

# EZURiO

# EZURiO

## UWScript Core Language 2.0.0.38

The information contained in this document is subject to change without notice. EZURiO makes no warranty of any kind with regard to this material including, but not limited to, the implied warranties of merchant ability and fitness for a particular purpose. EZURiO shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

© Copyright 2006 EZURiO Limited.

All rights reserved.

This document contains information that is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of EZURiO.

Other product or company names used in this publication are for identification purposes only and may be trademarks of their respective owners.

## Contents

<b>1.</b>	<b>INTRODUCTION</b> .....	<b>5</b>
<b>1.1</b>	<b>Types of Script</b> .....	<b>5</b>
<b>1.2</b>	<b>Script Storage</b> .....	<b>5</b>
<b>1.3</b>	<b>Language Basics</b> .....	<b>6</b>
1.3.1	Variables .....	6
1.3.5	Event Handling.....	6
<b>1.4</b>	<b>Language Constructs</b> .....	<b>7</b>
<b>2.0</b>	<b>SYNTAX</b> .....	<b>8</b>
<b>2.1</b>	<b>Standard Abbreviations</b> .....	<b>9</b>
<b>2.2</b>	<b>Formats</b> .....	<b>9</b>
<b>2.3</b>	<b>Command Responses</b> .....	<b>9</b>
<b>2.3.1</b>	<b>Implementation Status</b> .....	<b>9</b>
<b>3.</b>	<b>COMPILER AND MODULE INSTRUCTIONS</b> .....	<b>10</b>
<b>3.1</b>	<b>AT</b> .....	<b>10</b>
<b>3.2</b>	<b>AT I</b> .....	<b>10</b>
<b>3.3</b>	<b>AT+CMP</b> .....	<b>10</b>
<b>3.4</b>	<b>AT+DEL</b> .....	<b>10</b>
<b>3.5</b>	<b>AT+RUN</b> .....	<b>11</b>
<b>3.6</b>	<b>AT+DBG</b> .....	<b>11</b>
<b>3.7</b>	<b>AT+DIR</b> .....	<b>11</b>
<b>3.8</b>	<b>AT+GET</b> .....	<b>12</b>
<b>3.9</b>	<b>AT+SET</b> .....	<b>12</b>
<b>3.10</b>	<b>AT+FOW</b> .....	<b>12</b>
<b>3.11</b>	<b>AT+FCL</b> .....	<b>12</b>
<b>3.12</b>	<b>AT+FWR</b> .....	<b>12</b>
<b>3.13</b>	<b>COMPILE</b> .....	<b>13</b>
<b>3.10</b>	<b>RUN</b> .....	<b>13</b>
<b>3.11</b>	<b>UWORD FLASHDEL</b> .....	<b>13</b>
<b>4.</b>	<b>CORE LANGUAGE COMMANDS</b> .....	<b>14</b>
<b>4.1</b>	<b>COMMENTLINE</b> .....	<b>14</b>
<b>4.2</b>	<b>IMM_COMMAND</b> .....	<b>14</b>
<b>4.3</b>	<b>DIRECTIVE</b> .....	<b>14</b>

<b>5.</b>	<b>CONSTANTS</b> .....	15
<b>5.1</b>	<b>Integer Constants</b> .....	15
<b>5.2</b>	<b>String Constants</b> .....	15
<b>6.</b>	<b>VARIABLES</b> .....	16
<b>6.1</b>	<b>Arithmetic expressions</b> .....	16
<b>7.</b>	<b>FUNCTIONS AND SUBROUTINES</b> .....	18
<b>7.1</b>	<b>SUBROUTINE</b> .....	18
<b>7.2</b>	<b>EXITSUB</b> .....	18
<b>7.2</b>	<b>ENDFUNC</b> .....	18
<b>7.3</b>	<b>EXITFUNC</b> .....	18
<b>7.4</b>	<b>ONEVENT</b> .....	19
<b>7.5</b>	<b>ONERROR</b> .....	19
<b>7.6</b>	<b>ONFATALERROR</b> .....	19
<b>7.7</b>	<b>WAITEVENT</b> .....	19
<b>7.8</b>	<b>POLLEVENT</b> .....	19
<b>7.9</b>	<b>STOP</b> .....	20
<b>7.10</b>	<b>RESUME</b> .....	20
<b>8.0</b>	<b>EXTENDED COMMANDS</b> .....	22
<b>8.1</b>	<b>SLEEP</b> .....	22
<b>8.2</b>	<b>UWORD STRLEN</b> .....	22
<b>8.3</b>	<b>UWORD GETERROR</b> .....	22
<b>8.4</b>	<b>STRING LEFT\$</b> .....	22
<b>8.5</b>	<b>STRING RIGHT\$</b> .....	22
<b>8.6</b>	<b>STRING MID\$</b> .....	22
<b>8.7</b>	<b>UWORD STRPOS</b> .....	22
<b>8.8</b>	<b>XX xxCASTyy</b> .....	22
<b>8.9</b>	<b>XX MINxx</b> .....	23
<b>8.10</b>	<b>XX MAXxx</b> .....	23
<b>8.11</b>	<b>SWORD ABSW</b> .....	23
<b>8.12</b>	<b>UWORD STRCMP</b> .....	23
<b>8.13</b>	<b>SLONG STRVALDEC</b> .....	23
<b>8.14</b>	<b>STRING STRESCAPE</b> .....	23

8.15	UWORD STRGETCHR .....	24
8.16	SWORD STRSETCHR.....	24
8.17	SWORD STRSETBLOCK.....	24
8.18	SWORD STRFILL ( STRING BYREF strvar1, UWORD nChr, UWORD nCount).....	24
8.19	SWORD NVRECORDGET ( UWORD BYVAL userRecNum, STRING BYREF strvar) .....	25
8.20	SWORD NVRECORDSET ( UWORD BYVAL userRecNum, STRING BYREF strvar) .....	25
8.21	STRROTLEFT ( BYREF STRING strvar, UWORD numChrs) .....	25
8.22	RANDSEED ( ULONG BYVAL seed) .....	25
8.23	ULONG RAND ( ).....	26
8.24	SWORD TABLEINIT ( BYREF STRING strvar) .....	26
8.25	SWORD TABLEADD ( BYREF STRING strvar, STRING strtok, UWORD nIndex).....	26
8.26	SWORD TABLELOOKUP ( BYREF STRING strvar, STRING strtok) .....	26
8.27	UWORD STDIN (BYREF STRING strvar).....	26
9.0	CONSTRUCTION STATEMENTS.....	27
9.1	IF .....	27
9.2	WHILE .....	27
9.3	DO.....	27
9.5	BREAK .....	27
9.6	CONTINUE.....	27
9.7	FOR.....	27
9.8	NEXT .....	28
9.9	SELECT .....	28
9.10	ENDSELECT.....	28
9.11	PRINT.....	28
	Appendix 1 - Error Messages reported .....	30

# EZURiO

## 1. INTRODUCTION

UWScript is a programming language that derives from the BASIC programming language, but which has been extended with additional tokens that enable specific aspects of the wireless connection. It also contains a number of instructions that are used to control the compiler and to query the module. These latter instructions include a collection of "AT" commands which will be familiar to designers who have used modems.

The manual divides into a number of sections which present the different aspects of the UWScript .

Section 3 covers the basic commands that are used to control the module and the compile process. It also explains the compile and run process.

Section 4 describes the core language token structure.

Sections 5 and 6 introduce the constant and variable of UWScript .

Sections 7, 8 and 9 describes the fundamental commands within UWScript .

Section 10 explains the commands that are specific to connecting and using wireless LAN functionality.

To help acquaint the user with UWScript , the appendices give some sample scripts that illustrate common wireless network tasks.

### 1.1 Types of Script

There are three types of pre-stored scripts which are run at different times:

1. Factory default script. This is a script which is pre-stored during the module production and cannot be changed by the user. The factory default script is executed at power on or following a module reset. The aim of this script is to set the module into a factory default operating state and to initialise all global variables prior to execution of user scripts.
2. User defined 'auto-run' script. This is a script that is pre-loaded by the user into the module and which is automatically executed at power on following the factory configuration script. Typically this might be used to pre-configure the module ready for communication with the host (for example setting the correct baud rate for the UART interface), or to automatically establish connection with an end point at power on.
3. User defined script. User defined scripts are downloaded by the host and stored in non-volatile memory using a simple flat file system where each script is referred to by name. Once a script has been downloaded and stored, it can be run at any time using the RUN command from the host.

### 1.2 Script Storage

Scripts are downloaded to the module via the UART interface. Scripts are stored in the module non-volatile memory using a simple flat file-system. A maximum of 10 scripts can be stored in memory at any one time with a maximum of 1000 script lines in total. Currently stored scripts can be listed with the DIR command and deleted using the DELETE command. When a script file is deleted, the file system automatically re-orders the scripts in memory to avoid file system fragmentation.

## 1.3 Language Basics

### 1.3.1 Variables

The UWScript interpreter supports super global variables (whose existence is independent of script execution), normal global variables (which exist in the scope of the script) and local variables (which exist in the scope of a subroutine). The UWScript interpreter does not support forward declarations. If any script attempts to access a variable, label or script name that has not been previously declared then the parser will report an ERROR.

### 1.3.2 Super Global Variables

The UWScript scripting language supplies pre-defined super global variables which can be accessed across all scripts. If required these variables can be used to allow passing of data between scripts. The super global variables are always in scope during the power on cycle of the module, and any value that they are set to will be retained while module is powered (including low power modes). A soft or hard reset, or power off/on cycle will cause the variables to be reset to their default value.

The factory default script is used to define and initialise these super global variables. The factory default script is run at power on prior to any other script being executed.

An "Autorun script" can be defined by the user to run after the "factory default" script in order to configure the device for operation in the required mode. This script may contain any of the UWSCRIPT commands permitted in Runtime mode. This script may also invoke other scripts which have already been downloaded into the device.

### 1.3.3 Normal Global Variables

Each script may contain normal global variables, which are valid from the time they are declared within the script, until the start of a new script. The variables remain in scope at the end of the script to enable the user or host processor to interrogate the variables after the script has ended. These variables are discarded on entering a new execution mode script.

### 1.3.4 Local Variables

Local variables are declared in subroutines and remain in scope until the subroutine returns.

### 1.3.5 Event Handling

The UWScript language interpreter allows for the occurrence of asynchronous events within a UWScript module. An example of an asynchronous event might be an unexpected loss of connection or the completion of a connection.

