameters, but you may add as many as you want – they will be add as additional item options. The returned value is the text of the option selected.

params
4. string variable name to get the resulting input
5. text to display on input window
6. first list option
7. second list option

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.login

Starts the user login procedure. Login is done by sending a confirmation code to the provided e-mail address. All interface text must be set as text globals at the *Text* tab of the *Movie properties* window – if not set default English strings are used. The text names you must set to avoid using default values are shown on table below. Global texts can be changed at runtime by using the *string.setglobal* action.

Global	Description	Default value
logintitle	Window title	Login

logintext	Window message	Please type your e-mail address. You'll receive a 6 digit code to confirm your identity.
termsagree	Terms agree message	I agree with the above terms.
invalidemail	Invalid email address input	Please provide a valid e-mail address.
emailwait	Processing message	Please wait while a confirmation code is sent to your e-mail.
noemailsent	Send email error message	Error while sending the code by e-mail. Please try again.
checkforcode	Code check instructions	Please check out your e-mail inbox. Type below the 6 digit code you received. If you do not find the message, take a look at you spam folder.
codewait	Code check processing	Please wait while the provided code is checked.
invalidcode	Invalid code message	The provided code is invalid. Please try again.
emailsubject	E-mail message subject	TilBuci login
emailbody	E-mail message body	Hi, you are receiving this message to confirm your login at TilBuci. Please provide the code below to proceed:\r\n\r\n[CODE]\r\n\r\nIf you did'nt request this code, don't worry: just ignore this message.
emailsender	E-mail sender name	TilBuci

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.message

Shows a message to the user asking for confirmation. If no cancel action is set, the window will only show the ok button.

params
1. window title
2. window text

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.string

Shows a dialog window asking for a string input.

params
1. string variable name to get the resulting input
2. text to display on input window

additional optional object fields
- ok: actions to run after user clicks on ok
- cancel: actions to run after user click on cancel

input.add

Adds a text input element into your screen.

params
1. the input name
2. the x position
3. the y position
4. the width (height is automatically set from the font size of UI theme)
5. (optional) placeholder to show when no text is set

input.place

Sets a text input placement.

params
1. the input name
2. the x position
3. the y position
4. the width (height is automatically set from the font size of UI theme)

input.remove

Removes a text input from screen.

params
1. the input name

input.removeall

Removes all text inputs from screen.

input.settext

Sets the text of a text input element.

params
1. the input name
2. the new text

input.setpassword

Sets a text input password mask display

params
1. the input name
2. show as password? (true/false)

input.addtarea

Adds a text area element into your screen.

params
1. the text area name
2. the x position
3. the y position
4. the width
5. the height

input.placetarea

Sets a text area placement.

params

1. the text area name
2. the x position
3. the y position
4. the width
5. the height

input.removetarea

Removes a text area from screen.

params

1. the text area name

input.removealltareas

Removes all text areas from screen.

input.settextarea

Sets the content of a text area element.

params

2. the text area name
3. the new text

input.addnumeric

Adds a numeric stepper input element into your screen.

params

1. the input name
2. the initial value
3. the minimum value
4. the maximum value
5. the increase/decrease step

input.placenumeric

Sets a numeric stepper input placement.

params

1. the input name
2. the x position
3. the y position
4. the width (height is automatically set from the font size of UI theme)

input.removenumeric

Removes a numeric stepper input from screen.

params

1. the input name

input.removeallnumerics

Removes all numeric stepper inputs from screen.

input.setnumeric

Sets the value of a numeric stepper input element.

params
1. the input name
2. the new value

input.setnumericbounds

Sets a numeric stepper bounds.

params
1. the input name
2. minimum value
3. maximum value
4. increase/decrease step

input.addtoggle

Adds a toggle stepper input element into your screen.

params
1. the input name
2. toggle selected?
3. X position
4. Y position

input.placetoggle

Sets a toggle input placement.

params
1. the input name
2. the x position
3. the y position

input.removetoggle

Removes a toggle input from screen.

params
1. the input name

input.removealltoggles

Removes all toggle inputs from screen.

input.settoggle

Sets the value of a toggle input element.

params
1. the input name
2. the new value

input.inverttoggle

Inverts a toggle input value.

params
1. the input name

2.7. Instance

instance.clearall

Clears any previously set value (instance returns to design setting).

param
1. instance id

instance.clearalpha

Clears previously set alpha value (instance returns to design setting).

param
1. instance id

instance.clearcolor

Clears previously set color overlay value (instance returns to design setting).

param
1. instance id

instance.clearcoloralpha

Clears previously set overlay alpha value (instance returns to design setting).

param
1. instance id

instance.clearfont

Clears previously set font face name value (instance returns to design setting).

param
1. instance id

instance.clearfontalign

Clears previously set text align value (instance returns to design setting).

param
1. instance id

instance.clearfontbackground

Clears previously set font background value (instance returns to design setting).

param
1. instance id

instance.clearfontbold

Clears previously set font bold state value (instance returns to design setting).

param
1. instance id

instance.clearfontcolor

Clears previously set font color value (instance returns to design setting).

param
1. instance id

instance.clearfontitalic

Clears previously set font italic state value (instance returns to design setting).

param
1. instance id

instance.clearfontleading

Clears previously set font leading value (instance returns to design setting).

param
1. instance id

instance.clearfontsize

Clears previously set font size value (instance returns to design setting).

param
1. instance id

instance.clearheight

Clears previously set height value (instance returns to design setting).

param
1. instance id

instance.clearorder

Clears previously set order value (instance returns to design setting).

param
1. instance id

instance.clearrotation

Clears previously set rotation value (instance returns to design setting - experimental).

param
1. instance id

instance.clearvisible

Clears previously set visible state (instance returns to design setting).

param
1. instance id

instance.clearvolume

Clears previously set sound volume value (instance returns to design setting).

param
1. instance id

instance.clearwidth

Clears previously set width value (instance returns to design setting).

param
1. instance id

instance.clearx

Clears previously set x value (instance returns to design setting).

param
1. instance id

instance.cleary

Clears previously set y value (instance returns to design setting).

param
1. instance id

instance.loadasset

Loads a collection asset into the instance.

param
1. instance id
2. new asset id to load

instance.next

Load the next collection item on instance.

param
1. instance id

instance.pause

Pauses an instance content.

param
1. instance id

instance.play

Plays an instance content.

param
1. instance id

instance.playpause

Plays an instance content if it is paused, pauses it otherwise.

param
1. instance id

instance.previous

Load the previous collection item on instance.

param
1. instance id

instance.scrollbottom

Scrolls text to bottom.

param
1. instance id

instance.scrolldown

Scrolls text down.

param
1. instance id

instance.scrolltop

Scrolls text to top.

param

1. instance id

instance.scrollup

Scrolls text up.

param
1. instance id

instance.seek

Jumps an instance to a time position (integer, seconds).

param
1. instance id
2. time position

instance.setalpha

Changes the instance alpha level (0 to 1.0).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setcolor

Changes the instance overlay color (0x###### hex format).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setcoloralpha

Changes the instance color overlay alpha level (0 to 1.0).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfont

Changes the instance font face name.

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontalign

Changes the instance alignment (*left*, *right*, *center* or *justify*).

param

1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontbackground

Changes the instance font background color (0x###### hex format, empty string for none).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontbold

Changes the instance font to bold (boolean).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontcolor

Changes the instance text font color (0x###### hex format).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontitalic

Changes the instance font to italic (boolean).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontleading

Changes the instance font leading space (integer).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setfontsize

Changes the instance text font size (int value).

param
1. instance id
2. new value for horizontal display

3. new value for vertical display (optional)

instance.setheight

Changes the instance height size (numeric).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setorder

Changes the instance order on display list (int value).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setparagraph

Sets a paragraph instance content.

param
1. instance id
2. new text

instance.setrotation

Changes the instance rotation (experimental).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setvisible

Changes the instance visible state (boolean).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setvolume

Changes the instance sound volume (0 to 1.0).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setwidth

Changes the instance width size (numeric).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.setx

Changes the instance x position (numeric).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.sety

Changes the instance y position (numeric).

param
1. instance id
2. new value for horizontal display
3. new value for vertical display (optional)

instance.stop

Stops an instance playback.

param
1. instance id

instance.zoom

[consider using instance.morezoom and instance.lesszoom that provide better results] Enlarges the instance content to cover the stage. Works on picture and video content. The enlarged content will be hidden on user click.

param
1. instance id

instance.morezoom

Increases the zoom display of an instance from its center – the instance remains on place.

param
1. instance id
2. the zoom amount (in %)

instance.lesszoom

Decreases the zoom display of an instance from its center – the instance remains on place.

param
1. instance id
2. the zoom amount (in %)

instance.clearzoom

Removes all zoom applied to an instance and places it on the original position.

param
1. instance id

instance.moveup

Moves an instance up on display.

param
1. instance id
2. the amount to move

instance.movedown

Moves an instance down on display.

param
1. instance id
2. the amount to move

instance.moveleft

Moves an instance left on display.

param
1. instance id
2. the amount to move

instance.moveright

Moves an instance right on display.

param
1. instance id
2. the amount to move

instance.clearmove

Places an instance on display on its design position.

param
1. instance id

instance.startdrag

Grabs the instance and moves it following the mouse cursor or the target. The instance remains in drag mode until the mouse click/touch ends, the target trigger button is pressed again or the *instance.stopdrag* action is called. ***Warning: only use this action on the instance trigger event.***

param
1. instance id

additional optional object fields
- complete: actions to run when the instance is released from drag

instance.stopdrag

Stops any instance dragging.

instance.isoverlapping

Checks if one instance is overlapping another. Useful for verifying the status of an instance after the drag has finished.

param
1. first instance id
2. second instance id

additional optional object fields
- then: actions to run if the instances are overlapping
- else: actions to run if the instances are not overlapping

2.8. Integer conditions

Conditional statements based on integer values.

if.intsdifferent

Checks if the two int values are different.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.intsequal

Checks if the two int values are equal.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true

- else: actions to run if the result is false

if.intset

Checks if an int variable exists.

params
1. the variable name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

if.intgreater

Checks if the first value is greater than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.intgreaterequal

Checks if the first value is greater or equal than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.intlower

Checks if the first value is lower than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

if.intlowerequal

Checks if the first value is lower or equal than the second one.

params
1. first value
2. second value

additional optional object fields
- then: actions to run if the result is true
- else: actions to run if the result is false

2.9. Integer values

Management of integer variables.

int.abs

Returns the absolue value of an int.

params
1. the variable name to receive the result
2. the value

int.clear

Removes an int variable.

params
1. the variable name

int.clearall

Removes all int variables.

int.divide

Divides two or more int values. At least 3 parameters are required but you can provide as many as you want and their values are sequentially divided.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.max

Returns the maximum between two int values.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.min

Returns the minimum between two int values.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.multiply

Multiplies two or more int values. At least 3 parameters are required but you can provide as many as you want and their values are values are sequentially multiplied.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.random

Generates a random int value.

params
1. the variable name to receive the rando value
2. minimum value
3. maximum value

int.set

Sets an int variable.

params
1. the variable name
2. the variable value

int.subtract

Subtracts two or more int values. At least 3 parameters are required but you can provide as many as you want and their values are subtracted from the result.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.sum

Sums two or more int values. At least 3 parameters are required but you can provide as many as you want and their values are add to the result.

params
1. the variable name to receive the result
2. the first value
3. the second value

int.tofloat

Converts an int value to a float one.

params

1. the float variable name to receive the result
2. the int value

int.tostring

Converts an intvalue to a string.

params
1. the string variable name to receive the result
2. the int value

2.10. Mouse

mouse.hide

Hides the mouse cursor above the content area.

mouse.show

Displays the mouse cursor above the content area again.

2.11. Movie

movie.load

Loads a new movie into player.

params
1. new movie id

2.12. Replace

Use these actions to set automatic replaced both in string processing an file naming.

replace.clearfile

Removes a file replacement.

params
1. the needle (text to search) to remove

replace.clearstring

Removes a string replacement.

params
1. the needle (text to search) to remove

replace.clearallfiles

Clears all file replacement needles.

replace.clearallstrings

Clears all string replacement needles.

replace.origin

Replaces the current movie images origin set while loading scenes/keyframes. Possible values are “alpha”, “center”, “top”, “topkeep”, “bottom”, “bottomkeep”, “left”, “leftkeep”, “right” and “rightkeep”.

params
1. the new origin

replace.setfile

Sets a needle that will replace of its occurrences on loaded file names.

params
1. the needle (text to search)
2. the string to replace the needle on file names

replace.setstring

Sets a needle that will replace of its occurrences by the value given in both string processing and text display.

params
1. the needle (text to search)
2. the string to replace the needle

if.replacefileset

Checks if a file replacement exists.

params
1. the replacement name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

if.replacestringset

Checks if a string replacement exists.

params
1. the replacement name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

2.13. Runtime

runtime.ifbrowser

Check if the content is running from a browser (TilBuci player, web or PWA runtimes).
additional optional object fields

- then: actions to run if true
- else: actions to run if false

runtime.install

Installs the content as a native application.
Available on PWA runtime only.

runtime.quit

Ends the content execution.
Available on desktop app runtime only.

runtime.savedata

Saves the values of all current variables into an encrypted file. If running from a browser or as a mobile app, the file is downloaded to the device. If running from a desktop app, the file is automatically saved into the user's data folder.

params

1. the file name (.movieid will be add at the name end)

runtime.loaddata

Loads an encrypted variables file into memory. If running from a browser or as a mobile app, a file selection dialogue will appear. If running from a desktop app, the file is automatically loaded from the user's data folder.

params

1. the file name (.movieid will be add at the name end)

additional optional object fields

- success: the file was loaded and the variable values recovered
- error: the file couldn't be loaded

runtime.ifdataexist

Checks if a variables data file exists on the user's data folder.
Available on desktop app runtime only.

params

1. the file name (.movieid will be add at the name end)

runtime.startkiosk

Starts kiosk display mode (available for desktop and mobile apps).

runtime.endkiosk

Closes kiosk display mode (available for desktop and mobile apps).

2.14. Scene

scene.load

Loads a new scene.

params
1. new scene id

scene.navigate

Loads the scene set at the navigation settings of the current one.

params
1. the navigation direction: *up*, *down*, *left*, *right*, *nin* or *nout*

scene.pause

Pauses the current scene playback.

scene.play

Plays the current scene.

scene.playpause

Plays a stopped scene or pauses a playing one.

scene.nextkeyframe

Loads the next keyframe of a paused scene.

scene.previouskeyframe

Loads the previous keyframe of a paused scene.

scene.loadfirstkeyframe

Loads the first keyframe of a paused scene.

scene.loadlastkeyframe

Loads the last keyframe of a paused scene.

scene.loadkeyframe

Loads a keyframe from a paused scene.

params
1. the keyframe number to load (keyframe numbers start at 1)

scene.historyback

Loads the previously displayed scene (the history of loaded scenes is erased when adding a new movie).

scene.shake

Shakes the entire scene display.

params
1. shake effect duration in seconds (you may use float values like 1.5)
2. effect intensity

additional optional object fields
- end: actions to run when the effect ends

2.15. Snippets

Action snippets can be defined at the *Action snippets* tab of the *Movie properties* window.

run

Runs a named action (set at *Movie* > *Properties* window).

params
1. action snippet name

2.16. String conditions

Conditional statements based on string values.

if.stringcontains

Checks if the first string contains the second one.

params
1. first string to compare
2. second string to compare

additional optional object fields
- then: actions to run if the string contains the other one
- else: actions to run if the string does not contain the other one

if.stringendswith

Checks if the first string ends with the second one.

params
1. first string to compare
2. second string to compare

additional optional object fields
- then: actions to run if the string ends with the other one
- else: actions to run if the string does not end with the other one

if.stringsdifferent

Checks if two strings are different.

params
1. first string to compare
2. second string to compare

additional optional object fields
- then: actions to run if the strings are different
- else: actions to run if the strings are equal

if.stringsequal

Checks if two strings are equal.

params
1. first string to compare
2. second string to compare

additional optional object fields
- then: actions to run if the strings are equal
- else: actions to run if the strings are different

if.stringset

Checks if a string variable exists.

params
1. the variable name to check

additional optional object fields
- then: actions to run if true
- else: actions to run if false

if.stringstartswith

Checks if the first string starts with the second one.

params
1. first string to compare
2. second string to compare

additional optional object fields
- then: actions to run if the string starts with the other one
- else: actions to run if the string does not start with the other one

if.stringemail

Checks if the strig is a valid e-mail address.

params
1. the address to check

additional optional object fields
- then: actions to run if the string is a valid email
- else: actions to run if the string isn’t a valid email

