# TRIGGERS USER GUIDE

## TELLNEXT ESSENTIALS EDITION

Here you will find, a guide for configuring the triggers in TellNext.

# Table of Contents

| | |
|---|---|
| 09/02/2016 | Creation |
| 16/02/2016 | Some minor changes |
| 18/02/2016 | Added some explanations about special values, predefined Lua variables, about triggers in general. Refactored several names, split tables that didn't fit in a single page, modified a few descriptions, etc. |
| 26/02/2016 | Refactored several names to keep backward compatibility. |
| 26/05/2016 | Move Voice values to Modules |
| 13/06/2016 | Moving values to Configuration-file-trigger |
| 22/06/2016 | Rename: Profile -> Process and add the level "Application" to Profile. |
| 15/09/2016 | Application -> Profile et Profile -> Process. Adding the "OnConnectScript" and "OnKeyScript". |
| 22/11/2017 | Adding the Lua library « Keyboard ». |
| 14/01/2019 | "@" prefix denotes Process a Variable, and "$" prefix denotes a Profile variable |
| | |
| | |
| | |
| | |
| | |

# CONVENTIONS

## VALUE TYPES

| Name in this file | Description |
|---|---|
| “boolean” | A property that can be set or not set, enabled or disabled, with only two possible values, *true* and *false*. |
| “integral” | A number without decimal part such as *123*, *-50*, but not such as *3.14*.<br>The minimum and maximum values depend on the property. |
| “string” | A piece of text. The minimum and maximum length depend on the property. |
| “Lua expression” | A piece of text that contains a value Lua expression. |
| “scancode” | A value of type “string” that contains exactly 4 characters, each of them being a valid hexadecimal character (from 0 to 9 and from A to F). It represents the combination of a key and optional modifiers (control, shift, alt, etc...). |
| “color” | A value of type “string” that represents a color. Colors are defined as hexadecimal values. “AARRGGBB”, “RRGGBB”, “RGB”. |
| “enum” | A value of type string that can only have one value among several predefined values. |
| “object” | Represent a (fixed length) set of values. Each value is identified by a name. An object’s value is determined by it’s children, each of them being either a string, an integral, or any other value type (including objects). |
| “array” | Represent a (variable length) set of values. Each value is identified by a position (starting from 0 and consecutive). An array’s value is determined by its children, each of them being either a string, an integral, or any other value type (including arrays). |
| “array of objects” | An array that only contains values of type “object” as children. |

# CONFIGURATION

Main changes from current version:

- Go-Basic is replaced by LUA
- The configuration file is in JSON format
- The expressions and conditions will accept LUA operators.

## SECTIONS

### 1. TRIGGERS

A trigger is a procedure that may start when certain conditions are verified and that will perform a set of actions. Triggers can be used for automation tasks (Automator), voice directed transactions (TTS + ASR), screen reformatting (to be defined), etc...

#### *1.1. Trigger conditions*

For the procedure of a trigger to be started, some conditions must be verified. These conditions can depend on:

- The content of the terminal screen;
- The position of the cursor;
- The value of profile-variables and process-variables (see the scripting documentation for more information about these variables);
- Other more complex conditions.

The conditions of a trigger are tested under certain circumstances. Most of the time it will be as a result of a change in the content of the terminal screen, but it can occasionally be in response to more specific events. When all the conditions of a trigger are verified, the procedure is started. If the conditions of a trigger are verified, the conditions of the other triggers will not be tested as only one trigger can be processed each time. The conditions of triggers are processed in the same order as triggers are defined.

### *1.2. The actions*

When all the conditions of a trigger are verified, several actions can be performed such as running a script, assigning values to application/process-variables, automatically sending data to the Telnet server ("auto triggers"), initiating a vocal engine procedure (TTS + ASR), controlling a PutToLight or a WeightOnDemande device, etc... See the chapter about trigger properties for more detailed information.

### *1.3. Special tokens in text properties*

Several text (string) configuration properties may contain special "tokens". These special tokens are replaced by the value they represent when the property is used. There are several types of special tokens and each of them is defined further. The types of special tokens that are allowed for a given configuration property depends on that property. Each type of special tokens must follow a specific syntax.