Asynchronous events are analogous to interrupts and are handled by event handler routines within the UWScript interpreter. UWScript commands are provided to allow event handler routines to be installed for each possible event source. The list of possible events will change dependent on the wireless technology.

Event handler functions. If the event handler returns 1 then the handler will return to the same point in the script. If the event handler returns 0 then it returns to the first construct in the script.

## 1.4 Language Constructs

### 1.4.1 Variable Declaration

UWScript is a strongly typed language and variables must be declared before use. Numeric variables are limited to 16 or 32 bits in length and can be signed or unsigned. String variables are allocated when they are first used (rather than being allocated at declaration) and do not have any length restrictions. One side-effect of this is that, since UWScript is executing in a limited memory environment, it is possible to run out of variable memory space at run-time if too many variables and strings are declared. Under these circumstances the script operation will be terminated, an error message issued and the module will return to interactive mode. The point at which this happens depends on the free memory available for UWScript variables and this in turn will vary dependent on the particular wireless platform in use (WISM for example has significantly more memory available than Bluetooth).

### 1.4.2 Constants

Numeric constants can be defined in decimal, hexadecimal, octal or binary using the following methods:

```
1234      -- decimal
H'1234    -- hex
D'1234    -- decimal
O'1234    -- Octal
B'010101  -- binary
```

### 1.4.3 Expressions

Arithmetic expressions can be used in a number of UWScript constructs. Expressions can use the following operators with the highest precedence shown first:

#### UNITARY OPERATORS

=====

```
!      not
~      complement
-      negate
+      positive
```

#### BINARY OPERATORS

=====

```
*      multiply
/      divide
%      mod
+      add
-      subtract
<<    shift left
>>    shift right
<      less than
<=    less than or equal
>      greater than
>=    greater than or equal
==    equals
!=    not equal to
&      bitwise AND
^      bitwise exclusive OR
|      bitwise OR
&&    logical AND
```

# EZURiO

^^ logical exclusive OR  
|| logical OR

In logical expressions, zero indicates false and non-zero indicates true. Brackets can be used to force precedence where necessary.

## 1.4.4 Branching

Conditional execution is supported by the IF THEN ELSE construct.

## 1.4.5 Iteration

Various forms of code iteration are provided:

- FOR NEXT loops (loop with test at the top of the loop and automatic loop variable iteration)
- WHILE ENDWHILE (loop with test at the top of the loop)
- DO UNTIL (loop with test at the bottom of the loop)

## 1.4.6 Functions and Subroutines

UWScript supports both subroutines and functions. Both subroutines and functions can accept parameters and functions can return a value. By default, all numeric variables are passed by value and all string variables are passed by reference. This default behaviour can be overridden using the #SET command.

UWScript supports a large number of built-in functions – primarily to support wireless specific language extensions and string manipulation.

## 1.4.7 I/O

UWScript supports character based I/O using the PRINT and INPUT built in functions. UWScript supports character based I/O using streams (Section **Error! Reference source not found.**). By default, the UWScript output stream is bridged to the UART and input to the script is bridged from the UART stream. Hence, by default, character based I/O uses the UART.

## 1.4.8 Wireless Control

Each wireless technology is supported by a range of built in functions and subroutines that enable the user to manipulate and configure the wireless interface. Whilst the details of the functions may vary between technologies the aim is to keep the level of abstraction the same so that a similar level of understanding is required regardless of the technology being used.

UWScript consists of a command parser and a single pass compiler. The command parser reads in a line of text, which consists of a series of token words that are then either acted upon directly or compiled into an executable script.

The compiler is a single pass compiler that reads the contents of a script and compiles it directly into an executable that can be stored in the module's memory for later execution.

## 2.0 SYNTAX

Throughout the document, the following convention are used to describe functionality. These are more fully described in the relevant sections:

## 2.1 Standard Abbreviations

`/n` Line Feed    `/r` Carriage Return    `/t` Horizontal Tab

## 2.2 Formats

`{}` Enclose Mandatory Features  
`<>` Enclose Optional Features  
`|` Separate Alternatives  
`'h` Hexadecimal number    `'o` Octal number    `""` string constant

## 2.3 Command Responses

Some commands return a unsigned 16 bit integer or UWORD when run. These are identified in the documentation by prefixing their definition with UWORD. When they are used in immediate mode, or without a script the UWORD prefix must not be used.

### 2.3.1 Implementation Status

This document describes the UWScript commands that will be part of the first release of the language. Until that release is made, engineering releases will be available for evaluation.

Customers evaluating UWScript should be aware that not all functions listed in this document are available within an engineering release. Where a function is not fully tested it will be implemented using an alternative nomenclature that prefixes it with an underscore, e.g.

Final Release	Engineering Release
<code>attach()</code>	<code>_attach()</code>

Examples of this can be seen in the sample code in Appendix 1.

Customers should consult the function matrix that is supplied with each firmware release to identify the status of commands.

## 3. COMPILER AND MODULE INSTRUCTIONS

Note that AT and Compiler instructions do not use brackets for their arguments. This is to maintain compatibility with Hayes AT type commands. However they require spaces after the "AT" and between the arguments.

### 3.1 AT

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed. It performs no action other than to respond with \n00\r (linefeed character, 00, carriage return character). It exists to emulate the behaviour of a device which is controlled using the 'AT' protocol.

### 3.2 AT I integer\_constant

integer\_constant := in range 0 .. 65535

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

Note the requirement for a space between the "AT", the "I" and the integer constant.

The response will be: \n10\tMM\tInformation\r\n00\r

- \n = linefeed character 0x0A
- \t = horizontal tab character 0x09
- MM = integer\_constant
- Information = sting consisting of information requested
- \r = carriage return character 0x0D

The following specific information is catered for :-

- 0: Name of device
- 1: UWScript Build Number
- 3: Version number
- 10002 : "FileSystem in RAM" or 'Manufacturer name'
- 10003 : Build Date and Time
- 10005 : Either RAM or RASH or FLASH or 'Manufacturer name' Else :  
Manufacturer name

### 3.3 AT+**CMP** { string\_constant | END }

string\_constant := Must contain a valid filename.

The filename cannot contain any of the following seven characters : \* ? " < > |

**END** := This marks the end of the compilation.

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

This command is used to start the download and compilation of a script. If the filename specified is valid and does not yet exist in the file system then the response will be : \n00\r

If the file already exists it will respond with an error. If in the middle of a compile of an earlier file, it will respond with an error.

The maximum length of filename is FSA\_MAX\_FILENAME\_LEN

### 3.4 AT+**DEL** string\_constant < + >

string\_constant := Must contain a valid filename.

The filename cannot contain any of the following seven characters : \* ? " < > |

# EZURiO

This command will not delete the current running file unless the optional + is appended. This is used to force deletion of a file which is open due to it having been recently run.

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

If the file does not exist, or if it was successfully erased, it will respond with \n00\r.

The maximum length of filename is FSA\_MAX\_FILENAME\_LEN

### 3.5 **AT+RUN** string\_constant

string\_constant := Must contain a valid filename.

The filename cannot contain any of the following seven characters: \* ? " < > |

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

This command is used to run a precompiled script, if the script exists then the response will be \n00\r.

If any variables exist from a previous run, they will be destroyed before the new script is serviced. When or if the script reaches its end, a deferred OK response will be sent (\n00\r)

The maximum length of filename is FSA\_MAX\_FILENAME\_LEN

### 3.6 **AT+DBG** <string\_constant >

string\_constant := Must contain a valid filename which cannot contain any of the following seven characters : \* ? " < > |

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed. Debugging is enabled hence all BP nnnn statements will be active

This command is used to run a precompiled script. If it exists then the response will be \n00\r. If any variables exist from a previous run, they will be destroyed before the new script is serviced. When/If the script reaches it's end, a deferred ok response will be sent (\n00\r)

The maximum length of filename is FSA\_MAX\_FILENAME\_LEN

### 3.7 **AT+DIR** < string\_constant >

< string\_constant > := Optional pattern match string

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

This command is used to list the names of all compiled scripts. If the optional pattern match string is specified, then only the filenames with that substring embedded inside will be listed.

The response will be as follows:

```
\n06\tFILENAME1\r
\n06\tFILENAME2\r
\n06\tFILENAME\r
\n00\r
```

If the list is empty then only \n00\r will be sent.

### 3.8 **AT+GET** integer\_constant integer\_constant := ID of the config key

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed. This command is used to read the content of the configuration key. All config keys are stored as an array of 16 bit words.

The response to this command will be \n07\tiiii oooo hhhh hhhh hhhh hhhh\r\n00\r where each line starting with a 07 will have up to 8 words. If the key contains more data words, then more of these 07 lines are displayed.

- oooo value (hexadecimal) specifies the start offset of the data in the key.
- iiii (hexadecimal) is an echo of the config key ID specified in the command
- hhhh is the config key data in hexadecimal.

### 3.9 **AT+SET** integer\_constant = string\_constant integer\_constant := ID of the config key string\_constant := "hhhh hhhh hhhh"

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

This command is used to write a new value into a configuration key. All config keys are stored as an array of 16 bit words Unlike the GET command the entire value array is specified in a single command.

The new data array is specified as fixed format 4 digit hex numbers (with optional H' prefixes). For example AT+SET 1 = "1234 5678 9ABC" or AT+SET 1 = "H'1234 5678 H'9ABC" If the config key is successfully updated, the response will be \n00\r.

### 3.10 **AT+FOW** string\_constant string\_constant := Must contain a valid filename which cannot contain the following seven characters : \* ? " < > |

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

Open a file for writing to.

### 3.11 **AT+FCL** This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

Closes a file which was opened using FOPENWR

### 3.12 **AT+FWR** string\_constant string\_constant := Any \NN or \r or \n will get de-escaped before writing to file

This is an immediate mode command and **MUST** be terminated by a carriage return for it to be processed.

Write the string after de-escaping to a previously open file

### 3.13 **COMPILE** "string\_constant"

string\_constant := Must contain a valid filename. It may not contain the following seven characters : \* ? " < > |

The command recompiles an already existing script stored on the module, it assumes that the script file will have extension .UWScript and if it does not exist, then it will look for a .txt file, compile it and then run it.

This is an immediate mode command and MUST be terminated by a carriage return for it to be processed.

### 3.10 **RUN** "string\_constant"

string\_constant := Must contain a valid filename. It may not contain the following seven characters : \* ? " < > |

The command executes an already existing and compiled script on a module, it assumes that the script file will have extension .UWScript and if it does not exist, then it will look for a .txt file, compile it and then run it.