2.17. String values

Management of string variables.

string.clear

Removes a string variable.

params
1. the variable name

string.clearall

Removes all string variables.

string.clearglobal

Clears a text global.

params
1. the global name

string.concat

Concatenates two or more strings. Three parameters are required. Each additional one will be concatenated at the string end. You can use any type of variable (string, int, float or bool) – the value will be automatically converted to string.

params
1. the variable name to receive the concatenated one
2. the first string to concatenate
3. the second string to concatenate

string.replace

Replaces all occurrences of the needle string by the given one at the provided text.

params
1. the variable name to receive the replace result
2. the original string
3. the needle (string to look for)
4. the replacement text

string.set

Sets a string variable.

params
2. the variable name
3. the variable value

string.setglobal

Sets a text global.

params
1. the global name
2. the global value

string.setgroup

Sets a group of the *strings.json* file to look for string variable values.

params
1. the group name

string.tofloat

Converts a string value to a float one.

params
1. the float variable name to receive the result
2. the string to convert

string.toint

Converts a string value to an integer one.

params
1. the integer variable name to receive the result
2. the string to convert

2.18. System

system.copytext

Copies a string into the user clipboard.

params
1. the string to copy

system.fullscreen

Shows the content at fullscreen or brings back the display to normal state. Please note that you must trigger this action only on an user interaction. It won’t work on movie/scene/keyframe actions due to browser policies. No parameters are required.

system.visitoringroup

Checks if the current logged visitor is part of a give group. If no visitor is currently logged, always return false.

params
1. the group name to check (case sensitive)

additional optional object fields

- then: actions to run if the visitor belongs to the provided group
- else: actions to run if the visitor is not in the group or if no visitor is logged at all

system.logout

Logs out the current user.

system.openembed

Opens an overlay with a previously embedded HTML5 content (access Media > Embed on the left menu to send the HTM5 content to embed).

params
1. the embed name used when sending the content

Embed content can exchange information with the host movie using javascript methods. Since the embed content is displayed using an iframe, all melhods must be called from "parent".

parent.tilbuci_getstring($name)	gets the current value of the $name variable (empty string if it is not set)
parent.tilbuci_setstring($name, $value)	sets the value of the string variable $name
parent.tilbuci_getfloat($name)	gets the current value of the $name variable (0 if it is not set)
parent.tilbuci_setfloat($name, #value)	sets the value of the float variable $name
parent.tilbuci_getint($name)	gets the current value of the $name variable (0 if it is not set)
parent.tilbuci_setint($name, #value)	sets the value of the integer variable $name
parent.tilbuci_getbool($name)	gets the current value of the $name variable (false if it is not set)
parent.tilbuci_setbool($name, ?value)	sets the value of the boolean variable $name
parent.tilbuci_runaction($action)	runs any json-formatted action on movie

system.closeembed

Closes the HTML5 embed content display.

system.embedreset

Resets the HTML5 embed display site and position so it appears covering all display area.

system.embedplace

Sets the position and size of the HTML5 embed content area. The placement values are absolute, considering the entire stage, not only your movie display area. Because of that, the #_CONTENTX, #_CONTENTY, #_CONTENTWIDTH and #_CONTENTHEIGHT values can be useful to help figuring out the desired values.

params
1. x position
2. y position

3. width
4. height

system.openurl

Opens an URL on user browser.

params

1. the UL to open

system.sendevent

Sends a custom event (TilBuciEvent.EVENT) to everyone listening to the player object. The event info field will be filled with any string parameters you add to this action.

system.setkftime

Sets the time between keyframes in milliseconds. Values lower than 250 will be ignored.

params

1. the new time between keyframes in milliseconds

system.ifhorizontal

Checks if the current display orientation is horizontal.

additional optional object fields

- then: actions to run if true
- else: actions to run if false

system.ifvertical

Checks if the current display orientation is vertical.

additional optional object fields

- then: actions to run if true
- else: actions to run if false

system.ifwebsite

Checks if running from the website export runtime.

additional optional object fields

- then: actions to run if true
- else: actions to run if false

system.ifpwa

Checks if running from the PWA export runtime.

additional optional object fields

- then: actions to run if true
- else: actions to run if false

system.ifpwainstalled

Checks if running from an installed PWA.

additional optional object fields
- then: actions to run if true
- else: actions to run if false

system.ifdesktop

Checks if running from the desktop app export runtime.

additional optional object fields
- then: actions to run if true
- else: actions to run if false

system.ifmobile

Checks if running from the mobile app export runtime.

additional optional object fields
- then: actions to run if true
- else: actions to run if false

system.ifpublish

Checks if running from the publish services export runtime.

additional optional object fields
- then: actions to run if true
- else: actions to run if false

system.ifplayer

Checks if running from the standard TilBuci player.

additional optional object fields
- then: actions to run if true
- else: actions to run if false

2.19. Target

The target can be used for navigation using the keyboard or a game controller. It replaces the mouse cursor.

target.show

Starts the target navigation, showing the target graphic at the center of the screen.

target.hide

Stops the target navigation, hiding the target graphic.

target.toggle

Alternates the target current display state.

target.setposition

Sets the target graphic position (showing it if necessary).

params
1. x position
2. y position

target.clear

Clear current custom target graphic, returning to the default one.

target.set

Sets a custom target (configured at the contraptions menu).

params
1. the custom target name

2.20. Text

Text actions adjust the css styles used on html instances. You can provide a default css stylesheet at the *CSS* tab of the *Movie properties* window.

css.clear

Clears current css styles.

css.set

Sets css styles.

params
1. css styles text

2.21. Timer

Timers allow you to run actions in intervals and after some time.

timer.clear

Stops and clears a timer.

params
1. the timer name to clear

timer.clearall

Stops an clears all set timers.

timer.set

Sets a timer with a given interval in milliseconds and a number of iteractions.

params

2. timer name
3. interval among timer ticks in milliseconds (minimum of 250)
4. number of iteractions

additional optional object fields

- tick: actions to run after evey tick
- end: actions to tun on timer end

3. Plugin actions

Plugins may add actions and conditions to TilBuci. The software comes with some of them already available. Here are the actions they provide.

3.1. Debug plugin

This plugin is intended to be used during your content development. It is recommended to turn it off in movies already available to the public.

trace

This command will display a text on the browser's console. You can use it to show anything. It requires one parameter, but you may add as many as you want – all of them will displayed at the console.

params
1. content to display on browser's console

params

trace.bools

Displays all current boolean variables as their values at the browser console.

trace.ints

Displays all current integer variables as their values at the browser console.

trace.floats

Displays all current float variables as their values at the browser console.

trace.strings

Displays all current string variables as their values at the browser console.

debuginfo.hide

Hides the debug info window if it is being displayed (F8 on keyboard).

debuginfo.show

Shows the debug info window over your content with many . You may also press F8 on the keyboard to show this window.

3.2. Share

This plugin provides actions to open a browser tab ready for the user to share a content from your movies. They don't require any parameters – the movie url and title are automatically add to the share call – but you may add a boolean one set to true to force the share url to point to the movie index instead of the current scene, even if the share mode on movie properties is set to scene.

share.facebook

share.linkedin

share.pinterest

share.reddit

share.x

3.3. Google Analytics

This plugin enables measurement of user activities while checking out your content. It must be configured with the *Measurement ID* before use. Movie and scene load operations are automatically registered as events on the Analytics platform, but you may also create your custom events.

When a movie is loaded, a new campaign is automatically set on Analytics with these parameters:

- id: the movie ID
- name: the movie name
- source: your TilBuci base installation domain

Automatic movie load events set these parameters:

- movie_id: the loaded movie ID
- movie_name: the loaded movie title

Automatic scene load events hold these parameters:

- movie_id: the current movie ID
- movie_name: the current movie title
- scene_id: the loaded scene ID
- scene_name: the loaded scene title

analytics.event

Registers a custom event at the Google platform. You must provide the event name and its description.

The recorded event will set these parameters:

- movie_id: the current movie ID
- movie_name: the current movie title
- scene_id: the current scene ID
- scene_name: the current scene title
- about: the provided description

params
1. event name
2. event description

3.4. Server Call

This plugin is aimed at operations that must run at your TilBuci installation server.

call.process

Calls a server script to process and return data to TilBuci. The script can even be loaded from another domain, but it must always return a JSON text considering the format below.

This is a versatile function. It can be used to collect data from other internet sources, process data collected while viewing your content, and even create communication systems like your own user management and data storage.

The function requires at least two string parameters but you may add as many as you want. When called, it will make a POST request to the URL of the first parameter with the following values:

value	**content**
movieid	Current movie ID
sceneid	Current scene ID
movietitle	Current movie title
scenetitle	Current scene title
visitor	Current visitor e-mail or “system” if none is logged in
data	A JSON-encoded array with all string parameters set, in order, except the script url

Your script must process the information and return a JSON encoded object with values that will be loaded into TilBuci variables when received. The JSON object can contain as many variables as you want to return, described as:

```
“name”: { “t”:“the variable type”, “v”:“the value” }
```

Possible types are “B” (boolean), “I” (integer”), “F” (float) and “S” (string). A return example that sets four variables:

```
{
	"mystring":{"t":"S","v":"the string value"},
	"myfloat":{"t":"F","v":100.101},
	"myint":{"t":"I","v":10},
	"mybool":{"t":"B","v":true}
}
```

You may provide the “success” and “error” optional object fields to run after a successful or failure call.

params
1. the url to call
2. string parameter to send for processing

additional optional object fields
- success: actions to run on successful call
- error: actions to run on call failure

call.sdprocess

This action is like the previous one, but the URL to call must always be hosted at the same domain as your TilBuci movie is loaded from. Everything else works just like call.process. Because of the same domain URL limitation, you can still use this action to access server data on exported websites or PWA applications, unlike the other Server Call plugin actions.

call.url

Calls any URL from the server. By transferring the request to the server, TilBuci avoids browser “CORS” limitations. You may provide a string variable name to receive the return on a successful request. You may provide the “success” and “error” optional object fields to run after a successful or failure call.

params
1. the url to call
2. optional string variable name to receive the result

additional optional object fields
- success: actions to run on successful call
- error: actions to run on call failure

3.5. Overlay

This plugin can load an external content and display it above your TilBuci movie. You can even exchange data between them.

overlay.show

Loads an external content on an overlay frame above your movie.

params
1. the url to load
2. title to display above the overlay content (may be an empty string)
3. optional bool value to send additional information as get parameters

additional optional object fields
- success: actions to run after the overlay is successfully displayed and closed
- error: actions to run on overlay error

The overlay is a powerful feature to enhance your TilBuci creation. It can be anything loaded as a web page, from a simple page to a detailed form or a complex game.

The first parameter is the url of the overlay content. You can set it to an address on your domain (recommended) or event to other sites. TilBuci will automatically assign a unique key to every overlay call. This key will be sent as a get "key" parameter add to your url, like:

https:tilbuci.com.br/myoverlaycontent/?key=uniquekey

This key is very important: you can use it to exchange data between your movie and the overlay.

The second parameter is a title that will appear above the overlay content. If you do not want a title, just use a blank string.

The third parameter is optional. You can add as many additional parameters as you want to send to your overlay. If the third parameter is set to "true", this extra information will be included as get values on your url. Check out this example:

```
{
    "ac":"overlay.show",
    "param":[
        "https://tilbuci.com.br/myoverlaycontent/",
        "My overlay title",
        "true",
        "param4",
        "param5"
    ]
}
```

Will open an overlay calling this address:

https:tilbuci.com.br/myoverlaycontent/?key=uniquekey&v1=param4&v2=param5

Since there are limitations regarding the size of get requests, use this with caution. The "key" value is always sent.

Even if you do not add the additional information as get values, your overlay can still retrieve it by calling the TilBuci webservice interface. To do so, you must call the "ws" route of your TilBuci installation with a POST request, like *https://tilbuci.com.br/ws/*, with the following values:

POST parameter	Value
a	Overlay/GetKeyData
k	The overlay key you received
s	A signature made of MD5(secret key + overlay key)

You must set a secret key at your TilBuci overlay plugin configuration. This secret key, add to the unique key of your overlay and encoded as a MD5 string must always be sent as the "s" signature parameter to validate your request.

The response you receive will always be a JSON-formatted string. To evaluate it, you must first look for the "e" (error) value. If the error is "0", your request was successful, and you may proceed parsing the "data" value. You'll receive it as a JSON-encoded object with name/value pairs with the same names you'd receive as get values.

You can also use this webservice request to send data back to the TilBuci movie. This data will be loaded into variables when the overlay is closed. Here are the required POST parameters:

POST parameter	Value
a	Overlay/SetKeyData
k	The overlay key you received
s	A signature made of MD5(secret key + overlay key)
d	The data to send as a JSON-encoded string

The "d" parameter is a JSON-encoded string describing the variables to set on your TilBuci movie. It can contain as many variables as you want to return, described as:

```
"name": { "t":"the variable type", "v":"the value" }
```

Possible types are "B" (boolean), "I" (integer"), "F" (float) and "S" (string). A return example that sets four variables:

```
{
	"mystring":{"t":"S","v":"the string value"},
	"myfloat":{"t":"F","v":100.101},
	"myint":{"t":"I","v":10},
	"mybool":{"t":"B","v":true}
}
```

If you call this webservice more than once, new values will replace the previous ones. The TilBuci movie will only receive this information when the overlay is closed. This happens when the visitor clicks at the close button that will always appear above your overlay. You may also call this javascript function from your content to force it to close but notice that if the url of the overlay request is not at the same domain of your TilBuci installation, the browser may block this call.

```
parent.overlay_close();
```

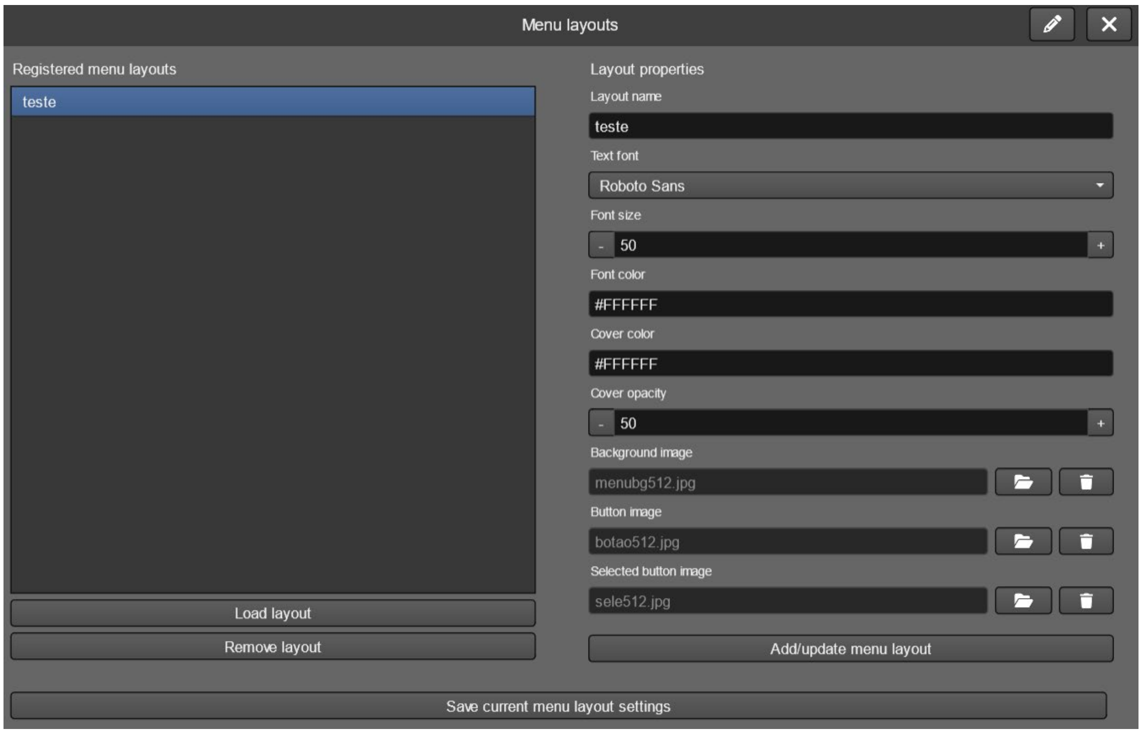
The *overlay.show* action supports the *success* and *error* additional optional object fields. Error actions are run on any problem regarding the overlay display, while the success ones will run after the overlay is closed and the return variables are set.

4. Contraptions

Contraptions are ways to simplify creation by grouping together tools that help you develop your content more quickly. They can include both settings found in the left menu and specific actions.

