# Action Programs

Revision 30, for EDDiscovery 19

Action programs are small programs which run when an event occurs in a program implementing Action, like a journal record being received or a user pressing a key.

Action programs are either written in the editor included in the program, or in a text editor.

A [N] marker in the text means the feature was introduced at a certain revision level.

[6]Text formats supported are Plain old ASCII/UTF-8 (note the Byte Order Mark is not written by this program but may be present) or Unicode format UCS-2 (BE or LE) [6].

Hit F8 or use the add-ons menu to bring up the condition editing page, where you set up entries to describe what events the program should look for. Then a program is assigned to each condition to process the events variables and produce an output (sound, text, speech, keypress).

## Examples

### Key assigned program

This example shows a program assigned to a key event.

ACTIONFILE V4

ENABLED True

INSTALL LongDescription="Long add on description"

INSTALL ShortDescription="Short add on description"

INSTALL Version=1.0.0.0

INSTALL MinEDVersion=12.1.0.0

INSTALL Location=Actions

INSTALL OtherFile1=A1.dll;DLL

INSTALL OtherFile2=A2.mp3;Sounds

INSTALL DownloadFile1=A2.mp3;Githubfolder;Sounds

INSTALL DownloadFolder=Githubfolder;LocalFolder

INSTALL NotDisablable=True

INSTALL NotEditable=False

EVENT onKeyPress, onKeyProgram, "var1=a,var2=b", KeyPress $== F4

PROGRAM onKeyProgram

Print Key Hit!

pragma dumpvars \*

END PROGRAM

Programs always start with ACTIONFILE V4 and ENABLED True or False.

Following this is an optional set of INSTALL information variables. These must be present if you want the pack to be installable via the programs plug in feature. Mandatory ones are:

* LongDescription : The long text description of the pack
* ShortDescription: Less than 30 characters
* Version: The version number of the pack
* MinEDVersion: Minimum version of the program that the pack works with
* Location: Where to store the pack in the programs appdata folder structure

Optionally, you can include:

* OtherFile[N]: Download additional file from the same folder that the ACT file is in, and store in the folder specified as the second parameter.
* DownloadFile[N] [30]: Download file from github, from githubfolder, and store in the localfolder
* DownloadFolder[N] [30]: Download the whole folder tree (including subfolders) from githubfolder, and store in the localfolder tree. Folders are synchronised when downloaded so that the folders mentioned only contain the files found in github. Any other files are deleted. Note sub folders not mentioned in github are not touched, so you can have local private folder storage.
* NotDisableable: If present, and set to True, the pack can only be deleted not disabled.
* NotEditable: If present, and set to True the pack can’t be edited by the user
* ItemType: If present, sets the item name displayed on the install screen
* ProgType: If present, defines the program the pack is allowed to work with. Semicolon list. Example would be “EDDLite;EDDiscovery”. If not present, allowed for all programs.

INSTALL variables must precede EVENT or PROGRAM statements [30]

Next is a set of EVENT statements to assign a program to an event. The executable program creates a wide set of events, which you can find in the programs own help. You assign an action program to an event as an reaction to that event happening. In this case its event is onKeyPress, assigned to program onKeyProgram. A condition is attached that the keypress must be F4. A set of variable values is also passed to the program when this event is triggered. The program prints to the log “Key Hit!” and then dumps all the variables known to the program to the log.

Finally the script defines PROGRAMs to run when events occur.

### Event assigned program

This triggers on an event from Elite:

ACTIONFILE V4

ENABLED True

EVENT FSDJump, FSDResponse, “”, Condition AlwaysTrue

PROGRAM FSDResponse

Print Welcome to system %(EventClass\_StarSystem)

pragma dumpvars EventClass\*

END PROGRAM

This triggers on an FSDJump, and prints to the log all the variables associated with this event and prints out a welcome message.

Each event has a long list of variables associated carrying the event information. Use Program dumpvars to find them all out, or look at EDD source code for each event. There are too many events and variables to document.

# Strings

Strings are double quoted (“) or single quoted [6] (‘) groups of characters, such as “hello there” or ‘goodbye my friend’. To include a quote character in a string, use \” or \’ (dependent on the quote character starting the string). So “Hello”there” would be written “Hello\”there”, or ‘goodbye’there’ would be written ‘goodbye\’there’. You don’t have to escape the other quote character, so ‘goodbye”there’ is just written as is. To write \ itself, use \\.

For output commands and dialogs (Print, Pragma Log, Pragma Debug, Message boxes) the escape sequences \r, \n and \\ are replaced by their non-escaped equivalents.

# Variables

Action does not have typed variables (double/int/string etc).

Variables are referenced using the following syntax: %(varname). For example, a Print command to print the commanders name would be: Print %(Commander) welcome!

Variable names are case sensitive. Variables are set using one of the Set, Let, Global functions, etc. Or they can set by the program passing down information to an event, or from user definition, or parameters to a program.

For most commands, the text of the command is scanned for variables and the text is replaced by the content of the variable. Variable expansion is not recursive, so if a variable contains another variable that other variable will not be expanded. Certain commands have options that can turn of variable expansion of text (Let $ operator for instance). For variables which do containing other variables, and you want them expanded, a function %expand() is provided which expands the contents of the variable text.

Variables are defined by different parts of the program:

* Persistent Globals: Defined by PersistentGlobal command, or by the user in the Global dialog box:
  + These are persistent between invocations of an action program.
  + These are visible to programs (irrespective of the action pack).
  + They are persistent between invocations of the EXE program.
* Program defined Globals: Defined by the Global or GlobalLet command.
  + These are persistent between invocations of an action program.
  + These are visible to programs (irrespective of the action pack).
  + They are not persistent between invocations of the EXE program.
* File Static variables: Defined by the Static or StaticLet command
  + These are persistent between invocations of an action program.
  + These are visible to programs within the action pack they are defined in.
  + They are not persistent between invocations of the EXE program.
* Local program variables: Define by Let or Set commands
  + Local to the current running program.
  + They are not persistent between program runs.
* Program information. Set by the program automatically at start up.
  + CurrentCulture : Culture name, such as en-gb
  + CurrentCultureInEnglish
  + CurrentCultureISO: ISO culture code
  + Also see the program specific section on this.
* Pack Information. Static variables containing information about the pack.
  + ActionPackName: Name of pack
  + ActionPackFilePath: Full File path of path
  + ActionPackFolder: Path of directory of file [30]
* Parameters in the event which causes the program to run
  + Any variables defined in the parameter field is passed to the event. These parameters are associated with the action condition itself, not the program, so a single program can be assigned to multiple action conditions and have unique parameters passed in for each condition. This is a powerful mechanism meaning you can limit the number of programs you need to write.
* Event information variables (if the program is run due to an event). See the event section.

# Conditions

Conditions, used in IF, While statements etc, can be used to change program flow or select actions.

A condition consists of a left hand side, operator and optional right hand side.

The left hand side is either a variable name, or a reference to a variable (%(varname) for instance) or a function (%indexof(varname,var2)).

The right side is a string or number, which can be expanded to resolve any variable or functions in it.

Conditions may be grouped using And, Or, Nor, Nand operators between each condition. An example is var Contains “fred” Or var Contains “Jim”.

Conditions can be grouped together using () operators, and the groups can be And, Or, Nor, Nand checked. An example is (var Contains “fred” Or var Contains “Jim”) And (var2 $== “xx”)

Condition types are:

## String Conditions

Compare one string against another string, i.e: var != “string” , %(var) != “%(var2) hello”

* Contains (Not case sensitive)
* NotContains (Not case sensitive)
* $== (String Equals, Not case sensitive)
* $!= (String not equal, Not case sensitive)
* CSContains (Case sensitive),
* CSNotContains (Case sensitive),
* CS== (String Equals, Case sensitive)
* CS!= (String not equal, Case sensitive)

## String Contents

Unary Empty/Not Empty test, i.e: var IsNotEmpty

* Empty
* IsNotEmpty

## Boolean Conditions

Unary Boolean test using zero as false, i.e: var IsTrue, %(var) isFalse

* IsTrue (non zero)
* IsFalse (zero)

## Numeric Conditions

Treats both sides as doubles, and compare, i.e: var == 20.2

* ==
* !=
* >
* <
* >=
* <=

## Date Conditions

Treat both sides as a US Date, and compare, either >= Date, or < Date.

* D>=
* D<

## Variable Present Conditions

Unary Variable Presence, is variable on the left present/not present, i.e: var IsPresent

* IsPresent
* IsNotPresent

## String List Conditions

Compare with multiple strings in a comma separated list:

* IsOneOf: Is the left side present on the right side string list: Var IsOneOf “a,b,c”. If any item in the list has spaces or commas in it, it must be quoted itself. Case insensitive.
* NotOneOf [26]: as per IsOneOf but true if the list does not contain the items
* AnyOfAny: Is one of the left side strings in a list present on the right side string list. Same format as IsOneOf on both sides, case insensitive. Example v1 AnyOfAny v2 where Set v1 = “notjim,\”jim 1\”” and Set v2 = "Fred,\"Jim 1\",Sheila"
* NotAnyOfAny[26]: as per AnyOfAny but true if the list does not contain the items
* MatchSemicolon [7]: Is the left side present on the right side string list of semicolon separated values.
  + Both sides are space trimmed before comparison, case insensitive. Ie. Var MatchSemicolon one;two;three.
* NotMatchSemicolon[26]: As per MatchSemicolon but true if the list does not contain the items
* MatchSemicolonList [7]: Is the left side present on the right side string list.
  + The string list consists of a set of semicolon separated values: <item> [‘;’<item>]..
  + Each item is composed of: <subpart> [ ‘|’ <subpart>]..
  + The Or allows you to indicate either subpart 1 or subpart 2 is acceptable, etc.
  + Each subpart is composed of: ‘[‘ ] <word> | <quoted string> [ ‘]’ ] | ‘[]’
  + A subpart can be a word, or a quoted string, or a brackets word/string [word], or just []
  + The optional brackets allow you to mark a word as optional: [word]
  + A pair of brackets together is used in an Or list to denote the whole entry is optional: Fred|Jim|[]
  + Examples are: [Set] phasers to stun, Set|Calibrate phasers to stun
  + ‘Hello’|’Goodday there’ Mr Finwen
* NotMatchSemicolonList[26]: As per MatchSemicolonList but true if the list does not contain the items
* MatchCommaList [7]: see MatchSemicolonList but with ‘;’ separator changed to ‘,’
* NotMatchCommaList[26]: As per MatchCommaList but true if the list does not contain the items.

## Always Conditions

Always true or false conditions. When written in text form, must be in the form: Condition AlwaysTrue etc.

* AlwaysTrue
* AlwaysFalse

# Statements

Statement names are case insensitive.

## Flow Control Statements

### Break

In a Loop, Do..While, ForEach or While, indicate that no more code is to be executed in the structure and the structure is terminated.

### Call

Call a subroutine.

Call <program name>[ ‘(‘ <input parameter list> ‘)’]

<program name> = [<Action File Set Name> ‘::’] [program title]

<Action File Set Name> the name of the file set the program is in, or if not present, search this file set first, then search all others for the program

<program title> = Name of program| “Name of program which can include macro expansions”[13]

<input parameter list> = <var def> [‘,’ <var def>]…

<var def> =( varname ‘=’ | ‘$=’ <value>) | ( <varwildcard> ‘=’ <immaterialvalue>’)

<varwildcard> = ‘wildcard name ending in \*’

<immaterialvalue> = Anything

<value> = <nonquotedbracketedorspacestring> | ‘”’ string ‘”’

<Nonquotedbracketedorspacestring> = string of characters without spaces, brackets (), commas or quotes.

Call a program and optionally pass parameters to it. The program will not have any local variables from the calling program. It will just have the parameters passed to it and the global variables available.

Parameters with ‘=’ are expanded as they are passed to the program. Use ‘$=’ for parameters that you don’t want to expand before passing.

To aid passing multiple values, use the pattern wildcardname\*=1 or wildcardname\* $=1 as a parameter name to pass in multiple variables starting with wildcardname (i.e f\_\*=1). The value is immaterial and ignored.

The program can return a value using Return statement and this value will be placed in the ReturnValue variable.

### Do

Perform a loop with a check at the end of the loop.

Do

.. statements - Indented to show relationship

While <condition> - if true, loop, else exit

<condition> = condition to check against variables defined, either global, event or local variables.

Errors will be produced if While is missing, or incorrectly indentation is found.

### End

End the program

END

Stops program and continue with next queued program if any.

### ErrorIf

If a condition is true, stop the program with a message

ErrorIf <message> ‘,’ <condition>

<message> = Quoted string, or non quoted non comma no space string, with escape characters allowed for control codes.

<condition> = condition to check against variables defined, either global, event or local variables.

Errors will be produced if the parameters are ill formed.

### ForEach

Iterate over a set of variables, given a pattern to search variables for, and set var to each found variable name in turn.

ForEach <var> in <searchpattern>

.. statements - Indented to show relationship

<var> = name of variable to set to each found variable name

<searchpattern> = pattern to search for. Wildcards \* and ? are supported. If no wildcards are given, its treated as a prefix and all instances of name\* will be found.

From Version 30, ‘in’ is case insensitive, and the variable ‘Index’ is written with the iteration number, 1,2,3..

### If

Change program flow on condition.

If <condition>

.. statements - Indented to show relationship

ElseIf <condition> - Optional

.. statements

Else If <condition> - As many as required

.. statements

Else - Optional

.. statements

<condition> = condition to check against variables defined, either global, event or local variables.

Errors will be produced if ElseIf, Else is found without an IF, or incorrectly indentation is found. Else If or ElseIf can be used interchangeably.

### Loop

Perform a loop a fixed number of times.

Loop <count> [‘,’ <loopvar>]

.. statements - Indented to show relationship

<count> = positive integer to indicate how many times to loop. <=0 mean no execution of the loop.

<loopvar> = optional name of variable to count the iterations, 1 onwards. If not present, a local variable, Loop<indentlevel> will be set each loop, counting from 1 upwards. For example, if the loop is at level 2, Loop2 will be set each time.

Errors will be produced if incorrectly indentation is found.

### Return

Return from a subroutine

Return [<string>]

<string> = value to return, if any. The returning program will see this value in ReturnValue variable.

### Sleep

Pause the program and sleep for a duration in milliseconds

Sleep [<time>]

<time> = in ms to sleep for.

### While

Perform a loop with a check of condition at the top of the loop.

While <condition>

.. statements - Indented to show relationship

<condition> = condition to check against variables defined, either global, event or local variables.

Errors will be produced if indentation is missing.

## Debugging and Comment Statements

### Rem or //

Make a program remark.

REM <string>

// <string>

<string> remark.

No action on this command.

Remarks can also be placed after a valid command on a line:

Print “Hello” // comment.

### Pragma

Control program behaviour and debug.

Pragma <commandname-opts>

Command Name-opts:

* DumpVars varnamewildcard : Dump a variable, or a variable wildcard, to the log window (i.e L\*)
* Log: Write quoted text to the log window
* Debug: Write quoted text to log window only on debug builds
* IgnoreErrors : Disable stopping on errors. LastError variable will be written if an error occurs and execution will continue
* AllowErrors: Enable stopping on errors.
* Enableasync[8]: Default, async processing on
* Disableasync[8]: Special, async processing off, continuous running.
* Enabletrace[8]: Special, trace output on
* Disabletrace[8]: Default, trace output off

## Variable Statements

### DeleteVariable

Delete a persistent, non-persistent or local variable or variables.

DeleteVariable <variablename> [‘,’ <variablename>]

No error is produced if variable does not exist. [8] You can now include the wild card character \* at the end of the name to indicate a range of variables.

### Expr

Evaluate an expression, and assign any result to the Result variable. This is useful for executing functions with side effects, such as file writing.

Expr <expression>

<expression> = unquoted, any valid string or string containing functions or variables.

Example

Expr %writeline(h1,”fred”)

### Global

Set a non-persistent global variable to a string value. See Set.

### GlobalLet

Evaluate expression and set a non-persistent global variable to the decimal value. See Let.

### Let

Set a local variable variable [6] to a numeric expression.

LET <varname> ‘=’|’$=’ <numeric expression>

<varname> = either a literal name, or an expansion.

<numeric expression> = expression with +/-/\*/divide etc.

A variable can be assigned and expanded (operator =) or just assigned ($=) without expansion.

The variables on the right side must be expanded using %() syntax.

Numeric expression is a C-Style expression with the following evaluation order:

|  |  |  |
| --- | --- | --- |
| Level | Operator | Binding |
| 0 | Unary minus (-) or plus (+) | To right, Unary |
|  | () bracket operator | To right |
| 1 | Logical Not operator ! | To right, Unary |
|  | Binary Not operator ~ | To right, Unary |
| 2 | \* Times | To left, Pair of value |
|  | / Divide | To left, Pair of value |
|  | % Modulo | To left, Pair of value |
| 3 | + | To left, Pair of value |
|  | - | To left, Pair of value |
| 4 | << Binary shift left | To left, Pair of value |
|  | >> Binary shift right | To left, Pair of value |
| 5 | < Less than | To left, Pair of value |
|  | > Greater than | To left, Pair of value |
|  | <= Less equal to | To left, Pair of value |
|  | >= Greater equal to | To left, Pair of value |
| 6 | == Equality | To left, Pair of value |
|  | != Inequality | To left, Pair of value |
| 7 | & Binary And | To left, Pair of value |
| 8 | ^ Binary eor | To left, Pair of value |
| 9 | | Binary Eor | To left, Pair of value |
| 10 | && And | To left, Pair of value |
| 11 | || Or | To left, Pair of value |

### PersistentGlobal

Set a persistent global (one saved between invocations of the program) variable to a string value. See Set.

### Set

Set a local variable [6] to a string.

SET <varname> ‘=’|’+=’|’$=’|’$+=’ <value>

<varname> = either a literal name, or an expansion.

<value> = <nonquoted> | ‘”’ string ‘”’

<nonquoted> = string of characters without quotes, or a space at the end, or an empty string.

A variable can be assigned and expanded (operator =) or just assigned ($=) without expansion. It can also be added to (+= operator), and finally added to and not expanded ($+=).

### Static

Set a non-persistent file local static variable or multiple variables to a string value. See Set.

### StaticLet [6]

Evaluate expressions and set a local static variable to the decimal value. See Let.

## User Interface Statements

### Dialog / NonModalDialog

Two forms exist:

**Dialog**: Show the user a modal program defined dialog, wait for user selection.

**NonModalDialog**: Show the user a non-modal program defined dialog. Continue executing program.

For modal dialogs, the action program is suspended while the user makes a control UI input (except see below for Resume option). The program resumes executing with the <dialog variable> set to a trigger value.

For non-modal dialogs, the action program continues. When the user selects a control, an onNonModalDialog event will occur, with the variables ‘Dialog’ set to the dialog name, and ‘Control’ set the same value as the dialog name variable above.

