BST Reference Sheet

This document references all of the current actions and checkers available in BST.
This version of the document has been updated for ***OpenBST 2.0 Beta 2***
This document is available under the CC-BY-NC-SA 4.0 International licence.

License details : https://creativecommons.org/licenses/by-nc-sa/4.0/

Website : https://utybo.github.io/BST/
More recent versions may be available here : https://utybo.github.io/BST/#downloads
Document version : 1.1
Changes since last version : Fixed typos

GENERAL INFORMATION

Most of the time, if an action, a checker or an NND expects you to provide a node, you can provide either a node ID, an alias (if you specified an alias before) or a variable that contains an alias or a node ID.
If an action or a checker expects you to provide a value, you can generally use a variable as well, as long as what is contained in the variable corresponds to the value expected

SYNTAX CHEATSHEET

Regular node beginning	1:
Logical node beginning	1:&
Virtual node beginning	1:>
Include a var. in a node	${variable}
Incl. a v. node in a node	${>1}
Incl. a l. node in a node	${&1}
Option	:Option text|nnd
Conditional option	:Option text|nnd|[checker:a,b]
Option with Scripts	:Option text|nnd|{action:a,b}{action:a,b}

Note : you can combine both the checker and the action(s) here. If you only have one checker or one action, you can ommit the brackets

QUESTIONS? SUGGESTIONS? SUBMIT A TICKET AND WE'LL HELP!
https://github.com/utybo/BST/issues

In this document :

BST REFERENCE TABLE : Actions in basic BST (vanilla, without modules)		
Name	Syntax	Description

add	`add:<putIn>,<a>,<b>`
	`add:<a>,<b>`
		Does a mathematical addition. The numbers *a* and *b* are added (they can be variables), and the variable named *putIn* is set to the result. If *putIn* is not specified, then the result will be placed in *a* which has to be a variable in this case.

assert	`assert:<checker>`	If the checker returns *true*, nothing is done. However, if it returns *false*, make the story crash. **Only available in test environments.**
bound	`bound:<variable>,<minimum>,<maximum>`	Ensures that *variable* is within the bounds determined by the *minimum* and the *maximum*. While *variable* has to be a variable, *minimum* or *maximum* can either be a variable or a raw value.
call	`call:<logical node>`	Executes the *logical node* specified, without actually « going to » it. Useful for subroutines.
clone	`clone:<to clone>,<to set>`	Duplicates a variable. The variable named *to set* will have its value set to the one of *to clone* – however, future modifications of *to clone* will have no effect on *to set* and vice versa.
crash	`crash:`	Throw an exception immediately, terminating the execution of the current test. **Only available in test environments.**
decr	`decr:<variable>`	Decrements variable by one.

div	`div:<putIn>,<a>,<b>`
	`div:<a>,<b>`
		Does a mathematical euclidian division. The numbers *a* and *b* will be divided (they can be variables), and the variable named *putIn* is set to the result. If *putIn* is not specified, then the result will be placed in *a* which has to be a variable in this case.
		Note that this is an euclidian division : the result will be rounded.

exit	`exit:`	Closes the story instantly
fail	`fail:`	Equivalent to the crash action. **Only available in test environments.**
incr	`incr:<variable>`	Increments *variable* by one
input	`input:<variable>,Text to show`	Asks the user for a value, which is then passed to *variable*. The text to show to the user can contain commas.

mod	`mod:<putIn>,<a>,<b>`
	`mod:<a>,<b>`
		Determine a modulo, which is the remainder of an euclidian division. The numbers *a* and *b* will be divided (they can be variables), and the variable named *putIn* is set to the remainder of the euclidian division. If *putIn* is not specified, then the remainder will be placed in *a* which has to be a variable in this case.
		Note that this is an euclidian division : the result will be rounded.

mul	`mul:<putIn>,<a>,<b>`
	`mul:<a>,<b>`
		Does a mathematical multiplication. The numbers *a* and *b* are multiplied (they can be variables), and the variable named *putIn* is set to the result. If *putIn* is not specified, then the result will be placed in *a* which has to be a variable in this case.

rand	`rand:<variable>,<maximum>`
	`rand:<variable>,<minimum>,<maximum>`
		Randomly pick a number between *minimum* (or 0 if no minimum is given) and *maximum*, inclusive, meaning that the result can be *minimum*, *maximum*, or any number inbetween.

