# Action Programs

Revision 3, for EDDiscovery 7.0

Action programs are small programs which run when an event occurs in ED Discovery, like a journal record being received, 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 written in the editor included in ED Discovery.

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).

# 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

## Close down of Program

TriggerName = onShutdown, TriggerType = ProgramEvent

## 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 = onNewEntry

The following variables will also be initialised:

* Section 4, Event History Variables
* Section 4, System Variables
* Section 4, Current Ship Variables

## User Key Press

TriggerName = onKeyPress, TriggerType = KeyPress

KeyPress = key name, see below for names

## User Right Click on entry

TriggerName = event name, TriggerType = UserRightClick

Event variables set up as per New Journal event.

## Event Action command has executed an event.

TriggerName = event name, TriggerType = ActionProgram

Event variables set up as per New Journal event.

## 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

## 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”

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)

# Key Presses

To define a key action, the trigger event needs to be onKeyPress, 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.

Keynames are:

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

\* A-Z, 0-9

\* Numpad 0-9

\* Oem1 (';' on UK keyboards)

\* Oem5 ('\' on UK keyboards)

\* Oem7 ('#' on UK keyboards)

\* Oem8 ('`' on UK keyboards)

\* tilde, comma, period, question, Minus, Plus, Home, End, Insert, Delete, Next (Pagedown) PageUp, Back, Down, Up, Left, Right, F1-F12, Decimal, Return, Enter, Add, Subtract, Multiply, Divide

# Variables

Parameters given to commands can contain variables. These variables are referenced using the following syntax: %(varname). Variable names are case sensitive.

So a Print command to print the commanders name would be : Print %(Commander) welcome!

Variables are defined by different parts of the program:

* Persistent Globals: Defined by the user in the Global dialog box, or by using the PersistentGlobal command.
  + They are persistent between invocations of ED Discovery.
* Program defined Globals: Defined by the Global command.
  + These are persistent between programs, but not between invocations of ED Discovery. Normally set up by a onStartup program.
* Program information. Set by ED Discovery automatically
  + Commander - commander name
  + RefreshCount - number of times refresh has been run on this commander. Reset to 1 when commander changes
* Parameters in the event which causes the program to run
  + Any variables defined in the parameter field (just pass the P button) 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 journal event). See above
* Local program variables, set by Let or Set commands

These variables are available for all commands to use.

## Event History Variables

These are written by an new journal event, or by a lookup of an event by the Event command. The prefix will be either ‘Event’ if it’s a new journal event, or the prefix selected in 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 - if docked, what station, empty otherwise
* 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 + 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, and running it (right click and run event) and using pragma dumpvars E\* to see what is available.

## System Variables

These are written by an new journal event, or by a lookup of an event by the Event command. The prefix will be either ‘Event’ if it’s a new journal event, or the prefix selected in the Event command.

* Prefix + StarSystem - star system name
* Prefix + StarSystemEDSMID – EDSMID
* Prefix + xpos, ypos, zpos – Position of star in galaxy
* Prefix + Event 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.

* 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 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\_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 or Ship commands.

* Prefix + Ship\_Module\_Count –Number of modules

For each module:

* 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.

# 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 can take variable names, literal strings (“text”) which are expanded themselves for embedded variables/functions, or literal data (LongDate in %date).

The following are defined:

## %Abs(varname,outputformat)

Given the double value of the variable, absolute it to positive. Print using outputformat, which is a c# tostring format (0.#). Additional formatting options for round, floor: ‘M’ at the start means replace a negative ‘-‘ with ‘Minus ‘, for speech use.

## %CloseFile(handle)

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

## %Date(varname | “string”,expansiontype)

Taking a date, in US form in varname, expand to one of these forms indicated by expansion type:

LongTime

ShortTime

LongDate

ShortDate

LongDateTime = (Long Date + Long Time)

DateTime =ShortDate+LongTime.

UTC = To UTC, printed as yyyy/dd/mm hh:mm:ss

Local = printed as yyyy/dd/mm hh:mm:ss

## %DateTimeNow(expansiontype)

Print the current time toformat as per %Date.

## %Datehour(varname | “string”)

Taking a date, in US form in varname, return the hour (0-23)

## %DirExist(varname | “path” [ varname | “path”]..)

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

## %EscapeChars(varname | “string”)

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

## %Eval(varname | “evaluation string” [‘,’ ‘Try’])

Evaluates either the variable or the string and returns the evaluated number. As per Let. If its not a number, it throws an error, unless you use ‘Try’ as the second parameter in which case it returns “NAN” for a bad evaluation and does not cause an error.

## %Exists(varname[‘,’ varname]..)

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

## %ExistsDefault(varname , defaultvarname | “string”)

If varname exists, use its text, else use the defaultvarname (which must exist) or a string.