‘DialogV

The DialogControl command for both types can then be used to control and close the dialog.

Modal Dialog names are local to the function (and passed to any called functions) they defined in.

Non modal dialog names are static to the file they are defined in, and are not visible to other files.

#### Command

Dialog/NonModalDialog <dialog name/variable>,<Caption>,<size>,<Variable prefix>,[<Options>]..

<dialog name/variable> = logical name of the dialog, and the variable name to use for this dialog to pass back information in for modal dialogs

<Caption> = Title caption

<size-position> = In General from version: [25]

<wmin> ‘,’ <hmin> ‘,’<xpos> ‘,’ <ypos> ‘,’ <wmax> ‘,’ <hmax> ‘,’ <wanted> ‘,’ <hwanted> :

Set Dialog sizing.

<wmin> = either ‘-‘ or width minimum size

<hmin> = either ‘-‘ or height minimum size

<xpos> = either ‘-‘ or missing meaning centred, or xpos

<ypos> = either ‘-‘ or missing meaning centred, or ypos

<wmax> either ‘-‘ or missing means no limit, or maximum width

<hmax> either ‘-‘ or missing means no limit, or maximum height

<wwanted> either ‘-‘ or missing means not set, or width wanted. Positive means recommended width (minimum contents dialog width will override this), Negative means forced width.

<hwanted> either ‘-‘ or missing means no limit, or height. Works the same <wwanted>

<Variable prefix> = Control definitions variables for this dialog starts with this prefix.

[21]<Options> = ‘AllowResize’ : Allow resize of box

= ‘Transparent’ : Box is transparent

= ‘NoCloseIcon’ : No close icon is present

= ‘AlwaysOnTop’ : Always on top of all other windows

= ’NoWindowsBorder’ : no windows border (close etc)

= ‘NoPanelBorder’ : no panel outline border.

= ‘TopPanel:<n>’ [30]: Top panel, where title and close icon is shown. Height in pixels. If not give, title and close icon go to content panel.

= ‘BottomPanel:<n>’ [30]: Bottom panel height in pixels.

= ‘FontScale:<n>’ [30]: Scale font by this floating point number (default is 1.0)

= ‘NoShow’[30]: Don’t show the dialog but just create it. Continue executing a script. To show the dialog, use DialogControl Show. To continue the modal operation and suspend the script until a user interaction was made, then use DialogControl Continue after the Show.

#### Control Definition

A set of one or more control variable definition define the dialog, they are in the form:

<Variable prefix><unique identifier> = <Control Definition>

<unique identifier> = any variable name characters to distinguish this from another variable.

Each control is defined by a string:

<Control Definition> = <Control Name> ‘,’ <Control Type> ‘,’ <Control Text> ‘,’

[‘In:’<panelname> ‘,’] [30]

[‘Dock:’<dockstyle> ‘,’] [30]

[‘Anchor:’<anchorstyle> ‘,’] [30]

[‘Margin:’<left> ‘,’ <top> ‘,’ <right> ‘,’ <bottom> ‘,’] [30]

<X> ‘,’ <Y> ‘,’ <W> ‘,’ <H> ‘,’ [<ToolTip> [‘,’ <control parameters>]]

<Control Name> = name of control.

<Control Type> = See below for types. The type is case insensitive

<Control Text> = value for initial setting of control, see below:

<panelname> = if given, this can be ‘Bottom’ place in bottom panel (you must have defined a BottomPanelHeight), ‘Top’ place in top panel (ditto), or a control name to place inside a panel control defined by the script. For a splitter, use <Name>.1 to target the top panel, and <Name>.2 to target the bottom panel. Panel targets must be defined in order first before a control using that panel target.

<dockstyle> = if given, dock the control (normally a panel) to Top, Bottom, Left, Right, Fill. Height should be set for Top/Bottom, width for Left/Right. All four XYZH integers must be present.

<anchor> = ‘Right’ | ‘Bottom’ | ‘BottomRight’. Anchor the control to the right side, or bottom size, or both. Move during resize.

<Margin parameters> = give margin around item for flow panels only. Default for margin is 3,3,3,3 if not given.

<X> <Y> = position. Each can be an absolute integer, or +N or -N to say its offset from the last control by this amount.

<W> <H> = size

<ToolTip> = tooltip to display. This is optional unless you need to give control parameters. Use an empty string to indicate no tooltip.

For the control types shown below, extra control parameters may or must be present.

##### Label

<Control Text> = Text to display

##### Button

<Control Text> = Either text to display, or Resource:full module path to resource for an image from the resource file, or File:full path to file to load an image from a file.

##### TextBox

<Control Text> = Text to display

< control parameter 1> = optional ‘1’ | ‘0’ indicating multiline control (not multiline by default)

< control parameter 2> = optional ‘1’ | ‘0’ indicating if clear box on first character. [8]

##### RichTextBox [30]

<Control Text> = Text to display

No optional parameters

##### CheckBox

<Control Text> = Either text to display, or Resource:full module path to resource for an image from the resource file, or File:full path to file to load an image from a file.

<control parameters> = optional ‘1’ | ‘0’ indicating checked or unchecked (unchecked default)

##### DateTime

<Control Text> = date/time to display

<control parameters> = “short” | “long” | “time” | custom date time format. See Microsoft documentation.

##### Numberboxlong, int or double

<Control Text> = Initial number, in invariant form.

<control parameter 1> = optional minimum value

<control parameter 2> = optional maximum value

<control parameter 3> = optional printing format (0.## etc)

##### ComboBox

<Control Text> = default entry to select. If empty or not present in the list, no item will be selected.

<control parameters> = mandatory list of combo box option values strings, as a comma separated list. Quotes are needed if items contain commas.

##### Panel [30]

<control parameters> = [<backcolour>]

<backcolour> = optional back colour of box, as a named colour (Blue) or as a quoted string of four colour values “a,r,g,b” where the values are from 0-255

##### FlowPanel [30]

<control parameters> = [<direction> [‘,’ <backcolour>]]

<direction> = ‘Horizontal’ (default) | ‘Vertical’

Use Margin: on items to control spacing.

##### PanelRollUp [30]

<control parameters> = [<pin state> [‘,’ <backcolour>]]

<pin state> = pin state of the roll up, 1/0. Default is unrolled (1).

##### DGV (Data Grid View)[30]

<Control Text> = ignored.

<control parameters> = <dgv configuration> ‘;’ <column definition> [‘,’ <column definition>]…

<dgv configuration> = <row header width pixels>

<column definitions> = ‘(‘ <col type> ‘,’ <col header text> ‘,’ <col % fill weight> [‘,’ <sort mode>]‘)’

<coltype> = ‘text’ (only as of now)

<col header text> = quoted string with column header text

<col % fill weight> = integer, fill weight for column

<sort mode> = optional string for sort mode: ‘Off’|‘Alpha’|’Date’|’Numeric’|’NumericAlpha’ |’AlphaInt’

Add a data grid view control. Note the semicolon between the DGV configuration and first column description. Control parameters must be present.

##### DropDownButton [30]

<Control Text> = Semicolon list of tags to select on entry

<control parameters> = <All or none shown 1/0> ‘, ‘

<All or none back 1/0> ‘,’

<multicolumns 1/0>

[‘,’ <sortitems 1/0> [ ‘,’ <imagexsize> ‘,’ <imageysize>]

‘;’

‘(‘<item>’)’ [, ‘(‘ item ‘)’ ]…

<All or none shown> = 1/0 show the options ‘All’ and ‘None’

<All or none back> = 1/0 return ‘All’ or ‘None’ if these are applicable to the return value

<multicolumns> = 1/0 Enable multicolumn mode

<sortitems> = 1/0 Alpha sort the items

<imagexsize> = for images in drop down list, set X size to this

<imageysize> = for images in drop down list, set Y size to this

<item> = <tag> ‘,’ <text> [ ‘,’ <imgpath> [ ‘,’ <exclusive> [‘,’ <type> ]]]

<tag> = string/quoted string giving logical tag for this item

<text> = string/quoted string giving text to display

<imgpath> = optional string/quoted string giving path (either a file path or a resource path) to the icon to display. Empty string means no icon. If path is not found the icon will not be displayed and the program will continue. Note you don’t give File: or Resource: prefix here.

<exclusive> = optional empty string, or list of options. See below.

<type> = optional string giving type, case insensitive..

‘Group’: Group option, Tags contain the group list of tags. Exclusive can be ‘All’ then the group tag is returned not the list of options.

‘Button’: Button. Present as a button, not a check box.

‘Checkbox’: Checkbox. Default if type is not given. Exclusive is a list of other options to turn off it this is selected.

##### Splitter [30]

<control parameters> = <Orientation> ‘,’ <Splitter percentage> [ ‘,’ <Panel 1 min size> [‘,’ <Panel 2 min size> ]]

<Orientation> = ‘Horizonal’ splitter or ‘Vertical’. Case insensitive

<Splitter Percentage> = 0.0 – 100.0 floating point splitter size of panel 1

<Panel X min size> = Minimum size in design pixels

Control parameters must be present.

Control parameters must be present. For each Item:

#### Triggers

When an event occurs on the UI, the program is informed.

For modal dialogs, the Dialog command will exit and the variable name given in the Dialog command will contain the event information.

For non modal dialogs, an onNonModalDialog event will be raised with the ‘Control’ variable set to the event information.

The format of the event information stored in Dialog variable or ‘Control’ is:

<event information> = <dialog name> ‘:’ <control name> [ ‘:’ <event> [ ‘:’ <value1> [ ‘:’ <value2>]]]

<dialog name> = name of dialog (given in Dialog commnad)

<control name> = name of control (not quoted)

See below for other meanings

In addition, two more variables can be created carrying data, dependent on the trigger, called <dialog name>\_Value and <dialog\_name>\_Value2 [30]

##### Label

No UI action is generated.

##### Button

In <event\_information> only control name is sent. No other fields.

##### TextBox

<event> = ‘Return’ - return pressed

##### RichTextBox

No UI action is generated.

##### CheckBox

In <event\_information> only control name is sent. No other fields.

<dialog name>\_Value will be set to Checked/Unchecked/Indeterminate

<dialog name>\_Value2 will be set to 1/0

##### DateTime

<event> = ‘ValueChanged’ - validity changed [30]

<value1> = new date value, in zulu format

<dialog name>\_Value will be set to Date Time representation

##### Number Boxes

In all below, <dialog name>\_Value will be set to value of control

<event> = ‘Return’ - return pressed

or

<event> = ‘Validity’ - validity changed

<value1> = ‘True’ or ‘False’

Or

<event> = ‘ValueChanged’ - Value changed changed [30]

<value1> = new value

##### Combo Boxes

In <event\_information> only control name is sent. No other fields.

<dialog name>\_Value will be set to selected index.

##### Panel, FlowPanel, PanelRollUp

No UI action is generated.

##### DGV [30]

<event> = ‘RowSelection’ - user selected rows

<value1> = list of rows, semi comma separated. Order is by selection, not by row number

<dialog name>\_Value will be set to the same.

or

<event> = ‘CellSelection’ - user selected cells

<value1> = list of cells (given by row,col), semi colon separated. Order is by selection, not by cell/row number.

<dialog name>\_Value will be set to the same.

or

<event> = ‘SortColumn’ - user sorted columns

<value1> = column number

<dialog name>\_Value will be set to sorted column

or

<event> = ‘RightClickMenu’ - right click menu selected

<value1> = tag on the right click menu

<value2> = row number clicked

<dialog name>\_Value will be set to the tag

<dialog name>\_Value2 will be set to the row number

##### DropDownButton

<event> = ‘DropDownButtonClosed’ - closed button

<value1> = selection made, semicolon list of tags

<dialog name>\_Value will be set to the same.

or

<event> = ‘DropDownButtonPressed’ - user selected a button

<value1> = button tag pressed

<dialog name>\_Value will be set to the same.

##### Splitter

<event> = ‘SplitterMoved’ - user moved the splitter

<value1> = splitter percentage, floating point number

<dialog name>\_Value will be set to the same.

##### Configurable Form Events

For a control made using Dialog/NonModalDialog, dialog wide events are sent using this form: <event information> = <dialog name> ‘:’ <event>

<event> = ‘Escape’ - if the escape key is pressed.

<event> = ‘Resize’ - if the dialog was resized.

<event> = ‘Reposition’ - if the dialog was repositioned

<event> = ‘Close’ - if the dialog was requested to close via the Close icon.

##### Configurable User Control Events

For a build in panel configuration user form (Python Panel), dialog wide events are sent using the form: <event information> = <dialog name> ‘:’ <event> ‘:’ <value1>

<event> = ‘Resize’ - if the dialog was resized.

<value1> = ‘width,height’

#### Example

Example:

Set dvar1 = "OK,Button,\"Button OK\",10,30,100,20,\"Press for OK\""

Set dvar2 = "Cancel,Button,\"Button Cancel\",10,60,100,20,\"Press for Cancel\""

Set dvar3 = "B1,Button,B1,10,110,100,20,\"Press for B1\""

Set dvar4 = "TB1,TextBox,Default,10,150,200,60,\"Enter text\",1"

Set dvar5 = "CB1,CheckBox,Check,10,250,200,20,\"Check Box\",1"

Set dvar6 = "DRP1,ComboBox,\"two\",100,60,200,20,\"Drop down box\",one,two,three,four,five"

Dialog D1, "Hello there", "800,600", dvar

### DialogControl

Indicate to the control the next action to take.

DialogControl <dialog name> [‘,’] <cmdname> [<optional parameters>]

<dialog name> = name of dialog

<cmdname> = case insensitive command name to execute, see below:

#### Exists

Return in variable “Exists” either 0 (does not exist) or 1 (exists). Useful mostly for non-modal dialogs to know if it is being presented. No optional parameters.

#### Show[30]

If the Dialog has been created with NoShow option, you must call this to show the dialog.

#### Continue

Modal dialogs only. Resumes processing of dialog actions and suspends the program until another action occurs. No optional parameters

#### Close

Close the dialog (if a pop out form dialog). No optional parameters.

#### Get

<optionalparameters> = <Control Name> (see Dialog).

Return the value of the control named in the variable DialogResult. From version 15 the name can be in quotes.

* Splitters return split percentage as a floating point number
* DropDownButtons return current settings as a semicolon string.
* Panel Roll Up return pin state

#### Set[1]/SetEscape[30]

<optionalparameters> = <Control Name> ‘=’ <value> . Set the control value to this value.

From version 15 the name can be in quotes.

Value is either a single word or a quoted string.

Error if the control does not have a value that can be set.

* Label, Button set the text
* For RichTextBox and TextBox, escape characters (\r\n etc) are replaced by their equivalents if SetEscape is used
* CheckBox sets the check mark (0/1)
* DateTime set the date/time
* NumberBoxDouble, ..Int, ..Long set the value
* ComboBox sets the current selection, the value must be present in the list (case sensitive), or an error will occur.
* Panel Roll Up set the pin state (1/0)
* DropDownButtons set the current settings as a semicolon string.
* Splitters set the split percentage (0.0-100.0)

#### AddText[30]

<optionalparameters> = <Control Name> ‘,’ <value> . Add this text to a rich text box at the end and scroll to it. Value can be a quoted string. Escape characters are replaced. Error if the control is not a rich text box.

#### Add[30]

<optionalparameters> = <Control Definition>

Add another control. Parameters are as per the initial control definition. Note the positions here are in design units, and will be scaled to font selected.

#### AddRange[30]

<optionalparameters> = <Variable prefix>

Add controls defined in variables beginning with variable prefix.

#### Remove[30]

<optionalparameters> = <Control Name>

Remove a control.

#### AddSetRows[30]

<optionalparameters> = <Control Name> [‘,’] <change list>

<change list> = <row definition> [ ‘;’ <row definition>]…

<row definition> = <row number> [ ‘,’ <cell start number> [‘,’<header text>] ]

[‘,’ <cell definition> [ ‘,’ <cell definition>]… ]

<row number> = -1 insert at top, -2 insert at end, else row number to change (0 based)

<cell start number> = cell to start insertion at (default 0 if not given)

<cell definition> = ‘(‘ <cell type> ‘,’ <cell value> [ ‘,’ <cell tooltip>] ‘)’

Add, remove, or change cells in a DGV. A JSON definition can also be given for <add/set> to add rows. Cell definitions are optional in which case you can just change the header text.

#### InsertColumns[30]

<optionalparameters> = <Control Name> [‘,’] <position> ‘;’ <column definition> [‘,’ <column definition>]…

<position> = zero based position to insert the column in front of.

<column definition> = See DGV Dialog entry for definition.

Add columns to a data grid view. Note the semicolon between the DGV configuration and first column description.

#### RemoveColumns[30]

<optionalparameters> = <Control Name> [‘,’] <position> ‘,’ <count>

<position> = zero based position to insert the column in front of.

<count> = Number of columns to be removed (can be bigger than that available).

Remove columns from a data grid view.

#### RightClickMenu[30]

<optionalparameters> = <Control Name> [‘,’] ( <tag> ‘,’ <text> )…

<tag> = tag to associate with this control, reported back

<text> = text to display in menu option

Configure the right click menu on a data grid view

#### GetColumnsSetting[30]

<optionalparameters> = <Control Name>

Return the column setting (JSON string) in a variable ColumnsSetting

Get the data grid view column settings.

#### SetColumnsSetting[30]

<optionalparameters> = <Control Name> ,<setting>

<setting> = Column setting (JSON string)

Set the data grid view column settings to this.

#### SetDGVSetting[30]

<optionalparameters> = <Control Name> , <column reorder>, <per column wordwrap allowed>, <allow row header visibility selection>, <single row select>

<column reorder> = allow column reorder (true)

<per column wordwrap allowed> = allow per column word wrap control (true)

<allow row header visibility selection> = allow row header visibility selection (false)

<single row select> = when right clicking, select whole row (true)

Configure a data grid view control.

#### SetWordWrap[30]

<optionalparameters> = <Control Name> , <word wrap>,

<word wrap> = 1/0 indicating if the whole grid is word wrapped (false default). All columns are set to the same word wrap state

Only applicable to data grid view control.

#### Clear[30]

<optionalparameters> = <Control Name>

Clear a control. For DGV controls it clears the grid. For rich text boxes, it clears the box. Other controls its not applicable to.

#### RemoveRows[30]

<optionalparameters> = <Control Name> [‘,’] <row start> ‘,’ <count>

Remove rows from grid.

<row start> is the row start number (0..N-1), or if negative, counting from the end forward (so -1 is the last row). If out of bounds, the command is ignored.

<count> number of rows to delete. Can be greater than rows available to remove without error.

The variable “Removed” will be set to the actual number of rows removed.

#### Enable [21]

<optionalparameters> = <Control Name> [‘,’ [<Enable 1/0>] ]

If no optional parameters are given after Control Name, report the enable state of the control in variable “Enabled”.