4.1. Menu

The menu contraption is a quick way to create menus and display them in your scenes. The process starts with creating layouts from the left menu. To do this, you need to provide some information such as the name of the font to be used and the images for both the background and the menu buttons. Once the layouts are created, they can be used in the *contraption.menu* action to start displaying them at any time.



teste

Layout name: teste
Text font: Roboto Sans
Font size: 50
Font color: #FFFFFF
Cover color: #FFFFFF
Cover opacity: 50
Background image: menubg512.jpg
Button image: botao512.jpg
Selected button image: sele512.jpg

contraption.menu

This command allows you to display a menu from a previously defined layout. It receives 6 parameters. When a button is clicked, its number is stored at the provided int variable (first button value is 1, second is 2 and so on). After setting the value, the actions set at “select” are run. The menu won’t automatically disappear after a button is clicked – you’ll need to add the *contraption.menuhide* when you want to remove it.

params
1. the layout name
2. the menu option texts (all of them, split by ;)
3. integer variable name to receive the click result
4. placement (top, topleft, topright, center, centerleft, centerright, bottom, bottomleft, bottomright, absolute)
5. x distance from border (or position for absolute placement)
6. y distance from border (or position for absolute placement)

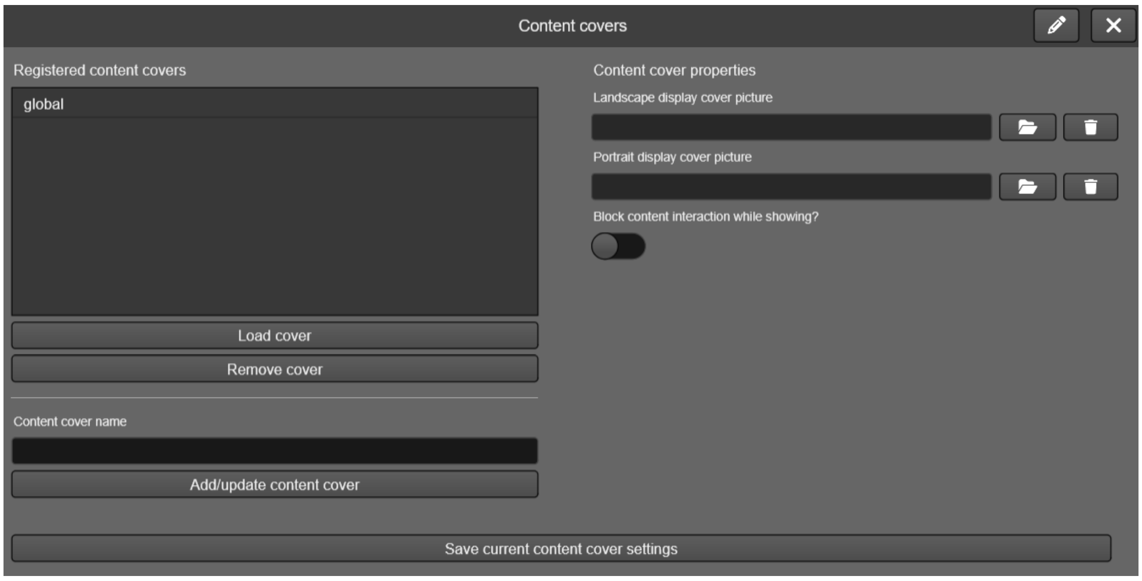
additional optional object fields
- select: actions to run on a menu button selection

contraption.menuhide

Hides any menu currently displayed. No parameters needed.

4.2. Cover

The cover contraption is a simple way to create an image layer on top of your content, easy to display at any time. To do this, covers must be configured in the left menu contraptions > content cover.


Content covers
Registered content covers
global
Load cover
Remove cover
Content cover name
Add/update content cover
Content cover properties
Landscape display cover picture
Portrait display cover picture
Block content interaction while showing?
Save current content cover settings

When configuring, in addition to giving a name, you must indicate at least one picture that will be used to cover the entire movie area. You can also define one image for horizontal display and one for vertical display. Finally, you can also indicate whether the cover will block click/touch interactions with the content below.

contraption.cover

This action will cause a defined cover to be displayed. If there is already a cover displayed, it will be replaced with the indicated one.

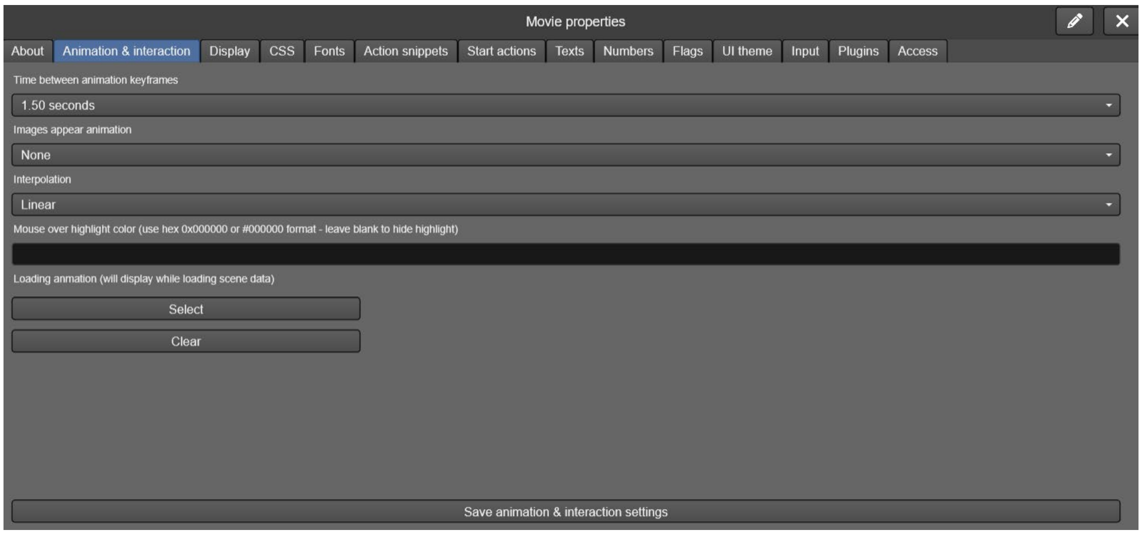
params
1. the cover name

contraption.coverhide

Removes any currently displayed cover.

4.3. Loading icon

The loading icon is a way to give visual feedback to the visitor while some processing is taking place, such as loading content. It is defined in the movie properties, as a spritemap type media.



Movie properties

Time between animation keyframes: 1.50 seconds
Images appear animation: None
Interpolation: Linear
Mouse over highlight color (use hex 0x000000 or #000000 format - leave blank to hide highlight)
Loading anmation (will display while loading scene data)

contraption.showloading

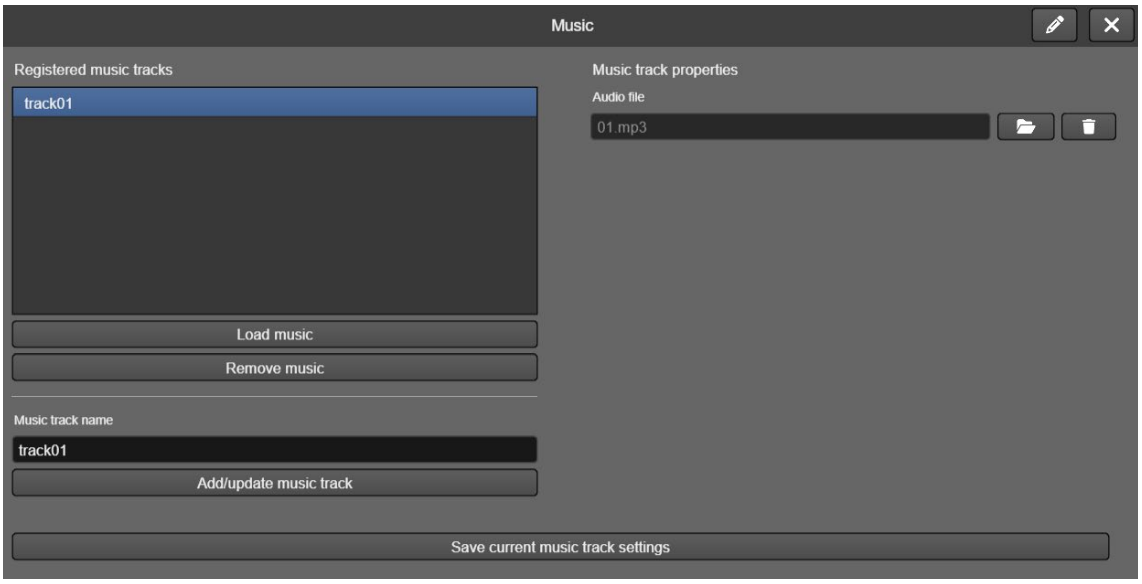
Displays the loading icon set for the movie at the display center (notice that scene loading automatically triggers this action).

contraption.hideloading

Hides the loading icon from display (notice that when a scene loading finishes, this action is automatically triggered).

4.4. Music tracks

While it's possible to use an audio instance to include music in your creations, this contraption makes the process easier. Simply create music tracks from your media and, with a simple command, play them when needed.



contraption.musicplay

Plays a music track, interrupting any other currently playing music. This action does not interfere with audio instances present in your movie.
params
1. the music track name

contraption.musicpause

Pauses the current track. If you start playing it again, it will resume from the same point.

contraption.musicstop

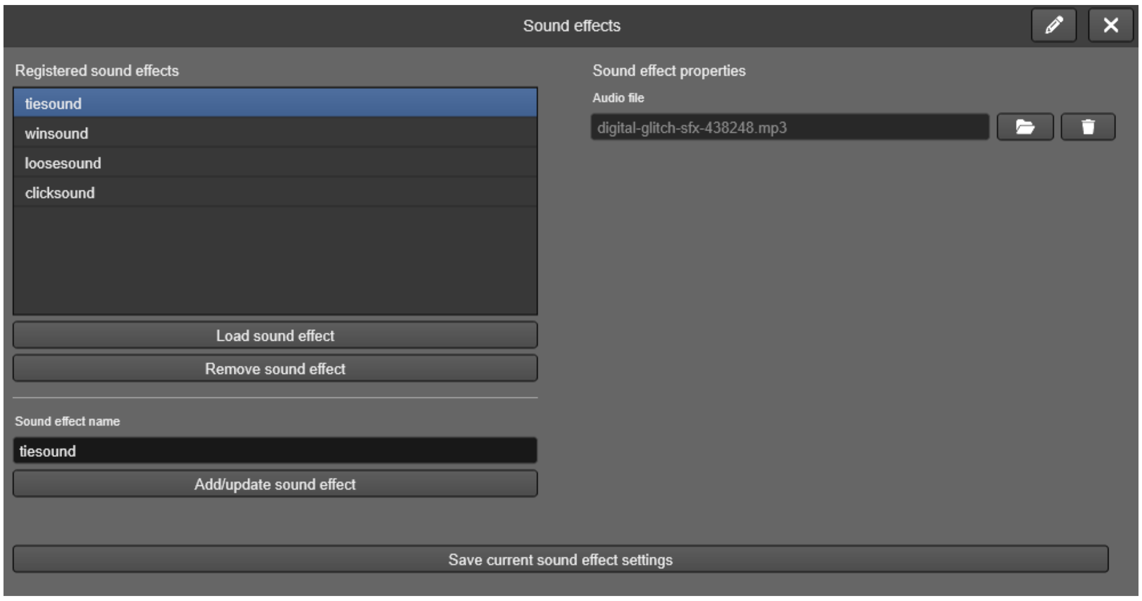
Stops the current track. If you start playing it again, it will start from the beginning.

contraption.musicvolume

Sets the music track sound volume from 0 to 100%.
params
1. the sound volume as an integer from 0 to 100

4.5. Sound effects

Just like with music, isolated audio tracks may be needed for your production, and you can include them using audio instances. Even so, TilBuci offers a feature that simplifies this process. You can register sound effects to be played via an action. Unlike music, playing a sound effect doesn't interrupt the previous one. Sound effects are played only once, without looping.



contraption.soundplay

Plays a sound effect. This action does not interfere with audio instances present in your movie.
params
1. the sound effect name

contraption.soundpause

Pauses a sound effect. If you start playing it again, it will resume from the same point.
params
1. the sound effect name (leave blank to pause all sound effects)

contraption.soundstop

Stops a sound effect. If you start playing it again, it will start from the beginning.
params
1. the sound effect name (leave blank to stop all sound effects)

contraption.soundvolume

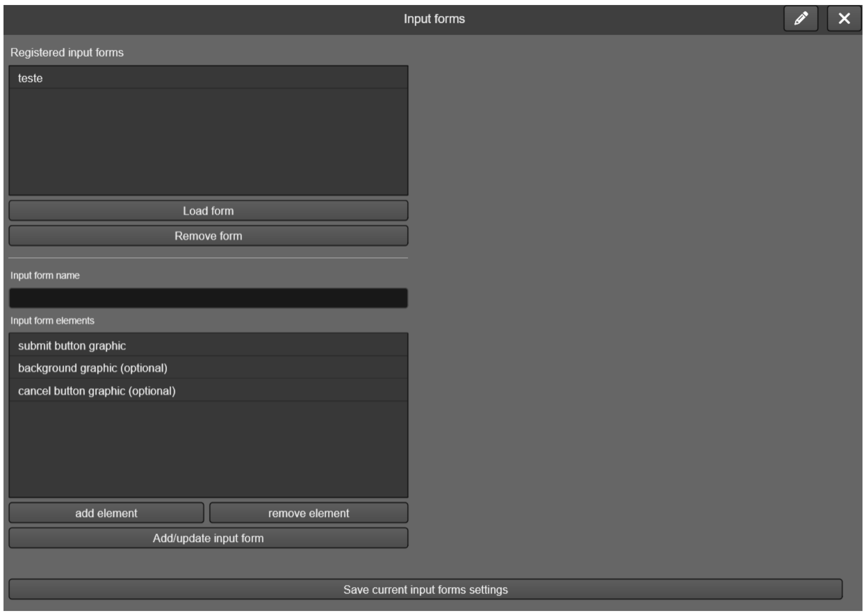
Sets a sound effect sound volume from 0 to 100%.
params

1. the sound volume as an integer from 0 to 100
2. the sound effect name (leave blank change all sound effects)

4.6. Forms

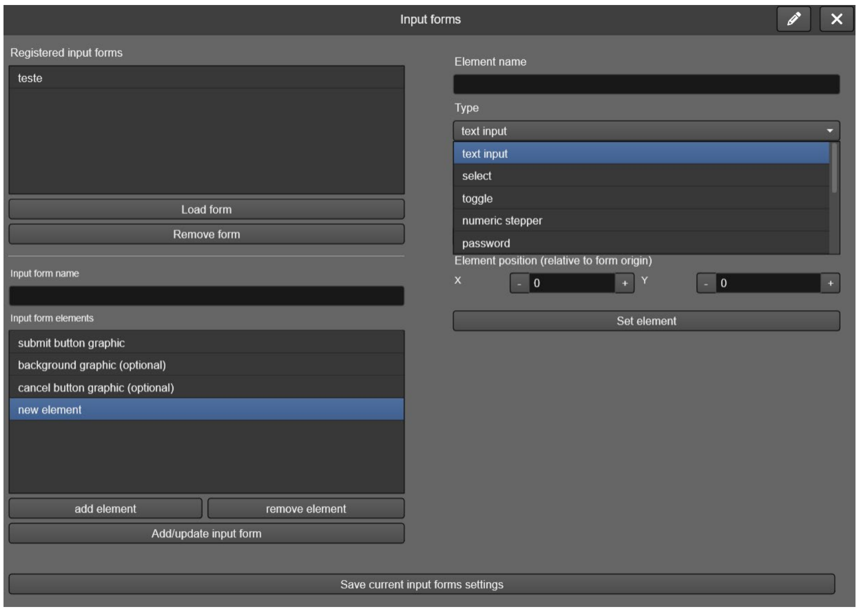
TilBuci offers several ways to retrieve information from the visitor. You can use the input actions to either display text areas at the scene display or to show the TiBuci's interface for getting data like numeric steppers, lists and so on. The form contraption is another option for that: you can create complex forms with several input types and show them using an action command.

First, you must create a fomr using the contraption window. You can create as many forms you want: just give them different names.



The form must have an "ok button" so the visitor can submit data. There is also optional background and cancel buttons.

Use the "add element" button to create your inputs form any of the several available types. Give each one an unique name (for the same form) its position (from the form display origin) and width. For select inputs you must also provide the options list, split by ;. You can add as many elements you want.