## %Expand(varname | “string” [‘,’ varname | “string”]..)

Given a variable or a string, 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.

## %ExpandArray(arrayrootname | “string”, separvarname | “string” , startinteger | var, lengthinteger | var, [ specialfunc ])

Given array variables in format rootname[index], expand them to a list separated by separ, from start to a maximum of length. If less are present, no error occurs.

If specialfunc is “splitcaps” then the split caps function is applied to each entry. This is the only value currently recognised. All others will be ignored.

## %Expandvars(rootname | ”string”, separvarname | “string” , startinteger | var, lengthinteger | var, [ specialfunc ])

Given a variable root name(either literal or in a string), find all entries beginning with this text, and join them in a list separated by separ. Each entry will be in the form <rest of string> = value.

Start and length determine the number of entries. 1 is the first one.

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

If specialfunc contains “valueonly”, only the name part will be printed

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

Any other special function characters are ignored.

## %FileExists(varname | “filepath” [‘,’ varname | “filepath”]..)

Does all files given by variables, or file paths if in quotes, exist?

## %FileLength(varname | “filepath”)

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

## %Findline(varname,stringmatchvar | “string” )

Given a variable varname, which has lines in it, try and find the string in the line given by the parameter 2, case insensitive, and return the whole line which it is in. Empty string if not found

## %Floor(varname,outputformat)

Given the double value of the variable, truncate to the integer using floor. See %Abs for outputformat.

## %If ..

Many forms of this are supported:

* %IfEmpty,%IfNotEmpty (varname1 | “string”, varname2 | “string” , [, varname3 | ”string” ])
* %IfTrue,%IfFalse, (v1,v2,v3)
* %IfZero, %IfNonZero (v1,v2,v3)
* %IfContains, %IfNotContains (v1,v2,v3, [v4 [,v5]] )
* %IfEqual, %IfNotEqual (v1,v2,v3, [v4 [,v5]] )
* %IfGT, %IfLT (v1,v2,v3, [v4 [,v5]] )
* %IfGE, %IfLE (v1,v2,v3, [v4 [,v5]] )
* %IfEQ, %IfNE (v1,v2,v3, [v4 [,v5]] )

Empty/NotEmpty: If v1 is in this state, expand v2, else expand v3 if present.

True/False: If numeric value v1 is in this state, expand v2, else expand v3 if present. V1 must be an integer.

Zero/NonZero: If numeric value v1 is in this state, expand v2, else expand v3 if present. V1 must be a number.

Contains/NotContains/Equal/NotEquals: If v1 is empty and v5 is present, expand v5. Else if condition between v1 and v2 matches using a string case insensitive comparison1, expand v3, else expand v4 if present.

Nums: If v1 is empty and v5 is present, expand v5. Else v1 and v2 must be a number, and if condition between v1 and v2 matches, expand v3, else expand v4 if present.

## %Indexof(varname | “string” ,varname | “string”)

Given the text of the variable, find where the text of the second variable (case sensitive) is present. -1 if not found, zero based.

## %Indirect(varname | “string” [‘,’ varname | “string”]..)

Given a varname, or given a string which expands to a var name, 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.

## %Join(delimvar | “string” ’,’ varname | ”string” ’,’ varname | ”string” [’,’ varname | ”string”]..)

Join together with the delimiter variables. Minimum 3 values.

## %Length(varname | “string”)

Given a variable or string, how many characters are in its value.

## %Lower(varname | “string” [ ‘,’ delimvar | “string” , varname | ”string” [’,’ varname | ”string”]..)

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

## %MkDir(varname | “string”)

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

## %OpenFile(handle,varname | “string”,mode)

Open a file, assign to variable handle. Mode can be either a literal, or a variable. Valid values are:

* 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.

Returns 1 if open, 0 if not. Handle contains either an internal ID, or the reason for failure.

## %Phrase(varname | “string”)

Using the same algorithm as Say, pick a phrase from the varname or string.

## %Random(varname | integer-constant)

Give a random number between 0 and the value minus -1, so %Random(10) gives 0-9.

## %ReadLine(handle,varname)

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

## %Replace(varname | “string” , varsearch | “string”, varreplace | “string”)

Replace any case insensitive varsearch strings in varname with varreplace.

## %ReplaceEscapeChars(varname | “string”)

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

## %ReplaceVar(varname | “string” , rootnamevar | literal | “string”) or %rv(..) or %rs

The rootnamevar defines the prefix of a set of variables to search for substitutions. Using either rootnamevar, or treating it literal or as a string, all variables prefixed with rootname will be searched for substitution patterns.

The substitution pattern is a semicolon separated list.

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.

## %Round(varname,digits,outputformat)

Given the double value of the variable, round to the number of digits (0 to N or variable). See %Abs for outputformat.

