# Action Programs

Revision 12, for EDDiscovery 10.4

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

The current events that trigger them are:

* Key presses
* New Journal entries
* Program Events (Start up, Show down, Refresh, Pop ups/down etc)

Action programs are either written in the editor included in ED Discovery, 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).

# Core Language

## 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, carriage return, line feed and backslash respectively.

## Variables

Action does not have typed variables (double/int/string etc), all variables are strings.

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.

~~[8] Variable names can have [N] embedded inside them, such as Main[N]Value. [N] will be replaced by the expansion, if available, of N surrounded by \_. So if N=4, Main[N]Value results in the variable name Main\_4\_Value. This is used for a quick syntax to look up some variable arrays written by the program. If the expansion of N does not exist, the variable name would be Main\_N\_Value.~~ For next version.

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: File path of path
* 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 (optionally quoted, comma separated list) of values, case insensitive. ie. Var IsOneOf “a”,”b”,”c”.
* AnyOfAny: Is one of the left side strings in a list present on the right side string list, case insensitive. Ie. Var containing “one”,”Two”, then var AnyOfAny “Three”,”Four”,”one”
* 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.
* 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
* MatchCommaList [7]: see MatchSemicolonList but with ‘;’ separator changed to ‘,’

### 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 [<Action File Set Name>::] <program name> [ ‘(‘ <input parameter list> ‘)’]

<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 name> = Name of program

<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.

#### 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.

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.

#### 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. The program resumes executing with the <dialog name> variable set to the control that causes resumption, or “Escape” if the escape key is pressed.

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 to the control name.

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.

Dialog/NonModalDialog <dialog name>,<Caption>,<size>,<Variable prefix>

<dialog name> = name to use for this dialog.

<Caption> = Optionally quoted string for the caption

<size> = <w> ‘,’ <h> [ ‘,’ <x> ‘,’ <y>] = Mandatory width and height, with optional position.

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

Each control variable definition is in the form:

<Variable prefix><unique identifier> = <Control Name> ‘,’ <Control Type> ‘,’ <Control Text> ‘,’ <X> ‘,’ <Y> ‘,’ <W> ‘,’ <H> ‘,’ <ToolTip> [‘,’ <control parameters>]

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

<Control Name> = name of control.

<Control Type> = ‘Button’ | ‘Textbox’ | ‘CheckBox’ | ‘Label’ | ‘ComboBox’

[8]| “DateTime” | “Numberboxlong” | “Numberboxdouble” (case insensitive)

<Control Text> = text for control, value to select for ComboBox, or date time for DateTime

<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 for all but combobox.

For the control types shown below, extra control parameters may be present:

For TextBox:

< 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]

For CheckBox:

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

For DateTime:

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

For Numberboxlong or NumberBoxDouble:

<control parameter 1> = optional minimum value

<control parameter 2> = optional maximum value

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

For 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 options, as a comma separated list. No need for quotes for spaces.

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.

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

<dialog name> = name of dialog

<cmdname> = ‘continue’ | ‘get’ | ‘set’ | ‘close’ | ‘exists’ | ‘position’

Continue resumes processing of dialog actions and suspends the program until another action occurs.

For continue:

Modal dialogs only. Re-enter modal dialog and wait for next user input. No optional parameters

For close:

Close the dialog. No optional parameters.

For 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.

For get:

<optionalparameters> = <Control Name> (see Dialog). Return the value of the control in the variable DialogResult.

For set:

<optionalparameters> = <Control Name> ‘=’ <value> . Set the control value to this. Value can be a quoted string. Error if the control does not have a value that can be set. For ComboBox, the value must be present in the list (case sensitive), or an error will occur.

For position:

Return in variable “X” and “Y” the position of the dialog. No optional parameters.

Example (continuing on from Dialog):

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

Future:

Transparent control

On Top control

Images loaded from a EDD folder

Icon buttons boxes

Transparent control

Draw on transparent control

Back selection

#### 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 type, pick one of [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

Return folder selected in FolderName variable, or empty for 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 file selected in FileName variable, or empty for cancel.

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

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

Return file selected in FileName variable, or empty for cancel.

#### InfoBox

Presents a information box with an OK button.

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.

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

<caption> = optionally quoted caption of input box

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

<default list> = optional, semicolon separated optionally quoted list of default values. May be blank or less or more than prompt list.

<features> = optional feature list, only option is ‘Multiline’ (case insensitive)

Output will be:

InputBoxOK = 1 on OK, or 0 on cancel.

InputBox1 to InputBoxN = value returned

#### 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> [ ‘,’ <configurationlist>]

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

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

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

<item> = configuration items: see below

<value> = value of item.