If the optional parameter Enable is given set the control enable state to this.

#### Visible [21]

<optionalparameters> = <Control Name> [ [‘,’] <Visible 1/0>] ]

If no optional parameters are given after Control Name, report the visible state of the control in variable “Visible”.

If the optional parameter Visible is given set the control visible state to this.

#### Position

<optionalparameters> = [ <X> ‘,’ <Y> ]

If no optional parameters are given, report the position of the dialog in the variables “X” and “Y”.

[21] If the optional parameters X and Y are given, set the window to this position.

#### **Size** [21]

<optionalparameters> = [ <W> ‘,’ <H> ]

If no optional parameters are given, report the size of the dialog in the variables “W” and “H”.

If the optional parameters W and H are given, set the window to this position.

#### Controlbounds [21]

<optionalparameters> = <ControlName> [ [‘,’] <X> ‘,’ <Y> ‘,’ <W> ‘,’ <H>] ]

If no optional parameters are given after ControlName, report the controls bounds in variables X,Y,W,H.

If the optional parameters X,Y, W and H are given, set the control to these bounds.

The positions are after theme scaling, so are not in design units.

#### CloseDropDownButton[30]

Close down the drop down.

#### IsAllValid[30]

Report in Valid variable (1/0) if all controls are in a valid state

#### Example:

Example (continuing on from Dialog example above):

Dialog D1, "Hello there", "800,600", dvar

While D1 $!= OK

Print %(D1)

if D1 $== "B1"

Print Button B1

DialogControl D1,Set TB1="Hello there"

DialogControl D1,Continue

DialogControl D1,Get TB1

Print textbox=%(DialogResult)

DialogControl D1,Get CB1

Print checkbox=%(DialogResult)

DialogControl D1, Close

### FileDialog

Prompts the user to select a folder or a file

FileDialog <type>

<type> = <folder> | <openfile> |<savefile>

**<folder>** = ’Folder’ [ ‘,’ <description> [‘,’ <folder root>]]

<description> = description string, optionally quoted if it contains commas

<folder root> = folder. Use one of the systems [Folder Location Types](https://msdn.microsoft.com/en-us/library/system.environment.specialfolder(v=vs.110).aspx) . MyComputer is the most useful. If the name is not recognised, no root folder is selected.

Return results in FolderName variable (folder, or empty for cancel) and DialogResult variable[30] ‘OK’ or ‘Cancel’.

**<openfile>** = ‘OpenFile’ [ ‘,’ <rootfolder> [ ‘,’ <filter> [ ‘,’ <defext> [ ‘,’ <check> ] ] ] ]

<rootfolder> = folder to start from, such as E:

<filter> = windows dialog filter string, such as “Text Files|\*.txt|Word Files|\*.doc”

<defext> = default extension, such as “.txt”

<check> = If “On”, check folder and file exists. If not present or any other, do not check.

Return results in FileName variable (file, or empty for cancel) and DialogResult variable[30] ‘OK’ or ‘Cancel’.

**<savefile>** = ‘SaveFile’ [ ‘,’ <rootfolder> [ ‘,’ <filter> [ ‘,’ <defext> [ ‘,’ <overwrite> ] ] ] ]

<overwrite> = If “On”, warn if overwriting. If not present or any other, do not check.

Return results in FileName variable (file, or empty for cancel) and DialogResult variable[30] ‘OK’ or ‘Cancel’.

### InfoBox

Presents an information box with an OK button. URLs can be clicked thru [30]

InfoBox <message> [ ‘,’ <caption> ]

<message> = message string, quoted if required, \r\n etc.

<caption> = optional caption, quoted if required

### InputBox

Presents the user with an input box. The prompt list, the default list and tooltips are a list of items separated by a semicolon (but this can be changed [30]) .

InputBox <caption> ‘,’ <prompt list> [ ‘,’ <default list> [‘,’ <features> [‘,’ <tooltips>] ]

<caption> = caption of input box, can be quoted

<prompt list> = prompt list. The number of prompts determines how many items to ask for.

<default list> = option list of default values. May be blank or less or more than prompt list.

<features> = optional feature selection, join them with a ‘;’.

‘Multiline’ (case insensitive) multiline dialog

‘0x2345’ – use this as the separation character[30]

‘196’ – use this as the separation character[30]

<tooltips> = optional list of tooltips for each text box

Output will be:

InputBoxOK = 1 on OK, or 0 on cancel.

InputBox1 to InputBoxN = value returned when InputBoxOK=1

### Key [6]

Sends keystrokes to another program.

Key <keystrokes> [ ‘,’ <configurationlist>]

<keystrokes> = See below for the format. Use quotes if spaces are present.

<configurationlist> = <configurationvalue> [ ‘,’ <configurationvalue>]..

<configurationvalue = <item> ‘=’ <value>

<value> = value of item.

<item> = configuration items, one of:

***To*** is the process to send the keystroke to. If not present, the global variable “KeyProcessTo” is checked, and if it exists, that is used. Finally, if neither is set, the current window in focus is the default (or use “Current window”). It is strongly recommended to use a named process here.

***Delay*** is the default delay to hold the keystroke down for, in milliseconds. If not present, the global variable “KeyDelay” is checked. Finally, if neither is set, 10ms is the default.

***UpDelay [7]*** is the default delay after raising the keystroke for, in milliseconds. If not present, the global variable “KeyUpDelay” is checked. Finally, if neither is set, 10ms is the default.

***ShiftDelay [7]***is the default delay after pressing a shift key (Ctrl+alt also) before pressing the keystroke, and the delay after releasing the shift key in milliseconds. If not present, the global variable “KeyShiftDelay” is checked. Finally, if neither is set, 10ms is the default.

***SilentOnError [7]*** allows Key *to* silently terminate the action sequence without showing an error if an error in the key string is detected. If not present, the global variable “KeySilentOnError” is checked. 0 (default) is don’t do this. 1 means error once and then swallow. 2 means never error.

***AnnounciateOnError [7*]** allows Keyto speak the error if encountered using Say command. The default voice will be used. If not present, the global variable “KeyAnnounciateOnError” is checked. Set to 1 to enable this. Default is disabled.

**Key Sequences:**

The following combinations of modifiers and key names are supported:

\* Shift , Ctrl , Alt , Shift+Ctrl, Shift+Alt, Shift+Alt+Ctrl (Key command only, and just press/release these keys)

\* Shift+Key, Ctrl+Key, Shift+Ctrl+Key, Shift+Alt+Key, Shift+Alt+Ctrl+Key

The following keys are defined:

Cancel,Back,Tab,LineFeed,Clear,Return,Pause,Captial,

KanaMode,JunjaMode,FinalMode,HanjaMode,Escape,IMEConvert,IMENonconvert,IMEAceept,IMEModeChange,

Space,PageUp,Next,End,Home,Left,Up,Right,Down,Select,Print,Execute,PrintScreen,Insert,Delete,Help

0 to 9

A-Z

LWin,RWin,Apps,Sleep,

NumPad0-9,NumEnter,Multiply,Add,Separator,Subtract,Decimal,Divide,

F1-F24,

NumLock,Scroll,BrowserBack,BrowserForward,BrowserRefresh,BrowserStop,BrowserSearch,BrowserFavorites,BrowserHome,VolumeMute,VolumeDown,VolumeUp,MediaNextTrack,MediaPreviousTrack,MediaStop,MediaPlayPause,LaunchMail,SelectMedia,LaunchApplication1,LaunchApplication2,

Semicolon,Plus,Comma,Minus,Period,Question,Tilde,CloseBrackets,Pipe,OpenBrackets,Quotes,Backquote,Backslash,ProcessKey,Packet,Attn,Crsel,Exsel,EraseEof,Play,Zoom,NoName,Pa1,OemClear

In some dialogs, Menu is used instead Alt. They mean the same.

From now on, applies to Key command only.

In front of the optional modifier and optional key can be a set of action modifiers to change the keystroke timing and indicate if the keystroke is a press and release, or just press (down) or release (up) sequence:

[d] : Define the time in ms to hold the key in the state defined (For keypresses, this is the hold down time. For key up/down, it’s the time to hold in this state before processing the next key. You don’t need to specify any other delays except just d for key up/down.)

[d,s] : Define d, and define the shift delay. The shift delay is the time between pressing the modifier and pressing the main key. Default is 2 ms.

[d,s,t] : Define d and s, and define the up delay. The up delay is the time to wait after sending up before continuing to process. Default is 2 ms.

Followed by an optional indicator to define if it’s a key press (no indicator) or one of the following:

^ or < : Send Key Up event only

! or > : Send Key Down event only

The action modifiers, shift/ctrl/alt modifiers and key comprise a single key sequence.

A group of key sequences can be sent using the current action modifier and shift/ctrl/alt modifiers by enclosing them in a bracket pair “(“ and “)”. Example: Shift+(ABC NumPad0 F1) would hold down shift, send ABC, then numpad0, then F1, then release shift. Single character keystrokes (0-9,A-Z) can be joined together without spacing them, other keystrokes need spaces between them.

The keystroke string in the Key command line comprises a set of key sequences, space or comma separated.

Examples:

A : Send A

Numpad0 : Send Numpad0

Alt+A : Send Alt+A (Press Alt, wait 2ms, Press A, wait the default time, release A, wait 2ms, release Alt)

Alt : Press and release Alt

!Alt : Press Alt and hold it down.

^Alt : Release Alt

!A : Press A and hold it down.

[250]A : Press A, hold it down for 250ms, release it.

[300,30]Shift+A : Press shift, wait 30m, press A, wait 300ms, release A, wait 2ms, release Shift, wait 2ms.

A B C : Press A, then press B, then press C

A B [2000]C : Press A, then press B, then press C for 2000ms

### MessageBox

Presents a message box with configurable buttons and icons.

MessageBox <message> [ ‘,’ <caption> [‘,’ <buttons> [‘,’ <icon>] ] ]

<message> = message string, quoted if required, /r/n etc.

<caption> = optional caption, quoted if required

<buttons> = ‘OK’ | ‘AbortRetryIgnore’ | ‘OKCancel’ | ‘RetryCancel’ | ‘YesNo’ | ‘YesNoCancel’

(case insensitive, if the entry does not match one of these an error will be produced).

<icon> = ‘Asterisk’ | ‘Error’ | ‘Exclamation’ | ‘Information’ | ‘Question’ | ‘Warning’

(case insensitive, if the entry does not match one of these an error will be produced).

DialogResult variable will be created afterwards with ‘Abort’, ‘Cancel’, ‘Ignore’, ‘No’, ‘OK’ , ‘Retry’ , ‘Yes’

### Print

Print to log window. Any escape sequences in the string is replaced by their non escaped equivalents.

Print <string>

<string> contents to log window.

## Audio Statements

### Play

Play audio

PLAY [<filename>|<tone>][ ‘,’ <configurationlist>]

<filename> = file name of file to play. This is treated as an un-escaped string.

<tone> = “TONE” [24]

<configurationlist> = <configurationvalue> [ ‘,’ <configurationvalue>]..

<configurationvalue = <item> ‘=’ <value>

<item> = configuration items: see below

<value> = value of item.

For All:

***Volume*** is the volume to play at, either Default, or 0-100. If -999 is used, use the default value. Default value is either 60, or global WaveVolume if defined.

***Wait*** if set and 1, wait for speech to finish before continuing executing more commands.

***StartEvent*** if set, generate an event onPlayStarted with EventName variable set to this contents. String must be not empty to generate this.

***FinishEvent*** if set, generate an event onPlayFinished with EventName variable set to this contents. String must be not empty to generate this.

***Priority*** if set sets the priority, Normal is the default

**Effects variables** see the Say section. WaveEffect holds the global wave effects

[29]***Para=”<name=value> [,<name>=value]..”*** means take this string of comma separated configuration settings and apply it to the play command. Using it with a variable such as Para=”%(playsettings)” will allow you to use one variable to store all of the configuration.

For Tone [24]:

**Frequency** is the frequency in Hz to produce the tone at.

**Duration** is the length of time for the tone in ms

**Attack** sets up an ADSR volume envelope. It must be present for ADSR to work. Set the attack time in ms from 0 up. Then:

**Decay** sets the ADSR decay time in ms. Default is 0.

**Sustain** sets the ADSR sustain time in ms. If not present, sustain is near infinite, there is no release phase.

**Release** sets the ADSR sustain time in ms. Default is 1000ms

**SustainVolume** indicates the relative level of sustain from the peak of attack (0-100). Default is 100 if Decay is not give, 50 if given

### Say

Speak a phrase using the systems text to speech engine.

SAY <phraselist|speechtext> [ ‘,’ <configurationlist>]

<phraselist> = When Literal is not present in the configuration list, or is zero, a phrase list is given as the speech to say. See below for the format of a phrase list.

<speechtext> = When Literal is present and non zero, literal text is given as the speech to say.

<configurationlist> = <configurationvalue> [ ‘,’ <configurationvalue>]..

<configurationvalue = <item> ‘=’ <value>

<value> = value of item.

<item> = configuration items, one of:

***Volume*** is the volume to speak at, either Default, or 0-100. If -999 is used, use the default value. Default value is either 60, or global SpeechVolume if defined.

***Voice*** is the voice to use instead of the one defined by the global SpeechVoice variable. If this value is empty the option is ignored and the global voice is used.

***Rate*** is the speed to speak, Default or -10 to 10. If -999 is used, use the default value. Default value is either the value in the global SpeechRate, or 0 if not present.

***Priority*** if present sets the priority, Normal is the default, or you can use Low, High or HighClear[18], or the value can be set by the global SpeechPriority[7] when defined.

Speech marked with a higher priority than the current top of the queue is inserted at the front of the queue, to be played next. All entries marked low priority are removed from the front of the queue in this situation. The current speech is stopped immediately if it’s Low or Normal priority, else it’s played out. High Clear inserts the speech at the start of the queue, and clears the queue afterwards of all speech.

***Culture*** if the option is present, and the value is not empty, sets the speech culture hint to windows (such as en-gb). Else use the culture given in the global SpeechCulture, or no hint if its not present.

***Wait*** if set and non zero, wait for speech to finish before continuing executing more commands.

***StartEvent*** if set, generate an event onSayStarted with EventName variable set to this content when sound starts. String must be not empty to generate this.

***FinishEvent*** if set, generate an event onSayFinished with EventName variable set to this content when all sound is complete. String must be not empty to generate this.

***Literal***, if set and non zero, says don’t process string for grouping, just say it literally

***DontSpeak***, if set and non zero, means don’t speak the phrase. The start/finish events occur but the audio is about 10ms long of blank audio. Prefix and postfix audio are played. Also check the global SpeechDisable. If this is non zero, don’t speak the phrase [13]

***PrefixSound,*** If the option is present, and the value is not empty, give an audio file to play before the speech starts.

***PostfixSound,*** If the option is present, and the value is not empty, give an audio file to play after the speech end.

***MixSound,*** If the option is present, and the value is not empty, give an audio file to play during speech. This sound is mixed with speech. If the speech is longer than the mix sound, the mix sound is looped.

***QueueLimit***, set to a positive (>0) integer number of milliseconds. If there is sound in the speech queue longer than this, the phrase is not said audiably. No events occur, SaySaid is set tot “!LIMIT”.

**Effects variables** see the Say section below. SpeechEffect holds the global wave effects

***Literal*,** If the option is present and the integer value is non zero, then say the text as

[29]***Para=”<name=value> [,<name>=value]..”*** means take this string of comma separated configuration settings and apply it to the say command. Using it with a variable such as Para=”%(saysettings)” will allow you to use one variable to store all of the configuration. If Para is empty, no variables are added.

The variable SaySaid will be written with the text actually spoken (including any SSML markup), or “!LIMIT” if the text is culled due to limiting.

SpeechDebug is a special variable which if defined allows special debugging of the say command. Use one or more of these constant strings to turn on specific debugging:

‘Print’ prints the speech text to the log window.

‘Global’ sets the global GlobalSaySaid to the same valid as SaySaid.

‘DontSpeak’ is the same as the DontSpeak option above.

‘Mute’ mutes all output. No sound is queued. No events occur.

‘SayLine’ prints to the logline extended information on say variables

#### Phrase Lists

When a <phraselist> is present, it is defined as following:

<speech> = [<group> | <phraseset>]..

<group> = ‘{‘ <phraseset> ‘}’

<phraseset> = [‘;’] <text> [ ‘;’ <text>]..

<text> = Unicode text to say, or SSML mark up.

The separator string ‘;’ and the group opening string ‘{‘ and closing string ‘}’ are fixed for the Say command. See %phrase for the ability to use this format but with definable separator and group strings.

The string consists of optional phase groups and phrase sets.

A Phrase group in the Say command is marked by using group opening and closing strings and contains inside one phrase set.

A phrase set is a set of one or more speech elements, each element separated by a the separator string. One of those speech elements will be picked as follows:

* If the phrase sets starts with the separator, it will pick the first element, and then randomly one of the following elements (example “;a;b;c” will pick a and randomly pick b or c).
* If the phrase sets does not start with the sepeator, it will randomly pick one of the elements (example “a;b;c” will randomly pick a or b or c).

An example complex string with groups and phrasesets is “{a;b;c}d{e;f;g}h”, resulting in either a b or c being picked, followed by d, followed by either e or f or g being picked, followed by h.

[6] After the phrases has been assembled, each phrase can contain both normal text and optionally Speech Synthesis Mark Up Language (SSML) elements. They may be freely intermingled. The following SSML mark-up is supported:

* <say-as interpret-as=’<t>’ format=’<f>’> text </say-as>
* <sub alias = ‘<t>’> ignoredtext </sub>
* <phoneme ph=’<p>’> ignoredtext </phoneme>
* <prosody rate=’<p>’> text </prosody>
* <prosody volume=’<p>’> text </prosody>
* <emphasis level=’<p>’> text </emphasis>

See the Microsoft speech synthesiser Mark-up Language Reference for detailed list. Only these elements shown above are supported. Not all options for these elements always work with all speech engines. Phoneme lists should use UPS or IPA phoneme characters sets. IPA requires the action file to be in Unicode-16.

#### Effects for Say and Play

The following effect variables are defined:

EchoMix, EchoFeedback, EchoDelay

ChorusMix, ChorusFeedback, ChorusDelay, ChorusDepth

ReverbMix, ReverbTime, ReverbRatio

DistortionGain, DistortionEdge, DistortionCF, DistortionWidth

GargleFreq

PitchShift

Global sound effects found in the global SpeechEffects are applied to the phrase unless an effect listed above is present on the command line.

If NoEffects=<any> is given on the command line then no effects are made

If MergeEffect=<any> is given on the command line the global and command line effects are merged, global first, command line second.

If NoGlobalEffects[29] is given on the command line then only local effects given on the command line are applied. The Global SpeechEffects is not used.