This is an immediate mode command and MUST be terminated by a carriage return for it to be processed.

- If filename.UWScript exists and filename.txt does not exist, then run it.
- If filename.UWScript does not exist but filename.txt exist, then compile it and then run the newly generated .UWScript file
- If filename.UWScript and filename.txt both exist, then compile first if the .txt file is newer than the .UWScript file

### 3.11 **UWORD FLASHDEL** ( )

Deletes the contents of the FLASH file system. The flash file system is used to store the UWSCRIPT scripts.

Returns 0 on success, non-zero on failure.

## 4. CORE LANGUAGE COMMANDS

A line of UWScript will generally start with a token of the following type:

**COMMENTLINE | COMMAND | STATEMENT | DIRECTIVE**

And appended with either a COMMENTLINE or TERMINATOR, a single line of UWScript can contain more than one statement but these must be separated by a colon `:`.

**{COMMENTLINE | COMMAND | STATEMENT | DIRECTIVE} :  
< COMMENTLINE | COMMAND | STATEMENT | DIRECTIVE >**

### 4.1 COMMENTLINE

Allows non executable statements to be embedded into a script, these text statements can be used as descriptions of the program flow or as a reminder for code statements.

***'this is a comment'***

Comment lines must start and end with an apostrophe ` ` and any text within these statement lines will not be executed even if they are valid UWScript commands.

***'this is a comment and the following will not be executed declare string var'***

Comments can be appended to the end of a structured executable command, the executable statement will be executed but the comments will be ignored.

***declare string var 'the code was executed but this statement was ignored'***

### 4.2 IMM\_COMMAND

Is the code token interpreted by the compiler, the token must be appended with the particular arguments attributed to that particular token.

IMM\_COMMAND <optional\_arguments>

### 4.3 DIRECTIVE

A line of UWScript starting with the # character is an instruction to modify the behaviour of the following variables.

DIRECTIVE < variable >

Example

# SET integer\_constant1, integer\_constant2

In this example the type of behaviour to modify is specified by the SET command with the integer\_constant1 and integer\_constant2 being the specific variables being modified.

## 5. CONSTANTS

### 5.1 Integer Constants

An integer constant is a value that is determined during compilation and that cannot change during the execution of the script. Integer constants supported in UWScript can be specified in either Decimal (default), Hexadecimal, Octal or Binary.

A decimal constant consists of a string consisting of characters 0 to 9 and can be prefixed by the two character token D' or d' to signify decimal.

Example,  
1234 D'1234 d'567890

The maximum decimal constant that can be entered in a script is 2147483647 and the minimum is -2147483647.

Example  
H'1234 h'56789

A hexadecimal constant consists of a string consisting of characters 0 to 9, and A to F or a to f. It must be prefixed by the two character token H' or h'.

Example  
O'1234 o'56439 An octal constant consists of a string consisting of characters 0 to 7. It must be prefixed by the two character token O' or o'.

Example  
B'110111000 b'111010 A binary constant consists of a string consisting of characters 0 and 1. It must be prefixed by the two character token B' or b'.

### 5.2 String Constants

String constant is any sequence of ASCII characters preceding and ending with double quotation marks "this is a string constant". The string constant is declared in the UWScript and its value is set during compilation and cannot change during execution.

Example  
declare string my\_string  
my\_string="hello world"  
print my\_string

UWScript allows control characters such as Linefeed \n or Carriage Return \r can be included into the string constant during compilation.

## 6. VARIABLES

UWScript supports two classes of variables, simple and complex.

There are four types of simple variable:

UWORD is an unsigned 16 bit integer,

SWORD is a signed 16 bit integer,

ULONG is an unsigned 32 bit integer and

SLONG is a signed 32 bit integer.

There is one type of complex variable: STRING which is used to manipulate strings. Unlike C programming UWScript STRING variables are not null terminated but rather length defined. All variables have name and scope. Names begin with the letters A to Z, 0 to 9 and underscore \_. In UWScript you can use a combination of these letters and numbers. All variables are case sensitive.

Each variable can be assigned one of three scopes: SUPER, GLOBAL and LOCAL. A SUPER variable is declared as a GLOBAL variable as a default and will always be visible from within immediate mode.

A GLOBAL variable is a variable that is declared outside of a function or subroutine. GLOBAL and SUPER variables cannot have the same name assigned.

A LOCAL variable is one which is declared within a function or subroutine and can share the same name as a SUPER or GLOBAL variable. When shared names are used the LOCAL variable will take precedence over the SUPER or GLOBAL within the function that it is declared.

Simple variables are scalar and as such can be used within arithmetic expressions.

No variable can be referenced before it is declared.

### 6.1 Arithmetic expressions

Arithmetic expressions are a sequence of numeric integer constants, variables and operators, which are evaluated during the execution of an UWScript to produce a simple value. The outcome of this value is always the same type as the variable on the left hand side of an expression. The default type of an expression is SWORD, this can be changed by preceding the expression with either UWORD, ULONG or SLONG.

Operators can be unitary or binary and affect the variable or constant that follows them. The latter operates on the two entities on either side. Operators in any expression observe a priority which is used to evaluate the final result using RPN (Reverse Polish Notation). In UWScript expressions the use of brackets () alter the priority of arithmetic expressions in the same manner as in other programming languages.

Unitary operators appear on the left hand side of an operand and consist of the following, ordered with the highest priority:

- ! logical NOT
- ~ bit complement
- - negative
- + positive

Binary operators appear in the middle of two operands and the list is as follows starting with the highest priority:

- \* Multiply

- / Divide
- % Modulus
- + Addition
- - Subtraction
- << Arithmetic Shift Left
- >> Arithmetic Shift Right
- < Less Than (results in a 0 or 1 value in the expression)
- <= Less Than Or Equal (results in a 0 or 1 value in the expression)
- > Greater Than (results in a 0 or 1 value in the expression)
- >= Greater Than Or Equal (results in a 0 or 1 value in the expression)
- == Equal To (results in a 0 or 1 value in the expression)
- != Not Equal To (results in a 0 or 1 value in the expression)
- & Bitwise AND
- ^ Bitwise XOR (exclusive OR)
- | Bitwise OR
- && Logical AND (results in a 0 or 1 value in the expression)
- ^^ Logical XOR (results in a 0 or 1 value in the expression)
- || Logical OR (results in a 0 or 1 value in the expression)

To declare one or more variables in UWScript we call the varlist declaration. If varlist is called within a subroutine or function, then the declared variables will have a local scope, otherwise they can be reference anywhere within a script.

## **< DECLARE > varlist**

varlist :=vardec < , varlist >

vardec :=vartype varname < [ integer\_constant ] >

vartype :={ UWORD | SWORD | ULONG | SLONG | STRING }

varname :=valid variable name < [ integer\_constant ] >

Variable will be an array with integer\_constant elements

integer\_constant :=decimal number | octal number | hex number | binary number

decimal number :=< + | - >ddigit<ddigits>

octal number :=O'odigit<odigits>

hex number :=H'hdigit<hdigits>

binary number :=B'bdigit<bdigits>

ddigit :=0|1|2|3|4|5|6|7|8|9

odigit :=0|1|2|3|4|5|6|7

hdigit :=0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|a|b|c|d|e|f

bdigit :=0|1

If a local variable name is declared and has already been declared in a global scope, then while in the particular function or subroutine the global variable will be ignored and the local variable used within the function or subroutine only.

There are 2 classes of variables, simple and complex. UWORD, SWORD, ULONG and SLONG are examples of simple variables and STRING is an example of a complex variable. Arithmetic operations (in expressions) can only be performed on simple variables. There is a very limited set of operations (in expressions) that can be performed on complex variables.

When the variable is declared as an array (only one dimension is allowed) it must be referenced in the script source with the [x] where 'x' is an integer constant. When the variable is referenced in an expression it MUST be indexed using the [x] syntax and in this case 'x' can either be an integer constant or a variable. In fact, if it is a variable index and it was also declared as an array, then it should also specify the [x]. This can happen recursively.

## 7. FUNCTIONS AND SUBROUTINES

**7.1 SUBROUTINE** `routinename ( < argdec > ) < argdec > :=vartype < BYREF | BYVAL > varname <,argdec >`  
`vartype := { UWORD | SWORD | ULONG | SLONG | STRING }`  
`varname := valid variable name - array var not allowed`  
`routinename := valid user routine name`

This statement marks the beginning of a block of statement which will consist of the body of a user routine. It takes arguments, but returns no values. The optional BYREF or BYVAL specifies whether the argument will be passed to it as a reference (pointer) or by value. By default, complex variables are passed BYREF and simple variables BYVAL. This default behaviour can be modified using the # SET directive as follows:

- # SET 1,0 Default Simple args are BYVAL
- # SET 1,1 Default Simple args are BYREF
- # SET 2,0 Default Complex args are BYVAL
- # SET 2,1 Default Complex args are BYREF ENDSUB

This statement marks the end of a block of statement belonging to a subroutine. It marks the end of scope on any variables declared within that block.

### 7.2 EXITSUB

This statement is used at run-time to perform an early exit from the subroutine  
`FUNCTION vartype routinename ( < argdec > )`  
`< argdec > := vartype < BYREF | BYVAL > varname <,argdec >`  
`vartype := { UWORD | SWORD | ULONG | SLONG | STRING }`  
`varname := valid variable name - array var not allowed`  
`routinename := valid user routine name`

This statement marks the beginning of a block of statements which will consist of the body of a user routine. It takes arguments AND returns a value. The optional BYREF or BYVAL specifies whether the argument will be passed to it as a reference (pointer) or by value. By default, complex variables are passed BYREF and simple variables BYVAL. This default behaviour can be modified using the #SET directive as follows:

- # SET 1,0 Default Simple args are BYVAL
- # SET 1,1 Default Simple args are BYREF
- # SET 2,0 Default Complex args are BYVAL
- # SET 2,1 Default Complex args are BYREF

Since this routine returns a value it must appear on the right hand side of an expression. Basically, its value HAS to be 'used up' so that the underlying expression evaluation stack does not have 'orphaned' values left on the stack.

### 7.2 ENDFUNC arithmetic\_expression

`arithmetic_expression :=` A simple or complex expression depending on return type of function This statement marks the end of a block of statement belonging to a function. It marks the end of scope on any variables declared within that block. The expression value is the return value of the function.