With your forms created, you can use actions to display and handle them. The data provided by the visitor using the forms will be available as a global form value.

contraption.form

Displays a form over the scene content. The selected form will replace any other form being displayed at the moment.
params
1. the form name
2. the form x position
3. the form y position

This action lets you set two additional fields:

- Ok: actions to run when the visitor clicks on the submit button
- Cancel: actions to run when the visitor clicks on the cancel button

contraption.formvalue

Sets the value of a current form element. You must call this action after showing up the form. Use strings for all parameters, including numbers and “true”/”false”.
params
1. the form element name
2. the value to set

contraption.formsetstepper

Numeric steppers are always created with a minimum of 0, a maximum of 10000 and a step of 1, but you can change these limits using this command.
params

1. the numeric stepper element name
2. the minimum value
3. the maximum value
4. the increase/decrease step

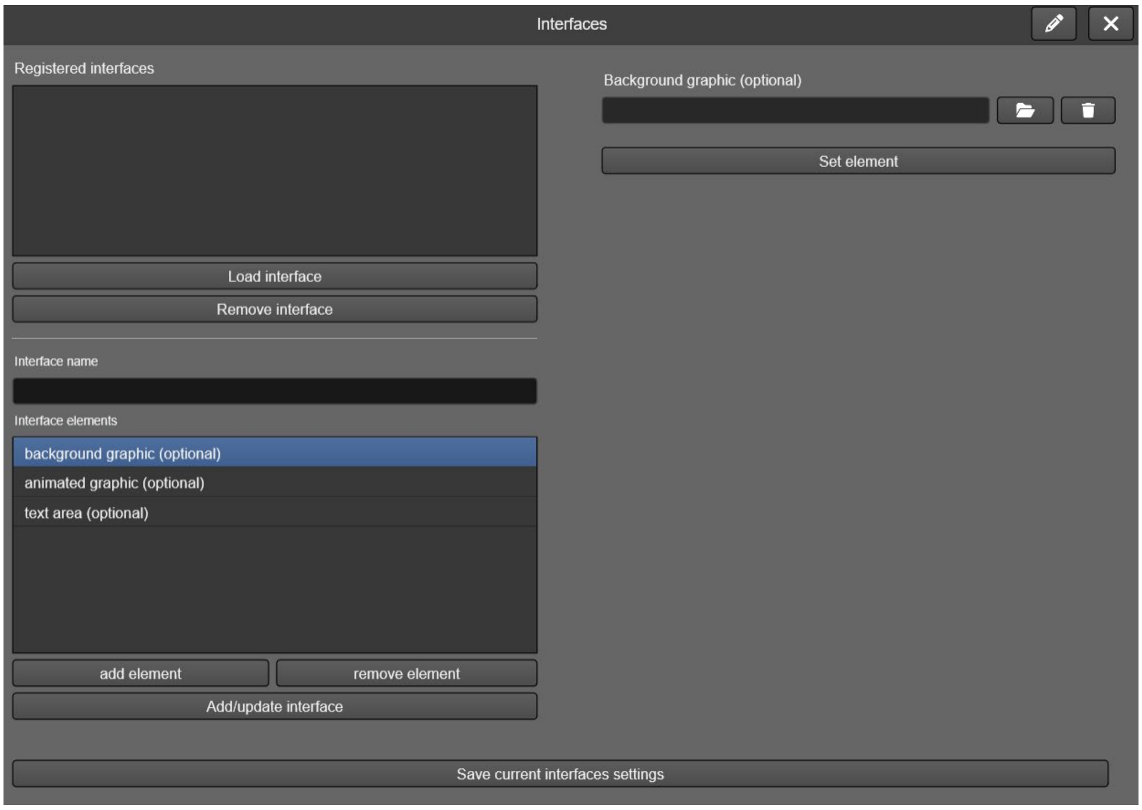
contraption.formhide

Hides the current form.

4.7. Interfaces

Visitor interfaces are very useful for your content engagement. TilBuci offers a contraption to simplify the creation of reusable interfaces that may contain graphics, buttons, texts and even animations. To create an interface, access "interfaces" at the "contraptions" right menu.

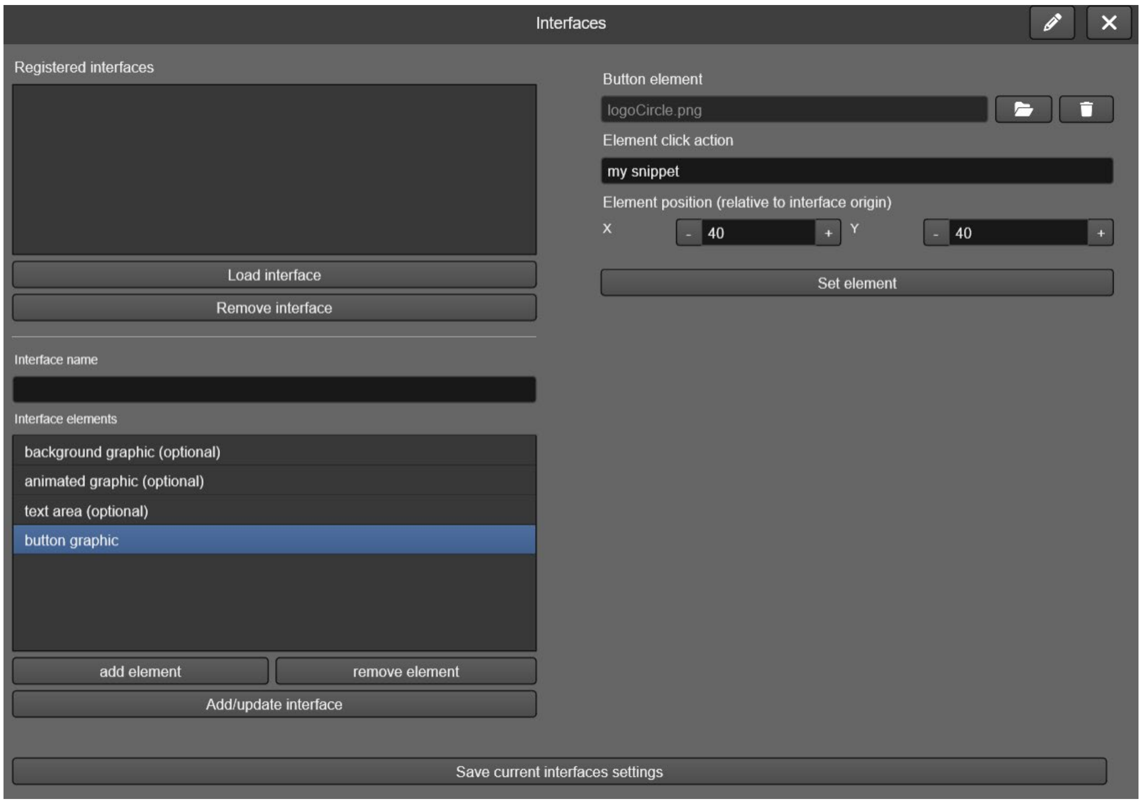
Unlike the forms, you may display as many interfaces as you want at the same time.



There are three pre-defined elements, but they are all optional: the interface background graphic, an animation (provided by a spritemap) and a text line. By selecting each of them at the list you can set the properties like the graphic and the text visuals.

The spritemap can be used as a button or just as a graphic. It always starts paused, but you can use actions to play it or even set a static frame to show, enabling you to use the interface as visual feedback for the visitor.

You may add additional graphics to the interface that can be used only for visual purposes or as buttons: just provide an action snippet name to run when clicked/tapped.



Button element: logoCircle.png
Element click action: my snippet
X: 40
Y: 40

contraption.interface

Shows an interface. You may display as many interfaces you want at the same time.
params
1. the interface name
2. the interface x position
3. the interface y position

contraption.interfacehide

Hides a single interface.
params
1. the interface name

contraption.interfacehideall

Hides all interfaces.

contraption.interfacetext

Sets the content of an interface text line.
params
1. the interface name
2. the text to display

contraption.interfaceanimframe

Sets the current frame of an interface animation (forcing it to pause).
params
1. the interface name
2. the frame (integer, 1-based)

contraption.interfaceanimplay

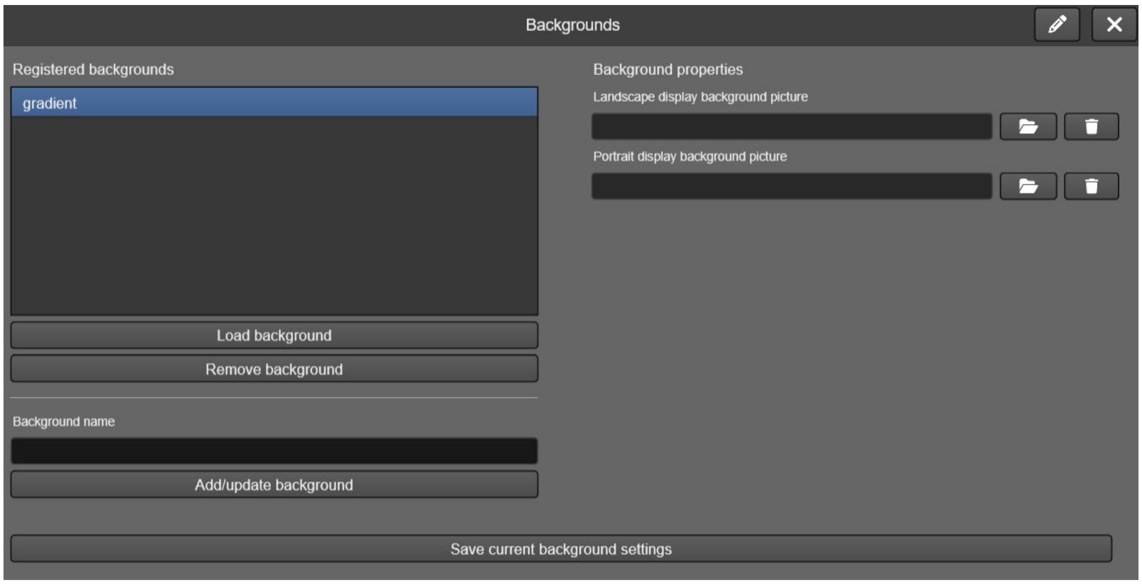
Starts playing an interface animation.
params
1. the interface name

contraption.interfaceanimpause

Pauses an interface animation.
params
1. the interface name

4.8. Background

You can set the background color of your TilBuci movies. You can also use picture instances to simulate backgrounds on your scenes, but with the background contraptions this process is much easier! When you show a background, it will be displayed even after a new scene loading.



When configuring, in addition to giving a name, you must indicate at least one picture that will be used to cover the entire movie background area. You can also set one image for horizontal display and one for vertical display.

contraption.background

This action will show the requested background picture below the scene content, replacing another background, if shown.

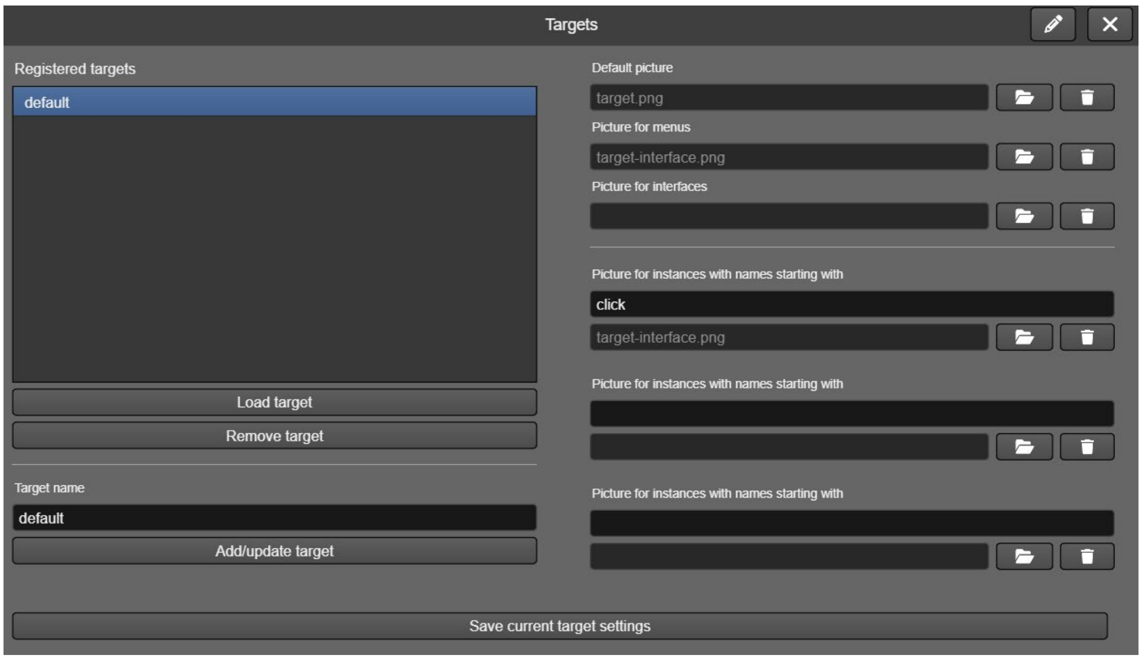
params
1. the background name

contraption.backgroundhide

Removes the current background picture.

4.9. Targets

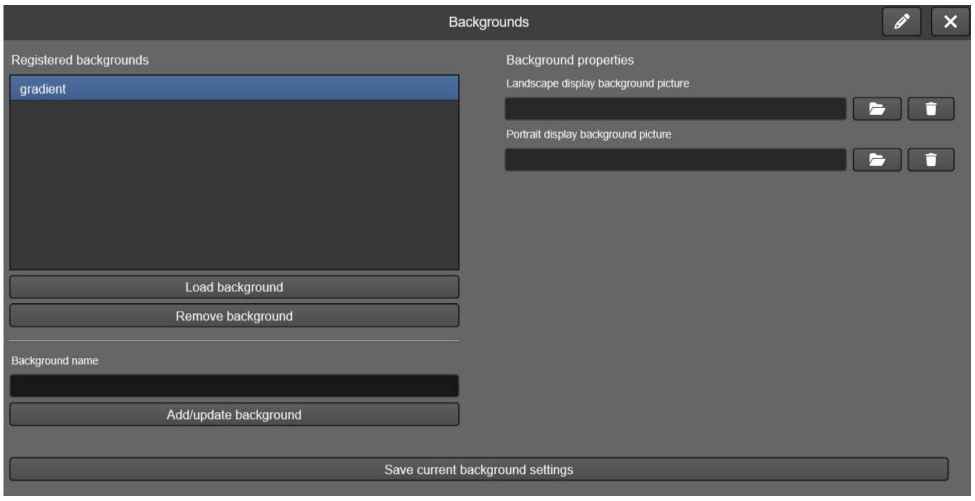
Targets allow navigation through content using both keyboards and game controllers. To do this, you can set groups of images that act as cursors controlled by these input methods. In addition to the default image, you can provide graphics that react to the target by moving over menu buttons, interfaces, or even instances with actions, ensuring visual feedback. If no image is specified, TilBuci will use a default one: a white circle with a black outline.



Actions involving targets were grouped under "input management" instead of "contraptions" due to the functions they manage.