## Other Statements

### Events

Enable, disable, add, remove events:

EVENTS [‘PREFIX’ <name> ] ‘ADD’ | ‘REMOVE’ | ‘DISABLE’ | ‘ENABLE’ | ‘DISABLEALLBUT’ | ‘ENABLEALLBUT’ | ‘LIST’ <eventdescription>

<name> = variable prefix, if not given EV\_ is the default. If <name> is missing after prefix an error will be produced.

<eventdescription> = <groupname>, <triggername>, <programname>, <variable list>, <condition>

<groupname> = group name of event to define, or to search for (wildcards permitted, case insensitive). Use “” to indicate don’t search on this.

<triggername> = trigger to define, or to search for (wildcards permitted for search). Use “” to indicate don’t search on this.

<programname> = program to refer to, or to search for (wildcards permitted for search). Use “” to indicate don’t search on this.

<variablelist> = variable list to define, or to search for (wildcards permitted for search). Use “” to indicate don’t search on this.

<condition> = condition to define, or to search for (wildcards permitted for search). Use “” to indicate don’t search on this.

For ADD: Add this event to the program event list. The <eventdescription> must be complete without wildcards. The following variable will be written:

<prefix> Result = 1 succeeded, 0 failed (event already exists)

For DELETE: Delete all events matching the pattern. The following variable will be written:

<prefix> Count = Number deleted

For DISABLE: Disable all events matching the pattern. The following variable will be written:

<prefix> Count = Number disabled

For ENABLE: Enable all events matching the pattern. The following variable will be written:

<prefix> Count = Number enabled

For DISABLEALLBUT: Disable all events NOT matching the pattern. The following variable will be written:

<prefix> Count = Number disabled

For ENABLEALLBUT: Enable all events NOT matching the pattern. The following variable will be written:

<prefix> Count = Number enabled

For LIST: list all matching events. The following variables will be written:

<prefix> Count = match count. N will be from 1 to count.

<prefix> Event[n] = event definition, same format as EVENT command

<prefix> Enabled[n] = 1 enabled, 0 not

<prefix> GroupName[n] = groupname, empty if no group assigned

# Functions

Functions operate on variables or fixed function parameter information and allow conversion and expansion functions on the variable mid string. Function names are case insensitive.

Functions take zero or more parameters. Optional parameters at the end of the line is supported.

Each parameter can be in the form of one of:

* Name of a variable. (quoted or unquoted)
  + Use either quoted or unquoted text. If a % is present the expression is expanded to provide an indirection method for variable names.
* Name of a variable (unquoted) or literal value (quoted)
  + Unquoted means it’s a variable name. If a % is present the expression is expanded to provide an indirection method for variable names.
  + Quoted string means it’s a literal value. If a % is present in the string the expression is expanded.
* Literal or a name of variable (unquoted), or literal value (quoted)
  + Unquoted means it’s a variable name or a literal value. If a % is present the expression is expanded. The result is tested to see if it’s a literal value (10.2), if not its treated as a variable name.
  + Quoted string means it’s a literal value. If a % is present in the string the expression is expanded.

## Variables

### %Exist(<varname> [ , <varname> ])

Does all of the variables exist, 1 if true, 0 otherwise.

<varname> = name of a variable.

### %ExistsDefault(<varname> , <default1> [,<default2> [,<default3>[,<default4>,<default5>]]])

If <varname> exists, use its text, else use the one of the default entries. The first default entry which is a string is returned. Of the first variable name found is returned. If no entry exists, the function returns an error.

Prior to [29] only default1 was supported.

<varname> = name of a variable

<defaultN> = variable name or “literal string”.

### %Expand(<varname> [ ,<varname>]..)

Given a variable, expand it (i.e resolve any % functions or variables). This is useful in conjunction with the ‘$’ operator on set to delay expansion of variables until required.

<varname> = name of a variable

### %ExpandArray(<arrayrootname>, <separ>, <start>, <length> [, <specialfunc>, [<postfix>],[‘lastsepar’])

Look up variables starting with <arrayrootname>[n]<postfix>, and expand the contents of them to a list, separated by <separ> or <lastsepar>, in the range from <start> with maximum of <length> items. Start and length determine the number of entries. Length can be greater than the number of entries.

<arrayvarname> = name of a variable

<separ> = variable or “string”

<start> = literal, variable or “string” containing the integer value. Note that Action normally uses 1 as the first element, so this normally be 1

<length> = literal, variable or “string” containing the integer value.

<specialfunc> = literal, either not present or one or more of (case ignored):

Splitcaps : Splitcaps runs the split caps function on each entry.

Ignoremissing [28]: Normally, if an index variable is missing, expansion stops. This prevents this and expands all present variables between start and start+length-1

<postfix>[8] = variable or “string”. If not present, its empty string

<lastsepar>[24] = variable or “string”. The last separator to use. If not given, use <separ> <24>

### %FindInArray(<arrayrootname>, <fieldname>, <findstring>, <start>, <length> [, <specialfunc>)[28]

Look up variables starting with <arrayrootname>[n]<fieldname>, from start to start+length-1

Compare them case insensitive with the findstring. The normal comparision is equals, unless overridden by <specialfunc>.

Return the index of the find, or -1 none.

Start and length determine the number of entries. Length can be greater than the number of entries.

<arrayvarname> = name of a variable

<fieldname> = field name or “string”

<findstring> = what to find, case insensitive.

<start> = <length> = literal, variable or “string” containing the integer value.

<specialfunc> = literal, either not present or one or more of:

casesensitive : make test case sensitive

contains : entry contains this

startswith : entry starts with

Ignoremissing : Normally, if an index variable is missing, checking stops. This prevents this and checks all variables between start and start+length-1

### %Expandvars(<rootname>, <separ>, <start>, <length> [, <specialfunc>])

Look up variables starting with <rootname> and expand the contents of them to a list, separated by <separ>. Each entry will be in the form <name of variable with prefix of rootname removed> = value. Start and length determine the number of entries. 1 is the first one. Length can be greater than the number of entries.

<rootname> = name of a variable

<separ> = variable or “string”

<start> = <length> = literal, variable or “string” containing the integer value.

<specialfunc> = literal, either not present or one of:

If specialfunc contains “nameonly”, only the name part will be written.

If specialfunc contains “valueonly”, only the value part will be written.

If specialfunc contains “splitcaps” then the split caps function is applied to each entry.

Any other special function characters are ignored.

### %FindArray(<arrayrootname>,<searchname> [‘,’ <startafter>] )

Look up variables starting with <arrayrootname> and search case insensitive inside the variables for searchname>. Return the name of the first variable with the contents matching <searchname>. If none is found, return empty string. If <startafter> is given, ignore all variables up to and including <startafter> and start searching after this. If <startafter> is empty, then it starts immediately.

<arrayrootname> = name of a variable

<searchname> = variable or “string” giving search criteria

<startafter> = variable or “string” giving variable name.

### i%Indirect(<varname> [‘,’ <varname>..)

Given a <varname>, look into that variable value and use that variable value as the name of another variable, and expand the variable (i.e resolve any % functions or variables). Useful for delayed expansion where a variable is set up to point to another variable. E.G pass in the name of the variable to a program which contains program data.

<varname> = name of a variable

### %I(<varname>, <postfix>) [8]

Given a <varname> and a literal string <postfix>, expand a variable called %<varname><literal> as per the indirect function. So it <varname> contained ”Root” and literal was 1, the function would lookup and expand the variable called Root1. A more compact form of %Indirect.

<varname> = name of a variable, must exist.

<postfix> = literal or “string”, postfix characters.

### %IsPresent(<varname> [’,’ <search> [’,’ <default>])

Detects if the contents of <search> is in <varname> and allows <varname> not to exist.

If <varname> is not defined, return the value in <default> or 0 if its <default> is not defined

If the <varname> is defined, and the <varname> contains the string in <search> (case insensitive), return 1, else return 0

<varname> = name of variable, looked up if it exists.

<search> = variable or ”string”, search term to look in varname for.

<default> = variable, or literal, or ”string”.

### %Variable(<varname> [‘,’ <varname>..) [24]

Given a <varname>, look into that variable value and use that variable value as the name of another variable, and return the contents of that variable. Do not expand the contents.

<varname> = name of a variable

## Numbers

### %Abs(<value>, <outputformat>)

Given the double <value> absolute it to positive. Print using <outputformat>.

<value> = Number literal, variable containing a number, or “string containing number”

<outputformat> = Format literal, variable or “string”.

This must be a c# tostring format (0.# , Nn, Pn, Gn, Fn, En). ‘n’ is optional.

‘M’ can be used in front of the format specifiers to mean replace a negative ‘-‘ with ‘Minus ‘, for speech use.

‘M=<text>;’ [29] can be used in front of the format specifiers to mean replace a negative ‘-‘ with the text up to the semicolon, for speech use.

“CurC”[10] can be used as a prefix to select the current UI culture instead of invariant.

See <https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings>

### %Int(<value>, <outputformat>) [6]

Given an integer value, print it using the format specifier provided.

<value> = Integer literal, variable containing a number, or “string containing number”

<outputformat> = see %Abs but with also “O” is to octal, “B” is to binary.

### %Eval(<expr> [‘,’ <option> [‘,’ <outputformat>[23]]])

Given the expression <expr>, evaluate it, and Print the value in decimal double in invariant culture (dot for decimal point).

<value> = Number literal, variable containing a number, expression (10-20), or “string containing expression”

<option> = Either “Error” (default) or “Try” is supported. The option “Try” supresses the error if the expression does not evaluate and returns “NAN” instead.

<outputformat> = See %abs. If not present, output cultural invariant with no specific formatting.

Expression is as per Let.

### %Floor(<value>, <outputformat>)

Given the double value, truncate to the integer using floor. See %Abs.

### %Hnum(<value>, <postfixlist>) [6]

Given the double value, make it human readable and reduce its complexity.

<value> = Number literal, variable containing a number, or “string containing number”

<postfixlist> = variable or “string” or literal. This must be six semi colon separated values to describe the name of the units in the chosen language. For English it is “Minus;Trillions;Billions;Millions;Thousands;Hundreds”.

### %If ..Numbers

Many forms of this are supported.

%IfTrue, %IfFalse (<vb>, <v-true> [, <v-false>])

If v1 is true (non zero) for IfTrue, false for IfFalse, expand v-true, else expand v-false if present, else empty string.

<vb> = Integer literal, variable containing a number, or “string containing number”

<v-true> = <v-false> = <v-not-present> = variable or “string”

%IfZero, %IfNonZero (<vf>, <v-true> [, <v-false>])

If vf is zero for IfZero, non zero for IfNonZero, expand v-true, else expand v-false if present, else empty string.

%IfGT, %IfLT,%ifGE,%ifLE,%IfEQ,%IfNE (<vf>,<v-compare>, <v-true>, [<v-false> [,<v-not-present>]])

If vf is empty and v-not-present is present, expand v-not-present.

Else Vf must then have a valid number in it. Compare vf with v-compare (according to condition) and expand v-true if condition is true, else expand v-false if present, else empty string.

<vf> = Empty string (must have v-not-present defined), or Number literal, variable containing a number, or “string containing number”

<v-compare> = Number literal, variable containing a number, or “string containing number”

### %Random(<range>)

Give a decimal random number between 0 and the <range>-1, so %Random(10) gives 0-9. Invariant culture.

<range> = literal integer, variable containing a number, or “String” containing an integer.

### %Round(<value>,<digits>,<outputformat>)

Given the double value, round to the number of digits, and print in decimal. Invariant culture.

<value> = Number literal, variable containing a number,, or “string containing a number”

<digits> = Integer literal, variable containing a number, or “string containing a number” containing the number of decimal places to round to. 0 to N.

<outputformat> = see %Abs

### %Roundnz(<value>, <digits>, <outputformat>, <extradigits>)

Given the double value, round to the number of digits . If this results in a zero, add on extra digits and add the same number of ‘#’’s onto the outputformat, and retry. Print in decimal.

<value> = Number literal, variable, or “string containing a number”

<digits> = Integer literal, variable or “string containing a number” containing the number of decimal places to round to. 0 to N.

<outputformat> = “CC” as per Abs, and must be in the 0.0# format at the end

<extradigits> = Integer literal, variable containing a number, or “string containing a number” to add on the number of decimal places to print to.

### %Roundscale(<value>, <digits>, <outputformat>, <extradigits> ,<scale>)

Given the double value, multiple by scale, and then do the same as %Roundnz.

<scale> = Number literal, variable containing a number,, or “string containing a number” to scale by.

### %SeedRandom(<seed>) [8]

Seed the random generator with a fixed seed to then give a fixed sequence. Seed can be any integer.

<seed> = literal integer, variable containing a number, or “String” containing an integer.

### %ToString(<value>, <outputformat>) [29]

Given a floating point value, print it using the format specifier provided.

<value> = Floating point or integer literal, variable containing a number, or “string containing number”

<outputformat> = see %Abs

## Strings

### %Alt(<text>, <alt-text> [,<alt-text>]…)

If <text> has text, use it, else try <alt-text>, then try any further ones given. If none has text, return empty string.

<text> <alt-text> = variable or “String”.

### %EscapeChar(<text>)

Replace \r, \n or \ with their escaped equivalents, [\\r](file:///\\r), [\\n](file:///\\n) or \\.

<text> = variable or “String”.

### %ICAO(<text> [, <enabledash>])

Replace A-Z and 0-9 with ICAO (Alpha,Brave,Charlie) phonemes for speech. Optionally allow dash through. All other characters are ignored and not copied to the output.

<text> = variable or “String”.

<enabledash> = literal, variable or “String” containing a string. Use “Dash” to enable dash to be outputted.

### %If ..Strings

Many forms of this are supported.

%IfEmpty,%IfNotEmpty(<v1> , <v-true> [, <v-false>])

If <v1> is empty (zero length) or not empty, expand <v-true>, else expand <v-false> when present, else empty string.

%IfEqual, %IfNotEqual, %IfContains, %IfNotContains (<v1>, <v-compare>, <v-true> [,<v-false> [, <v-not-present>]] )

If <v1> is empty and <v-not-present> is present, expand <v-not-present>. Else if condition between <v1> and <v-compare> matches (string case insensitive comparison), expand <v-true>, else expand <v-false> if present, else empty string.

<All items> = variable or “String”. Note for items which are not expanded because the condition did not select them, no expansion or check for variable existence is performed.

### %Indexof(<text> ,<search>)

Given <text>, find the position of <search> (case insensitive) is present. -1 if not found, zero based.

<text> <search> = variable or “String”.

### %JsonParse(<text>, <root>) [30\*]

Decode JSON in <text> into variables starting with <root>. Error if JSON is invalid.

<text> = variable or ”String”.

<root> = variable name root as a literal

Note in [30] the output format has changed to:

<root> [[.<objectname>] [<[N]>] [<[]\_Count]]…

So ["Fred"] = new JArray { 1, 2, 3 } will be represented as, given the root name json as

Json.Fred[]\_Count = 3

Json.Fred[1] = 1

Json.Fred[2] = 2

Json.Fred[3] = 3

And {"Fred":10,"Jim":20,"Abby":{"Clancy":10,"George":{"david":10,"edward":20}},”Flag”:true,"End":20} as:

json.Fred = 10

json.Jim = 20

json.Abby.Clancy = 10

json.Abby.George.david = 10

json.Abby.George.edward = 20

json.Flag\_BOOL=1

json.End = 20

JSON Bools are converted to 1 or 0 with a \_BOOL on the end of their name.

### %ToJson(<root>)[30]

Return a JSON representation of the variables starting with <root> (as per the rules defined by JsonParse). Error if no root variables exist.

### %Join(<delimiter>, <text>, <text-2>, [, <add-text>]..)

Join <text> together with <text-2> and any <add-text>, using the delimiter <delimiter> between them. Minimum 3 values.

<delimited> <text> <text-2> <add-text> = variable or “String”.

### %Length(<text>)

Given <text>, how many characters are in its value.

<text> = variable or “String”.

### %Lower(<text> [, <text>] ..) %LowerInvariant

Given <text>, return its value all converted to lower case. If three or more <text> entries are present, the first is the <text> value, the second is the delimiter to be placed between values, and the third entry on is further values: lower(v1,”,”,v2,v3,v4).

Lower respects current culture of the machine. Use LowerInvariant to use a standard translation.

<text> = variable or “String”.

### %Phrase(<text> [,<phase separator string> [,<group open separator string> [,<group close separator string>]]])

Using the same algorithm as Say, pick a phrase from the <text>.

<text> = variable or “String”

<phrase separator> allows you to pick a different phrase separator string than ‘;’. [24]

<group open separator> allows you to pick a different group opener string than ‘{’. [24]

<group close separator> allows you to pick a different group opener string than ‘}’. [24]

### %PhraseL(<text>) [30]

Using the same algorithm as Say, pick a phrase from the <text>, using fixed phrase separators “<;>”, “{{{“, “}}}”

<text> = variable or “String”

### %Regex(<text>, <pattern>, <replace>)

Perform a regular expression search on <text> using <pattern>, and replace with the <replace> pattern.

See <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions> and <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference> for help on the patterns accepted.

<text> <pattern> <replace> = variable or “String”.

### %Replace(<text>, <search>, <replace>)

Replace any case insensitive <search> parts in <text> with <replace>.

<text> <search> <replace> = variable or ”String”

### %ReplaceVar(<text>, <variable-rootname>) or %rv(..) or %rs

The <variable-rootname> defines the prefix of a set of variables to search for substitutions. All variables prefixed with this will be searched for substitution patterns for replacing parts of <text>.

<text> = variable or ”String”

<variable-rootname> = Literal, variable or ”String”. Root name of replacement variables.

The substitution pattern in each variable is a semicolon separated list. If exactly two items are given (A;B) then A is searched for in any position in the string and replaced by B, case insensitive.

If the pattern (R;<pattern>;<rep-pattern>) [6] is used, then the Regex function is used to perform the replacement on that item only, case sensitive at start. Use ‘r’ for case insensitive. See %RegEx. This is slower than the above method.

For instance, if a variable Replace\_1=fred;jim, and you applied %ReplaceVar(“fred”,Replace\_), then the variables would be searched for Replace\_, Replace\_1 would be found, and the substitution pattern fred;jim would be applied. “fred” would be replaced with “jim”.

No error occurs if no substitution pattern variables are found.

%rv is a short synonym for the longer name. %rs performs the replacement, then runs the %splitcaps function on the result.

### %ReplaceEscapeChar(<text>)

Replace \\r, \\n or \\ with their single character equivalents, [\r](file:///C:\r), [\n](file:///\\n) or \.

<text> = variable or ”String”

### %ReplaceIfStartsWith(<text>, <startswith>, [<replace>]) [13]

Replace any case insensitive <startswith> in <text> at the start of the string with parts in <replace> (optional, if not present remove the text entirely)

<text> <search> <replace> = variable or ”String”

### %Splitcaps(<text>) (or %sc..)

Given <text>, expand out Camel case form (HelloFred = Hello Fred) and remove any underscores.