### 7.3 EXITFUNC arithmetic\_expression

`arithmetic_expression :=` A simple or complex expression depending on return type of function

At run-time the function is exited with the value of the expression

- 7.4 ONEVENT** integer\_constant { callfunc | DISABLE }  
integer\_constant := In range 0 to MAX\_EVENT\_NUMBER  
callfunc := CALL routinename  
routinename := It must be function (not a subroutine) which returns a UWORD value and takes no arguments

This statement is used to bind a user routine to the event identified by integer\_constant when the syntax is "ONEVENT nn CALL userrtn" and to unbind when the syntax is "ONEVENT nn DISABLE". The event is thrown from within the run-time engine and events will be polled ONLY WHEN the WAITEVENT statement is being processed. The user routine that is specified MUST be a function which takes no arguments and MUST return a UWORD value. When WAITEVENT services an event handler and the return value from that routine is non-zero, then it will continue to wait for more events. A zero value will force the next statement after WAITEVENT to be processed.

- 7.5 ONERROR** { REDO | NEXT } routinename  
routinename := It must be subroutine (not a function) which takes no parameters

This statement is used to bind a user routine which gets called when an error is encountered at run-time. Use the built-in routine GETERROR() in the user routine to determine the type of error that was generated.

If REDO is specified, then on return from the error handler, the offending statement is reprocessed.

If NEXT is specified, then on return the next statement is process on return from the error handler.

See UWRESULTCODE\_RUN\_xxx for typical error values

- 7.6 ONFATALERROR** routinename  
routinename := It must be subroutine (not a function) which takes no parameters

This statement is used to bind a user routine which gets called when a fatal error is encountered at run-time. Use the built-in routine GETERROR() in the user routine to determine the type of error that got generated  
See UWRESULTCODE\_RUN\_xxx for typical error values

- 7.7 WAITEVENT**  
Event handlers will be called. If an event handler which must be a FUNCTION taking no arguments and returning a UWORD returns a zero value then the next statement after the WAITEVENT will be processed.

- 7.8 POLLEVENT**  
POLLEVENT checks if any events are pending and handles the highest priority one. Only one event is handled, the highest priority event. After handling the event, execution continues to the statement after the POLLEVENT statement. If no event is pending, the next statement after the POLLEVENT will be processed.

The return value from event handlers are ignored.

## 7.9 STOP

Use this statement in a script to stop it running so that the device falls back into immediate command line mode. In immediate mode the command RESUME is used to restart the script from the next statement after the STOP statement

## 7.10 RESUME

This is an immediate mode command and MUST be terminated by a carriage return for it to be processed. When a script has stopped running after a STOP command was encountered. This immediate mode command is used to resume operation at the next statement after the STOP statement. If there is nothing to resume (e.g. immediately after reset), then an error response is sent.

```
? < D' | H' | O' | B' > varname  
< D' | H' | O' | B' > :
```

This is an optional integer type prefix.

D' := Decimal

H' := Hexadecimal

O' := Octal

B' := Binary

varname : Valid variable name

This is an immediate mode command and MUST be terminated by a carriage return for it to be processed.

When a script encounters a STOP or END statement, it will fall into the immediate mode of operation and will not discard the global variables; this means that they can be referenced in immediate mode. This command is used to display the content of a known variable.

If the variable exists and it is a simple type then the response to this command will be :-\n08\tnnnnnn\rn00\r,

If the variable is o string type, then the response will be \n08\t"Hello World"\r \n00\r

If the variable does not exist then the response to this command will be :-\n01\t35\r or \n01\t35\tVAR\_EXPECTED\r

Note: if the optional type prefix is present, the output value, when it is an integer constant will be displayed in that base.

For example

```
\n08\tH'nnnnnn\r\n00\r = varname {integer_constant | string_constant}
```

```
varname :=Valid variable name {integer_constant | string_constant} :Either an integer constant or a string constant
```

This is an immediate mode command and MUST be terminated by a carriage return for it to be processed.

This command can be used to allow the content of a known variable to be changed whilst in immediate mode. If the script then is resumed, the variable will contain the new value. If the variable exists and it is of compatible type then the response to this command will be :-\n00\r

If the variable exists and it is NOT of compatible type then the response to this command will be :-\n01\t39\r\n01\t39\tTYPE\_MISMATCH\r

If the variable exists but the new value is missing, then the response will be\n01\t38\r\n01\t38\tVALUE\_EXPECTED\r

# EZURiO

If the variable does not exist the response will be `:-\n01\t35\r` or `\n01\t35\tVAR_EXPECTED\r`

After a running script ends or is stopped, its context, including stacks is not discarded. This immediate mode command allows any built in routine or a script routine to be invoked from immediate mode. One limitation is that, whereas in a script the arguments to a routine can be the result of an arithmetic or string expression, in immediate mode, only variables are allowed.

## 8.0 EXTENDED COMMANDS

### 8.1 SLEEP ( ULONG msec )

ULONG msec := A 32 bit unsigned value in milliseconds

Puts the module to sleep for 'msec' milliseconds

### 8.2 UWORD STRLEN ( BYREF STRING strvar )

BYREF STRING strvar := Reference to a string variable

Returns the length of the string in bytes

### 8.3 UWORD GETERROR ( )

Returns the most recent error value

### 8.4 STRING LEFT\$ ( BYREF STRING strvar, UWORD len )

BYREF STRING strvar := Reference to a string variable

UWORD len := Leftmost 'len' characters

Returns a string which consists of leftmost 'len' characters of the input string 'strvar'

### 8.5 STRING RIGHT\$ ( BYREF STRING strvar, UWORD len )

BYREF STRING strvar := Reference to a string variable

UWORD len := Rightmost 'len' characters

Returns a string which consists of rightmost 'len' characters of the input string 'strvar'

### 8.6 STRING MID\$ ( BYREF STRING strvar, UWORD pos, UWORD len )

BYREF STRING strvar := Reference to a string variable

UWORD pos := Start position (0 based)

UWORD len := 'len' characters from 'pos'

Returns a string which consists of 'len' characters of the input string 'strvar' starting at position 'pos'

### 8.7 UWORD STRPOS ( BYREF STRING strvar1, BYREF STRING strvar2, BYREF UWORD pos )

BYREF STRING strvar1 := Reference to a string variable

BYREF STRING strvar2 := Reference to a string variable which is embedded inside strvar1

BYREF UWORD pos := Start position (0 based)

Returns 1 if strvar2 is found embedded in strvar1. On exit 'pos' variable will have been updated with the actual position.

### 8.8 XX xxCASTyy ( yy varname )

yy varname :=

A variable named varname of type yy

xx is a valid variable type :- UW, SW, UL, SL

yy is a valid variable type :- UW, SW, UL, SL

XX is a valid variable type :- UWORD, SWORD, ULONG, SLONG

Converts the variable varname of type yy into type xx and returns it

- 8.9 XX MINxx** ( xx var1, xx var2 )  
xx var1 :=  
xx var2 := Return the minimum of two variable of type xx where xx is a valid variable type :- UW, SW, UL, SL  
XX is a valid variable type :- UWORD, SWORD, ULONG, SLONG return the minimum of the two variables
- 8.10 XX MAXxx** ( xx var1, xx var2 )  
xx var1 :=  
xx var2 := Return the maximum of two variable of type xx where xx is a valid variable type :- UW, SW, UL, SL XX is a valid variable type :- UWORD, SWORD, ULONG, SLONG return the maximum of the two variables
- 8.11 SWORD ABSW** ( sword var1)  
SLONG ABSL ( slong var1)  
sword var1 :=  
slong var1 := Signed variable return the absolute value of the input value
- 8.12 UWORD STRCMP** ( BYREF STRING strvar1, BYREF STRING strvar2)  
BYREF STRING strvar1 := Reference to a string variable  
BYREF STRING strvar2 := Reference to a string variable which is embedded inside strvar1

This is a builtin function.

Returns 0 if strvar1 matches strvar2.

The comparison is case sensitive Returns -1 if ascii comparison of strvar1 is less than strvar2

Returns 1 if ascii comparison of strvar1 is more than strvar2 This is similar functionality to strcmp function in 'C'

- 8.13 SLONG STRVALDEC** ( BYREF STRING strvar )  
BYREF STRING strvar := Reference to a string variable

This is a builtin function.

The input string is assumed to consist of decimal digits and that string is converted into a value and returned. Any leading whitespace is ignored.

The conversion stops when a non-decimal-digit is encountered in the string

- 8.14 STRING STRESCAPE** ( BYREF STRING strvar )  
BYREF STRING strvar := Reference to a string variable

This is a builtin function.

The input string is escaped so that the following characters are replaced by a multicharacter byte sequences

carriage return	= \r
linefeed	= \n
horizontal tab	= \t
\	= \\
"	= \"
chr < ' '	= \HH
chr >= 0x7F	= \HH

If a parsing error is encountered a nonfatal error will be generated which needs to be handled otherwise the script will abort.

## 8.15 **UWORD STRGETCHR** ( BYREF STRING strvar1, UWORD nIndex)

BYREF STRING strvar1 := Reference to a string variable

UWORD nIndex := Zero based index into the string.

This is a builtin function.

Returns the ascii value of the nIndex'th character in the string, where nIndex is zero based.

If the nIndex value specifies a position beyond the end of the string, a value of -1 is returned

## 8.16 **SWORD STRSETCHR** ( BYREF STRING strvar1, UWORD nChr, UWORD nIndex)

BYREF STRING strvar1 := Reference to a string variable that needs to be updated  
UWORD nChr := The chr to write into the string this should be a value in the range 0 to 255 inclusive

UWORD nIndex := Zero based index into the string where the character will be updated.

This is a builtin function.

Allows the nIndex'th character in the string to be replaced by the value nChr. If nIndex is beyond the end of the string, then string is expanded to allow that character to be updated.

The fill character is nChr. If the character gets updated, then the return value is zero.

If nChr is greater than 255 the function will return -1 and also will be -2 if the string cannot be extended. A -3 will be returned if the resultant string will be longer than allowed

## 8.17 **SWORD STRSETBLOCK** ( BYREF STRING strvar1, UWORD nChr, UWORD nIndex, UWORD nBlockLen)

BYREF STRING strvar1 := Reference to a string variable that needs to be updated  
UWORD nChr := The chr to write into the string. Should be a value in the range 0 to 255 inclusive

UWORD nIndex := Zero based index into the string where the character will be updated.

UWORD nBlockLen := Number of characters to update

This is a builtin function.

Allows nBlockLen characters starting from the nIndex'th character in the string to be replaced by the value nChr.