## %Roundnz(varname,digits,outputformat,extradigits)

Given the double value of the variable, round to the number of digits (0 to N or variable). If this results in a zero, add on extra digits (1..N or variable) and add the same number of ‘#’’s onto the outputformat, and retry. Outputformat must be in the 0.0# format as # will be added on afterwards if it needs more precision.

## %Roundscale(varname,digits,outputformat,extradigits,scale)

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

## Seek(handle,varname | position)

Seek the position of file to this position, which is a variable or a literal position.

## %Splitcaps(varname | “string”) (or %sc..)

Given the text of the variable, expand out Camel case form (HelloFred = Hello Fred) and remove any underscores.

## %Ship(varname | “string” )

Performs a split caps, then tries to fix up the name for phonetics to help the speech engine work.

## %Substring(varname | “string”,index|var,length|var)

Given the text of the variable, cut it at index for length characters. If index is out of range, return empty string. If length is too long, return as many as possible. Index and length can either be a integer directly or a variable name.

## Tell(handle)

Tell the position of this file.

## %Trim(varname | “string”)

Given a variable, return its value with whitespace at start and end trimmed.

## %Upper(varname | “string” [ ‘,’ delimvar | “string” , varname | ”string” [’,’ varname | ”string”]..)

As per lower, but upper case.

## %Version(version part)

Return version part of the ED Discovery four digit version number. 1 is the highest part, 4 is the lowest.

## %WordOf(varname | “String”, varcount | number, [varsepar | “string”])

Given a list, separated by a single character from varsepar or string (default is ‘;’), give the varcount word. If varcount is too high, it returns the last, if too low, it returns the first.

## %Write(handle,varname | “string”)

Write the varname or string into file. 1 if succeeds, 0 otherwise.

## %WriteLine(handle,varname | “string”)

Write the varname or string into file with a line feed. 1 if succeeds, 0 otherwise.

# Conditions

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

For conditions used in programs, 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 expanded as well to resolve any variable or functions in it.

Condition types are:

* String comparisons:
  + Compare string against another string : Contains, Not Contains, Contains (Case Sensitive), Not Contains (Case Sensitive), Equals (==), Not Equals (!=), Equals(Case Sensitive), Not Equals (Case Sensitive)
* Variable Presence:
  + IsPresent, IsNotPresent (is variable on the left present/not present)
* Empty/Not Empty:
  + Is Empty/Is Not Empty (does variable have contents or not)
* Boolean:
  + Is True/Is False (is the variable non zero, ie. True or zero, False)
* Numeric comparators
  + Treat both sides a doubles, and compare : Equals (==), Not Equals (!=), Less (<), Greater (>), Less Equals (<=), Greater Equals (>=)
* Date
  + Treat both sides as a US Date, and compare, either >= Date, or < Date.
* Is One Of
  + Is the left side present in a string list (optionally quoted, comma separated list) of values, case insensitive.
* Always True
  + Condition is always true (useful for action event checking mostly).

Conditions can be Or-ed together, And, Nor, Nand. Conditions can be placed in groups, and groups can then be checked against one another.

# Strings

To include a quote in strings, use \”. So “Hello”there” would be written “Hello\”there”.

For certain commands, \\r, \\n and \\ are replaced by their non escaped equivalents, \r,\n and \\. For example Print, user dialogs.

# Commands

Statements are case insensitive. The following pages list these commands.

## Break

In a Loop, Do..While 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.

## 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.

## DeleteVariable

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

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

No error is produced if variable does not exist.

## Dialog

Show the user a program defined dialog, wait for user selection

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

<dialog name> = name to use for this dialog. When an event occurs on this dialog, the program will resume and a variable of this name will hold the dialog control name which caused the event.

<Caption> = Optionally quoted string for the caption

<size> = <w> ‘,’ <h> Size

<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> [<optional control parameters>]

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

<Control Name> = name of control.

<Control Type> = ‘Button’ | ‘Textbox’ | ‘CheckBox’ (case insensitive)

<Control Text> = text for control

<X> <Y> = position

<W> <H> = size

<ToolTip> = tooltip to display

For TextBox:

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

For CheckBox:

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

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"

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’

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

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.

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=%(Value)

DialogControl D1,Get CB1

Print checkbox=%(Value)

DialogControl D1, Close

## 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.

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

Errors will be produced if the parameters are ill formed.

## 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. If the JID does not exist, or the travel history position is not set, the command does not execute, but returns without error and Prefix+JID = 0 to indicate that the JID is not found.

Return Values:

* Prefix + Count = count of events found in the search – 0, or more.
* Section 4, Event History Variables are written if an event is found.
* Section 4, System Variables are written if an event is found.
* Section 4, Ship Variables are written if an event is found, about the ship being used at that event.
* System 4, Ship Module Variables are written if an event is found.