<text> = variable or ”String”

### %Substring(<text>, <start-index>, <length>)

Given <text>, cut it at <start-index> for <length> characters. If index is out of range, return empty string. If length is too long, return as many as possible.

<text> = variable or ”String”

<start-index> = Literal Integer, variable or “String” containing an integer. Index is 0 based (first character).

<length> = Literal Integer, variable or “String” containing an integer.

### %Trim(<text>)

Given <text>, return its value with whitespace at start and end trimmed.

<text> = variable or ”String”

### %Upper(<text> [,<text>]..) %UpperInvariant

As per lower, but for upper case.

### %Version(<version>)

Return version part of the program in a four digit version number. 1 is the highest part, 4 is the lowest. Or use 0 to get all the digits as X.Y.Z.A

<version> = literal integer, variable or “String” containing an integer.

### %WordCount(<wordlist> [, <delimiter>]) [24]

Given a list in <wordlist>, separated by a single character from <delimiter> (default is ‘;’), return the number of words.

<wordlist> <delimiter>= variable or “String”.

### %WordFind(<wordlist>, <text> [, <delimiter>], [<caseinsensitive>], [<contains>]) [24]

Given a list in <wordlist>, separated by a single character from <delimiter> (default is ‘;’), find the text in the list. Returns 1 to N, or 0 for not found.

<wordlist> <text> <delimiter>= variable or “String”.

<caseinsensitive> = literal integer, variable or “String” containing an integer. 0 = sensitive, 1 = insensitive. Default 1.

<contains> = literal integer, variable or “String” containing an integer. 0 = text must fully match, 1 = any part of text in word matches

### %WordOf(<wordlist>, <wordnumber> [, <delimiter>])

Given a list in <wordlist>, separated by a single character from <delimiter> (default is ‘;’), give the <wordcount> word. If the count is too high, it returns the last, if too low, it returns the first. <wordcount> is 1 to N.

<wordlist> <delimiter>= variable or “String”.

<wordnumber> = literal integer, variable or “String” containing an integer.

### %WordListCount(<text> [, <delimiter> [8]])

Given a list of strings in <text>, optionally quoted, separated by any characters from <delimiter> (default is ‘,’) or space, give the number of entries. 0 if no words are found (due to a formatting problem).

<wordlist> = variable or “String”.

### %WordListEntry(<text>, <wordnumber> [, <delimiter> [8]])

Given a list of strings in <text>, optionally quoted, separated by any characters from <delimiter> (default is ‘,’) or space, give the <wordnumber> entry. <wordnumber> entry is 0 to N-1. If it’s too high or too low or the text is not a word list, it returns an empty string.

<wordlist> = <delimiter>= variable or “String”.

<wordnumber> = literal integer, variable or “String” containing an integer.

## Files

### %CloseFile(<handle>)

Close file associated with handle. No error will be produced if the handle is already closed.

<handle> = name of handle, or “string” containing a handle name.

### %CombinePaths(<pathpart1>,<pathpart2>…)

<pathpart> = variable or “string”, part of path

Produce a combined path, given the path parts. Example is CombinePaths( ActionPackFolder,"fred.png")

### %DirExists(<path> [, <path>]..)

If all folders named exist, return 1. Else return 0.

<path> = variable or “string” of path to folder.

### %DirectoryName(<filepath>) [30]

Return directory name of path.

<filepath> = variable or “string” of path of file.

### %Extension(<filepath>) [30]

Return extension of path.

<filepath> = variable or “string” of path of file.

### %FileExists(<filepath> [,<filepath>]..)

Does all files given by variables, or file paths if in quotes, exist? Return 1 or 0.

<filepath> = variable or “string” of path of file.

### %FileList(<path>, <wildcard>) [8]

Pass back a list of quoted list of files found in this path using the wildcard filename as the search pattern. Error if the path does not exist.

<filepath> = variable or “string” of path of file.

<wildcard> = variable or “string” containing a filename or filename wildcard pattern using \* and ?

### %FileLength(<filepath>)

What is the file length of this file or -1 if file does not exist.

<filepath> = variable or “string” of path of file.

### %FileName(<filepath>) [30]

Return file name of path (inc extension).

<filepath> = variable or “string” of path of file.

### %FileNameNoExtension(<filepath>) [30]

Return file name of path (without extension).

<filepath> = variable or “string” of path of file.

### %Findline(<filepath>,<matchstring>)

Find in the <filepath> file the first line containing <matchstring>, case insensitive, and return the whole line it is on. Empty string if not found.

<filepath> = variable or “string” of path of file.

<matchstring> = variable or “string” containing the match string.

### %FullPath(<filepath>) [30]

Return normalised path of file path.

<filepath> = variable or “string” of path of file.

### %MkDir(<path>)

Ensure directory exists. Return 1 if made, 0 otherwise.

<path> = variable or “string” of path to folder.

### %OpenFile(<handle>, <filepath>, <mode>)

Open the <filepath> file in <mode> and use the <handle> variable to hold the reference to it. Return 1 if opened, 0 if not. If 0, the handle will be set to the reason why it did not open.

<handle> = name of handle, or “string” containing a handle name.

<filepath> = variable or “string” of path of file.

<mode> = Literal, variable or “string” containing the mode to open file in, one of:

* Append : File must exist, write to end
* Create: Write a new file, overwrite current, write to beginning.
* CreateNew,: Write a new file, file must not exist, write to beginning
* Open: Read from a file, file must exist
* OpenOrCreate: Write to a file, if file exists overwrites, else makes a new file
* Truncate: Write to an existing file and truncate it, write to beginning.

### %ReadLine(<handle>, <linevar>)

Read a line from <handle> into <linevar>. Return 1 if success, 0 if end of file.

<handle> = name of handle, or “string” containing a handle name.

<linevar> = name or “string” giving the name of a variable to set to receive the line.

### %ReadAllText(<filepath>)

Read the whole file and return it. Error if the file does not exist.

<filepath> = variable or “string” of path of file.

### %RmDir(<path>)

Remove the directory if it exists and is empty. Return 1 if made, 0 otherwise.

<path> = variable or “string” of path to folder.

### %SafeVarName(<unsafevarname>)

Return a variable name made out of the <unsafefilename> replacing any invalid characters with “\_”.

<unsafevarname> = variable or “string” of variable name.

### %Seek(<handle>, <seekpos>)

Seek the position of file to this <seekpos> position. 1 if succeeds, 0 otherwise.

<handle> = name of handle, or “string” containing a handle name.

<seekpos> = Integer, variable or “string” giving the position to move to.

### %Tell(<handle>)

Tell the position of this file, as a decimal value.

<handle> = name of handle, or “string” containing a handle name.

### %Write(<handle>, <text>)

Write the <text> into the file. 1 if succeeds, 0 otherwise.

<handle> = name of handle, or “string” containing a handle name.

<text> = variable or “string” containing the text to write.

### %WriteLine(<handle>, <text>)

Write the <text> plus a line feed into the file. 1 if succeeds, 0 otherwise.

<handle> = name of handle, or “string” containing a handle name.

<text> = variable or “string” containing the text to write.

## Processes

### %CloseProcess(<handle>)

Kill the process given by the handle. Returns “1” if the process is found. Closing is not guaranteed.

<handle> = variable name of handle assigned in %startprocess.

### %FindProcess(<processname>)

Find the ID of a process, or 0 if it does not exist.

<processname> = variable or “string”.

### %HasProcessExited(<handle>)

Return if process has exited. If it has, it returns the exit code. If not, it returns “NOTEXITED” string.

<handle> = variable name of handle assigned in %startprocess.

### %Killprocess(<handle>)

Kill the process given by the handle. Returns “1” if the process is found. Killing is not guaranteed.

<handle> = variable name of handle assigned in %startprocess.

### %Listprocesses(<literalarrayroot>)

List all the processes running on the machine at this point, and store them in variables called arrayroot[n], n = 1 onwards.

<literalarrayroot> = variable name.

### %Startprocess(<processpath>, <cmdline>)

Run the process given by cmdname with the cmdline. Return the handle (PID) of the process.

The first time a process is requested to run, a confirmation dialog will appear which the user must accept to allow the process to run. Afterwards, it will run without prompting.

<processpath> = variable or “string”. Fully qualified or partial name of process

<cmdline> = command line to pass to process.

Returns the handle to use. Store this in a variable and use the variable for the <handle> parameter.

### %WaitForProcess(<handle>, <timeout>)

Wait for the process to terminate for timeout ms. Return 1 if the process terminated, 0 if not. Beware use – this will stall EDD until the timeout or process exits.

<handle> = variable name of handle assigned in %startprocess.

<timeout> = Literal Integer, or variable, or “string” containing an integer.

## Date and Time

### %Date(<date>, <date-format>)

Taking a date, the default format being US Form, read the date, then expand to a string in a defined format. The date is presumed to be a UTC date time, unless you use Local in the format. Any timezone information in the string will cause local to be ignored.

<date> = variable or “string”. Date in US form.

<date-format> = literal, variable or “string”. This is a semicolon separated case insensitive list of options, in any order. If multiple formats are given, the first one is used. Names are case insensitive:

* LongTime
* ShortTime
* LongDate
* ShortDate
* LongDateTime = (Long Date + Long Time)
* DateTime =ShortDate+LongTime.
* Ticks = Seconds since 1 January 1.
* Or default: yyyy/dd/mm hh:mm:ss if none of the above is found.

To convert to UTC/Local:

* ToLocal to convert a UTC to local.
* ToUTC to convert local to UTC.

[8] Output will be in the current culture of the machine, unless:

* Culture:<culture id> where id is the IOS code for culture (see https://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo(vs.71).aspx)

Example: %date(..,local;longdate;toutc)

### %DateTimeNow(<date-format>)

Print the current time as per %Date. Returns the UTC time unless options is set to “ToLocal”.

### %DateDelta(<date1>, <date2> [, <timezone-option>]) [6]

Return the number of seconds (may be fractional) between these two dates. Positive if datetime2 is younger than 1. If a date has a timezone marker, that will be used in the difference.

<date1> <date2> = variable or “string”. Date in US form

<timezone-option> = literal, variable or “string”. Only valid option is “local” to override the presumption of UTC for both dates, the only effect will be if one is timezone marked at the other is not.

### %DateDeltaFormat(<delta>, <beforetimeformat>, <aftertimeformat>)

Taking a delta time in seconds (may be a fractional number), return the delta time in <beforetimeformat> if the delta time is positive or <aftertimeformat> if delta time is negative or zero.

<delta> = fractional number or variable or “string”. Fractional value in seconds between two times.

<beforetimeformat> <aftertimeformat> = variable or “string”. The format string consists of text, with parameters embedded in [] brackets. The first characters after the opening bracket tell the formatter what to place in the text instead of the [].

[s<opttext>] Seconds, if non zero

[S<opttext>] Seconds, if non zero, and Min=Hours=Days=0.

[Sh<opttext>] Seconds, if non zero, and Hours=Days=0.

[m<opttext>] Minutes, if non zero

[M<opttext>] Minutes, if non zero, and Hours=Days=0.

[Md<opttext>] Minutes, if non zero, and Days=0.

[h<opttext>] Hours, if non zero

[H<opttext>] Hours, if non zero, and Days = 0

[d<opttext>] Days

[D<opttext>] Date (only for %datedeltaformatnow and %datedeltadiffformat)

<opttext> = Add on the opttext if present.

If the condition fails, such as a parameter is zero, all the text and the brackets are removed from the string.

Inside the opttext, plural characters may be indicates using {plural|nonplural} form. The plural will be used if the value is greater than one.

An example is:

'Expires in [d day{s|} ][h hour{s|} ][m minute{s|} ][s second{s|}]

### %DateDeltaFormatNow(<date>, <beforetimeformat>, <aftertimeformat>, [, <format-options>])

As per DateDeltaFormat but giving a date vs the current time.

<date> = variable or “string”. Date in US form.

<beforetimeformat> <aftertimeformat> = as per %DateDeltaFormat above.

<format-options> = = literal, variable or “string”. As per %Date above and used to format the ‘D’ option. Also, can apply ‘local’ time zone modifier to the date.

### %DateDeltaDiffFormat(<date1>, <date2>, <beforetimeformat>, <aftertimeformat>, [, <format-options>])

As per DateDeltaFormat but giving two dates instead of against the current time.

### %TickCount()

Current tick counter, as an integer.

# Events

Events are introduced in the Action script outside of a program, in the form:

EVENT <triggername>, <programname>, <variable list>, <condition>

<triggername> is the logical trigger name which is case sensitive. Use ALL or All to trap all events.

<programname> is the program to run. Case sensitive

<variable list> is optional, and may just be “”. Quotes should be used if there are any spaces in the variable definition. The variables listed are passed to the program on run. Example is “Volume=100,Voice=\”Female\””. Note the quoting.

<condition> is a condition to match, as per IF. Normally its Condition Always True.

When an event triggers, the variables TriggerName and TriggerType are set to tell the program that runs what triggered it. TriggerLocalTime will have the local time in US format when the event ran, and TriggerUTCTime the UTC time. Other variables can be set up by the event to describe more information about the event.

The following events can occur in the core system.

## Start of Program

TriggerName = onStartup, TriggerType = ProgramEvent

Followed by:

TriggerName = onPostStartup, TriggerType = ProgramEvent

Use only onPostStartup if you need to do something after all other packs have installed using startup. This is only to be used if a dependency exists between two packs.

## Close down of Program

TriggerName = onShutdown, TriggerType = ProgramEvent

## Timer has timed out

TriggerName = onTimer, TriggerType = ActionProgram

TimerName = set to timer name which timed out

If a timer has been associated with an event, the Event variables will also be set up

## Say has started

TriggerName = onSayStarted, TriggerType = ActionProgram

EventName = set to name set by the StartEvent parameter in say

## Say has ended

TriggerName = onSayFinished, TriggerType = ActionProgram

EventName = set to name set by the FinishEvent parameter in say

## Play has started

TriggerName = onPlayStarted, TriggerType = ActionProgram

EventName = set to name set by the StartEvent parameter in play

## Play has ended

TriggerName = onPlayFinished, TriggerType = ActionProgram

EventName = set to name set by the FinishEvent parameter in play

## Non-Modal Dialog User Event

A non-modal dialog has a control event.

TriggerName = onNonModalDialog, TriggerType = UserUIEvent

Dialog = the dialog name

Control = name of the control.

# Elite Dangerous Discovery Specific

## Events

### Refresh

TriggerName = onRefreshStart, TriggerType = ProgramEvent: At the start of refresh

*Then*

For any events which have the variable RunAtRefresh=X (any value) set in their event line, these will be executed by the program at the refresh point.

Event variables set up as per New Journal event.

*Then*

TriggerName = onRefreshEnd, TriggerType = ProgramEvent: At the end of refresh

### Install

TriggerName = onInstall, TriggerType = ProgramEvent: At install

When a new action file is installed or removed.

* InstallList: Semicolon list of packs installed or updated
* UpdateList: Semicolon list of packs already installed but which have been replaced by a newer version [23]
* RemoveList: Semicolon list of packs removed (note the removed pack is not called) [23]

### New Journal event received

TriggerName = event name, TriggerType = NewEntry

The following variables will also be initialised:

* Section 3.2, Event History Variables
* Section 3.2, System Variables
* Section 3.2, Current Ship Variables

The prefix “Event” will be used for these sections.

### User Right Click on entry

When a entry is right click and Run actions on this entry is selected, in the journal or travel history.

TriggerName = event name, TriggerType = UserRightClick

Event variables set up as per New Journal event.

### Event Action command has executed an event.

When Event <jid> Action command is issued.

TriggerName = event name, TriggerType = ActionProgram

Event variables set up as per New Journal event.

### Popup has opened

TriggerName = onPopUp, TriggerType = UserUIEvent

PopOutName = pop out name and index (spanel1)

PopOutTitle = expanded pop out name (Journal History)

PopOutIndex = index, 1 to N.

### Popup has closed

TriggerName = onPopDown, TriggerType = UserUIEvent

PopOutName = pop out name and index (spanel1)

PopOutTitle = expanded pop out name (Journal History)

### Major tab has changed

TriggerName = onTabChange, TriggerType = UserUIEvent

TabName = new tab

### Add in menu item selected

TriggerName = onMenuItem, TriggerType = UserUIEvent

MenuName is set to the trigger name

MenuText is set to the menu text

TopLevelMenuName is set to the top level menu name (full name)

### Elite UI input

A UI input to Elite Dangerous program (only) has been detected. Only generated once elite input detection has been turned on via Perform command.

The first event raised is OnEliteInputRaw

TriggerName = onEliteInputRaw, TriggerType = EliteUIEvent

Device = Device name

EventName = Key\_<name> (using windows key names, not frontiers)

or Joy\_<butno>

or Joy\_POV<num><dir>

or Joy\_<axis>Axis

Pressed=1/0. Valid for Key\_ and POVs only, not for axis.

Value = For POV, -1 for centred, else direction in degrees \*100

For Joysticks, from 0 to 1000.

The raw event can be used to map keys/joystick buttons to actions in EDDiscovery, if they are unused by Elite.

The second type of event raised is produced if the UI event maps to a Frontier binding, defined by the currently selected control binding in Elite. You get one of these events for each Frontier Binding (there may be more than one binding per key/button/axis):

TriggerName = onEliteInput, TriggerType = EliteUIEvent

Binding = Frontier defined binding name for this function, such as “ToggleDriveAssist”

BindlingList = All bindings associated with this UI event

EventName, Device, Pressed, Value as above

When a key/button/pov associated with an Frontier binding is released, making the binding now invalid, the following is reported.

TriggerName = onEliteInputOff, TriggerType = EliteUIEvent

Binding = Frontier defined binding name for this function, such as “ToggleDriveAssist”

Axis cannot be released so they don’t generate the Off event.

### Elite Dangerous UI event notified

Elite Dangerous has generated a notification of a UI event. [8]

TriggerName = onEliteUIEvent, TriggerType = EliteUIEvent

EventClass\_EventTypeID = EventClass\_EventTypeStr = UI Event name (no UI prefix)

EventClass\_UIDisplayed = Boolean, 1/0, indicating if EDD is showing journal UI events in its journal log.

EventClass\_\* = any variables associated with the event. See the code base around UIEvents

Also issued after the above event is a trigger named after the particular event:

TriggerName = “UI” + UI Event Name, TriggerType = EliteUIEvent

EventClass\_\* variables as above

Either may be used to pick up a specific event. [8]

Ship Basic variables from section 3.2 are also presented to the event.