4.10. Messages

TIlBuci offers various ways to present content to the user, including dialog boxes with buttons. This contraption is another one of these methods and is fully compatible with target navigation. Here, it's possible to display a "popup" with a message that can contain up to three buttons. In the configuration window of this tool, it's possible to adjust all the visual aspects, including the font, its size and color, as well as images for the message background and the button graphics.



contraption.message

This action will display a dialog box with the indicated text and buttons, which will await user interaction. After user interaction (the "when selected" event), the indicated integer variable will receive a number from 0 to 2 that registers which button was clicked and can be used to take the necessary actions.
params

1. the message layout name (configured at the messages contraption settiings)
2. the message text
3. the buttons labels, split by ; (maximum of 3)
4. the name of an integer variable that will receive the clicked button number

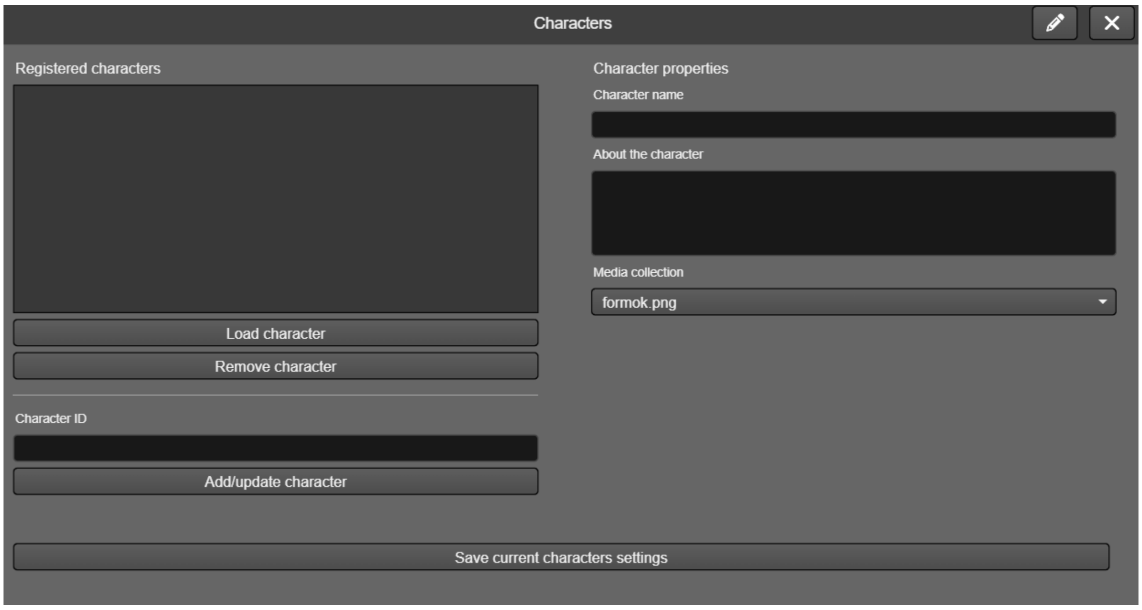
contraption.messagehide

Remove the message currently being displayed.

5. Narrative

Narrative tools offer simple ways to manage story-focused content, including character and dialogue management. These tools can be found at the left menu "narrative".

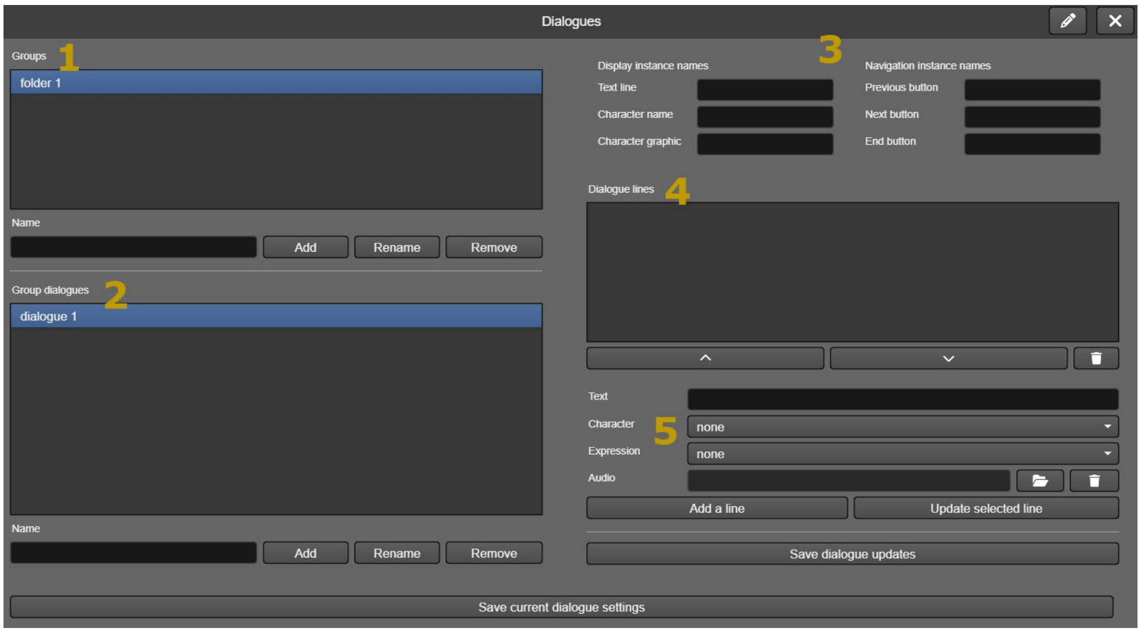
5.1. Characters



In this window you can register the characters in your story, including their names and a collection of media that will be used later as images of them in dialogues.

5.2. Dialogues

In the dialogue window, you can create conversations between characters in your story that will be displayed based on actions in the content display. These dialogues are stored in groups that can be loaded at any time.



In the dialog editing window, groups are registered in area 1 of the previous image. When a group is selected, its dialogs appear in area 2, from where they can be managed. Once one of these dialogs is selected, area 3 allows you to specify the names of the instances that will be used for navigation and content display, while area 4 shows the conversation lines that can be edited in area 5.

Dialogues are always handled by actions while playing your content.

dialogue.loadgroup

Loads a group of dialogs into memory, replacing any that were previously loaded.

params
1. the dialogue group name

additional optional object fields
- success: actions to run if the dialogue group is successfully loaded
- error: actions to run if the dialogue group fails to load

dialogue.start

Begins the display of a dialog from the currently loaded group. If no group is loaded or the specified dialog does not exist, the action has no effect.

params
1. the dialogue name

dialogue.next

Shows the next line of the current dialogue.

dialogue.previous

Shows the previous line of the current dialogue.

dialogue.last

Shows the last line of the current dialogue.

dialogue.first

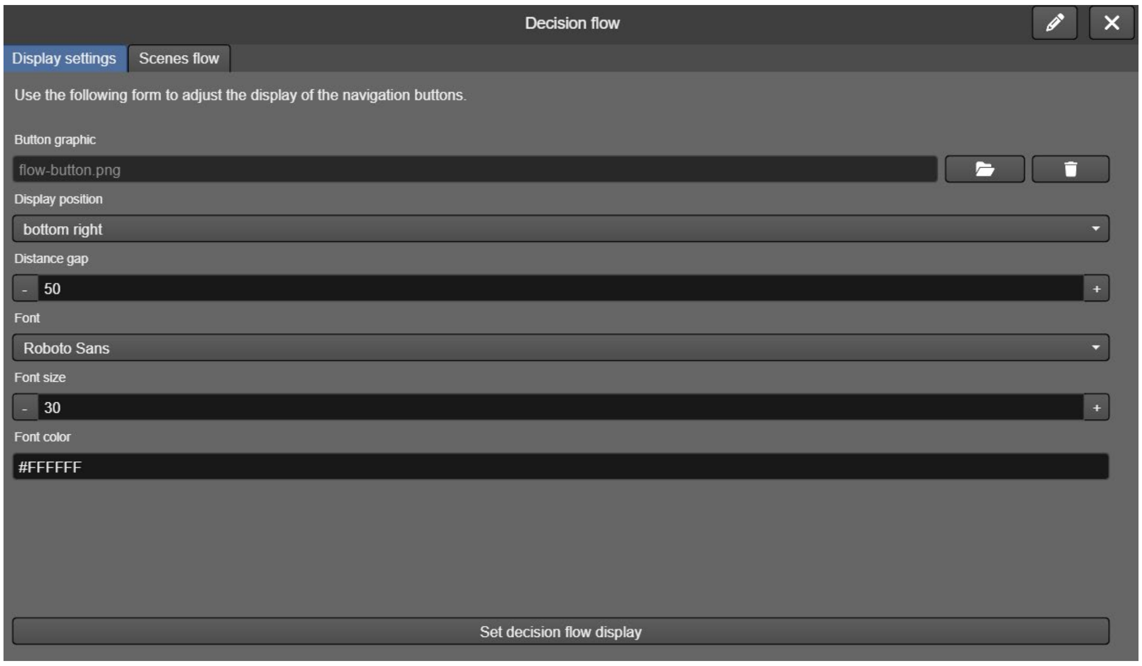
Shows the first line of the current dialogue.

dialogue.close

Finishes the current dialogue display.

5.3. Decision flow

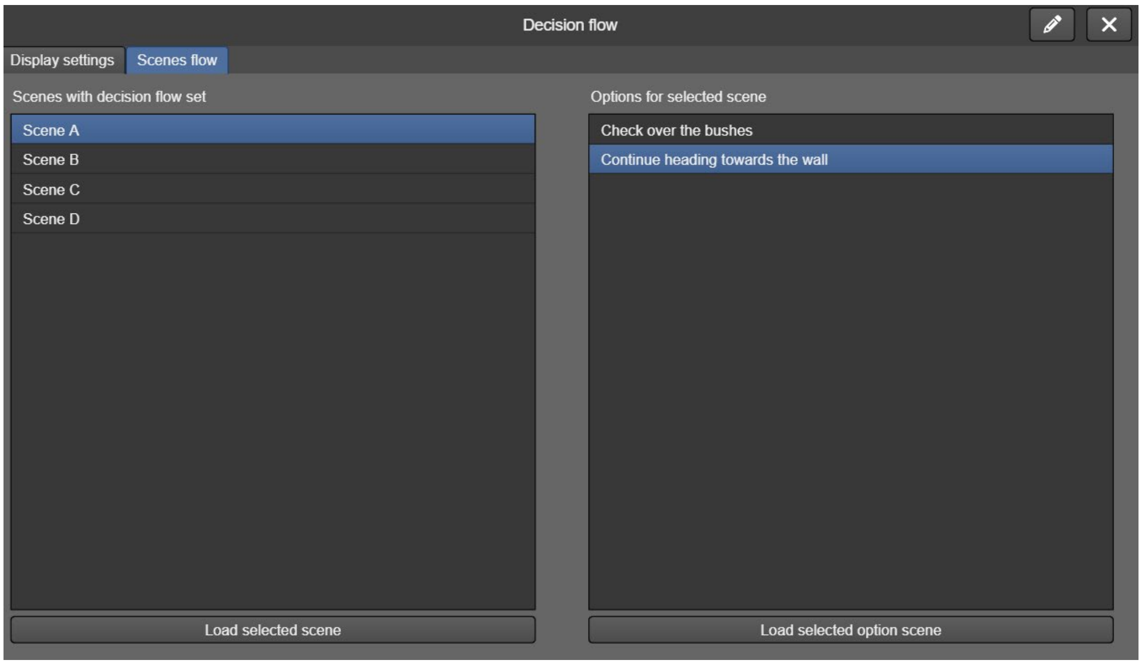
Decision flow is a tool that allows for a simple connection between scenes, displaying options as buttons when the last keyframe of each scene is accessed. In the left "narrative" menu, under the "decision flow" button, you can adjust the graphic properties of these buttons, such as image, font and position.



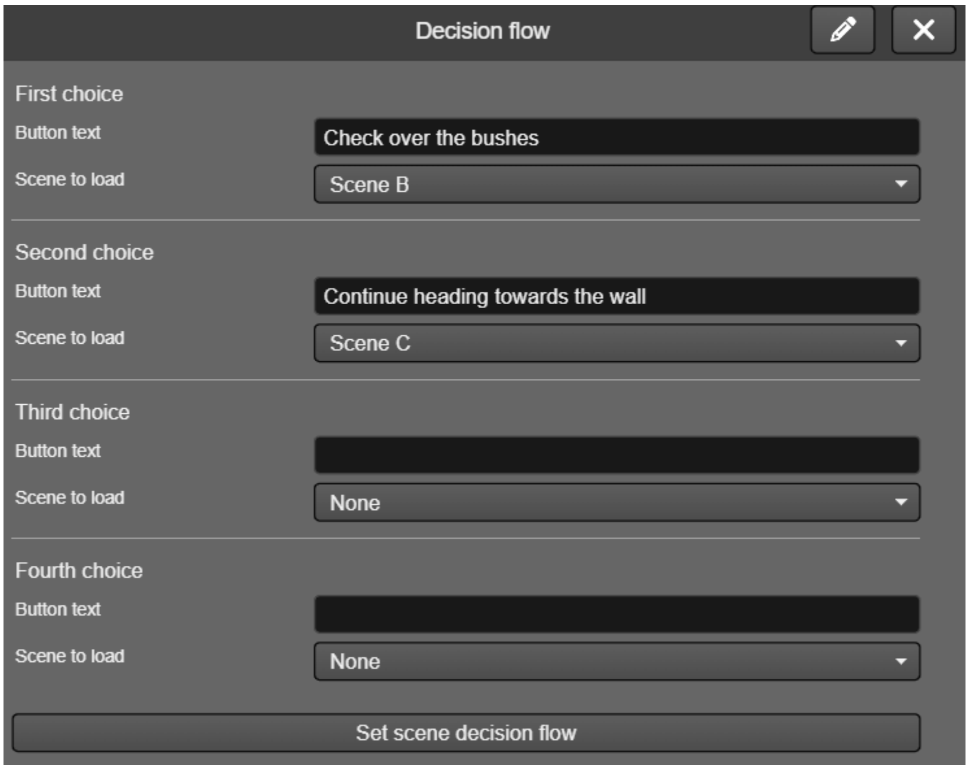
Use the following form to adjust the display of the navigation buttons.

Button graphic: flow-button.png
Display position: bottom right
Distance gap: 50
Font: Roboto Sans
Font size: 30
Font color: #FFFFFF

The second tab in this window allows you to view all scenes that have options set for this decision flow.


Decision flow
Display settings
Scenes flow
Scenes with decision flow set
Scene A
Scene B
Scene C
Scene D
Load selected scene
Options for selected scene
Check over the bushes
Continue heading towards the wall
Load selected option scene

This narrative feature does not have its own actions. Instead, the options are set scene by scene from the left-hand menu "scene", button "decision flow".



First choice
Button text: Check over the bushes
Scene to load: Scene B

Second choice
Button text: Continue heading towards the wall
Scene to load: Scene C

Third choice
Button text:
Scene to load: None

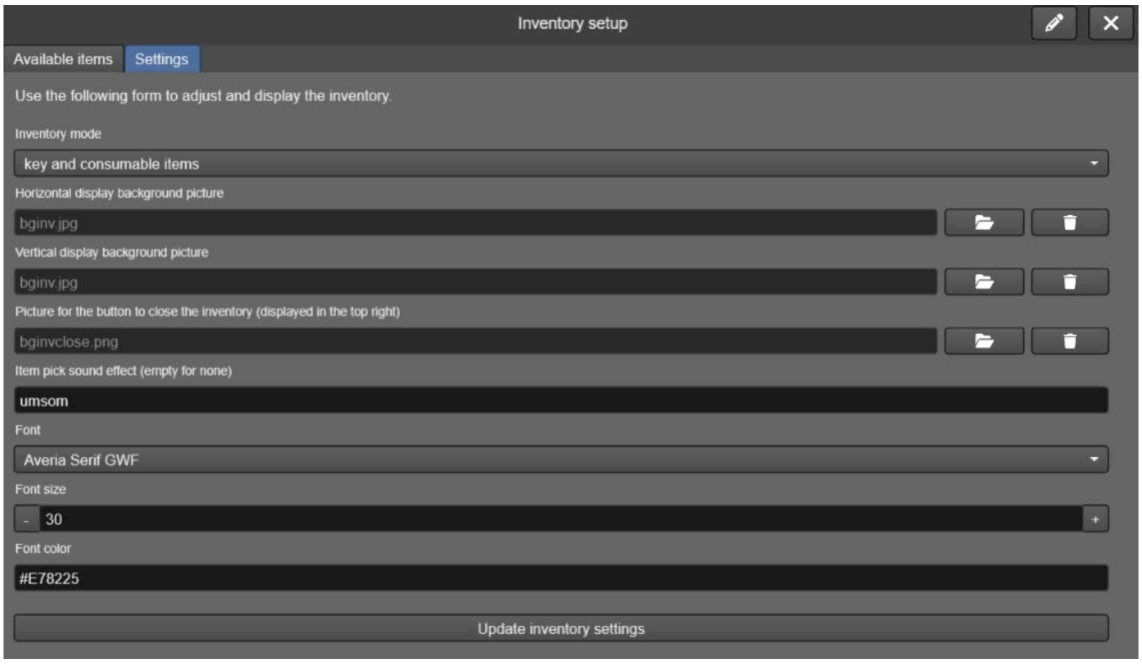
Fourth choice
Button text:
Scene to load: None

5.4. Inventory

TilBuci offers an inventory management system as a way to increase the arsenal of tools available for creating interactive narratives. The items used in this inventory are configured with a name, an image, and optionally, actions to be performed when they are activated, and can be divided into two groups: key items and consumables.

In addition to registering and managing these items, TilBuci also takes care of displaying the inventory, which can be freely configured.