If nIndex+nBlockLen is beyond the end of the string, then string is expanded to allow that character to be updated.

The fill character is nChr. If the block gets updated, then the return value is zero

If nChr is greater than 255 the function will return -1 and also will be -2 if the string cannot be extended. A -3 will be returned if the resultant string will be longer than allowed

## 8.18 **SWORD STRFILL** ( STRING BYREF strvar1, UWORD nChr, UWORD nCount)

STRING BYREF strvar1 := Reference to a string variable that needs to be updated  
UWORD BYVAL nChr := The chr to write into the string. Should be a value in the range 0 to 255 inclusive

UWORD BYVAL nCount := Number of characters to fill. Must be < 32767 (0x7FFF)

This is a builtin function.

Fills the string specified with nCount characters of nChr. If nChr is greater than 255 the function will return -1 and also will be -2 if the string cannot be extended. A -3 will be returned if the resultant string will be longer than allowed

## 8.19 **SWORD NVRECORDGET ( UWORD BYVAL userRecNum, STRING BYREF strvar )**

UWORD BYVAL userRecNum := Record number to read, in range 1 to N (build dependent)

STRING BYREF strvar := Reference to a string variable into which the record data will be read

This is a builtin function.

Read the record identified by 'userRecNum' into string 'strvar'. Returns the number of bytes read. If invalid record number is specified then -1 is returned. There are a limited number of user records which can be written to.

This is a build dependent value

## 8.20 **SWORD NVRECORDSET ( UWORD BYVAL userRecNum, STRING BYREF strvar )**

UWORD BYVAL userRecNum := Record number to read, in range 1 to N (build dependent)

STRING BYREF strvar := Reference to a string variable from which the record data will be written.

This is a builtin function.

Write the record identified by 'userRecNum' into string 'strvar'. Returns the number of bytes written.

Note it will be rounded up to the nearest even value. If the data needs to be padded to make it even bytes long then the pad byte will have random data.

If invalid record number is specified then -1 is returned. There are a limited number of user records which can be written too.

This is a build dependent value

WARNING: Minimise writes as each time a record is changed, flash is used up and at some point there will be no more free flash memory and at that point a power cycle will be required to force a defrag of the flash.

## 8.21 **STRROTLEFT ( BYREF STRING strvar, UWORD numChrs )**

BYREF STRING strvar := Reference to a string variable

UWORD numChrs := number of characters to rotate by.

This is a builtin subroutine.

Left rotate the string by the number of characters specified. If the current string is shorter than numChrs then the result is an empty string.

## 8.22 **RANDSEED ( ULONG BYVAL seed )**

ULONG BYVAL seed := Seed value to use.

This is a builtin subroutine.

The RANDSEED function sets the starting point for generating a series of pseudorandom integers. To reinitialize the generator, use 1 as the seed argument. Any other value for seed sets the generator to a random starting point. rand retrieves the pseudorandom numbers that are generated. Calling rand

before any call to srand generates the same sequence as calling srand with seed passed as 1.

## 8.23 ULONG RAND ( )

This is a builtin function.

The RAND function returns a pseudorandom integer in the range 0 to RAND\_MAX. Use the RunExcRANDSEED function to seed the pseudorandom-number generator before calling rand.

See also RANDSEED subroutine.

## 8.24 SWORD TABLEINIT ( BYREF STRING strvar)

BYREF STRING strvar := Reference to a string variable

This is a builtin function.

Initialises the string variable so that it is prepared for token storage so that a lookup table can be created. Returns 0 for success, other values for fail.

See also TABLEADD and TABLELOOKUP

## 8.25 SWORD TABLEADD ( BYREF STRING strvar, STRING strtok, UWORD nIndex)

BYREF STRING strvar := Reference to a string variable

STRING strtok := Token to add to the lookup table

UWORD nIndex := Index to associate with this token

This is a builtin function.

Adds the token specified to the lookup table in strvar and associate the index specified with it.

Returns 0 if it was successfully added 1 if nIndex > 255 2 if no memory available to store token 3 if the token is too large 4 if the token is empty

See also TABLEINIT amd TABLELOOKUP

## 8.26 SWORD TABLELOOKUP ( BYREF STRING strvar, STRING strtok)

BYREF STRING strvar := Reference to a string variable

STRING strtok := Token to add to the lookup table

This is a builtin function.

Search the table defined by strvar for the token strtok and Returns the index number associated with that token, otherwise -1 if no match found -2 if table is invalid -3 if token is empty or >255 characters long. If multiple tokens exist in the table with the same associated then the first instance in the table is returned.

See also TABLEINIT amd TABLEADD

## 8.27 UWORD STDIN (BYREF STRING strvar)

BYREF STRING strvar := Reference to a string variable which will get updated

This is a builtin function

Used to read ALL the data in the stdin stream into the string specified. The data is APPENDED to any existing data already in the string.

## 9.0 CONSTRUCTION STATEMENTS

- 9.1 IF** <vartype> arithmetic\_expression THEN statement\_block\_A { ENDIF | elseblock} elseblock : ELSE statement\_block\_B ENDIF  
vartype := { UWORD | SWORD | ULONG | SLONG }  
arithmetic\_expression := Arithmetic expression with precedence observed  
statement\_block\_A := One or more valid statements  
statement\_block\_B := One or more valid statements

If arithmetic\_expression evaluates to a non-zero value then statement\_block\_A is processed until the corresponding ELSE or ENDIF is reached. If the expression evaluates to 0, then the statements after the corresponding ELSE or ENDIF is processed.

If vartype is not specified then the expression is assumed to be of type SWORD and any variables not of that type will result in a compile error

- 9.2 WHILE** <vartype> arithmetic\_expression statement\_block ENDWHILE  
vartype := { UWORD | SWORD | ULONG | SLONG }  
arithmetic\_expression := Arithmetic expression with precedence observed  
statement\_block := One or more valid statements

If arithmetic\_expression evaluates to a non-zero value then statement\_block is processed until the corresponding ENDWHILE is reached. If the expression evaluates to 0, then the statements after the corresponding ENDWHILE is processed.

If vartype is not specified then the expression is assumed to be of type SWORD and any variables not of that type will result in a compile error

- 9.3 DO** statement\_block UNTIL <vartype> arithmetic\_expression  
vartype := { UWORD | SWORD | ULONG | SLONG }  
arithmetic\_expression := Arithmetic expression with precedence observed  
statement\_block := One or more valid statements

### 9.5 BREAK

This is relevant in a WHILE/ENDWHILE, DO/UNTIL, FOR/NEXT. or SELECT compound statement. It forces the program counter to exit the block. For example, in a WHILE/ENDWHILE loop the statement after the ENDWHILE is processed. In a DO/UNTIL, the statement immediately after the UNTIL is processed

### 9.6 CONTINUE

This is relevant in a WHILE/ENDWHILE, DO/UNTIL or FOR/NEXT compound statement. It forces the program counter to restart the block.

- 9.7 FOR** varname = arithmetic\_expression1 {TO | DOWNT } arithmetic\_expression2 < STEP arithmetic\_expression3> statement\_block  
arithmetic\_expression1 := Arithmetic expression of same type as varname  
arithmetic\_expression2 := Arithmetic expression of same type as varname  
arithmetic\_expression3 := Arithmetic expression of same type as varname  
statement\_block := One or more valid statements  
All three scalar expressions must result in a value of the same type as varname.

The `statement_block` will be processed with `varname` starting with value `arithmetic_expression1`. When `NEXT` is processed, the value as a result of `arithmetic_expression3` is added to variable `varname` if `TO` was specified, or subtracted if `DOWNTO` was specified.

It continues to loop until the variable `varname` contains a value  $\geq$  `arithmetic_expression2` when `TO` was specified or  $\leq$  `arithmetic_expression2` when `DOWNTO` was specified. If the optional `<STEP arithmetic_expression3>` is not present, then a step value of 1 is assumed.

## 9.8 NEXT

See FOR.

## 9.9 SELECT <vartype> arithmetic\_expression <statement\_block\_A>

```
CASE integer_constant1
    statement_block_B
CASE integer_constant2 TO integer_constant3
    statement_block_C
CASE integer_constant4
    statement_block_D
CASE ELSE
    statement_block_E
```

## 9.10 ENDSELECT

`vartype` := { UWORD | SWORD | ULONG | SLONG }  
`arithmetic_expression` := Arithmetic expression with precedence observed  
`statement_block_n` := One or more valid statements

At run time `arithmetic_expression` is evaluated and then the optional `<statement_block_A>` is always processed (a bit different to what you encounter in switch statements in C) until the first CASE statement is encountered. At this point, it will search for a CASE statement with an integer value (or range) which matches the value produced by the `arithmetic_expression`. The statement block after that CASE statement is then processed until a CASE or BREAK is encountered when it will jump to the statement immediately after the ENDSELECT.

When searching for a matching CASE statement, and a match is not found then the CASE ELSE statement block is processed. Unlike in 'C' programming language, the CASE ELSE statement is mandatory.

## 9.11 PRINT < # uword\_expr > exprlist

`uword_expr` := An arithmetic expression of type UWORD. This specifies the output channel

If this field is not present, the output is assumed to be to STDOUT <<In some builds this field will be compiled out>>

`exprlist` := <type.<decimal\_const.> { D' | H' | O' | B' } >>  
`arith_expr` <{ , | ; } exprlist> or <STRING > `string_expr` < { , | ; } exprlist >

# EZURiO

When H', O' or B' is specified it is used to force the output of the integer to be in that base. A , or ; separator between expressions is used to either send a TAB character or nothing respectively. If the first token of the expression is a literal string (hence starting with a ") or it is a variable, then 'type' can be omitted as the compiler can resolve the type of expression and tokenise accordingly

type := One of UWORD, SWORD, ULONG, SLONG

decimal\_const := Minimum characters to be output - fill with leading spaces <<In some builds this field will be compiled out>>

arith\_expr := An arithmetic expression

string\_expr := A String expression

This keyword is used to print the output of a series of either arithmetic or string expression to stdout if <#uword\_expr> is missing or to the channel specified if it is present. In some builds the <#uword\_expr> field will be missing and so the output shall always be to stdout.