UI Event list:

GUIFocus, Music, Pips, Position, FireGroup

Docked,Landed ,LandingGear ,ShieldsUp ,Supercruise ,FlightAssist ,HardpointsDeployed ,

InWing ,Lights ,CargoScoopDeployed ,SilentRunning ,ScoopingFuel ,SrvHandbrake ,

SrvTurret ,SrvUnderShip ,SrvDriveAssist ,FsdMassLocked ,FsdCharging ,FsdCooldown ,

LowFuel ,OverHeating ,HasLatLong ,IsInDanger ,BeingInterdicted

HUDInAnalysisMode, NightVision, Fuel, Cargo

Mode [23], OverallStatus, Command

The following variables are written by the UIOverallStatus event. Other UI Events have their own variable set.

* EventClass\_Mode= one of:
  + None,
  + MainShipNormalSpace,
  + MainShipDockedStarPort,
  + MainShipDockedPlanet,
  + MainShipSupercruise,
  + MainShipLanded,
  + TaxiDocked,
  + TaxiNormalSpace,
  + TaxiSupercruise,
  + TaxiLanded,
  + MulticrewDockedStarPort,
  + MulticrewDockedPlanet,
  + MulticrewNormalSpace,
  + MulticrewSupercruise,
  + MulticrewLanded,
  + MulticrewSRV,
  + SRV,
  + Fighter,
  + OnFootStarPortHangar,
  + OnFootStarPortSocialSpace,
  + OnFootPlantaryPortHangar,
  + OnFootPlantaryPortSocialSpace,
  + OnFootInstallation,
  + OnFootPlanet
* EventClass\_MajorMode = one of
  + None, Ship, Taxi, Multicrew, SRV, Fighter, OnFoot
* EventClass\_FlagsCount= number of below flags, some examples are shown:
  + EventClass\_Flags[1]=Docked
  + EventClass\_Flags[2]=LandingGear
  + EventClass\_Flags[3]=FsdMassLocked
  + EventClass\_Flags[4]=ShieldsUp
* EventClass\_Focus=NoFocus GUI FOCUS
* EventClass\_Pips\_Valid=1
  + EventClass\_Pips\_Systems=1
  + EventClass\_Pips\_Engines=4
  + EventClass\_Pips\_Weapons=1
* EventClass\_Firegroup=1
* EventClass\_Fuel=32 tons on board
* EventClass\_Reserve=0.32 reserve on board
* EventClass\_Cargo=0
* EventClass\_Pos\_ValidPosition=0/1
* EventClass\_Pos\_ValidAltitude=0/1
  + EventClass\_Pos\_Latitude=-999999
  + EventClass\_Pos\_Longitude=-999999
  + EventClass\_Pos\_Altitude=-999999
  + EventClass\_Pos\_AltitudeFromAverageRadius=0
* EventClass\_ValidHeading=0
  + EventClass\_Heading=-999999
* EventClass\_ValidRadius=0
  + EventClass\_PlanetRadius=-999999
* EventClass\_LegalState=Clean
* EventClass\_BodyName= Name if has one
* EventClass\_Health=-1 not on foot, or 0-100
* EventClass\_LowHealth=0/1
* EventClass\_Gravity=-1 not on planet, or G
* EventClass\_Temperature=-1 not on foot, or temperature in K
* EventClass\_TemperatureState=Normal/VeryCold/Cold/Warm/VeryWarm
* EventClass\_Oxygen=-1 not on foot, or 0-100
* EventClass\_LowOxygen=0/1
* EventClass\_BreathableAtmosphere=0/1
* EventClass\_FSDState=Normal/Charging/Jumping/Gliding/Cooldown
* EventClass\_SelectedWeapon= set if weapon is held
* EventClass\_SelectedWeapon\_Localised= local name
* EventClass\_EventTimeUTC=5/20/2021 11:39:35 AM
* EventClass\_EventTypeID=OverallStatus
* EventClass\_EventTypeStr=OverallStatus
* EventClass\_EventRefresh=0/1 set to 1 when change of ship type

### EDDN/EDSM Sync Sent

EDD has sent a EDDN/EDSM sync event [6][16]

Note from 16 onEGOSync event has been removed as EGO is no longer supported.

Note from 17 onIGAUSync event has been removed as IGAU is no longer supported.

TriggerName = onEDDNSync/onEDSMSync, TriggerType = ProgramEvent

All of these set up EventCount to indicate number of events sent.

EDSM sets up EventStarList which contains the list of stars which are first discoveries to EDSM.

### Voice Recognition [7]

Voice recognition must be explicitly enabled, see Perform Voice Recognition.

To define a phrase to recognise, and to associate a program with a phase, use the following event:

EVENT onVoiceInput, <Program to run>, <Variables to set>, VoiceInput MatchSemicolonList|MatchesemiColonList <Voice Phrase>

<Program to run> = program name to run on recognising this phrase.

<Variables to set> = Variables to pass to program

<voice phrase> = Phrases to recognise.

The event condition must be in the form VoiceInput MatchesSemiColon string or VoiceInput MatchesSemiColonList string for the event to be recognised as a valid voice input event . See the conditions section for the format of MatchesSemiColon or MatchesSemiColonList.

For MatchesSemiColonList you can build complicated phrase recognition lists, such as:

* "Increase|More Engine Power;More Engines;Engines Up;Increase Engines"
  + Recognises: Increase Engine Power, More Engine Power, More Engines, Engines Up, Increase Engines
* "[Open] Friends Menu"
  + Recognises: Open Friends Menu, Friends Menu
* "Deploy|Fire|Launch|[] Heat Sink"
  + Recognise Deploy Heat Sink, Fire Heat Sink, Launch Heat Sink, or Heat Sink.

Once a phrase has been recognised, an event will be generated with the following parameters:

TriggerName = onVoiceInput, TriggerType = Voice

VoiceInput will contain the phrase recognised, fully expanded out (not in the form given in the event).

VoiceConfidence will be a float with the % confidence

If voice recognition fails, an event will be generated with the following parameters:

TriggerName = onVoiceInputFailed, TriggerType = Voice

VoiceInput will contain the phrase not recognised, or an empty string

VoiceConfidence will be a float with the % confidence

### DLL Event [9]

When a DLL issues an Event.

TriggerName = <from DLL>, TriggerType = DLLEvent

Parameters defined by DLL event.

### User Key Press Down while Program is focused

Only when the main program is focused and not in specific dialogs. Only generated for keys which are registered for listening to via a condition entry.

TriggerName = onKeyPress, TriggerType = KeyPress

KeyPress = key name, see Key for names.

The condition must be in the form KeyPress string equals or IsOneOf, and either a single keyname (string equals) or a comma separated list of keynames (IsOneOf) given:

i.e. EVENT onKeyPress, ProgramWindow, cmd=topmost, KeyPress $== Alt+F1

Alt+F1 will be added to the keys to generate events on.

### User Key Press Released while Program is focused[24]

Only when the main program is focused and not in specific dialogs. Only generated for keys which are registered for listening to via a condition entry.

TriggerName = onKeyReleased TriggerType = KeyPress

KeyPress = key name, see Key for names.

The condition must be in the form KeyPress string equals or IsOneOf, and either a single keyname (string equals) or a comma separated list of keynames (IsOneOf) given:

i.e. EVENT onKeyReleased, ProgramWindow, cmd=topmost, KeyPress $== Alt+F1

Alt+F1 will be added to the keys to generate events on when released.

### Surveyor discovered new bodies [30]

The surveyor panel has discovered new discoveries.

TriggerName = onSurveyor, TriggerType = PanelEvent

Body = full body name

BodyShortName = short body name (such as 1 a) or the full body name

EventName<N> = the event name, see later

Value<N> = any data associated with the event

N iterates from 1 upwards, giving all the events that are associated with the body.

Event names are:

* MiningSignals
* GeoSignals
* BioSignals
* ThargoidSignals
* GuardianSignals
* HumanSignals
* OtherSignals
* Terraformable
* Landable
* Volcanism
* Rings
* Belts
* Eccentric
* TinyPlanetRadius
* HugePlanetRadius
* Atmosphere: The Value field will have the atmosphere type in English
* Discovery: The Value field will have the name of the discovery
* HighGravity: The Value field will have the G of the planet (non invariant). Gravities over 2 generate this trigger.
* Earthlike
* WaterWorld
* AmmoniaWorld

## Variables

### EDD Specific Variables

Commander information:

* Commander - commander name
* RefreshCount - number of times refresh has been run on this commander. Reset to 1 when commander changes

### Event History Variables

These are written by a new journal event (with the prefix Event), or by a lookup of an event by the Event command (where EC\_ or a user selected prefix will be used).

* Prefix + UTCTime - UTC time of event, US format. [13]It
* Prefix + LocalTime - Local time of event, US format.
* Prefix + CurrentMode – journal derived mode. See also UIShipType. One of:
  + Unknown,
  + Docked,
  + NormalSpace,
  + Supercruise,
  + Landed,
  + TaxiDocked,
  + TaxiNormalSpace,
  + TaxiSupercruise,
  + DropShipDocked,
  + DropShipNormalSpace,
  + DropShipSupercruise,
  + MulticrewDocked,
  + MulticrewNormalSpace,
  + MulticrewSupercruise,
  + MulticrewLanded,
  + MulticrewSRV,
  + SRV,
  + Fighter,
  + OnFootStation,
  + OnFootPlanet,
* Prefix + DockedState - docked state, 1 or 0
* Prefix + LandedState - landed state, 1 or 0
* Prefix + WhereAmI - what station/body am I at. May be empty if not known.
* Prefix + BodyType [10] – type of location : Star, Planet, PlanetaryRing, StellarRing, Station, AsteroidCluster.
* Prefix + BodyName – body name, empty if none [24]
* Prefix + BodyID – body ID, -1 if none [24]
* Prefix + ShipType - ship type
* Prefix + ShipTypeFD - ship type FD name [13]
* Prefix + OnFoot- is on foot [24]
* Prefix + IsSRV- is it an SRV [24]
* Prefix + IsFighter- is it a Fighter [24]
* Prefix + BodyApproached – near a body (1/0) [24]
* Prefix + BookedTaxi – booked a taxi (1/0) [24]
* Prefix + BookedDropship – booked a drop ship (1/0) [24]
* Prefix + OnCrewWithCaptain – in multicrew, empty if not [24]
* Prefix + ShipId - ship ID number
* Prefix + IndexOf - index displayed on screen for this entry
* Prefix + JID - Journal unique ID for this entry (also Prefix+Id)
* Prefix + Credits – Weath at this entry
* Prefix + TravelledDistance – Travelled information in ly
* Prefix + TravelledSeconds – Travelled information
* Prefix + IsTravelling – is event in a travel section.
* Prefix + TravelledJumps - Jumps
* Prefix + TravelledMissingJumps – Missing data jumps
* Prefix + MultiPlayer – 1 or 0 if in multiplayer
* Prefix + ContainsRares – 1 or 0 if hauling rares
* Prefix + EventSummary – As per the travel history summary column.
* Prefix + EventDescription – Short description
* Prefix + EventDetailedInfo – Detailed info
* Prefix + Note – Any note against this entry. [10.4.3]
* Prefix + Class\_EventTypeStr and Prefix+ Class\_EventTypeID – name of event, compressed form.
* Prefix + Class\_EventTimeUTC - UTC time of event, US date format
* Prefix + Class\_EventTimeLocal - Local time of event, US date format
* Prefix + Class\_SyncedEDSM - has synced to EDSM, 0/1
* Prefix + Class\_EdsmID – system EDSM ID number
* Prefix + Class\_SyncedEDDN - has synced to EDDN, 0/1
* Prefix + Class\_StartMarker - Is it a start marker event associated with calculating distances
* Prefix + Class\_StopMarker - Is it a stop marker event
* Prefix + Class\_TLUId – internal ID of file containing event, not useful.
* Prefix + StartMarker – 1 if the start marker is present on this entry, 0 otherwise [13]
* Prefix + StopMarker – 1 if the stop marker is present on this entry, 0 otherwise [13]
* Prefix + EdsmSync – 1 if synced, 0 if not [13]
* Prefix + EddnSync – 1 if synced, 0 if not [13]
* Prefix + EgoSync – 1 if synced, 0 if not [13]
* Prefix + Beta – 1 if journal record is a beta record, 0 if not [13]
* Prefix + Horizons – 1 if journal record is a horizons record, 0 if not [23]
* Prefix + Odyssey – 1 if journal record is a odyssey record, 0 if not [23]
* Prefix + GameMode – mode (open etc) [13]
* Prefix + Group – group name if in private group, else blank [13]
* Prefix + Wanted – if known wanted [13]
* Prefix + MarketId– market ID if know, else 0 [13]
* Prefix + StationName – empty if none [24]
* Prefix + StationType – empty if none, Orbis etc [24]
* Prefix + StationFaction– when docked, else empty [20]
* Prefix + Class\_<name> - Event data from EDDiscovery internal representation of the JSON journal event fields. This data is decoded into more meaningful form. Embedded class variables and arrays use the ‘\_’ character to separate fields (EventClass\_Rings[0]\_InnerRad). The best way to see what is available is setting up an event, run it (right click and run event) and using pragma dumpvars E\* to see what is available. Or look at the c# journal event code.

### System Variables

These are written by a new journal event (with the prefix Event), Scan or Event command.

* Prefix + StarSystem - star system name
* Prefix + StarSystemEDSMID – EDSMID
* Prefix + xpos, ypos, zpos – Position of star in galaxy
* Prefix + EDDB has been removed [25]

### System Variables Further Information

These are written by the Event From jid INFO command or Scan command.

* Prefix + VisitCount – total number of visits to this system

[21] Note ScanCount and FSDJumpsTotal removed due to computational load

### Event Variables Further Information

[21] Note FSDJump withdrawn due to computational load

### Ship Basic Variables

These are written when an event is triggered (with the prefix Event) or when the Event or Ship commands are used.

* Prefix + Ship – ship information. Type of ship, such as Sidewinder
* Prefix + Ship\_ID – Ship ID, decimal number, frontier assigned.
* Prefix + Ship\_Name – User given name, or empty
* Prefix + Ship\_Ident – User given ident, or empty,
* Prefix + Ship\_SubVehicle – Current subvehicle, None, SRV, Fighter
* Prefix + Ship\_FullInfo – Long info list
* Prefix + Ship\_ShortName – Short name, either the ship name or type with the ship ID
* Prefix + Ship\_FuelCapacity – fuel capacity
* Prefix + Ship\_FuelLevel – Fuel level
* Prefix + Ship\_CargoCapacity – cargo capacity

If no ship information is available at event point (due to the age of the entry), Unknown, None or 0 will be used as the values.

### Ship Module Variables

These are written by the Event INFO or Ship commands.

* Prefix + Ship\_Module\_Count –Number of modules

For each module (changed for [8]):

* Prefix + Ship\_Module[N]Slot – This is the slot name
* Prefix + Ship\_Module[N]\_Item – This is the item name
* Prefix + Ship\_Module[N]\_ItemLocalised – This is item name localised (or failing that the item name)
* Prefix + Ship\_Module[N]\_Enabled – This is either blank (don’t know) or 0 = off, 1 = enabled
* Prefix + Ship\_Module[N]\_AmmoClip – This is either blank (don’t know) or clip size
* Prefix + Ship\_Module[N]\_AmmoHopper – This is either blank (don’t know) or hopper size
* Prefix + Ship\_Module[N]\_Blueprint – This is either blank (don’t know or no blueprint) or the blueprint
* Prefix + Ship\_Module[N]\_Health – This is either blank (don’t know) or the health value, 0 -100
* Prefix + Ship\_Module[N]\_Value – This is either blank (don’t know) or the value of the module in credits.

N is 0 to number of modules -1.

### Mission Variables

These are written by the Event MISSIONS command.

* Prefix + MissionCount –Number of missions

For each module:

* Prefix +Mission[N]\_Name, Prefix +Mission[N]\_NameLocalised [14], Prefix +Mission[N]\_ID, Prefix +Mission[N]\_UTC, Prefix +Mission[N]\_Local, Prefix +Mission[N]\_ExpiryUTC, Prefix +Mission[N]\_ExpiryLocal
* Prefix +Mission[N]\_System, Prefix +Mission[N]\_Station, Prefix +Mission[N]\_Faction, Prefix +Mission[N]\_DestSystem, Prefix +Mission[N]\_DestStation, Prefix +Mission[N]\_TargetFaction
* Prefix +Mission[N]\_Influence, Prefix +Mission[N]\_Reputation, Prefix +Mission[N]\_Commodity,
* Prefix +Mission[N]\_Target, Prefix +Mission[N]\_TargetLocalised [14], Prefix +Mission[N]\_TargetFriendly[23],
* Prefix +Mission[N]\_TargetType, Prefix +Mission[N]\_TargetTypeFriendly [23], Prefix +Mission[N]\_TargetTypeLocalised [23],
* Prefix +Mission[N]\_KillCount [23]
* Prefix +Mission[N]\_Passengers, Prefix +Mission[N]\_Completed
* Prefix +Mission[N]\_Reward, Prefix +Mission[N]\_Donation, Prefix +Mission[N]\_RewardCommodity, Prefix +Mission[N]\_RewardPermit, Prefix +Mission[N]\_FactionEffects [29],

N is 0 to number of mission -1.

## Functions

### %Body[13] (<name> ,<systemname>,<variable-rootname> [,<splitcapscontrol[29]>])

First look to see if <name> starts with <systemname>, if so remove <systemname> from <name>. Then performs the same as %Star(). This is useful just to report the body name ”A 2 A” instead of the complete name.

<Name> = variable or ”String”

<systemname> = variable or ”String”

<variable-rootname> = Literal, variable or ”String”. Root name of replacement variables.

<splitcapscontrol> = Literal, as per %Star

### %Events(<prefix>,<postfix>)

List all journal events, using <prefix> in the front and <postfix> at the end of each event.

<prefix> =<postfix>= variable or “String”

### %Ship(<name>)

Given a ship name, performs a replacement of MK <X> to Mark <X> to make the speech better. Then splits the resulting with the SplitCaps function.

<name>= variable or “String”

### %Star [6] (<name> ,<variable-rootname>,[,<splitcapscontrol][29]>)

First looks for the elite dangerous star pattern in <name>, and expands it out so it is spaced so the speech engine says each letter individually.

Then performs the same action as %rs or %rv, dependent if splitcapscontrol is present or set to SplitCaps.

<Name> = variable or ”String”

<variable-rootname> = Literal, variable or ”String”. Root name of replacement variables.

<splitcapscontrol> = Literal, variable or ”String”. Optional. Controls if %rs or %rv is used. %rs is used if its not present or set to “SplitCaps” (case insensitive). %rv is used if present and set to anything but set to “NoSplitCaps” or “NS”.

### %SystemPath(<name>)

Given the path of a system folder.