These special tokens allow having the configuration properties whose effective value is defined during the execution of the application.

SEE ALSO THE "NOTES" CHAPTER ABOUT QUOTE ESCAPING.

#### 1.3.1. RCL From displayed screen.

RCL stands for "Row, Column, and Length". An RCL token designates a piece of text within the emulated terminal screen of the parent process. An RCL token embedded into a configuration property is replaced by the text contained within the screen when the property is used. They are defined as follows: {R, C, L}. The coordinates range from 1 to the number of rows and columns for the row and column coordinates respectively; from 0 to the number of rows times the number of columns for the length. The RCL token may contain whitespace characters.

Below is a list of valid and invalid RCL examples embedded into configuration properties The screen is assumed to be 24 rows by 80 columns:

- "Say": "Take {10,42,3} units" VALID;
- "Say": "Take {  10  ,  42  ,  3  } units" VALID;
- "Say": "Take {0,42,3} units" INVALID The coordinates must start from 1;
- "Say": "Take {10,42,2000} units" INVALID The length exceeds the maximum (i.e. 1920).

RCL tokens also provide a few formatting options.

#### 1.3.1.1. Case transformations.

Case transformations will "upperize" or "lowerize" the designated text. At this day, this option only has an effect on ASCII characters; non-ASCII characters remain unchanged. To specify a case transformation, either "U" (upper case) or "L" (lower case) must be prepended to the length parameter of the RCL token. The two available cases are mutually exclusive.

Examples:

- "Value": "Take {10,42,U3} units" VALID ;
- "Value": "Take {10,42,UL3} units" INVALID Cannot combine both cases ;
- "Value": "Take {10,42,2000L} units" INVALID The case must come before the length parameter.

#### 1.3.1.2. Trims.

It is possible to perform an alphabetic trim using the option "T" and a numeric trim using the option "N". These options can be combined and must be specified into a fourth parameter into the RCL token. The alphabetic trim removes all the leading and trailing space-like characters (whitespace, tab, newline, etc.) The numeric trim removes the first chain of consecutive "0" characters starting at the very beginning of the designated text, if any. The order of the options does not matter.

Examples:

- "Value": "Take {10,42,3,T} units" VALID ;
- "Value": "Take {10,42,3,NT} units" VALID ;
- "Value": "Take {10,42,3 T} units" INVALID These options must be defined into a fourth parameter;
- "Value": "Take {10,42,3, } units" INVALID If there is a fourth parameter, there must be at least one option.

#### 1.3.1.3. Leftmost and rightmost characters trunk.

It is possible to remove the leftmost and/or the right characters from the designated text. The options "L" and "R" allow to trunkate N characters at the beginning and at the end of the designated text respectively. These lengths can range from 0 (i.e no effect) up to the length of the selected text.

Examples:

- "Value": "Take {10,42,10,L3 R4} units" VALID Removes 3 characters from left and 4 from the right ;
- "Value": "Take {10,42,10,L3 R4} units" VALID Same as above ;
- "Value": "Take {10,42,5,L10} units" INVALID Trying to truncate more characters than selected characters ;
- "Value": "Take {10,42,3,R} units" INVALID No length specified ;
- "Value": "Take {10,42,3, } units" INVALID If there is a fourth parameter, there must be at least one option.

### 1.3.2. Variables.

Variable tokens can designate a profile-level, a process-level or a ScreenTracker variable. They are replaced by the value of the designated variable when the property is used. Variables are identified by their name -or by a number for ScreenTracker variables-. The name or number of the variable is preceded by a character indicating whether the designated variable is at **Profile**-level, **Process**-level or **ScreenTracker** (@, $ or #).

The character “@” indicates a Profile-level variable i/e: {@name}.

The character “$” indicates a Process-level variable i/e: {$name}.

The character “#” indicates a Process-level numbered ScreenTracker variable i/e {#10}.