## Appendix 1 - Error Messages reported

0200 ( 512) GENERIC\_FAIL  
0201 ( 513) MALLOC\_FAIL  
0202 ( 514) NULL\_POINTER  
0204 ( 516) CODE\_TOBE\_WRITTEN  
0205 ( 517) RESOURCE\_FULL  
0206 ( 518) RESOURCE\_EMPTY  
0207 ( 519) RESOURCE\_DELETED  
0208 ( 520) ACCESS\_DENIED  
0400 ( 1024) TOK\_FILE\_READ\_ERROR  
0401 ( 1025) TOK\_SCRIPTNAME\_TOO\_LONG  
0402 ( 1026) TOK\_PCODE\_WRITE\_OVERRUN  
0403 ( 1027) TOK\_PCODE\_WRITE\_ERROR  
0404 ( 1028) TOK\_UNTERMINATED\_STRING  
0405 ( 1029) TOK\_UNKNOWN\_KEYWORD  
0406 ( 1030) TOK\_NOT\_PRIMARY\_KEYWORD  
0407 ( 1031) TOK\_INCOMPLETE\_STATEMENT  
0408 ( 1032) TOK\_NOT\_ARRAY\_BRACKET  
0409 ( 1033) TOK\_MISSING\_NAME  
040A ( 1034) TOK\_VARIABLE\_NAME\_IS\_KEYWORD  
040B ( 1035) TOK\_VARIABLE\_NAME\_TOO\_LONG  
040C ( 1036) TOK\_ILLEGAL\_NAME  
040D ( 1037) TOK\_UNKNOWN\_VARIABLE\_TYPE  
040E ( 1038) TOK\_VARIABLE\_MISSING  
040F ( 1039) TOK\_FILE\_SEEK\_ERROR  
0410 ( 1040) TOK\_ARRAY\_INDEX\_TOO\_LARGE  
0411 ( 1041) TOK\_INVALID\_CONSTANT\_CHAR  
0412 ( 1042) TOK\_ARRAY\_INDEX\_NEGATIVE  
0413 ( 1043) TOK\_ARRAY\_INDEX\_0\_OR\_1  
0414 ( 1044) TOK\_ARRAY\_NOT\_ALLOWED  
0415 ( 1045) TOK\_MISSING\_CLOSING\_SQUARE\_BRACKET  
0416 ( 1046) TOK\_NOT\_END\_OF\_STATEMENT  
0417 ( 1047) TOK\_UNEXPECTED\_CLOSE\_BRACKET  
0418 ( 1048) TOK\_MISSING\_OPEN\_BRACKET  
0419 ( 1049) TOK\_MISSING\_CLOSE\_BRACKET  
041A ( 1050) TOK\_BRACKET\_MISMATCH  
041B ( 1051) TOK\_UNEXPECTED\_WHITESPACE\_IN\_CONST  
041C ( 1052) TOK\_EMPTY  
041D ( 1053) TOK\_INCOMPLETE\_VAR\_LIST  
041E ( 1054) TOK\_LABEL\_MISSING  
041F ( 1055) TOK\_BUFFER\_TOO\_SMALL  
0420 ( 1056) TOK\_DUPLICATE\_LABEL  
0421 ( 1057) TOK\_NESTED\_ROUTINE\_BLOCK  
0422 ( 1058) TOK\_ORPHAN\_END\_EXIT  
0423 ( 1059) TOK\_NAME\_REDEFINITION  
0424 ( 1060) TOK\_PCODEFILE\_INVALID  
0425 ( 1061) TOK\_HEADER\_POS\_NOT\_AS\_EXPECTED  
0426 ( 1062) TOK\_EMPTY\_EXPRESSION  
0427 ( 1063) TOK\_INVALID\_EXPRESSION\_TYPE  
0428 ( 1064) TOK\_CONSTANT\_OUTOF\_RANGE  
0429 ( 1065) TOK\_TYPE\_MISMATCH  
042A ( 1066) TOK\_VARIABLE\_NOT\_AN\_ARRAY  
042B ( 1067) TOK\_VARIABLE\_IS\_ARRAY  
042C ( 1068) TOK\_TOO\_FEW\_EXPR  
042D ( 1069) TOK\_INVALID\_LINE\_POSITION

# EZURiO

042E ( 1070) TOK\_UNEXPECTED\_OPERATOR  
042F ( 1071) TOK\_EXPRESSION\_INCOMPLETE  
0430 ( 1072) TOK\_NOT\_CONSTANT  
0431 ( 1073) TOK\_CONST\_STACK\_OVERFLOW  
0432 ( 1074) TOK\_EXPRESSION\_NOT\_CONSTANT  
0433 ( 1075) TOK\_UNEXPECTED\_TOKEN\_IN\_EXPR  
0434 ( 1076) TOK\_UNEXPECTED\_OPERATOR\_IN\_EXPR  
0435 ( 1077) TOK\_EXPR\_TOO\_COMPLEX  
0436 ( 1078) TOK\_EXPR\_RPN\_UNDERFLOW  
0437 ( 1079) TOK\_EXPR\_RPN\_OVERFLOW  
0438 ( 1080) TOK\_CONST\_EXPR\_ERROR  
0439 ( 1081) TOK\_ARRINDEX\_OUTOF\_RANGE  
043A ( 1082) TOK\_VAR\_MEMSPACE\_OVERFLOW  
043B ( 1083) TOK\_TOKEN\_IS\_NOT\_SUBROUTINE  
043C ( 1084) TOK\_MISSING\_COMMA  
043D ( 1085) TOK\_MISSING\_EQUAL  
043E ( 1086) TOK\_MISSING\_OPEN\_SQUARE\_BRACKET  
043F ( 1087) TOK\_UNKNOWN\_VARTYPE\_INBUILTIN  
0440 ( 1088) TOK\_VARIABLE\_EXPECTED  
0441 ( 1089) TOK\_INVALID\_STRING  
0442 ( 1090) TOK\_UNEXPECTED\_ARRAY  
0443 ( 1091) TOK\_CANNOT\_END\_IN\_ROUTINE\_BLOCK  
0444 ( 1092) TOK\_BUILTIN\_FNC\_LHS\_NOTALLOWED  
0445 ( 1093) TOK\_FUNCTION\_MUST\_RETURN\_TYPE  
0446 ( 1094) TOK\_TOKEN\_IS\_NOT\_FUNCTION  
0447 ( 1095) TOK\_UNKNOWN\_TOKENISER\_CMD  
0448 ( 1096) TOK\_UNEXPECTED\_TOKEN\_IN\_CMD  
0449 ( 1097) TOK\_INVALID\_TOKENISER\_CMD  
044A ( 1098) TOK\_UNKNOWN\_TOKENISER\_CMDID  
044B ( 1099) TOK\_INVALID\_TOKENISER\_CMDVAL  
044C ( 1100) TOK\_MISSING\_THEN  
044D ( 1101) TOK\_IF\_NESTING\_OVERFLOW  
044E ( 1102) TOK\_IF\_NESTING\_UNDERFLOW  
044F ( 1103) TOK\_OPEN\_CODE\_BLOCK  
0450 ( 1104) TOK\_ROUTINE\_LOCATION  
0451 ( 1105) TOK\_BLOCK\_END\_MISMATCH  
0452 ( 1106) TOK\_INVALID\_EVENT\_NUMBER  
0453 ( 1107) TOK\_UNKNOWN\_EVENTFUNC  
0454 ( 1108) TOK\_INCOMPATIBLE\_ROUTINE  
0455 ( 1109) TOK\_SYNTAX\_ERROR  
0456 ( 1110) TOK\_UNEXPECTED\_BREAK  
0457 ( 1111) TOK\_UNEXPECTED\_CONTINUE  
0458 ( 1112) TOK\_EXPECTED\_TO\_OR\_DOWNT0  
0459 ( 1113) TOK\_EXPECTING\_STEP  
045A ( 1114) TOK\_VARINDEX\_NOT\_ALLOWED  
045B ( 1115) TOK\_LOCAL\_SIMPLE\_FRAME\_OVERFLOW  
045C ( 1116) TOK\_LOCAL\_COMPLEX\_FRAME\_OVERFLOW  
045D ( 1117) TOK\_DIVIDE\_BY\_0\_IN\_CONST\_EXPR  
045E ( 1118) TOK\_EXPECTED\_ENDSUB  
045F ( 1119) TOK\_EXPECTED\_ENDFUNC  
0460 ( 1120) TOK\_EXPECTED\_EXITSUB  
0461 ( 1121) TOK\_EXPECTED\_EXITFUNC  
0462 ( 1122) TOK\_MISSING\_CASE\_ELSE  
0463 ( 1123) TOK\_UNEXPECTED\_CASE  
0464 ( 1124) TOK\_RANGE\_ERROR  
0465 ( 1125) TOK\_ILLEGAL\_PRINT\_WIDTH  
0466 ( 1126) TOK\_DISK\_FULL