<Name> = literal, variable or “String”. Name is case insensitive and one of:

AdminTools, ApplicationData, CDBurning, CommonAdminTools, CommonApplicationData, CommonDesktopDirectory, CommonDocuments

CommonMusic, CommonOemLinks, CommonPictures, CommonProgramFiles, CommonProgramFilesX86, CommonPrograms, CommonStartMenu

CommonStartup, CommonTemplates, CommonVideos, Cookies, Desktop, DesktopDirectory, Favorites, Fonts, History, InternetCache

LocalApplicationData, LocalizedResources, MyComputer, MyDocuments, MyMusic, MyPictures, MyVideos, NetworkShortcuts, Personal

PrinterShortcuts, ProgramFiles, ProgramFilesX86, Programs, Recent, Resources, SendTo, StartMenu, Startup, System, SystemX86, Templates,

UserProfile, Windows

Also

EDDAppFolder

EDDActionsFolder

EDDVideoFolder

EDDSoundFolder

## Statements

Statement names are case insensitive.

### Bookmarks

Read, add and delete bookmarks [8].

Bookmarks [PREFIX <name>] <command>

<name> = variable prefix, if not given B\_ is the default. If <name> is missing after prefix an error will be produced.

<command> = Command to execute. Zero or more parameters can follow each command. The following commands are supported:

* List [<wildcard>]: List all bookmarks. Optionally a name or a wildcard can be used.
  + Prefix + TotalCount will hold the number of bookmarks in total.
  + Prefix + MatchCount will hold the number of bookmarks in total.
  + Prefix\_1 to Prefix\_N will hold the bookmarks found.
  + Each bookmark will write a list of variables as follows, prefixed by Prefix + N + “\_”:
    - isstar : 1 if star, 0 if region mark
    - name,x,y,z,time (US format), note
  + If any planetary marks are present on planets:
    - Prefix + N + “Planet\_Count “ will list the number of planets marked
    - For each Planet, prefixed by Prefix + N + “\_Planet\_” + P + “\_”
      * name
      * Count = number of locations
      * For each Location, prefixed by Prefix + N + “\_Planet\_” + P + “\_” + L + “\_”
        + name, comment, latitude, longitude
        + Example name would beB\_1\_Planet\_1\_2\_name.
* EXIST [“REGION”] <star/regionname> [13]
  + Prefix + Exists = 1 if it exists, 0 otherwise.
* ADD [“REGION”] <star/regionname> <x> <y> <z> [<notes>] [<tags>[29]]
  + Add or update a bookmark, of a star or region dependent on presence of “REGION”, at x/y/z. Note may be updated or not. If it’s a new bookmark and notes is not present, it will be blank.
* ADDSTAR <starname> [<notes>][<tags>[29]]
  + Add or update a star bookmark. The database must have the star listed or else an error occurs. Note may be updated or not. If it’s a new bookmark and notes is not present, it will be blank.
* DELETE [“REGION”] <star/regionname>
  + Delete a star or region bookmark. Error if the bookmark is not found.
* UPDATENOTE [“REGION”] <star/regionname> <note>
  + Update the note on either a star or region bookmark. Error if the bookmark is not found.
* UPDATETAG [“REGION”] <star/regionname> <tag;tag;>
  + Update the tag list on either a star or region bookmark. Error if the bookmark is not found.
* ADDPLANET <starname> <planet> <placename> <lat> <long> [<comment>] [<tags>[29]]
  + Add a planetary bookmark to an existing bookmark (which must exist).
* DELETEPLANET <starname> <planet> <placename>
  + Delete a planetary bookmark. Error if it does not exist
* UPDATEPLANETNOTE <name> <planet> <placename> <comment>
  + Update a planetary bookmark (which must exist).
* UPDATEPLANETTAG[29] <name> <planet> <placename> <tag;tag>
  + Update a planetary bookmark (which must exist).
* PLANETMARKEXISTS <name> <planet> <placename>
  + Return <Prefix>\_Exists = 1 if the planetary mark exists, 0 otherwise.

### Captains Log [13]

Read, add and delete Captains log entries for the current commander. [13]

CAPTAINSLOG [PREFIX <name>] [CMDR <cmdr>] <command>

<name> = variable prefix, if not given CL\_ is the default. If <name> is missing after prefix an error will be produced.

<cmdr> = commander name. Error will be produced if commander not found.

<command> = Command to execute. Zero or more parameters can follow each command. The following commands are supported:

* List [<wildcard>] [<field>]: List all log entries.
  + <wildcard> is the text to search for, normal wildcards can be used
  + If <field> is not present, it is matched against either the system or body name
  + If <field> is present, it can be set to one of the following:
    - System
    - Body
    - Note
    - Tag
  + Each commander entry will have a prefix set to: Prefix + “\_” + Index + “\_”. Index is 1,2,3 etc.
  + Prefix + TotalCount will hold the number of bookmarks in total.
  + Prefix + MatchCount will hold the number of bookmarks in total.

The following commands allow you to find an entry:

* FROM <ID>: Select ID.
* LAST: Select last entry.
* FIRST: Select first entry.
* TIME <time>: Select first entry on or after this UTC time if one exists. Quotes may be used if the time contains spaces.
* FROM/LAST/FIRST/TIME FORWARD: Select the next entry after the one found by one of the first types.
* FROM/LAST/FIRST/TIME BACKWARD: Select the previous entry after the one found by one of the first types.

If no entry exists from one of the searches above, Prefix + ID will be -1 and the other variables will not be reported. No error will be reported for searches.

The following allow you to alter the log:

* FROM/LAST/FIRST/TIME [FORWARD|BACKWARD] DELETE: delete the item identified. An error will be produced if an entry to delete is not found.
* FROM/LAST/FIRST/TIME [FORWARD|BACKWARD] NOTE <text>: Change the text of the note.
* FROM/LAST/FIRST/TIME [FORWARD|BACKWARD] SYSTEM <text>: Change the system of the note.
* FROM/LAST/FIRST/TIME [FORWARD|BACKWARD] BODY <text>: Change the body of the note.
* ADD <systemname> <bodyname> <TIMEUTC> [<note> [<taglist>]]: Add a new entry
* ADDHERE [<note> [<taglist>]]: Add a new entry using the current time and location.

The following allow manipulation of the tagging lists.

* TAGLIST: Report tags and icons. Prefix + Tags will have the tag list.
* SETTAGLIST <tags>: Set Tag list. Any unknown icons will cause an entry to be ignored. Use the same format as reported in TAGLIST.
* APPENDTAGLIST <tags>: Append to Tag list. Any unknown icons will cause an entry to be ignored. Use the same format as reported in TAGLIST.

Options output the following variable, per Captains Log:

* Prefix + Id: Internal index of CL entry
* Prefix + TimeUTC: Time
* Prefix + TimeLocal: Time local
* Prefix + SystemName: system name
* Prefix + BodyName: body name
* Prefix + Note: Note text
* CLPrefix + Tags: Tag list, semicolon separated, semicolon at end.

### Commanders [13]

List commanders and control commander.

COMMANDERS [PREFIX <name>] [CMDR <cmdr>] <command>

<name> = variable prefix, if not given CL\_ is the default. If <name> is missing after prefix an error will be produced.

<cmdr> = commander name. Error will be produced if commander not found.

<command> = Command to execute. Zero or more parameters can follow each command. The following commands are supported:

* List [<wildcard>]: List all commanders matching wildcard.
  + <wildcard> is the text to search for, normal wildcards can be used
  + Each entry will have a prefix set to: Prefix + “\_” + CLIndex + “\_”. CLIndex is 1,2,3 etc.
  + Prefix + TotalCount will hold the number of bookmarks in total.
  + Prefix + MatchCount will hold the number of bookmarks in total.

The following commands allow you:

* CMDR <cmdr> CHANGETO: Change to commander.

Options output the following variable, per Commander:

* Prefix + Id: Internal index of commander
* Prefix + Name
* Prefix + EDSMName
* Prefix + EGOName
* Prefix + INARAName
* Prefix + JournalDir
* Prefix + SyncToEDSM: 0/1
* Prefix + SyncFromEDSM: 0/1
* Prefix + SyncToEDDN: 0/1
* Prefix + SyncToEGO: 0/1
* Prefix + SyncToINARA: 0/1
* Prefix + SyncToIGAU: 0/1
* Prefix + Deleted: 0/1, 1 if deleted.

### Commodities

Find information about the commodities at a particular journal entry.

Commodities [PREFIX <name>] <jid>

<name> = variable prefix, if not given M\_ is the default. If <name> is missing after prefix an error will be produced.

<jid> = Journal identification number. JID must be present or an error will be produced.

Forms:

* Commodities <jid> : Return information about the commodities at this JID.

Return Values:

* Prefix + Count = Number of commodities returned. May be zero.
* Prefix + Name + IndexNumber =Name of item
* Prefix + Category + IndexNumber = Category of item
* Prefix + fdname + IndexNumber = Frontier name of item
* Prefix + type + IndexNumber = Type of item
* Prefix + shortname + IndexNumber = Short name of item

IndexNumber is 1 to Count.

### DLLCall

Call a function in a DLL [10] and pass it parameters. Receive a string back from it.

DLLCall <dllname>, <action> [ ‘,’ <parameter>]

<dllname> = name of dll, without extension, case insensitive, or “All” (case insensitive)

<action> = action string to pass to DLL indicating what to do

<parameter> = optional comma separated list of parameters, optionally quoted.

Return Values:

* DLLCalled = number of DLLs called
* DLL[n] = DLL name called (n=1 to DLLCalled)
* DLLResult[n] = DLL result string (n=1 to N)

The DLL can return an error in which case the variables are created still then an error is issued to Action. An error is also created if no DLLs are found to call

Also

DLLCall <dllname, <Pre-set command>

<Pre-set command> = “JournalEntry” <JournalID> : Pass journal record described by the Journal JID into the DLL.

See the EDDiscoveryAdditionalDLLs project folder on the EDDiscovery git pages for an example DLL.

### EliteBindings

Find out information about elite bindings to keys. Note that Perform EnableEliteInput must be called first.

EliteBindings [PREFIX <name>] <searchname> [<searchname>]…

<prefix> = variable prefix, if not given EB\_ is the default. . If <name> is missing after prefix an error will be produced.

<searchname> = either the name of a binding value, or a name of a key , or use wildcard \* to find all or a subset.

Output is a set of:

<prefix> Binding\_<N> entries where <N> is 1 to N, giving the device:key pairs bound to an assigned function

<prefix> <variable> = <Value> entries given binding values

<prefix> Text = a list of binding and values found

### Event

Get information about an event or perform an action on it.

#### Event Information

Event [PREFIX <name>] [FROM jid | THPOS] [‘FORWARD’ | ‘BACKWARD’ | ‘FIRST’ | ‘LAST’ [<event-names>] [‘WHERE’ <condition>] ]

<name> = variable prefix, if not given EC\_ is the default. If <name> is missing after prefix an error will be produced.

<event-names> = a single event name, i.e. FSDJump or a bracketed comma list of names, i.e. (FSDJump, LoadGame)

<condition> = a condition, as per the IF command, relating to event fields. Used to filter returned events out.

Forms:

* FROM jid: Report on this Journal identification number
* THPOS: Report on the currently selected travel history entry in the main travel history page
* FROM jid NEXT: Report on next event after JID (in time)
* FROM jid LAST: Report on previous event before JID (in time)
* THPOS NEXT | LAST: Report on next/previous entry from travel history position
* FIRST: Report on first event in history
* LAST: Report on last event in history
* [FROM jid] FORWARD [NOT][13] <event-names>
  + Report on first event name matching event-names (or not matching) found after this JID (or from start if JID is not present)
* [FROM jid] BACKWARD [NOT] <event-names>
  + Report on first event name matching event-names (or not matching) found before this JID (or from last entry if JID is not present)
* [FROM jid] FORWARD|BACKWARD Where <condition>
  + As per above, but report on first event name which match condition. Condition is a normal Action condition as per IF command.
* [FROM jid] FORWARD|BACKWARD [NOT] <event-names> Where <condition>
  + As per above, but report on first event name matching event-names and which match condition.

As this is a query, no error will be produced if a journal event is not found.

Return Values:

* Prefix + JID will be written with the JID found, or 0 to indicate not found.
* Prefix + Count will return the number of events found in the search. 0 if none is found, else number found. Only one event is reported on however.
* Section 3.2, Event History Variables are written if an event is found.
* Section 3.2, System Variables are written if an event is found.
* Section 3.2, Ship Basic Variables are written if an event is found, about the ship being used at that event.

#### Event Perform Action

Event [PREFIX name] [FROM jid | THPOS] [<command see below>]

Commands (case insensitive):

* ACTION: run any action events on JID
  + Prefix + Count will return the number of actions found to run
* EDSM: Open the EDSM web page on this entry
  + Prefix + URL will have the URL of the EDSM page, or empty if system is not found in EDSM.
* ROSS: Removed in [21]
* EDDB: Removed in [27]
* INFO: Return more information on the event, as per:
  + Section 3.2, System Variables Further Information
  + Section 3.2, Event Variables Further Information
  + Section 3.2, Ship Module Variables.
* MISSIONS: Return more information on the event, as per:
  + Section 3.2, Mission Information at that point in history.
* NOTE “note” : Set note on journal entry [12]
* SETSTARTMARKER: Set start marker, clear stop. You must call Perform Refresh to update the screen. [13]
* SETSTOPMARKER: Set stop marker, clear start. You must call Perform Refresh to update the screen. [13]
* CLEARSTARTSTOPMARKER: Clear both flags. You must call Perform Refresh to update the screen. [13]

If jid is not given or not found, an error will be produced.

### GMO [13]

Get information about Galactic Map Objects from EDSM. EDD downloads the GMO list from EDSM on startup. The GMO list contains community driven points of interest and regions.

GMO [PREFIX <name>] <command>

<name> = variable prefix, if not given G\_ is the default. If <name> is missing after prefix an error will be produced.

Command Name (case insensitive):

* LIST: [<wildcard>]: List all GMOs. Optionally a name or a wildcard can be used which is matched against the GMO name. Each GMO will have a prefix set to:
  + GMO Prefix = Prefix + “\_” + GMOIndex + “\_”. GMO index is 1,2,3 etc.
  + Prefix + TotalCount will hold the number of bookmarks in total.
  + Prefix + MatchCount will hold the number of bookmarks in total.
* EXISTS <name> : Check if <name> exists and output its information if it does.
  + Prefix + Exists = 1 if found, 0 if not
  + GMO Prefix = Prefix

Options output the following variable, per GMO

* GMOPrefix + Name: Name of GMO
* GMOPrefix + Type: Type of GMO as a textual representation from EDSM.
* GMOPrefix + Group: Type of GMO: Region, Markers, Routes, Quadrants as classified by EDD.
* GMOPrefix + Search: EDSM Search URL
* GMOPrefix + MapURL: EDSM Map URL
* GMOPrefix + Description: Long description text.

### Historytab

Change the panel type of one of the history configurable panels.

Historytab <panelname> ‘Toggle’ | ‘paneltype’

<panelname> = Bottom | Bottom-Right | Middle-Right | Top-Right

<paneltype> = Log | StarDistance | Materials | Commodities | Ledger | JournalHistory | TravelHistory | ScreenShot | Stats | Scan

Forms:

* Historytab Bottom-Right Toggle : Toggle thru the types of panels in this area
* Historytab Bottom JournalHistory : Set bottom to journal history

Errors will be produced if panel name or panel type is not recognised.

### Ledger

Find information about a ledger entry

Ledger [PREFIX <name>] [AtOrBefore] <jid>

<name> = variable prefix, if not given L\_ is the default. If <name> is missing after prefix an error will be produced.

<jid> = Journal identification number. JID must be present or an error will be produced.

Forms:

* Ledger <jid> : Return information about this entry tied to JID. If a ledger entry is not associated with this entry, an error will be produced
* Ledger AtOrBefore <jid> : Find the first ledger entry at or before this JID.

Return Values:

* Prefix + JID = JID of event returned
* Prefix + IndexOf = index of entry in history
* Prefix + UTCTime = time in US format
* Prefix + EntryType = entry type string
* Prefix + Notes = notes on entry
* Prefix + Value = value of entry in credits, may be blank
* Prefix + PPU = profit per unit, may be blank
* Prefix + Credits = credits at this entry

### Materials

Find information about the materials at a particular journal entry.

Materials [PREFIX <name>] <jid>

<name> = variable prefix, if not given M\_ is the default. If <name> is missing after prefix an error will be produced.

<jid> = Journal identification number. JID must be present or an error will be produced.

Forms:

* Materials <jid> : Return information about the commodities at this JID.

Return Values:

* Prefix + Count = Number of materials returned. May be 0.
* Prefix + Name + IndexNumber =Name of item
* Prefix + Category + IndexNumber = Category of item
* Prefix + fdname + IndexNumber = Frontier name of item
* Prefix + type + IndexNumber = Type of item
* Prefix + shortname + IndexNumber = Short name of item

IndexNumber is 1 to Count.

### MenuItem

Adds a menu item to one of ED Discovery menus, or discovery if a menu item has been installed. Menu items will be removed if the pack is disabled. Clicking on the menu runs an event which the program can pick up.

Add:

MenuItem <menuname> ‘,’ <menu> ‘,’ <menutext> [‘,’ <icon>]

<menuname> = logival name of menu trigger name. Passed to the event for menu recognition purposes.

<menu> = ‘add-ons’ | ‘help’ | ‘tools’ | ‘admin’ (case insensitive)

<menutext> optionally quoted menu text to present

<icon> = optional, one of the built bitmap resources as its name (scan, microphone..) or None

Interrogate:

MenuItem <menuname>

Return in variable MenuPresent=1/0 if this logical menu is installed.

### Perform

Perform discrete acts.

Perform <commandname>

<commandname> : command to perform, case insensitive.