Variable tokens are defined as follows:

- {$variableName} for Profile-level
- {@variableName} for Process-level variables
- {#nn} for ScreenTracker defined variables (only strings are allowed)

The name of the Profile and Process variables can be either alphabetic (from “a” to “z”, from “A” to “Z” and from “0” to “9” as well as underscore “_”). These tokens may not contain whitespace characters. The Screen Tracker variables are numbered from “#1” to “#99”, and can hold only strings.

Examples:

- “Say”: “Welcome mister {@userName}” VALID;
- “Say”: “Welcome mister {@ user Name}” INVALID (contains a whitespace in the name);
- “Say”: “Welcome mister {$varname}” VALID;
- “Say”: “Welcome mister {$ varname}” INVALID (contains a whitespace in the name);
- “Say”: “Welcome mister {#12}” VALID;
- “Say”: “Welcome mister {# 12}” INVALID (contains a whitespace in the name);
- “Say”: “Welcome mister {#0012}” INVALID (contains leading zeros).

### 1.3.3. Expressions.

Expression token must contain a piece of Lua script that defines an expression that is or can be converted to a value of type string. The token is replaced by the evaluated string value when the expression is used. Expressions are defined as follows {{expression}}. Examples:

- "Say": "Take {{#10}} at location {{#15}}" VALID ScreenTracker variable;
- "Say": "Take {{# 10}} at location {{# 15}}" INVALID has spaces in name
- "Say": "Take {{ vars["totalUnits"] - vars["takenUnits"] + 1 }} at location {{ vars["locationNum"] }}" VALID ;
- "Say": "Welcome mister {{ screen:setText( 1, 1, "Hello" ) }}" INVALID Cannot be evaluated as a string ;
-

### 1.3.4. Non-printable characters.

Some characters that have no associated symbol such as TAB or BACKSPACE can still be included in certain string properties using an hexadecimal notation. Non-printable character tokens are replaced by the character they represent when the property is used. Examples:

- "Format": "Followed by tab {\09}" VALID ;
- "Format": "Followed by tab {\ 09}" INVALID Contains a whitespace in the value.

### 1.3.5. Notes.

- Double quote escaping: in the above examples, escaping double quotes characters was omitted for simplicity. However when writing directly in JSON config files, one must escape each double quote character with a backslash as follows: "MsgWelcome" : "Welcome mister {{ vars["userName"] }}" should be written "Say" : "Welcome mister {{ vars[\"userName\"] }}".
- "Say": "Welcome mister {@userName}" is equivalent to : "Welcome mister {{ gvars["userName"] }}".
- "Say": "Welcome mister {24, 10, 30}" is equivalent to : "Welcome mister {{ screen:getText( 24, 10, 30 ) }}".

### *1.4. Pre-defined values in Lua scripts and expressions*

As stated in the previous chapter, certain properties may contain Lua expressions to be evaluated or reference Lua script files to be executed.

Depending on the property, some Lua variables may be pre-defined so that they can be used within the value of the property or within the script file. The values that are pre-defined for a given property are listed in the description of each property. Below is a list of all the possible values and what they represent. More detailed information about Lua expressions and scripts are available in the scripting documentation.

| **PRE-DEFINED VALUES** | | |
|---|---|---|
| **Name** | **Lua type** | **Description** |
| **Log** | table | Contains a set of functions allowing to use the logging feature. |
| **Profile** | table | Contains a set of functions allowing to use features directly related to the parent profile itself. |
| **Process** | table | Contains a set of functions allowing to use features directly related to the parent process itself. |
| **Trg** | table | Contains a set of functions allowing to use features directly related to the triggers system. |
| **Screen** | table | Contains a set of functions allowing to use the terminal screen. |
| **Keyboard** | table | Contains a set of functions allowing to use the virtual keyboard and some keyboard-related features. |
| **Scanner** | table | Contains a set of functions allowing to use the scanner module. |
| **Voice** | table | Contains a set of functions allowing to use the vocal engine module. |
| **Ptl** | table | Contains a set of functions allowing to use the Pick To Light module. |
| **Tdc** | table | Contains a set of functions allowing to use the Telnet Data Connector module. |
| **Wod** | table | Contains a set of functions allowing to use the Weight On Demand module. |
| **gvars** | table | Contains user variables shared across all processes of a profile. |