# EZURiO

0467 ( 1127) TOK\_FP\_NULL  
0468 ( 1128) TOK\_MISSING\_CASE\_XX  
0469 ( 1129) TOK\_ILLEGAL\_CASE\_RANGE  
046A ( 1130) TOK\_EXPRESSION\_NOT\_TYPED  
046B ( 1131) TOK\_IS\_SUBROUTINE  
046C ( 1132) TOK\_SUBROUTINE\_IN\_EXPR  
0601 ( 1537) RUN\_INDEX\_TOO\_LARGE  
0602 ( 1538) RUN\_DIVIDE\_BY\_0  
0603 ( 1539) RUN\_CONTINUE  
0680 ( 1664) NON-FATAL TAR ERROR = 0x00 (0)  
0681 ( 1665) NON-FATAL TAR ERROR = 0x01 (1)  
0682 ( 1666) NON-FATAL TAR ERROR = 0x02 (2)  
0683 ( 1667) NON-FATAL TAR ERROR = 0x03 (3)  
0684 ( 1668) NON-FATAL TAR ERROR = 0x04 (4)  
0685 ( 1669) NON-FATAL TAR ERROR = 0x05 (5)  
0686 ( 1670) NON-FATAL TAR ERROR = 0x06 (6)  
0687 ( 1671) NON-FATAL TAR ERROR = 0x07 (7)  
0688 ( 1672) NON-FATAL TAR ERROR = 0x08 (8)  
0689 ( 1673) NON-FATAL TAR ERROR = 0x09 (9)  
068A ( 1674) NON-FATAL TAR ERROR = 0x0A (10)  
068B ( 1675) NON-FATAL TAR ERROR = 0x0B (11)  
068C ( 1676) NON-FATAL TAR ERROR = 0x0C (12)  
068D ( 1677) NON-FATAL TAR ERROR = 0x0D (13)  
068E ( 1678) NON-FATAL TAR ERROR = 0x0E (14)  
068F ( 1679) NON-FATAL TAR ERROR = 0x0F (15)  
0690 ( 1680) NON-FATAL TAR ERROR = 0x10 (16)  
0691 ( 1681) NON-FATAL TAR ERROR = 0x11 (17)  
0692 ( 1682) NON-FATAL TAR ERROR = 0x12 (18)  
0693 ( 1683) NON-FATAL TAR ERROR = 0x13 (19)  
0694 ( 1684) NON-FATAL TAR ERROR = 0x14 (20)  
0695 ( 1685) NON-FATAL TAR ERROR = 0x15 (21)  
0696 ( 1686) NON-FATAL TAR ERROR = 0x16 (22)  
0697 ( 1687) NON-FATAL TAR ERROR = 0x17 (23)  
0698 ( 1688) NON-FATAL TAR ERROR = 0x18 (24)  
0699 ( 1689) NON-FATAL TAR ERROR = 0x19 (25)  
069A ( 1690) NON-FATAL TAR ERROR = 0x1A (26)  
069B ( 1691) NON-FATAL TAR ERROR = 0x1B (27)  
069C ( 1692) NON-FATAL TAR ERROR = 0x1C (28)  
069D ( 1693) NON-FATAL TAR ERROR = 0x1D (29)  
069E ( 1694) NON-FATAL TAR ERROR = 0x1E (30)  
069F ( 1695) NON-FATAL TAR ERROR = 0x1F (31)  
06A0 ( 1696) NON-FATAL TAR ERROR = 0x20 (32)  
06A1 ( 1697) NON-FATAL TAR ERROR = 0x21 (33)  
06A2 ( 1698) NON-FATAL TAR ERROR = 0x22 (34)  
06A3 ( 1699) NON-FATAL TAR ERROR = 0x23 (35)  
06A4 ( 1700) NON-FATAL TAR ERROR = 0x24 (36)  
06A5 ( 1701) NON-FATAL TAR ERROR = 0x25 (37)  
06A6 ( 1702) NON-FATAL TAR ERROR = 0x26 (38)  
06A7 ( 1703) NON-FATAL TAR ERROR = 0x27 (39)  
06A8 ( 1704) NON-FATAL TAR ERROR = 0x28 (40)  
06A9 ( 1705) NON-FATAL TAR ERROR = 0x29 (41)  
06AA ( 1706) NON-FATAL TAR ERROR = 0x2A (42)  
06AB ( 1707) NON-FATAL TAR ERROR = 0x2B (43)  
06AC ( 1708) NON-FATAL TAR ERROR = 0x2C (44)  
06AD ( 1709) NON-FATAL TAR ERROR = 0x2D (45)  
06AE ( 1710) NON-FATAL TAR ERROR = 0x2E (46)  
06AF ( 1711) NON-FATAL TAR ERROR = 0x2F (47)

# EZURiO

06B0 ( 1712) NON-FATAL TAR ERROR = 0x30 (48)  
06B1 ( 1713) NON-FATAL TAR ERROR = 0x31 (49)  
06B2 ( 1714) NON-FATAL TAR ERROR = 0x32 (50)  
06B3 ( 1715) NON-FATAL TAR ERROR = 0x33 (51)  
06B4 ( 1716) NON-FATAL TAR ERROR = 0x34 (52)  
06B5 ( 1717) NON-FATAL TAR ERROR = 0x35 (53)  
06B6 ( 1718) NON-FATAL TAR ERROR = 0x36 (54)  
06B7 ( 1719) NON-FATAL TAR ERROR = 0x37 (55)  
06B8 ( 1720) NON-FATAL TAR ERROR = 0x38 (56)  
06B9 ( 1721) NON-FATAL TAR ERROR = 0x39 (57)  
06BA ( 1722) NON-FATAL TAR ERROR = 0x3A (58)  
06BB ( 1723) NON-FATAL TAR ERROR = 0x3B (59)  
06BC ( 1724) NON-FATAL TAR ERROR = 0x3C (60)  
06BD ( 1725) NON-FATAL TAR ERROR = 0x3D (61)  
06BE ( 1726) NON-FATAL TAR ERROR = 0x3E (62)  
06BF ( 1727) NON-FATAL TAR ERROR = 0x3F (63)  
06C0 ( 1728) NON-FATAL TAR ERROR = 0x40 (64)  
06C1 ( 1729) NON-FATAL TAR ERROR = 0x41 (65)  
06C2 ( 1730) NON-FATAL TAR ERROR = 0x42 (66)  
06C3 ( 1731) NON-FATAL TAR ERROR = 0x43 (67)  
06C4 ( 1732) NON-FATAL TAR ERROR = 0x44 (68)  
06C5 ( 1733) NON-FATAL TAR ERROR = 0x45 (69)  
06C6 ( 1734) NON-FATAL TAR ERROR = 0x46 (70)  
06C7 ( 1735) NON-FATAL TAR ERROR = 0x47 (71)  
06C8 ( 1736) NON-FATAL TAR ERROR = 0x48 (72)  
06C9 ( 1737) NON-FATAL TAR ERROR = 0x49 (73)  
06CA ( 1738) NON-FATAL TAR ERROR = 0x4A (74)  
06CB ( 1739) NON-FATAL TAR ERROR = 0x4B (75)  
06CC ( 1740) NON-FATAL TAR ERROR = 0x4C (76)  
06CD ( 1741) NON-FATAL TAR ERROR = 0x4D (77)  
06CE ( 1742) NON-FATAL TAR ERROR = 0x4E (78)  
06CF ( 1743) NON-FATAL TAR ERROR = 0x4F (79)  
06D0 ( 1744) NON-FATAL TAR ERROR = 0x50 (80)  
06D1 ( 1745) NON-FATAL TAR ERROR = 0x51 (81)  
06D2 ( 1746) NON-FATAL TAR ERROR = 0x52 (82)  
06D3 ( 1747) NON-FATAL TAR ERROR = 0x53 (83)  
06D4 ( 1748) NON-FATAL TAR ERROR = 0x54 (84)  
06D5 ( 1749) NON-FATAL TAR ERROR = 0x55 (85)  
06D6 ( 1750) NON-FATAL TAR ERROR = 0x56 (86)  
06D7 ( 1751) NON-FATAL TAR ERROR = 0x57 (87)  
06D8 ( 1752) NON-FATAL TAR ERROR = 0x58 (88)  
06D9 ( 1753) NON-FATAL TAR ERROR = 0x59 (89)  
06DA ( 1754) NON-FATAL TAR ERROR = 0x5A (90)  
06DB ( 1755) NON-FATAL TAR ERROR = 0x5B (91)  
06DC ( 1756) NON-FATAL TAR ERROR = 0x5C (92)  
06DD ( 1757) NON-FATAL TAR ERROR = 0x5D (93)  
06DE ( 1758) NON-FATAL TAR ERROR = 0x5E (94)  
06DF ( 1759) NON-FATAL TAR ERROR = 0x5F (95)  
06E0 ( 1760) NON-FATAL TAR ERROR = 0x60 (96)  
06E1 ( 1761) NON-FATAL TAR ERROR = 0x61 (97)  
06E2 ( 1762) NON-FATAL TAR ERROR = 0x62 (98)  
06E3 ( 1763) NON-FATAL TAR ERROR = 0x63 (99)  
06E4 ( 1764) NON-FATAL TAR ERROR = 0x64 (100)  
06E5 ( 1765) NON-FATAL TAR ERROR = 0x65 (101)  
06E6 ( 1766) NON-FATAL TAR ERROR = 0x66 (102)  
06E7 ( 1767) NON-FATAL TAR ERROR = 0x67 (103)  
06E8 ( 1768) NON-FATAL TAR ERROR = 0x68 (104)

# EZURiO

06E9 ( 1769) NON-FATAL TAR ERROR = 0x69 (105)  
06EA ( 1770) NON-FATAL TAR ERROR = 0x6A (106)  
06EB ( 1771) NON-FATAL TAR ERROR = 0x6B (107)  
06EC ( 1772) NON-FATAL TAR ERROR = 0x6C (108)  
06ED ( 1773) NON-FATAL TAR ERROR = 0x6D (109)  
06EE ( 1774) NON-FATAL TAR ERROR = 0x6E (110)  
06EF ( 1775) NON-FATAL TAR ERROR = 0x6F (111)  
06F0 ( 1776) NON-FATAL TAR ERROR = 0x70 (112)  
06F1 ( 1777) NON-FATAL TAR ERROR = 0x71 (113)  
06F2 ( 1778) NON-FATAL TAR ERROR = 0x72 (114)  
06F3 ( 1779) NON-FATAL TAR ERROR = 0x73 (115)  
06F4 ( 1780) NON-FATAL TAR ERROR = 0x74 (116)  
06F5 ( 1781) NON-FATAL TAR ERROR = 0x75 (117)  
06F6 ( 1782) NON-FATAL TAR ERROR = 0x76 (118)  
06F7 ( 1783) NON-FATAL TAR ERROR = 0x77 (119)  
06F8 ( 1784) NON-FATAL TAR ERROR = 0x78 (120)  
06F9 ( 1785) NON-FATAL TAR ERROR = 0x79 (121)  
06FA ( 1786) NON-FATAL TAR ERROR = 0x7A (122)  
06FB ( 1787) NON-FATAL TAR ERROR = 0x7B (123)  
06FC ( 1788) NON-FATAL TAR ERROR = 0x7C (124)  
06FD ( 1789) NON-FATAL TAR ERROR = 0x7D (125)  
06FE ( 1790) NON-FATAL TAR ERROR = 0x7E (126)  
06FF ( 1791) NON-FATAL TAR ERROR = 0x7F (127)  
0701 ( 1793) RUN\_END  
0702 ( 1794) RUN\_STOP  
0703 ( 1795) RUN\_TOCODE  
0704 ( 1796) RUN\_UNDEFINED\_PCODE  
0705 ( 1797) RUN\_NOT\_DIRECT\_ADDRESSABLE  
0706 ( 1798) RUN\_INVALID\_CONTEXT  
0707 ( 1799) RUN\_CORRUPT\_FILE  
0708 ( 1800) RUN\_UNKNOWN\_VAR\_TYPE  
0709 ( 1801) RUN\_UNKNOWN\_VAR\_SPACE  
070A ( 1802) RUN\_INVALID\_VAR\_OFFSET  
070B ( 1803) RUN\_SUPERSPACE\_MISSING  
070C ( 1804) RUN\_VERSION\_MISMATCH  
070D ( 1805) RUN\_NOGLOBALS\_TO\_PROMOTE  
070E ( 1806) RUN\_SIMPLESTACK\_OVERFLOW  
070F ( 1807) RUN\_SIMPLESTACK\_UNDERFLOW  
0710 ( 1808) RUN\_COMPLEXSTACK\_OVERFLOW  
0711 ( 1809) RUN\_COMPLEXSTACK\_UNDERFLOW  
0712 ( 1810) RUN\_CANNOT\_DEREFERENCE  
0713 ( 1811) RUN\_USER\_RTN\_NEST\_OVERFLOW  
0714 ( 1812) RUN\_USER\_RTN\_NEST\_UNDERFLOW  
0715 ( 1813) RUN\_SIMPLE\_LOCALFRAME\_OVERFLOW  
0716 ( 1814) RUN\_COMPLEX\_LOCALFRAME\_OVERFLOW  
0717 ( 1815) RUN\_TYPE\_MISMATCH  
0718 ( 1816) RUN\_REFTYPE\_EXPECTED  
0719 ( 1817) RUN\_UNKNOWN\_OPERATION  
071A ( 1818) RUN\_UNTESTED\_PCODE  
071B ( 1819) RUN\_CANNOT\_RESUME  
071C ( 1820) RUN\_CACHE\_FAIL  
071D ( 1821) RUN\_FILE\_READ\_ERROR  
071E ( 1822) RUN\_ABORTED\_FILE  
071F ( 1823) RUN\_UNEXPECTED\_END  
0780 ( 1920) FATAL TAR ERROR = 0x00 (0)  
0781 ( 1921) FATAL TAR ERROR = 0x01 (1)  
0782 ( 1922) FATAL TAR ERROR = 0x02 (2)