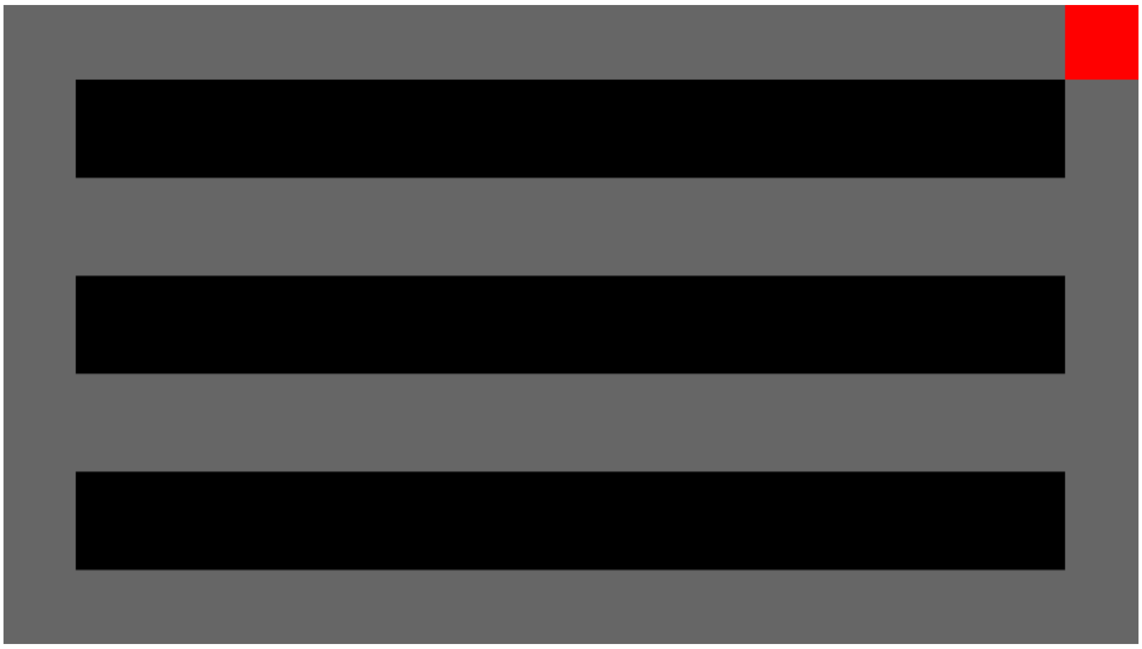
To begin configuration, access the inventory window from the left-hand narrative menu. The settings tab allows you to choose how the inventory will be managed, background images (for horizontal and vertical screens), an image for the close inventory button, and a sound effect to play when an item is activated, in addition to text adjustments for displaying the quantities of consumable items.



Use the following form to adjust and display the inventory.

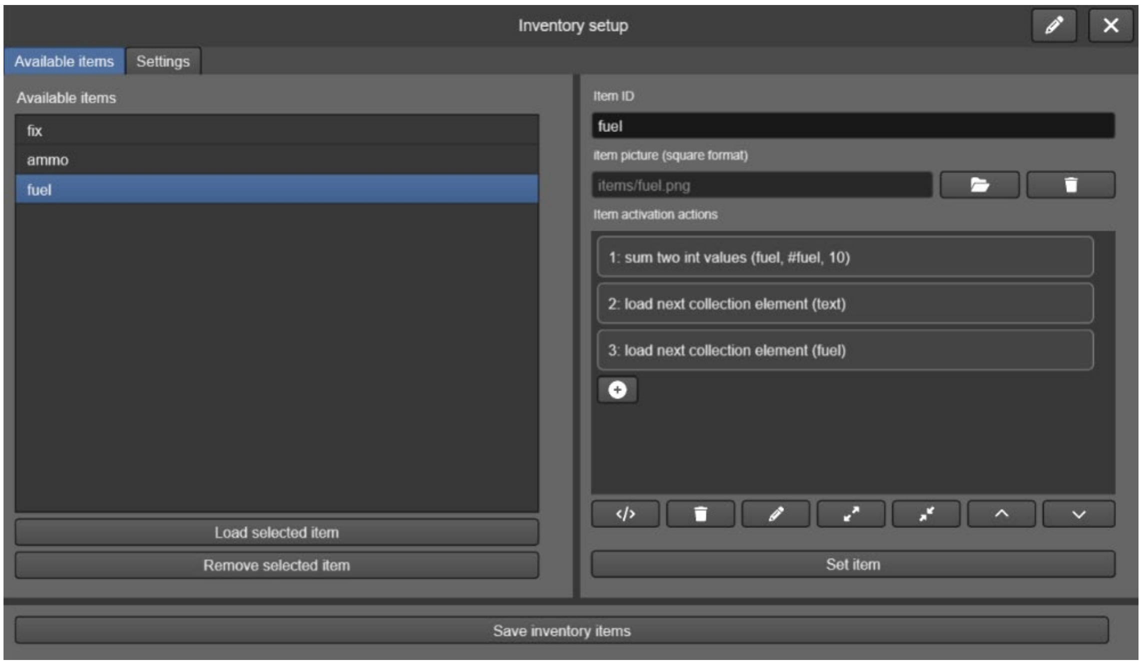
Inventory mode	key and consumable items
Horizontal display background picture	bginv.jpg
Vertical display background picture	bginv.jpg
Picture for the button to close the inventory (displayed in the top right)	bginvclose.png
Item pick sound effect (empty for none)	umsom
Font	Averia Serif GWF
Font size	30
Font color	#E78225

The following picture shows the inventory display layout. The gray area is covered by the background image. The black area shows the rows where the items are arranged, up to 4 in each row. The red area is occupied by the close button image. The height and width of the close button image define the dimensions around the item rows. For vertical screens, the principle is the same, with the close button always remaining in the top right corner.



Tilbuci manages inventories of up to 4 key items, which are always shown in the top row (or on the right in vertical screens), and 8 consumable items, each with its individual quantity, which are displayed 4 in each remaining row.

To register an item, you need to specify a name and an image. The actions, which are optional, are executed whenever the item is activated in the inventory. There is no distinction between key items and consumables for registration purposes. You can register as many items as you want.



Available items
- fix
- ammo
- fuel

Item ID: fuel

Item picture (square format): items/fuel.png

Item activation actions
1: sum two int values (fuel, #fuel, 10)
2: load next collection element (text)
3: load next collection element (fuel)

The actions available for manipulating items are in the "narrative" group.

inventory.show

Displays the inventory screen above the current scene image.
additional optional object fields

- complete: actions to run when the inventory display is closed

inventory.close

Closes the inventory window (clicking the close button has the same effect).

inventory.addkeyitem

Adds a key item to the inventory. If 4 key items have already been added, the action is ignored.

params

1. the item ID

inventory.removekeyitem

Removes a key item from your inventory if it has been added.
params

1. the item ID

inventory.clearkeyitems

Removes all current key items.

inventory.haskeyitem

Check if there is a key item in the inventory.
params

1. the item ID to check

additional optional object fields

- then: actions to run if the item is found
- else: actions to run if the item is not found

inventory.addconsumable

Adds a consumable item to the inventory. If it's already in the inventory, the indicated quantity is added to the current quantity.
params

1. the item ID
2. the amount to add

inventory.removeconsumable

Remove a consumable item completely from the inventory.
params

1. the item ID

inventory.consumeitem

Consumes one unit of an item, removing it from inventory if the quantity reaches zero. This action is performed automatically when a consumable item is clicked in the inventory interface.
params

1. the item ID

inventory.clearconsumables

Removes all current consumable items.

inventory.tostring

Stores all items in the current inventory in a string variable. This action, along with *fromstring*, allows you to record the inventory status along with other player data.
params

1. the string variable to receive the inventory data

inventory.fromstring

Restores the state of an inventory based on the value of a string variable.
params

1. the string variable to load the data from

inventory.keytostring

Stores all key items in the current inventory in a string variable.
params

1. the string variable to receive the inventory data

inventory.keyfromstring

Restores the key items of an inventory based on the value of a string variable.
params

1. the string variable to load the data from

inventory.constostring

Stores all consumable items in the current inventory in a string variable.
params

1. the string variable to receive the inventory data

inventory.consfromstring

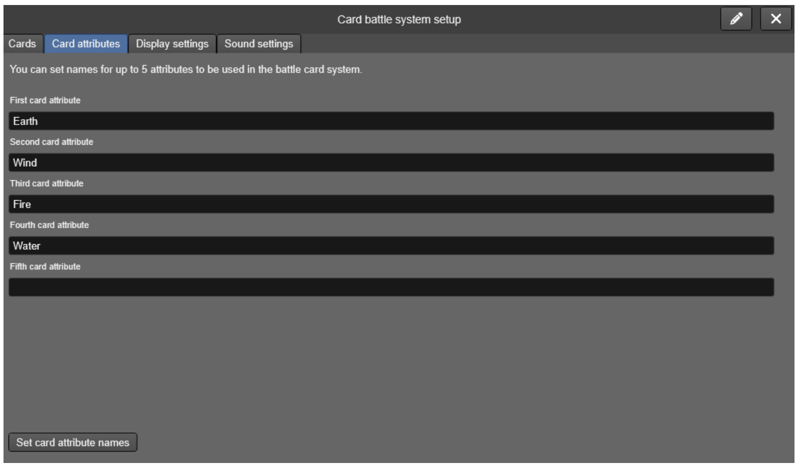
Restores the consumable items of an inventory based on the value of a string variable.
params

1. the string variable to load the data from

5.5. Card battle

To help add elements of randomness to the narratives, TilBuci features a simplified battle system in the form of card comparisons. This allows for creating situations in the stories where the player faces opponents with varying outcomes. This system is quite simplified in TilBuci to make it easy to configure and use.

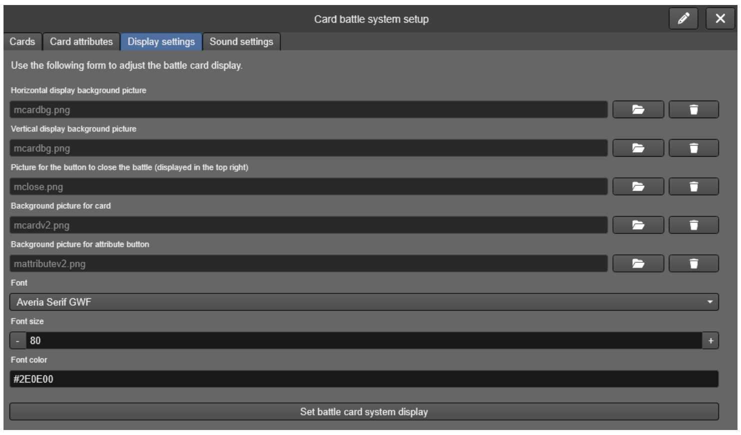
Each card used contains up to 5 values, which are called attributes. In the battle system configuration window, accessed via the left-hand narrative menu, you can name the attributes according to the theme of your production - attributes with blank names are ignored. You can either type the name directly or specify a string variable containing that name, such as $attr1, which can be useful for translations.



You can set names for up to 5 attributes to be used in the battle card system.

First card attribute
Earth
Second card attribute
Wind
Third card attribute
Fire
Fourth card attribute
Water
Fifth card attribute

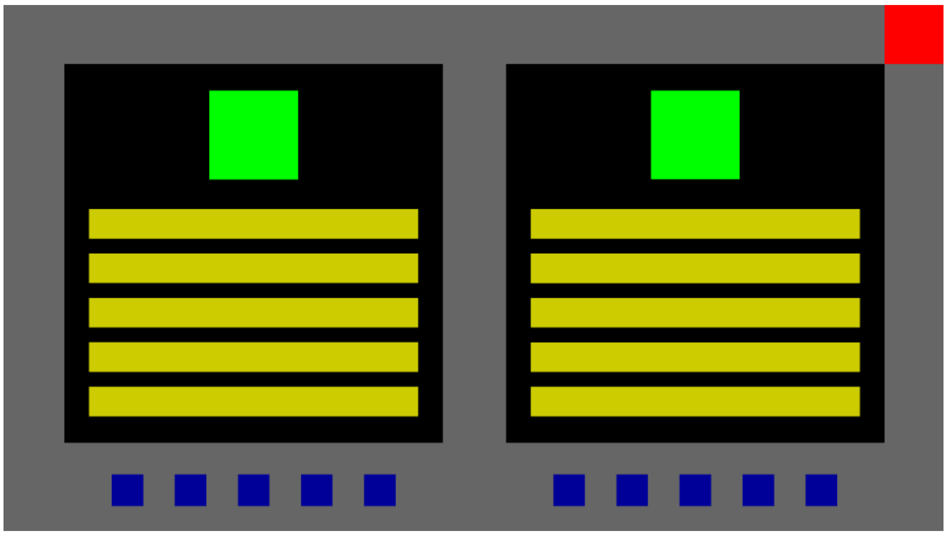
For TilBuci to manage the display of battles, it is necessary to configure the screen's graphic elements, including background images (horizontal and/or vertical), a button to end the battle, background images for cards and texts, attribute values, and adjustments to the font used.



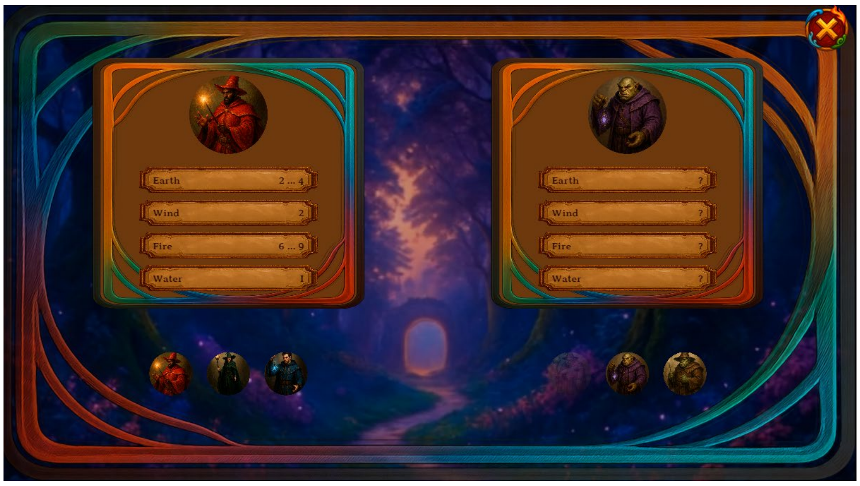
Use the following form to adjust the battle card display.

Horizontal display background picture
mcardbg.png
Vertical display background picture
mcardbg.png
Picture for the button to close the battle (displayed in the top right)
mclose.png
Background picture for card
mcardv2.png
Background picture for attribute button
mattributev2.png
Font
Averia Serif GWF
Font size
80
Font color
#2E0E00

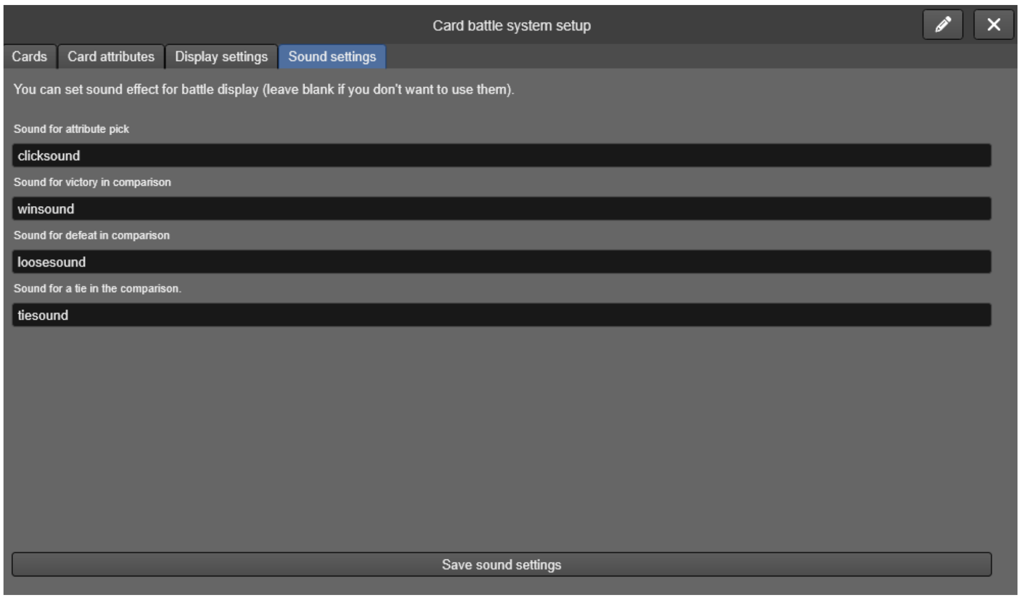
The following picture shows how these elements are displayed on the screen.