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

***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 are numerous, see the effects dialog and its output for usage.

Default means use either the windows system default, or values defined by global variables WaveVolume and WaveEffect.

#### Say

Speak a phrase using the systems text to speech engine.

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

<phraselist> = See below for the format.

<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 100, 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 set sets the priority, Normal is the default, Low or High, or set by the value in the global SpeechPriority[7] if defined.

***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 1, 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 1, says don’t process string for grouping, just say it literally

***DontSpeak***, if set and 1, says don’t speak the phrase. The start/finish events occur but the audio is around 10ms long of blank audio. Prefix and postfix audio is played.

***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*** are numerous, see the effects dialog and its output for usage. If no speech effects are given on the line, an NoEffects is not present, then effects in the global SpeechEffects are applied.

***NoEffects***=<any>means don’t apply the global speech effects (defined in SpeechEffects global) to the speech

***MergeEffects****=<any>* means apply the global settings (if they exist), then apply any on the say command line.

The variable SaySaid will be written with the text actually spoken (including any SSML markup).

SpeechDebug is a special variable which if defined does the following:

‘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.

**Phrases Lists:**

The phrase defined by <phraselist> is interpreted as follows:

With no literal override set, the <phraselist> is treated as a group of phrases to pick randomly from.

The string consists of optional phase groups and phrase sets. A Phrase group is marked by using { } group markers and contains inside one phrase set.

Each phrase set is a set of one or more speech elements, each element separated by a semicolon, and Action will pick one element at random to say (see exception below to this general rule).

An example phrase set is “a;b;c” and Action will pick at random one of either a,b or c to say. If the phrase set starts with a semicolon “;a;b;c” then always pick a, and randomly pick between b and c (equivalent to “{a}b;c”).

An example using groups is “{a;b;c}{x;y;z}” or “{a;b;c}x;y;z” and Action will pick at random one of a,b,c and one of x,y,z.

With literal override set, the text is treated as is and is not parsed for phrases.

[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.

## 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> , <defaultvarname>)

If <varname> exists, use its text, else use the <defaultvarname> value. <defaultvarname> must exist if it’s a macro name.

<varname> = name of a variable

<defaultvarname> = variable or “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>])

Look up variables starting with <arrayrootname>[n]<postfix>, and expand the contents of them to a list, separated by <separ>, 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> = <length> = literal, variable or “string” containing the integer value.

<specialfunc> = literal, either not present or splitcaps (case ignored). Any others will be ignored. Splitcaps runs the split caps function on each entry.

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

#### %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.

#### %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”.

### 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 specifier to mean replace a negative ‘-‘ with ‘Minus ‘, 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>])

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> = if present, only “Try” is supported. This supresses the error if the expression does not evaluate and returns “NAN” instead.

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 comma 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.

### 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-variablename>)

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

<text> = variable or ”String”.

<root-variablename> = variable name root.

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

Join <text> together with <text-2> and any <add-text>, using the the delimiter <delimter> 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>)

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

<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;<patter>;<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”

#### %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.

#### %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.

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

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

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

#### %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.

#### %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.

#### %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> K2[, <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

The following events can occur. 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.

### 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

### User Key Press

Only when the main program is focused and not in a dialog. 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. This is a global variable and records the last user key press that activated this event.

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.

### 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*

TriggerName = event name, TriggerType = onRefresh: for each event which in the history list, in time order (oldest first). Only conditions with ‘run at refresh’ set to true will be considered for action. Event variables set up as per New Journal event.

*Then*

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

### 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

### 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

### ~~History Panel has changed type~~

~~TriggerName = onPanelChange, TriggerType = UserUIEvent~~

~~PanelName = panel name, either “Bottom”, “Bottom-Right”,”Middle-Right”,”Top-Right”~~

~~PanelTabName = new tab name (spanel)~~

~~PanelTabTitle = new tab title (Journal History)~~

### ~~User selected entry in History page~~

~~TriggerName = onHistorySelection, TriggerType = UserUIEvent~~

~~Event variables set up as per New Journal event.~~

### 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 has been detected. Only generated once elite input detection has been turned on via Perform command. Two types events are raised:

TriggerName = onEliteInputRaw, TriggerType = EliteUIEvent

Device = Device name

EventName = Key\_ 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.

Note the naming convention for keys is Frontiers, not the standard windows naming used in Keys.

If the UI event maps to a Frontier binding, defined by the currently selected control binding in Elite, you also get:

When the action occurs, the key/button/pov/axis which causes the binding to be true is reported:

TriggerName = onEliteInput, TriggerType = EliteUIEvent

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

EventName, Device, Pressed, Value as above

When a key/button/pov 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.