**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 EDDB.
* Event [PREFXIX name] FROM jid INFO : Return more information on the event. See Section 4, System Variables Further Information and Section 4, Event Variables Further Information.

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

## 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”)

## 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.

## Global

Set a non-persistent global variable or multiple variables to a string value.

Global [<varlist>]

<varlist> = <var def> [‘,’ <var def>]…

<var def> = varname ‘=’|’+=’|’$=’|’$+=’ <value>

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

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

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

## Historytab

Change the panel type of one of the history configurable panels.

Historytab <panelname> ‘Toggle’ | ‘paneltype’

<panelname> = Bottom | Bottom-Right | Middle-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.

## If

Change program flow on condition.

If <condition>

.. statements - Indented to show relationship

ElseIf <condition> - Optional

.. statements

ElseIf <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.

## InputBox

Presents the user with a input box.

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

<caption> = optionally quoted caption of input box

<prompt list> = semicolon separated optionally quoted list of prompt. 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

## 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

## Let

Set a variable or multiple variables to a numeric expression

SET [<varlist>]

<varlist> = <var def> [‘,’ <var def>]…

<var def> = varname ‘=’|’$=’ <numeric expression>

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

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

## Loop

Perform a loop a fixed number of times.

Loop <count>

.. statements - Indented to show relationship

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

Errors will be produced if incorrectly indentation is found.

A local variable, Loop<indentlevel> will be set each loop, counting from 1 upwards. So if the loop is at level 2, Loop2 will be set each time.

## 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.

## MessageBox

Presents a message box with configurable buttons and icons.

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

<message> = message string, quoted if required

<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’

## MenuItem

Adds a menu item to one of ED Discovery menus. Menu items will be removed if the pack is disabled. Clicking on the menu runs a program from the pack the menu item is declared in.

MenuItem <menuname> ‘,’ <menu> ‘,’ <menutext> [‘,’ <icon>]

<menuname> = name of menu trigger name, see the event list. Normally used to recognise the menu

<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

## 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.

## 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
* ConfigureWave [<title>]: open the wave control menu
* EditSpeechText : Speech configuration screen
* ManageAddOns : Manage add on action files
* EditAddOns : Edit the add ons

## PersistentGlobal

Set a persistent global variable or multiple variables to a string value.

PersistentGlobal [<varlist>]

As per Global.

## 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> = Log | StarDistance | Materials | Commodities | Ledger | JournalHistory | TravelHistory | ScreenShot | Stats | Scan | Spanel | Trippanel | Notepanel | RouteTracker | Exploration

<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.

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 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.

## 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.

## 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.

## Rem

Make a program remark

REM <string>

<string> remark.

No action on this command.

## Return

Return from a subroutine

Return [<string>]

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

## Say

Speak a phrase. The phrase is picked from <phraselist> or say the whole <phraselist> if Literal is defined.

With no literal override set, a phrase list is a grouping of phrases to pick randomly from. Phrases are separated by semi colons and/or { } group markers . Group markers allow you to group a set of phrases randomly picked together. Examples are: “a;b;c” pick one of a,b or c. “{a;b;c}{x;y;z}” or “{a;b;c}x;y;z” pick one of a,b,c and one of x,y,z. If the phrase group starts with a semicolon “;a;b;c” then always pick a, and randomly pick between b and c. You can also do this with “{a}b;c”.

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

<phraselist> = if not literal set : <phrase group> | ‘{‘ <phrase group> ‘}’ [‘{‘ <phrase group> ‘}’]..

<phrase group> = semi colon separated list of phrases (a;b;c)

<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.

***Voice*** is the voice to use, either Default, or a installed voice name

***Rate*** is the speed to speak, Default or -10 to 10. 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 onSayStarted with EventName variable set to this contents. String must be not empty to generate this.

***FinishEvent*** if set, generate an event onSayFinished 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

***Culture*** if set, sets the speech culture hint to windows (such as en-gb).

***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.

***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 SpeechVoice, SpeechVolume, SpeechEffects, SpeechCulture and SpeechRate.

The variable SaySaid will be written with the text actually spoken.

SpeechDebug is a special variable which if defined and contains ‘Print’ redirects speech output to the log window only and mutes the voice as per DontSpeak.

## 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 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.
* 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.
* 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’
* 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

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.

## 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 4, Ship Variables are written if the ship is found
  + Section 4, 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 4, 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 4, System Variables are written if the star is found
* Section 4, System Variables Further Information are written if the star is found

## Set

Set a variable or multiple variables to a string

SET [<varlist>]

<varlist> = <var def> [‘,’ <var def>]…

<var def> = varname ‘=’|’+=’|’$=’|’$+=’ <value>

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

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

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

## Sleep

Pause the program and sleep for a duration in milliseconds

Sleep [<time>]

<time> = in ms to sleep for.

## 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.

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)

## 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.