General:

* 3dmap : Open 3dmap
* 2dmap : Open 2dmap
* Edsm : Perform EDSM sync
* Refresh : Do a refresh of the system

Audio:

* ConfigureVoice [<title>]: open the voice control menu allowing device selection.
  + This defined the persistent globals: SpeechVoice, SpeechVolume, SpeechRate, SpeechEffects via a menu interface. DialogResult[29] = 1 OK, 0 Cancel
* ConfigureVoice[29] <title>,<voicename>,<volume>,<rate>,<effects> : open the voice control menu with these parameters and no device selection.
  + If voicename=NOVOICENAME then no voice name option will be presented.
  + If rate=NORATE then no rate option will be presented.
  + Returns VoiceName, Volume, Rate, Effects and DialogResult=1 OK, 0 Cancel variables.
* ConfigureWave [<title>]: open the wave control menu allowing device selection.
  + This defines the persistent globals: PlayVolume, PlayEffects via a menu interface. DialogResult[29] = 1 OK, 0 Cancel
* ConfigureWave[29] <title>,<volume>,<effects> : open the wave control menu with no device selection.
  + Returns Volume, Effects and DialogResult=1 OK, 0 Cancel variables
* VoiceNames. Store into a variable “VoiceNames” a comma separated quoted list of voices available on this machine

Voice Recognition:

* EnableVoiceRecognition <culture>: Enable voice recognition with this culture (en-gb). This turns the VR engine on with no phrases loaded.
  + VoiceRecognitionEnabled = 1 or 0 dependent on if it worked
* DisableVoiceRecognition : Disable voice recognition
* VoiceRecognitionConfidenceLevel <confidence>: Set confidence level to accept a recognition attempt. Up to 0.99. Must be done after Enable
* VoiceRecognitionParameters <babble timeout>, <initial silence timeout>, <end timeout>, <end timeout ambiguous>: Reconfigured the VR engine with these times in ms. Default is 0,0,150,500. Must be done after Enable.
* BeginVoiceRecognition: Load all the voice phrases from events and begin recognition. Must be done after Enable. Note that you can repeatedly call this, each time, it will reload all active voice prompt to the voice recognition system.
* VoiceRecognitionEvent <state> [22] : control if voice recognition events are issues to the system. Values are 1 (on), 0 (off), Toggle, or Status. Status means don’t change just report state. After the command is executed, a variable is created with the new state:
  + VoiceRecognitionEvent = 1 if on, 0 if off after command.

Keys:

* LoadKeys [23]: Reload the key recognition system with the updated key events. Call if you have programmatically changed any key events.

Add Ons:

* ManageAddOns : Manage add on action files
* EditAddOns : Edit the add on action files.
* EditLastPack : Edit last pack which the user edited, or if none, no action.
* EditPack <name>: [7] Edit pack of this name. Error if pack does not exist.
* EditSpeechText : Speech configuration screen

UI Input:

* EnableEliteInput: turn on reading of input (joysticks, keyboards) into the Elite program
* EnableEliteInputnoaxis: turn on reading of input (joysticks, keyboards) into the Elite program, but don’t generate events on axis moves.
* DisableEliteInput: turn off reading of input (joysticks, keyboards) into the Elite program
* ListEliteInput: Store into a variable “EliteInput” the devices detected, store into “EliteInputCheck” the binding check result (all bindings and any errors will be listed), store into “EliteInputButtons”[22] a comma separated list of buttons/keys pressable.
* Bindings: Dump Elite bindings to the variable Bindings.
* BindingValues: Dump Elite bindings to the variable BindingValues

Internet:

* URL <urlname> : Open a web page at this urlname. Urlname must start with http: or https:
* Datadownload <gitfolder> <filewildcard> <download folder> [<optclean-boolean>] [8]
  + Download from the ED Discovery Data repository, in folder <gitfolder>, the files given in <filewildcard>, to the <download folder> which must exist. Optionally empty the folder first (if set to 1).
  + The variable Downloaded is set to 1 if it succeeds.
  + The variable DownloadedCount is set to the number of files downloaded [30]

Panels:

* PanelAction[27]: Ask a panel to perform a specific action
  + EditNotePrimary [27] : ask the primary history panel to edit the top entry note

Action System:

* ActionFile <actionpackname-noextension> : Dump info on action file [11]
  + Events\_Count will show the number of hooked events [23]
  + Events[n] will show all hooked events (UI or journal), example: WingInvite, GenericResponse, "", Condition AlwaysTrue
  + Events\_<name> will show the same information
  + Journal\_Count will show the number of journal events [23]
  + JEvents[n] will iterate through all journal events and indicate if they are hooked, or “None” if not.
  + JEvents\_<name> will indicate the condition or “None” if not set.
  + UI\_Count will show the number of UI Events [23]
  + UIEvents[n] will iterate through all UI events and indicate if they are hooked, or “None” if not.
  + UIEvents\_<name> will indicate the condition or “None” if not set.
  + Install\_Count will show the number of install variables [22]
  + Install[n] will iterate through all install variables, example : Location,Actions
  + FileVar\_Count will show the number of file variables [22]
  + FileVar[n] will iterate through all current static file variables, example: ActionPackName,VoicePack10
  + Enabled will be set to 1 = enabled, 0 not.
* GenerateEvent - Debug command [8]
  + “Json of event” – Generate and issue an event on this json event. GenerateEventName variable will be written with the name of the event executed.
  + “on<eventname>” [<varnames>].. – where <eventName> is the name of an internal event. Following the event name is:
    - <varnames> = list of zero or more variable names to pass into the event. Wildcard \* allows multiple variables with the same root name to be passed in. Names only, no definitions allowed.
  + “UI<eventname>” [<varnames>].. – name of an UI event with variables as above.
  + Both event types run now, pre-empting the executing script.
* ActionEvent - Debug command [30]
  + Format is : <tiggername> <triggertype> [<vars>]..
  + The event runs now, pre-empting the executing script.

### Popout

Control the popouts.

Popout [PREFIX <name>] STATUS | ( <panelname> STATUS ) | ( <paneltype> TOGGLE|ON ) | ( <panelname> <command-opts>)

<name> = variable prefix, if not given P\_ is the default. If <name> is missing after prefix an error will be produced.

<paneltype> = use these logical names

Log | JournalHistory | TravelHistory | StarList | Materials | Commodities | MicroResources | Ledger | Missions | Factions | Modules | Stats | SuitsWeapons | MarketData | ShipYards | Outfitting | Synthesis | Engineering | ShoppingList | MaterialTrader | Scan | EDSM | Spansh | Inara | ScanGrid | StarDistance | EstimatedValues | SearchFinder | Trilateration | map2d | map3d | localmap3d | Bookmarks | CaptainsLog | Combat | RouteFinder | Expedition | Exploration | SystemInfo | Spanel | Surveyor | Trippanel | NotePanel | RouteTracker | Compass | MissionOV | MiningOV | SettingsPanel | ScreenShot | TheGrid | TheSplitter

<panelname> = <paneltype> <instance count>. If instance count is missing, refers to the first instance of the panel type.

<instance count> = 1 to N. This indicates a particular instance of a panel (spanel1, spanel2 etc)

<command-opts> = See below

Forms:

* Popout Status : Return information on popouts
* Popout <panelname> Status : Return if panelname exists, if so, its status
* Popout <panetype> Toggle | On : Turn on/off or turn on a panel type (Spanel, Stats etc). This refers to the first instance of a particular type of window.
* Popout <panelname> <command-opt> : Execute a command on this instance of a panel

Popout Status returns the following return values:

* Prefix + Count = Number of popouts, 0 onwards
* Prefix + index = Popout name (in the form panelname + instance count, so spanel1 for instance)

Popout <panelname> Status returns the following return values:

* Prefix + Exists = 1 if panelname exists, 0 otherwise.

If panelname exists:

* Prefix + Transparent : Is transparent, 1 or 0
* Prefix + TopMost : Is top most, 1 or 0
* Prefix + DisplayTitle : Is display title on, 1 or 0
* Prefix + ShowInTaskBar : Is shown in taskbar, 1 or 0
* Prefix + WindowState : Normal, Minimized or Maximized
* Prefix + Top : Top pixel position
* Prefix + Left: Left pixel position
* Prefix + Width: Width of window
* Prefix + Height: Height of window

Popout <panelname> <command-opts> allows control of a particular panel. If panel name does not have an instance number, refers to the first instance (1).

* Toggle : Toggle on/off this panel type
* Off : Turn off this panel (if already off, no action).
* On : Turn on this panel (if already on, no action).
* Transparent : Set transparent
* Opaque : Set opaque
* Title : Turn on titles
* NoTitle : Turn off titles
* TopMost : Make topmost
* NormalZ : Disable topmost, normal Z order
* ShowinTaskBar : Show in task bar
* NotShowinTaskBar : Don’t show in task bar
* Minimize : Minimize it
* Normal : Make window normal, de max/min it
* Maximized : Make window maximized
* Location <x>,<y>,<w>,<h> : Set window position and size
* Position <x>,<y> : Set window position
* Size <w>,<h> : Set window size

If the command is not recognised, an error will be produced.

### Pragma

Control program behaviour and debug. These are in addition to standard pragma’s.

Pragma <commandname-opts>

Command Name-opts:

* Bindings: Dump elite bindings.
* BindingValues: Dump elite binding values

### ProgramWindow

Configure the main program window.

ProgramWindow <commandname>

Command Name (case insensitive):

* Tab <tabname> : Select tab <tabname>, case insensitive.
  + Tab names are the same as shown on screen, or the logical name of the panel (see Popout).
  + An error will be produced if the tab is not open.
* TabExists [27]: Does tab exists? Tab names are the same as shown on screen, or the logical name of the panel. Returns a variable “Exists” which is either 1 (exists) or 0.
* OpenTab <tabname> [24] : Select or open tab <tabname>, case insensitive.
  + If you use the logical name the tab will be selected, or opened
  + If you use the tab name, you can only select it. An error will be produced if it’s not open
* CloseTab <tabname> [24] : Close tab <tabname>, case insensitive.
  + Tab names are the same as shown on screen, or the logical name of the panel (see Popout).
  + An error will be produced if the tab is not open.
* TopMost : Make topmost
* NormalZ : Disable topmost, normal Z order
* ShowinTaskBar : Show in task bar
* NotShowinTaskBar : Don’t show in task bar
* Minimize : Minimize it
* Normal : Make window normal, de max/min it
* Maximized : Make window maximized
* Location <x>,<y>,<w>,<h> : Set window position and size
* Position <x>,<y> : Set window position
* Size <w>,<h> : Set window size

Errors will be produced if command name or other parts are not recognised.

### Scan

Find information about the star scan at a particular system.

Scan [PREFIX <name>] [‘EDSM’] [‘SPANSH’] “<systemname>” [“bodyname”]

<name> = variable prefix, if not given S\_ is the default. If <name> is missing after prefix an error will be produced.

‘EDSM’ = if present, check EDSM for star information as well (must be first)

‘SPANSH’ [28]= if present, check SPANSH for star information as well. If both present, check SPANSH then EDSM.

<systemname> = Name of system. Use quotes if there are spaces, which there normally is.

<bodyname>[29] = optional, name of body in system to report on. Can either be just its short name “B 1” or its full name.

When body name is not present:

**Primary Return Values**:

* Prefix + Stars = Number of stars in that system, or 0 if system not known/found.

For each star:

* Prefix + Star\_<star index> = Short name of star (A,B etc)
* Prefix + Star\_<star index>\_... = Information on that star, see below.
* Prefix + Star\_<star index>\_Planets = Number of sub objects in this star (Planets + belts etc)
* Prefix + Star\_<star index>\_Planets\_Only = Number of planets in this star (excluding bets etc)

For each planet of each star:

* Prefix + Planets\_<star index>\_<planet index> = Short name of planet
* Prefix + Planets\_<star index>\_<planet index>\_... = information on that planet, see below.
* Prefix + Planets\_<star index>\_<planet index>\_Moons = number of children of this planet. (Plus \_Only to exclude all but moons)

For each moon of each planet of each star:

* Prefix + Moon\_<star index>\_<planet index>\_.<moon\_index> = Short name of moon
* Prefix + Moon\_<star index>\_<planet index>\_.<moon\_index>.. = information on that moon, see below.
* Prefix + Moon\_<star index>\_<planet index>\_<moon\_index>\_Submoons = number of children of this moon (plus \_Only to exclude all but submoons).

For each sub moon of each moon of each planet of each star:

* Prefix + SubMoon\_<star index>\_<planet index>\_.<moon\_index>\_<submoon index>. = Short name of sub moon
* Prefix + SubMoon\_<star index>\_<planet index>\_.<moon\_index>\_<submoon index>.. = information on that sub moon, see below.

These variable indicate if the data came from EDSM or Spansh

* Prefix + EDSMLookup : set to 1 if EDSM lookup occurred
* Prefix + EDSMNoData : set to 1 if EDSM lookup returned no data
* Prefix + SpanshLookup[29] : set to 1 if Spansh lookup occurred
* Prefix + SpanshNoData[29]: set to 1 if Spansh lookup returned no data

Each Star, Planet, Moon, Sub moon the following information is given. The full prefix is the string given above:

* Full Prefix + \_type = ‘body’ | ‘star’ | ‘barycentre’ | ‘belt’
* Full Prefix + \_assignedname = name assigned by scan system, just the body name (A1)
* Full Prefix + \_assignedfullname = name assigned by scan system, full name including star name
* Full Prefix + \_data = 0 we don’t have any more data, 1 we do
* Full Prefix + \_signals = empty if none, else a list of signals, comma separ [15]
* Full Prefix + \_genuses = empty if none, else a list of genuses found on the planet, comma separ [15]

If we have a scan for this body:

* Full Prefix + \_isstar = 1 if star, 0 if not star
* Full Prefix + \_edsmbody = 1 if from EDSM
* Full Prefix + \_source = FromJournal, FromEDSM or FromSpansh
* Full Prefix + \_bodyname = Body name from scan
* Full Prefix + \_bodydesignation = Alternative Body name (from local EDD data) or bodyname
* Full Prefix + \_orbitalperiod = empty or orbital period in seconds
* Full Prefix + \_rotationalperiod = empty or rotational period in seconds
* Full Prefix + \_surfacetemperature = empty or temperature in kelvins
* Full Prefix + \_distls = distance from main star in ls, 0 indicates main star
* Full Prefix + \_text = text description of item
* Full Prefix + \_value = estimated value of body [7]

If it’s a star:

* Full Prefix + \_startype = Star type, K, A, H etc
* Full Prefix + \_startypeid = EDD classification of body type
* Full Prefix + \_startypetext = Star type in English text
* Full Prefix + \_stellarmass = blank or mass in Sols
* Full Prefix + \_age = blank or age in millions of years
* Full Prefix + \_mag = blank or absolute magnitude
* Full Prefix + \_habinner = blank or habitation inner border in ls
* Full Prefix + \_habouter = blank or habitation outer border in ls

If it’s a body:

* Full Prefix + \_class = Body class, in English
* Full Prefix + \_typeid = EDD classification of body type
* Full Prefix + \_landable = ‘Landable’ | ‘Not Landable’
* Full Prefix + \_atmosphere = blank or atmosphere type
* Full Prefix + \_terraformstate = blank or terraform state
* Full Prefix + \_volcanism = blank or volcanism
* Full Prefix + \_gravity = blank or gravity in m/s
* Full Prefix + \_pressure = blank or pressure in pascal
* Full Prefix + \_mass = blank or mass in earth masses.
* Full Prefix + \_materials\_<materialname> = percentage composition of materials on planet

When body name is present:

* Prefix + BodyFound = 0 no body found (below will not be present) 1 = found.
* Prefix + Body = Short name of body
* Prefix + Body\_... = information on that planet, see above
* Prefix + Body\_Moons = number of children of this planet. (Plus \_Only to exclude all but real moons)

### Ship

Find information about a ship

Star [PREFIX <name>] “<ship name>”

<name> = variable prefix, if not given SH\_ is the default. If <name> is missing after prefix an error will be produced.

<ship name> = Name of system. Use quotes if there are spaces, which there normally is. The list of known ships is searched for this string, using the ship Full Info string as the matching source. If the ship name is part of one of those, the ship information is returned. Prefix + “Found” indicates if the ship was found. If the string is empty, no ship is attempted to be found.

Return Values:

* If a ship name is given (Ship name is not an empty string):
  + Prefix + Found = 0 if ship is not found, 1 if found. Only written if a ship is asked for.
  + Section 3.2, Ship Variables are written if the ship is found
  + Section 3.2, Ship Module Variables are written if the ship is found
* Irrespective of if a ship is found or given, the following variables are also made:
  + Prefix + “Ships” is written with the total number of ships, SRVs, Fighters known about.
  + For each Ship known about, Prefix + Ships[N]\_ + <..> is written with the information found in Section 3.2, Ship Variables for that particular ship. N is 0 to the total number of ships-1. For instance SH\_Ships[2]\_Ship\_ID = 28.

### Star

Find information about a star.

Star [PREFIX <name>] [‘EDSM’] [‘SPANSH’] “<systemname>” [Options..]

<name> = variable prefix, if not given ST\_ is the default. If <name> is missing after prefix an error will be produced.

‘EDSM’ [21] = if present, check EDSM for star information as well

‘SPANSH’ [28]= if present, check SPANSH for star information as well. If both present, check SPANSH then EDSM.

<system name> = Name of system. Use quotes if there are spaces, which there normally is.

Options can be one of:

* NEAREST [13] [MinDist [MaxDist [MaxReport [Spherical|Cube]]]]: Report on nearest stars.
  + Default is maximum of 50 within 20 lys in a sphere.
  + Variables Prefix+index+”\_” is used, with the same variables in section 3.2 as below.
  + Note this can take a while and the script will be paused while this is computed.
  + Prefix + Count will hold the number of stars found

Return Values:

* Prefix + Found = 0 if not found, 1 if found.
* Section 3.2, System Variables are written if the star is found (EDDB information will be empty as we no longer store that information against stars 21)
* Section 3.2, System Variables Further Information are written if the star is found

### Target

Set target.

Target [PREFIX <name>] <command>

<name> = variable prefix, if not given T\_ is the default. If <name> is missing after prefix an error will be produced.

Command Name (case insensitive):

* BOOKMARK bookmark-system-name : Set target to system which already has a bookmark. If system is not bookmarked, an error will occur. Use BOOKMARKS to make or check if a bookmark exists first.
* GMO gmo-name: Set target to a galaxy object called gmo-name. If the gmo does not exist, an error will occur.
* SYSTEM system-name: Set target to a system. System needs to have co-ords and be in the DB, else an error will occur [25]
* Note: Removed in [25]
* CLEAR: Clear target
* GET: Report current target.
  + Prefix + TargetSet = 1 if set,0 if not. If not, none of the following are written.
  + Prefix+ TargetPositionName = name, or empty if not set. If empty, the following are not written.
  + Prefix + TargetPositionFullName = name with any internal prefixes on it
  + Prefix + TargetType = Bookmark, GMO or Notemark.
  + Prefix + TargetX
  + Prefix + TargetY
  + Prefix + TargetZ

### Timer

Set a timer ticking, and when it counts out, generate the event onTimer with the variable TimerName set to the timer name.

Timers are one shot, but you can just ask for the timer to run again using the same name. Multiple timers with the same <name> can be running at any one time.

If you want a unique timer, which is reset to the given interval if Timer is used again before it times out, precede the name with a plus symbol (note if JID is given, the original JID will be applied when the timer does tick out).

Timer [‘+’ ] <name>,<timeinms> [‘,’ <JID>]

<name> = unique timer name

<timeinms> = time in milliseconds before timer counts out

<JID> = event JID to associate with this timer. The event variables will be set to this JID. If the JID is entered but it is not found, an error will occur. You can obtain the current JID when responding to a journal event using %(EventJID)

To delete a timer if present (no error is produced if its not there): [28]

Timer -<name>