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

### 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]

UI Event list:

JournalMusic, 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,

GUIFocus, ShipType

### EDDN/EDSM/EGO Sync Sent

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

TriggerName = onEDDNSync/onEDSMSync/onEGOSync, TriggerType = ProgramEvent

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

EGO and EDSM sets up EventStarList to indicate if a submission of a star is associated with a EGO record or is a new EDSM star. One or more stars can be returned, semicolon separated.

### Voice Recognition [7]

To enable voice recognition, see Perform. To define a phrase to recognise, and to associate a program with a phase, use the following event:

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

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.

String is the phrase or phrases to listen for. See the conditions section for the format.

TriggerName = onVoiceInputFailed, TriggerType = Voice

VoiceInput will contain the phrase not recognised, or an empty string

VoiceConfidence will be a float with the % confidence

Any condition may be used in the Event entry.

### DLL Event [9]

When a DLL issues an Event.

TriggerName = <from DLL>, TriggerType = DLLEvent

Parameters defined by DLL event.

## 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, or by a lookup of an event by the Event command.

* Prefix + LocalTime - Local time of event, 24 hr clock
* 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 + ShipType - ship type
* 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 + Class\_<name> - Event data from EDDiscovery internal representation of the JSON journal event fields. This data is decoded into more meaningful form. 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, Scan or Event command.

* Prefix + StarSystem - star system name
* Prefix + StarSystemEDSMID – EDSMID
* Prefix + xpos, ypos, zpos – Position of star in galaxy
* Prefix + EDDB + info returns EDDB info on the star system with info set to: Government, Allegiance, State, Security, PrimaryEconomy, Faction, Population and NeedsPermit.

### 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
* Prefix + ScanCount – total number of scans in this system
* Prefix + FSDJumpsTotal – total number of jumps

### Event Variables Further Information

These are written by the Event From jid INFO command.

* Prefix + FSDJump = FSD jump count number at this event.

### Ship Basic Variables

These are written when an event is triggered 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]\_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]\_TargetType, Prefix +Mission[N]\_Passengers, Prefix +Mission[N]\_Completed
* Prefix +Mission[N]\_Reward, Prefix +Mission[N]\_Donation, Prefix +Mission[N]\_RewardCommodity, Prefix +Mission[N]\_RewardPermit

N is 0 to number of mission -1.

## Functions

### %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>)

Performs a split caps function, then tries to fix up the name if it’s a star name for phonetics to help the speech engine work.

<name>= variable or “String”

### %Star(<name> ,<variable-rootname>) [6]

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.

<Name> = variable or ”String”

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

### %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.
* ADD [“REGION”] <star/regionname> <x> <y> <z> [<notes>]
  + 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>]
  + 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.
* ADDPLANET <starname> <planet> <placename> <lat> <long> [<comment>]
  + 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).
* PLANETMARKEXISTS <name> <planet> <placename>
  + Return <Prefix>\_Exists = 1 if the planetary mark exists, 0 otherwise.

### 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’ [<eventnames>] [‘WHERE’ <condition>] ]

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

<eventnames> = 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:

* Event [PREFXIX name] FROM jid : Report on event by Journal identification number
* Event [PREFIX name] THPOS : Report on currently selected travel history entry
* Event [PREFIX name] FROM jid NEXT : Report on next event after JID (in time)
* Event [PREFIX name] FROM jid LAST : Report on previous event before JID (in time)
* Event [PREFIX name] THPOS NEXT | LAST : Report on next/previous entry from travel history position
* Event [PREFIX name] FIRST : Report on first event in history
* Event [PREFIX name] LAST : Report on last event in history
* Event [PREFIX name] [FROM jid] FORWARD <eventnames> : Report on first event name matching eventnames found after this JID (or from start if JID is not present)
* Event [PREFIX name] [FROM jid] BACKWARD <eventnames> : Report on first event name matching eventnames found before this JID (or from last entry if JID is not present)
* Event [PREFIX name] [FROM jid] FORWARD <eventnames> Where <condition>: Report on first event name matching eventnames and which match condition, found after this JID (or from first entry if JID is not present). Replace FORWARD with BACKWARD for a reverse search.
* Event FROM jid <command> : Run a command on event, see below for commands.

FROM or THPOS sets the journal id to search from, or to use. FROM allows a JID to be specified, THPOS means the current selected cursor position in the main travel history page.

Prefix+JID will be written with the JID found, or 0 to indicate an entry is not found (without error).