Name	Syntax	Description
set	`set:<variable>,<value>`	Sets the variable *variable* to the value *value*

sub	`sub:<putIn>,<a>,<b>`
	`sub:<a>,<b>`
		Does a mathematical substraction. The numbers *a* and *b* are subtracted (they can be variables), and the variable named *putIn* is set to the result. If *putIn* is not specified, then the result will be placed in *a* which has to be a variable in this case.

BST REFERENCE TABLE : Actions i

Module	Name	Syntax
BDF	bdf_apply	bdf_apply:<name>,<prefix> bdf_apply:<name> bdf_apply:!<variable>,<prefix> bdf_apply:!<variable>
BRM	brm_load	DEPRECATED
HTB	htb_base64	htb_base64:<resource>,<variable>
HTB	htb_import	htb_import:<resource>,<variable>
JSE	jse_autoimport	DEPRECATED
JSE	jse_eval	jse_eval:<variable>,<javascript>
JSE	jse_import	DEPRECATED
JSE	jse_reset	DEPRECATED
SSB	ssb_ambient	ssb_ambient:<resource>
SSB	ssb_play	ssb_play:<resource>
SSB	ssb_stop	ssb_stop:<resource>
UIB	uib_init	uib_init:
UIB	uib_set	uib_set:<component id>,<value>
UIB	uib_setprop	uib_setprop:<component id>,<property name>,<value>
UIB	uib_setvisible	uib_setvisible:<boolean>
XBF	xbf_call	xbf_call:<name>,<node>
XSF	xsf_exec	xsf_exec:<name>,<function> OR xsf_exec:<name>,<function>,<putIn>

n BST modules
Description
Import all the values from the bdf file named *name*, or named by the content of the variable *variable*. Each import is prefixed by *prefix* if specified.
This action used to trigger the loading of all resources. This is now useless, as resources are loaded automatically.
Import the content of resource named *resource* in the variable named *variable*, encoded into Base64
Import the raw content of resource named *resource* in the variable named *variable*.
This action is deprecated, but is still recognized by OpenBST so that stories with it do not crash.
Puts the result of the javascript operation *javascript* into the variable named *variable*
This action is deprecated, but is still recognized by OpenBST so that stories with it do not crash.
This action is deprecated, but is still recognized by OpenBST so that stories with it do not crash.
Plays the music resource named *resource* in background, as an ambient music. It is automatically looped.
Plays the sound resource named *resource* as a one-shot sound.
Stops the currently playing ambient sound, if any. Does not do anything if there are no ambient sounds currently playing.
Initializes UIB
Main entry point for setting component texts or values. Please refer to the manual for further details
Sets the property named *property name* of component *component id* to *value*
If *boolean* is true, show UIB. If *boolean* is false, hide UIB.
XBF equivalent of the action *call*, with the only addition being that the logical node is in the resource story named *name*
Launches the function named *function* in Javascript file named *name* and, if *putIn* is specified, puts the result from the function in the variable *putIn*

BST REFERENCE TABLE : Checkers in BST modules

Name	Syntax	Description
greater	`greater:<a>,<b>`	Checks if *a* is strictly greater than *b*
greaterequ	`greaterequ:<a>,<b>`	Checks if *a* is greater than or equal to *b*
less	`less:<a>,<b>`	Checks if *a* is strictly less than *b*
lessequ	`lessequ:<a>,<b>`	Checks if *a* is less than or equal to *b*
equ	`equ:<a>,<b>`	Checks if *a* is equal to *b*
not	`not:<a>,<b>`	Checks if *a* is not equal to *b*
jse_eval	`jse_eval:<javascript>`	Checks if the *javascript* code returns true. If it does, than this checker returns true. If it returns false or any other value, the checker returns fase.