The gray area in the previous image is occupied by the picture set as the background. The red field shows the image of the end battle button. The black square shows the card's background – prefer square images for this. In horizontal view, the cards are shown side-by-side, while in vertical view, one above the other. The yellow rectangles contain the attribute buttons (they are displayed according to the number of attributes used) – produce their image considering the size of the card's background. The green area shows the illustration of the current card, while the blue areas show the images of all cards involved in the battle. The distribution of graphics on the screen considers the width and height of both the "End Battle" button and the attribute background. The following picture shows the application of graphics in this layout.



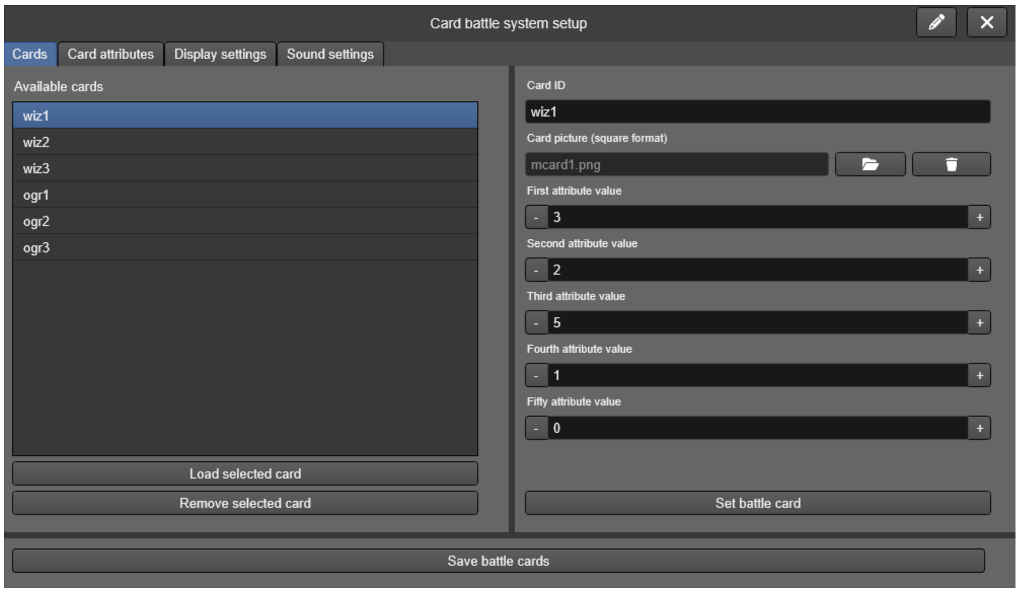
It is also possible to set audio feedback for some battle actions: selecting an attribute for comparison, winning a comparison, losing, or drawing it. Simply use the name of the sound effect to play (configured in the sound effects contraptions) - it is also possible to use a string variable that contains the name of the audio to be used, such as *$mysound*.



You can set sound effect for battle display (leave blank if you don't want to use them).

Sound for attribute pick: clicksound
Sound for victory in comparison: winsound
Sound for defeat in comparison: loosesound
Sound for a tie in the comparison.: tiesound

Finally, you need to configure the cards. This step does not distinguish between cards used by the player or the opponent. For each card, you need to provide a name, a picture (always square), and the basic values for all attributes used. You can register as many cards as you want.



Available cards: wiz1, wiz2, wiz3, ogr1, ogr2, ogr3

Card ID: wiz1
Card picture (square format): mcard1.png
First attribute value: 3
Second attribute value: 2
Third attribute value: 5
Fourth attribute value: 1
Fifty attribute value: 0

The battle works according to a simple rule. Two cards are displayed, one for the player and one for the opponent. These cards show the current values of each attribute for the player, while the opponent's card shows these values hidden. The value of each attribute is calculated from the card's base value, adding or removing a fixed gain, plus a random amount that can be set via actions. When the comparison value is fixed, only a number is displayed. When it has a random influence, it is displayed as a range, such as 1...5. The losing card is discarded. At the end of the battle, if the player still has cards available, they are the winner. If they have run out of cards, they lose.

The actions related to the card battle are found at the narrative group.

battle.show

Start a card battle. You need to specify one to five cards for yourself and your opponent. The card names should be separated by commas, and you can specify them as a string variable sequence, such as *$card1,$card2,$card3*.
params
1. the player cards list
2. the opponent cards list

additional optional object fields
- win: actions to run if the player wins the match
- loose: actions to run if the player looses the match or clicks the end battle button

battle.close

Ends a card battle. This action is automatically executed at the end of the battle or when the player clicks the button to end the battle.

battle.setattribute

Sets adjustments to be applied to one of the player's card attributes. The fixed gain (which can be negative) is always added (subtracted) from the base value registered on the card. The random gain value indicates a number to be randomly added each time the card is used (for example, the value 3 indicates that, at the time of comparison, a total from 0 to 3 will be added to the current value). The random reduction works in reverse, indicating a total that can be subtracted from the current value. This action is quite useful when associated with the use of inventory items.
params
1. the attribute to affect (1 to 5)
2. fixed gain value
3. random gain value
4. random reduction value

battle.setopponent

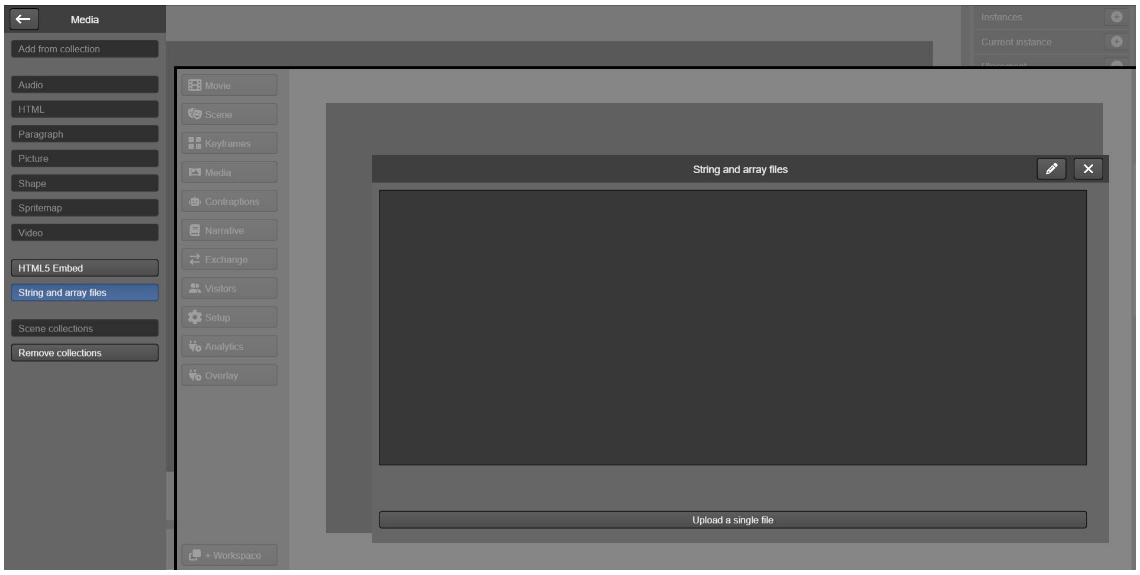
It works similarly to the *battle.setattribute* action, but applies the values to the opponent's cards.

params

1. the attribute to affect (1 to 5)
2. fixed gain value
3. random gain value
4. random reduction value

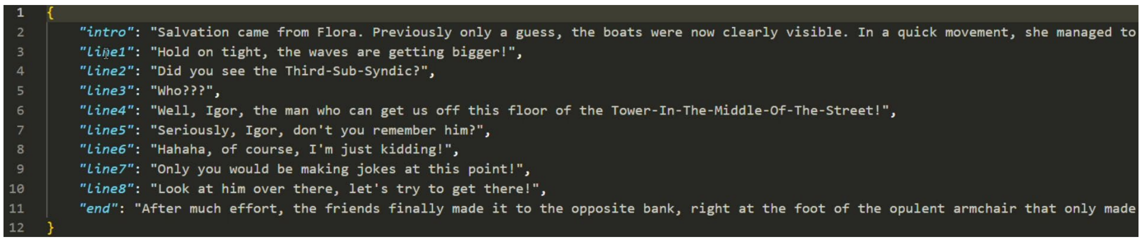
6. String files and Arrays

In addition to traditional media files such as pictures, videos, and audio, TilBuci supports sending text files in JSON format that can contain information used in your productions. These files can be used both as text sources and as values in a list, which we call an "array". You can upload as many JSON files of strings and arrays as you want for your movie. Avoid using spaces or special characters in the filename for best results.



6.1. String files

These text files are JSON-formatted ones that can be loaded and unloaded at any time into variables, allowing your creations to contain large volumes of text with reduced impact on both load time and memory usage.



```
{
    "intro": "Salvation came from Flora. Previously only a guess, the boats were now clearly visible. In a quick movement, she managed to
    "line1": "Hold on tight, the waves are getting bigger!",
    "line2": "Did you see the Third-Sub-Syndic?",
    "line3": "Who???",
    "line4": "Well, Igor, the man who can get us off this floor of the Tower-In-The-Middle-Of-The-Street!",
    "line5": "Seriously, Igor, don't you remember him?",
    "line6": "Hahaha, of course, I'm just kidding!",
    "line7": "Only you would be making jokes at this point!",
    "line8": "Look at him over there, let's try to get there!",
    "end": "After much effort, the friends finally made it to the opposite bank, right at the foot of the opulent armchair that only made
}
```

The string files used in TilBuci are quite simple: they contain name-value pairs. Once the file is loaded, string variables with the names found in the file become available with the indicated content. When loading a new text file, the variables defined by the previous file are deleted.

string.loadfile

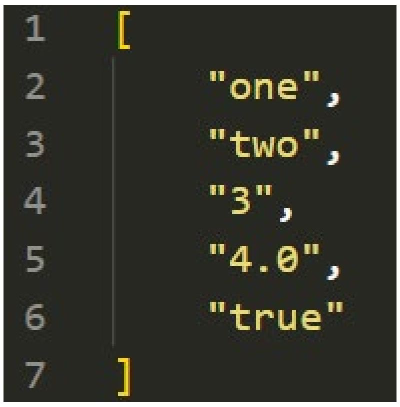
Loads a string JSON file registered using the media manager (replaces the content of any previously loaded strings file).

params
1. the string file name (without ".json")

additional optional object fields
- success: actions to run on successful file load
- error: actions to run if the file fails to load

6.2. Arrays

Array files also follow the JSON format and consist of only a list of strings.



```
[
    "one",
    "two",
    "3",
    "4.0",
    "true"
]
```

These strings can contain actual text, but also numeric or boolean values. When a file is loaded, the values become available as an array with the same name, and can be manipulated using common array actions. Unlike string files, loading a new array does not delete the previous one from memory - you must use the corresponding action for that.

The information in an array is stored and retrieved from its index, a number that indicates the current position for reading/writing information.

When current information is saved, such as when the *data.savestate* action is called, loaded arrays are not saved. To save them, use the *array.tostring* action beforehand to save them into a string variable (and *array.formstring* to load them back into memory afterward).

array.loadfile

Loads a file into an array with the same name.
params
1. the array file name (without ".json")

additional optional object fields

- success: actions to run on successful file load
- error: actions to run if the file fails to load

array.create

Create an empty array if it doesn't already exist.
params
1. the array name

array.remove

Deletes an existing array.
params
1. the array name

array.push

Adds a value to the end of the array.
params
1. the array name
2. the value to add

array.set

Sets the value of a position in the array. If the array length is smaller than the specified position, empty values will be added until the specified value is reached. This action does not change the current index of the array.
params
1. the array name
2. the position to add
3. the value to add

array.get

Retrieves the value of a position in the array as a string. If the array is smaller than the indicated position and does not exist, an empty string will be returned. This action does not change the current index of the array.
params
1. the array name
2. the position to load
3. the string variable to receive the value

array.getint

Retrieves the value of a position in the array as an int. If the array is smaller than the indicated position and does not exist, 0 will be returned. This action does not change the current index of the array.
params
1. the array name
2. the position to load
3. the int variable to receive the value

array.getfloat

Retrieves the value of a position in the array as a float. If the array is smaller than the indicated position and does not exist, 0 will be returned. This action does not change the current index of the array.
params
1. the array name
2. the position to load
3. the float variable to receive the value

array.getbool

Retrieves the value of a position in the array as a boolean. If the array is smaller than the indicated position and does not exist, false will be returned. This action does not change the current index of the array.
params
1. the array name
2. the position to load
3. the bool variable to receive the value

array.clear

Remove all values from an array.
params
1. the array name

array.current

Retrieves the value of the current index position in the array as a string. If the array is empty, an empty string will be returned.
params
1. the array name
2. the string variable to receive the value

array.currentint

Retrieves the value of the current index position in the array as an int. If the array is empty, 0 will be returned.
params
1. the array name
2. the int variable to receive the value

array.currentfloat

Retrieves the value of the current index position in the array as a float. If the array is empty, 0 will be returned.
params
1. the array name
2. the float variable to receive the value

array.currentbool

Retrieves the value of the current index position in the array as a boolean. If the array is empty, false will be returned.
params
1. the array name
2. the bool variable to receive the value

array.next

Moves the array index by one position and returns its value as a string. If the end of the array is reached, the index is moved to the first position.
params
1. the array name
2. the string variable to receive the value

array.nextint

Moves the array index by one position and returns its value as an int. If the end of the array is reached, the index is moved to the first position.
params
1. the array name
2. the int variable to receive the value

array.nextfloat

Moves the array index by one position and returns its value as a float. If the end of the array is reached, the index is moved to the first position.
params
1. the array name
2. the float variable to receive the value

array.nextbool

Moves the array index by one position and returns its value as a boolean. If the end of the array is reached, the index is moved to the first position.
params
1. the array name
2. the bool variable to receive the value

array.previous

Moves the array index to the previous position and returns its value as a string. If the beginning of the array is reached, the index is moved to the last position.
params
1. the array name
2. the string variable to receive the value

array.previousint

Moves the array index to the previous position and returns its value as an int. If the beginning of the array is reached, the index is moved to the last position.
params
1. the array name
2. the int variable to receive the value

array.previousfloat

Moves the array index to the previous position and returns its value as a float. If the beginning of the array is reached, the index is moved to the last position.
params
1. the array name
2. the float variable to receive the value

array.previousbool

Moves the array index to the previous position and returns its value as a boolean. If the beginning of the array is reached, the index is moved to the last position.
params
1. the array name
2. the bool variable to receive the value

array.setindex

Sets the current index of the array. If the specified position is greater than the size of the array, the index is moved to the last available position.
params
1. the array name
2. the index position to set

array.getindex

Retrieves the current index position of an array.
params
1. the array name
2. the int variable name to receive the value

array.tostring

Stores the current array's entire contents in a string variable.
params
1. the array name
2. the string variable name to receive the data

array.fromstring

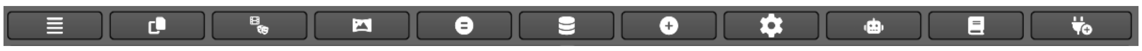
Recreates an array from the contents of a string variable.

params

1. the array name
2. the string variable name to get the data from

7. Code assist

To simplify the action coding process, TilBuci provides some assistants to help the script creation. On every scripting window there are some buttons to display these tools.