Return Values:

* Prefix + Count = count of events found in the search – 0 none found, else number found. Only one event is reported however, which is dependent on the command issued.
* 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 [‘ACTION’ | ‘EDSM’ | ‘ROSS’]

Forms:

* Event [PREFXIX name] FROM jid ACTION: run any action events on JID
  + Prefix + Count will return the number of actions found to run
* Event [PREFXIX name] FROM jid 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.
* Event [PREFXIX name] FROM jid ROSS : Open the ROSS web page on this entry
  + Prefix + URL will have the URL of the ROSS page, or empty if system is not known to the program in ROSS.
* Event [PREFXIX name] FROM jid EDDB : Open the EDDB web page on this entry [10.4.2]
  + Prefix + URL will have the URL of the EDDB page, or empty if system is not known to the program in EDDB.
* Event [PREFXIX name] FROM jid 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.
* Event [PREFXIX name] FROM jid MISSIONS : Return more information on the event, as per:
  + Section 3.2, Mission Information at that point in history.
* Event [PREFXIX name] FROM jid NOTE “note” : Set note on journal entry [10.4.2]

If jid is not given or not found, an error will be produced.

### 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>

Command Name (case insensitive):

* 3dmap : Open 3dmap
* 2dmap : Open 2dmap
* Edsm : Perform EDSM sync
* Refresh : Do a refresh of the system
* URL <urlname> : Open a web page at this urlname. Urlname must start with http: or https:
* ConfigureVoice [<title>]: open the voice control menu.
  + This defined the globals: SpeechVoice, SpeechVolume, SpeechRate, SpeechEffects via a menu interface
* ConfigureWave [<title>]: open the wave control menu
  + This defines the globals: PlayVolume, PlayEffects via a menu interface.
* 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
* 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 on 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 erorrs will be listed).
* VoiceNames. Store into a variable “VoiceNames” a comma separated quoted list of voices available on this machine
* EnableVoiceRecognition <culture>: Enable voice recognition with this culture (en-gb). This turns the VR engine on but does not stop the recognition
  + 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.
* 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: Must be done after Enable. Load all the voice prompts from events and begin recognition.
* Bindings: Dump Elite bindings to the variable Bindings.
* BindingValues: Dump Elite bindings to the variable BindingValues
* ActionFile <actionpackname-noextension> : Dump info on action file [11]
  + Events[n] will show all hooked events (UI or journal), example: WingInvite, GenericResponse, "", Condition AlwaysTrue
  + Events\_<name> will show the same information
  + 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.
  + 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[n] will iterate through all install variables, example : Location,Actions
  + FileVar[n] will iterate through all current static file variables, example: ActionPackName,VoicePack10
  + Enabled will be set to 1 = enabled, 0 not.
* 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.
* GenerateEvent - Debug command [8]
  + “Json of event” – Generate and issue an event on this json event. GenerateEventName will be written with the name of the event executed.
  + “onEventname” [<varnames>].. – 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.

### 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 | StarDistance | Materials | Commodities | Ledger | JournalHistory | TravelHistory | StarList | MarketData | Missions | Synthesis | Engineering | ShoppingList |Scan | EstimatedValues | Modules | Exploration | ScreenShot | Stats | SystemInfo | EDSMStarFinder | Expedition | Trilateration | Spanel | Trippanel | Notepanel | RouteTracker | SettingsPanel | TheGrid

<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.
* 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’] “<systemname>”

<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

<system name> = Name of system. Use quotes if there are spaces, which there normally is.

Return Values:

* Prefix + Stars = Number of stars in that system, or 0 if system not known/found.
* 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.
* Prefix + \_Planets\_<star index>\_<planet index>\_... = information on that planet, see below.
* Prefix + \_Planets\_<star index>\_<planet index>\_Moons = number of moons of this planet. (Plus \_Only to exclude all but moons)
* 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 sub moons of this moon (plus \_Only to exclude all but submoons).
* Prefix + \_SubMoon\_<star index>\_<planet index>\_.<moon\_index>\_<submoon index>.. = information on that sub moon, see below.

Each Star, Planet, Moon, Sub moon information consists of:

* 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

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, 0 if from your own scans
* Full Prefix + \_bodyname = Body name from scan
* 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 + \_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 + \_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.

Section 3.2, System Variables and Section 3.2, System Variables Further Info are also written about the system.

### 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>] “<systemname>”

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

<system name> = Name of system. Use quotes if there are spaces, which there normally is.

Return Values:

* Prefix + Found = 0 if not found, 1 if found.
* Section 3.2, System Variables are written if the star is found
* Section 3.2, System Variables Further Information are written if the star is found

### 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, preceed 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)