| | | |
|---|---|---|
| **vars** | table | Contains user variables local to a single process. |
| **garrs** | table | Contains user array values shared across all processes of a profile. |
| **arrs** | table | Contains user array values local to a profile. |
| **prev** | integer | The number of the previously activated trigger, starting from 1, or 0 if no trigger has been activated yet. |
| **current** | integer | The number of the current trigger, starting from 1, or 0 if no trigger is currently active. |
| **next** | integer | The number of the next trigger to be (force) fired, starting from 1, or 0 if no next trigger has been defined. |
| **lastAsr** | string | The most representative value of the last voice input. |
| **lastBarcode** | string | The most representative value of last scanner input. |
| **lastWeight** | string | The most representative value of last W.O.D. input. |

Here is a grid, indicating, for each trigger configuration property, which are the available pre-defined Lua variables. Properties that do not appear in this table do not use Lua and thus have no predefined variables. Green is available, orange is conditionally available and red is unavailable. Variables that are conditionally defined depend on the configuration of TellNext Emulator. For example, if the vocal engine is not available, vocal-engine related variables ("voice", "lastAsr") will not be available.

| | Log | Profile | Process | Trg | Screen | Keyboard | Scanner | Voice | Ptl | Tdc | Wod | Csv | gvars | vars | garrs | arrs | prev | current | next | lastAsr | lastBarcode | lastWeight |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Conditions/Matches[]/Value | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Conditions/Previous | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Conditions/DoIf | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Conditions/VirtualKbdLayout | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Conditions/CustomKbdLayout | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Initialize/VarAssign[]/Value | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Initialize/Next | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Initialize/ScriptOnInit | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Tts/Phrases[]/Previous | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Tts/Phrases[]/Say | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Asr/Confirm/Phrase | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Asr/Confirm/Info | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Asr/Information | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Asr/AltTerminator/Phrase | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| | Log | Profile | Process | Trg | Screen | Keyboard | Scanner | Voice | Ptl | Tdc | Wod | Csv | gvars | vars | garrs | arrs | prev | current | next | lastAsr | lastBarcode | lastWeight |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Asr/Expected | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Asr/Value | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Asr/Msg | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Bc/Value | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Bc/Msg | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Aux/Value | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Check/Aux/Msg | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Format/Asr/Data | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Format/Bc/Data | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Input/Format/Aux/Data | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/Expected | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| | Log | Profile | Process | Trg | Screen | Keyboard | Scanner | Voice | Ptl | Tdc | Wod | Csv | gvars | vars | garrs | arrs | prev | current | next | lastAsr | lastBarcode | lastWeight |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Input/ScriptOnInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/ScriptOnScannerInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/ScriptOnVocalInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/ScriptOnPtlInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/ScriptOnTdcInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| Triggers[]/Input/ScriptOnWodInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Triggers[]/Input/ScriptOnWidgetInput | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| OnConnectScript | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| OnKeyScript | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
| OnScanScript | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

Structure of a triggers definition file

| **TRIGGERS** | | | |
|---|---|---|---|
| **Name** | **Type** | **Detail** | **Comments** |
| Trriggers[] | Array of objects | | Root element of the file. Array of trigger definition objects. |
| - Trigger.Conditions | Object | Section | Defines the activation conditions to execute this trigger.<br>Analyzes the screen contents and other conditions. |
| - Trigger.Initialize | Object | Section | Defines the initialization tasks to perform to prepare the trigger to be performed.<br>Manages the keyboard, the scanner, the display, etc. |
| - Trigger.TTS | Object | Section | Defines the TTS texts to say during a voice trigger, if the voice mode is enabled. |
| - Trigger.ASR | Object | Section | Defines the ASR configuration to perform a voice input. |
| - Trigger.Input | Object | Section | Defines the processing of the received input (ASR, scanner, WOD, PTL, etc) for this trigger. |

See Configuration-File-Triggers document for values:
**TellNext_configuration_triggers.pdf**