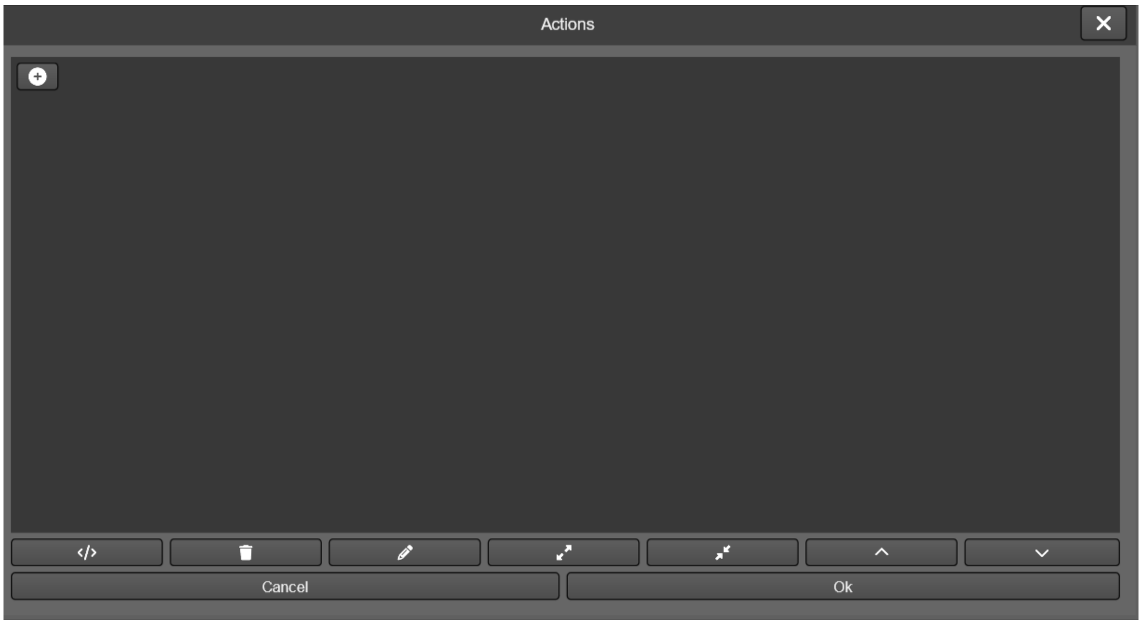
These buttons actions are, in order:

1. return to the block editor
2. copy current action script
3. show movie and scene actions
4. show media/instance actions
5. show variable management actions and conditions
6. show data and input handling actions
7. show additional actions, like system and timer
8. show runtime actions
9. show contraption-related actions
10. show narrative actions
11. show available plugin actions

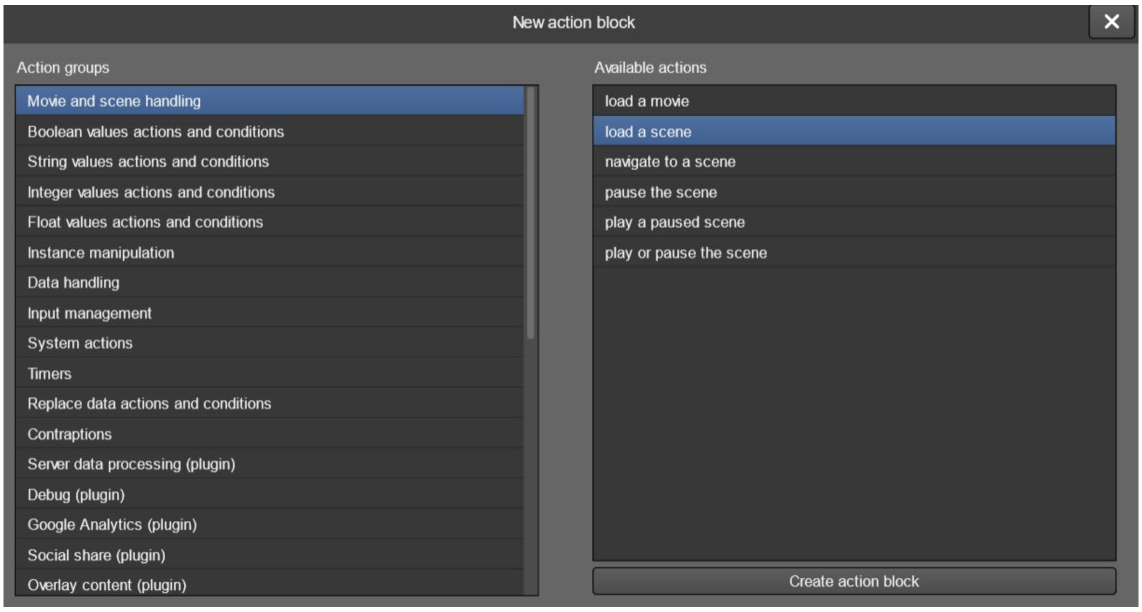
All windows shown from these buttons have tools to help code creation. Just set what you want and click the buttons with the copy icon – the code will be copied to clipboard so you can paste it on your creations. Please note that this copy functions will only work if TilBuci is running from secure "https" urls due to browser policies. You will also find the "show action" buttons that will display the action at the "Action/value" area, from which you can manually copy the result – this will work from both https and http.

8. Block editor

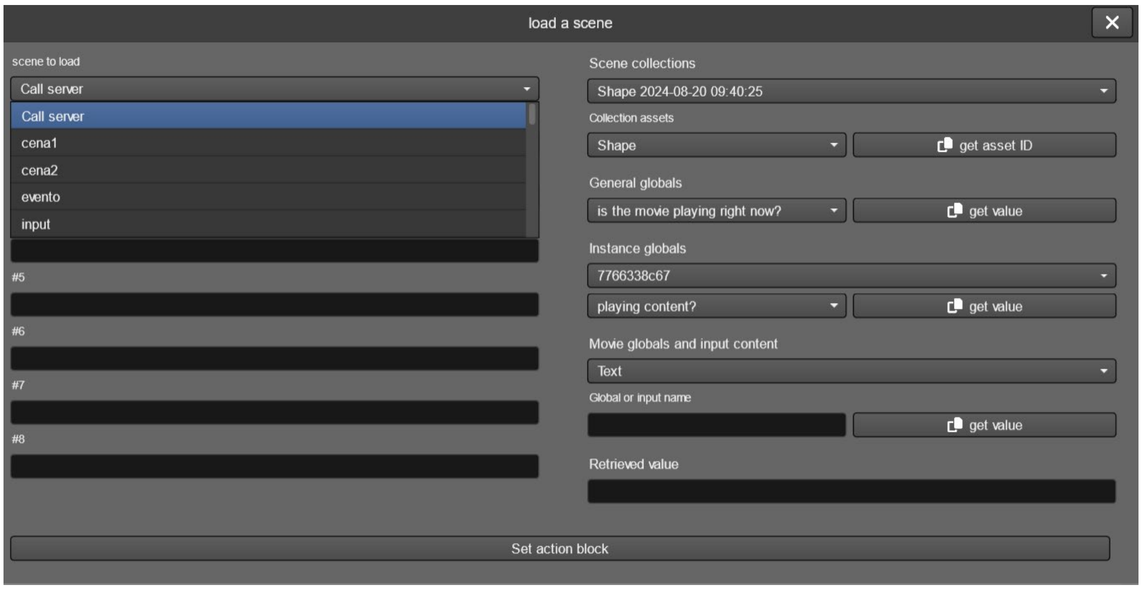
Even though you can write the code for your actions in TilBuci, using or not the assistants, the simplest way to create interactions is through the block editor. The action blocks are a visual way to enter commands and the software's default method.



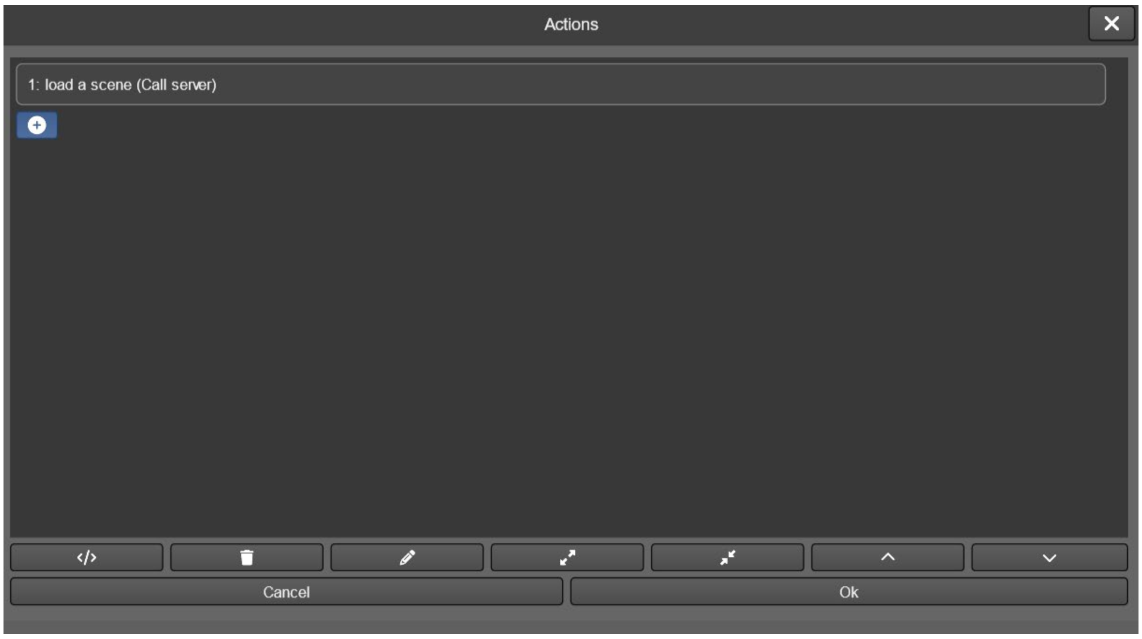
The block area always displays a + button used to create a block. Click it to show the action groups available. By clicking on one of them, you'll see the available actions. Select the one you want and, them click on "create action block".



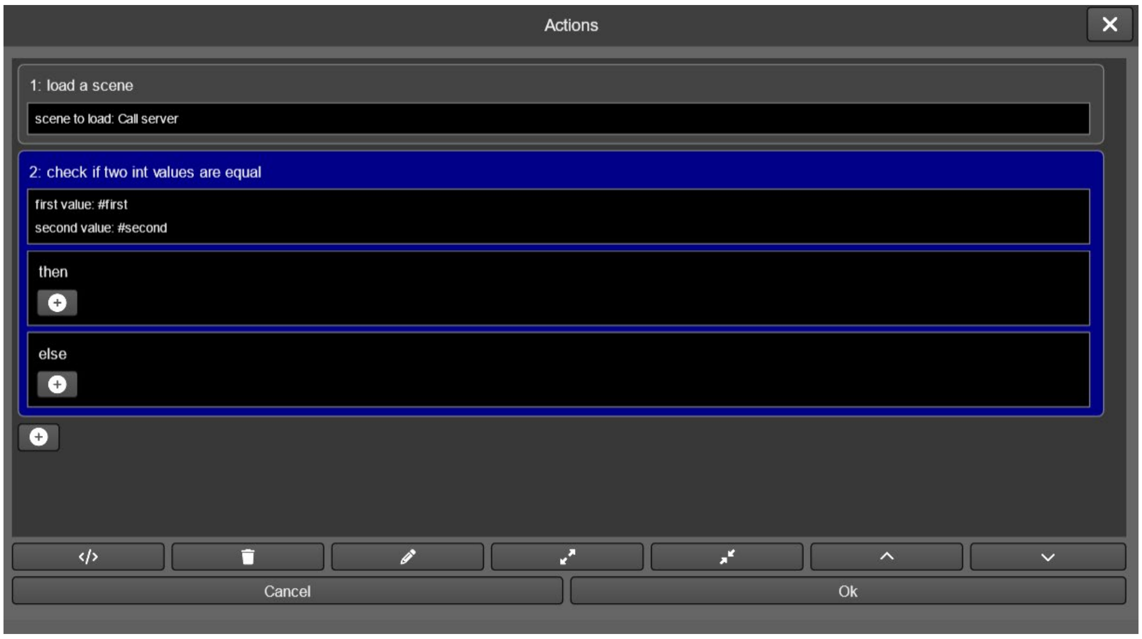
The next window will show all necessary parameters to finish setting your action.



You can add as many blocks as you want to an action: just press the + button to add another one. The actions will be run at the numbered order.

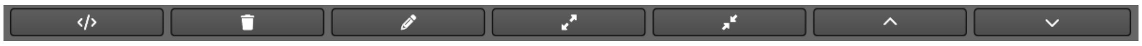


Some blocks, like conditions, can hold themselves further actions. In these cases, additional + buttons are displayed where necessary.



1: load a scene
scene to load: Call server
2: check if two int values are equal
first value: #first
second value: #second
then
else

Just below the blocks area you'll find some buttons to help your coding.



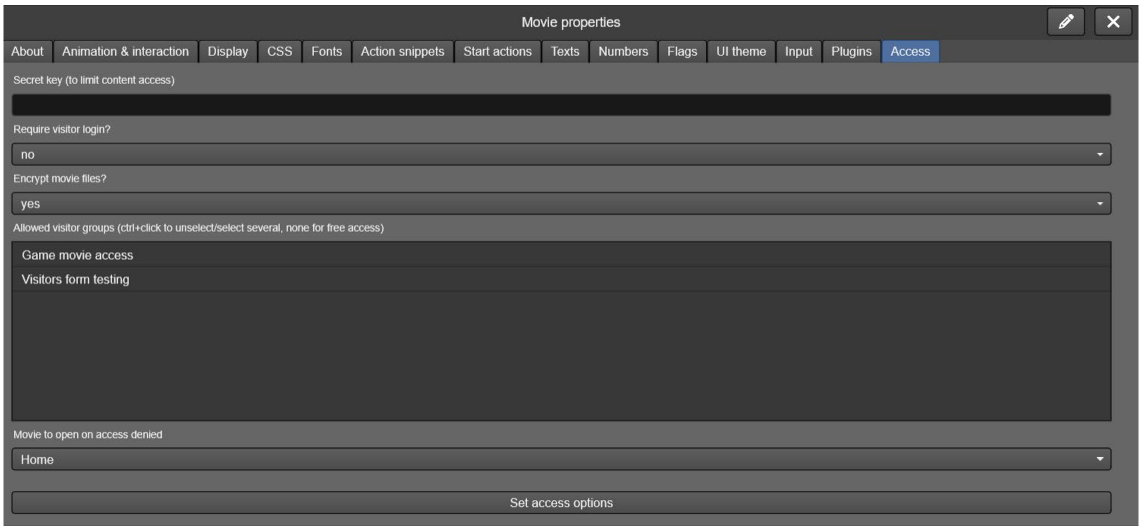
They represent, in order:

1. switch to the code view (you can switch between block and code display at any time, both methods are 100% compatible)
2. delete the selected block
3. edit the selected block
4. expand the selected block (or mouse center button click)
5. reduces the selected block view
6. move selected block up in execution order
7. mode selected block down in execution order

9. Content encryption

TilBuci movies are saved as folders containing the media files and also several plain text, json-encoded ones that describe all your movie settings and interactions. These json files are, by default, saved unencrypted. While this is good for many cases, you may need to protect your creation from access by other people.

TilBuci offers an encryption option for your movies. Just open the access tab of the movie properties, change the "encrypt movie files?" setting to true and click the set access options button.



Movie properties

Secret key (to limit content access)

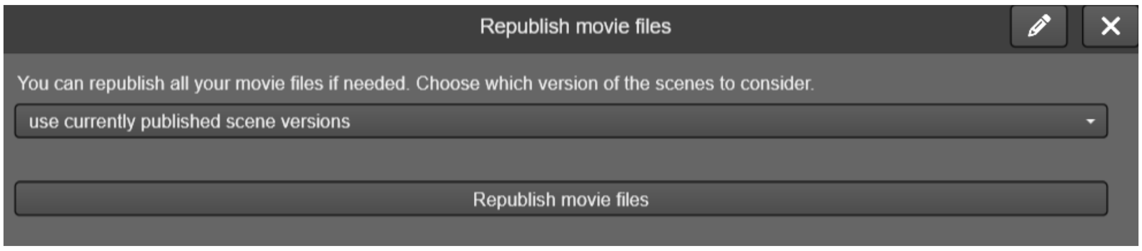
Require visitor login?
no

Encrypt movie files?
yes

Allowed visitor groups (ctrl+click to unselect/select several, none for free access)
Game movie access
Visitors form testing

Movie to open on access denied
Home

Every time you change this setting, go to the movie left menu and click the republish button. This will force all movie json files to be rewritten, forcing the encryption.



Republish movie files

You can republish all your movie files if needed. Choose which version of the scenes to consider.
use currently published scene versions

Encrypted movies are compatible with all exporters, but remember that movies exported for importing on other TilBuci installations are always saved unencrypted.

9.1. Custom encryption

TilBuci uses a standard AES-256 encryption method which should be sufficient for most cases. If you need to bring your own encryption method/key, however, you can do it following some simple steps.

First, you must save your movie unencrypted – do not use the TilBuci default encryption. Then, after exporting your content, load and encrypt all movie json files using your custom encryption.

Now, access the index.html file used to load your content. You must add your own decryption method to it – this method must receive a single parameter, the encrypted text, and return a string containing the decrypted one.

The last thing to do is look for the TilBuci initialization script at the index.html file. It looks like this:

```
<script type="text/javascript">
    lime.embed ("TilBuci", "openfl-content", 0, 0, { parameters: { "mode" : "player", "movie":
    "mymovie", "scene": "", "decrypt": myDecryptMethod } });
</script>
```

You must add the "decrypt" property to the initialization parameters, referring to your custom decryption method.

For a simple example of custom decryption, please download this exported website:

http://tilbuci.com.br/files/customencryption.zip