# EZURiO

0783 ( 1923) FATAL TAR ERROR = 0x03 (3)  
0784 ( 1924) FATAL TAR ERROR = 0x04 (4)  
0785 ( 1925) FATAL TAR ERROR = 0x05 (5)  
0786 ( 1926) FATAL TAR ERROR = 0x06 (6)  
0787 ( 1927) FATAL TAR ERROR = 0x07 (7)  
0788 ( 1928) FATAL TAR ERROR = 0x08 (8)  
0789 ( 1929) FATAL TAR ERROR = 0x09 (9)  
078A ( 1930) FATAL TAR ERROR = 0x0A (10)  
078B ( 1931) FATAL TAR ERROR = 0x0B (11)  
078C ( 1932) FATAL TAR ERROR = 0x0C (12)  
078D ( 1933) FATAL TAR ERROR = 0x0D (13)  
078E ( 1934) FATAL TAR ERROR = 0x0E (14)  
078F ( 1935) FATAL TAR ERROR = 0x0F (15)  
0790 ( 1936) FATAL TAR ERROR = 0x10 (16)  
0791 ( 1937) FATAL TAR ERROR = 0x11 (17)  
0792 ( 1938) FATAL TAR ERROR = 0x12 (18)  
0793 ( 1939) FATAL TAR ERROR = 0x13 (19)  
0794 ( 1940) FATAL TAR ERROR = 0x14 (20)  
0795 ( 1941) FATAL TAR ERROR = 0x15 (21)  
0796 ( 1942) FATAL TAR ERROR = 0x16 (22)  
0797 ( 1943) FATAL TAR ERROR = 0x17 (23)  
0798 ( 1944) FATAL TAR ERROR = 0x18 (24)  
0799 ( 1945) FATAL TAR ERROR = 0x19 (25)  
079A ( 1946) FATAL TAR ERROR = 0x1A (26)  
079B ( 1947) FATAL TAR ERROR = 0x1B (27)  
079C ( 1948) FATAL TAR ERROR = 0x1C (28)  
079D ( 1949) FATAL TAR ERROR = 0x1D (29)  
079E ( 1950) FATAL TAR ERROR = 0x1E (30)  
079F ( 1951) FATAL TAR ERROR = 0x1F (31)  
07A0 ( 1952) FATAL TAR ERROR = 0x20 (32)  
07A1 ( 1953) FATAL TAR ERROR = 0x21 (33)  
07A2 ( 1954) FATAL TAR ERROR = 0x22 (34)  
07A3 ( 1955) FATAL TAR ERROR = 0x23 (35)  
07A4 ( 1956) FATAL TAR ERROR = 0x24 (36)  
07A5 ( 1957) FATAL TAR ERROR = 0x25 (37)  
07A6 ( 1958) FATAL TAR ERROR = 0x26 (38)  
07A7 ( 1959) FATAL TAR ERROR = 0x27 (39)  
07A8 ( 1960) FATAL TAR ERROR = 0x28 (40)  
07A9 ( 1961) FATAL TAR ERROR = 0x29 (41)  
07AA ( 1962) FATAL TAR ERROR = 0x2A (42)  
07AB ( 1963) FATAL TAR ERROR = 0x2B (43)  
07AC ( 1964) FATAL TAR ERROR = 0x2C (44)  
07AD ( 1965) FATAL TAR ERROR = 0x2D (45)  
07AE ( 1966) FATAL TAR ERROR = 0x2E (46)  
07AF ( 1967) FATAL TAR ERROR = 0x2F (47)  
07B0 ( 1968) FATAL TAR ERROR = 0x30 (48)  
07B1 ( 1969) FATAL TAR ERROR = 0x31 (49)  
07B2 ( 1970) FATAL TAR ERROR = 0x32 (50)  
07B3 ( 1971) FATAL TAR ERROR = 0x33 (51)  
07B4 ( 1972) FATAL TAR ERROR = 0x34 (52)  
07B5 ( 1973) FATAL TAR ERROR = 0x35 (53)  
07B6 ( 1974) FATAL TAR ERROR = 0x36 (54)  
07B7 ( 1975) FATAL TAR ERROR = 0x37 (55)  
07B8 ( 1976) FATAL TAR ERROR = 0x38 (56)  
07B9 ( 1977) FATAL TAR ERROR = 0x39 (57)  
07BA ( 1978) FATAL TAR ERROR = 0x3A (58)  
07BB ( 1979) FATAL TAR ERROR = 0x3B (59)

# EZURiO

07BC ( 1980) FATAL TAR ERROR = 0x3C (60)  
07BD ( 1981) FATAL TAR ERROR = 0x3D (61)  
07BE ( 1982) FATAL TAR ERROR = 0x3E (62)  
07BF ( 1983) FATAL TAR ERROR = 0x3F (63)  
07C0 ( 1984) FATAL TAR ERROR = 0x40 (64)  
07C1 ( 1985) FATAL TAR ERROR = 0x41 (65)  
07C2 ( 1986) FATAL TAR ERROR = 0x42 (66)  
07C3 ( 1987) FATAL TAR ERROR = 0x43 (67)  
07C4 ( 1988) FATAL TAR ERROR = 0x44 (68)  
07C5 ( 1989) FATAL TAR ERROR = 0x45 (69)  
07C6 ( 1990) FATAL TAR ERROR = 0x46 (70)  
07C7 ( 1991) FATAL TAR ERROR = 0x47 (71)  
07C8 ( 1992) FATAL TAR ERROR = 0x48 (72)  
07C9 ( 1993) FATAL TAR ERROR = 0x49 (73)  
07CA ( 1994) FATAL TAR ERROR = 0x4A (74)  
07CB ( 1995) FATAL TAR ERROR = 0x4B (75)  
07CC ( 1996) FATAL TAR ERROR = 0x4C (76)  
07CD ( 1997) FATAL TAR ERROR = 0x4D (77)  
07CE ( 1998) FATAL TAR ERROR = 0x4E (78)  
07CF ( 1999) FATAL TAR ERROR = 0x4F (79)  
07D0 ( 2000) FATAL TAR ERROR = 0x50 (80)  
07D1 ( 2001) FATAL TAR ERROR = 0x51 (81)  
07D2 ( 2002) FATAL TAR ERROR = 0x52 (82)  
07D3 ( 2003) FATAL TAR ERROR = 0x53 (83)  
07D4 ( 2004) FATAL TAR ERROR = 0x54 (84)  
07D5 ( 2005) FATAL TAR ERROR = 0x55 (85)  
07D6 ( 2006) FATAL TAR ERROR = 0x56 (86)  
07D7 ( 2007) FATAL TAR ERROR = 0x57 (87)  
07D8 ( 2008) FATAL TAR ERROR = 0x58 (88)  
07D9 ( 2009) FATAL TAR ERROR = 0x59 (89)  
07DA ( 2010) FATAL TAR ERROR = 0x5A (90)  
07DB ( 2011) FATAL TAR ERROR = 0x5B (91)  
07DC ( 2012) FATAL TAR ERROR = 0x5C (92)  
07DD ( 2013) FATAL TAR ERROR = 0x5D (93)  
07DE ( 2014) FATAL TAR ERROR = 0x5E (94)  
07DF ( 2015) FATAL TAR ERROR = 0x5F (95)  
07E0 ( 2016) FATAL TAR ERROR = 0x60 (96)  
07E1 ( 2017) FATAL TAR ERROR = 0x61 (97)  
07E2 ( 2018) FATAL TAR ERROR = 0x62 (98)  
07E3 ( 2019) FATAL TAR ERROR = 0x63 (99)  
07E4 ( 2020) FATAL TAR ERROR = 0x64 (100)  
07E5 ( 2021) FATAL TAR ERROR = 0x65 (101)  
07E6 ( 2022) FATAL TAR ERROR = 0x66 (102)  
07E7 ( 2023) FATAL TAR ERROR = 0x67 (103)  
07E8 ( 2024) FATAL TAR ERROR = 0x68 (104)  
07E9 ( 2025) FATAL TAR ERROR = 0x69 (105)  
07EA ( 2026) FATAL TAR ERROR = 0x6A (106)  
07EB ( 2027) FATAL TAR ERROR = 0x6B (107)  
07EC ( 2028) FATAL TAR ERROR = 0x6C (108)  
07ED ( 2029) FATAL TAR ERROR = 0x6D (109)  
07EE ( 2030) FATAL TAR ERROR = 0x6E (110)  
07EF ( 2031) FATAL TAR ERROR = 0x6F (111)  
07F0 ( 2032) FATAL TAR ERROR = 0x70 (112)  
07F1 ( 2033) FATAL TAR ERROR = 0x71 (113)  
07F2 ( 2034) FATAL TAR ERROR = 0x72 (114)  
07F3 ( 2035) FATAL TAR ERROR = 0x73 (115)  
07F4 ( 2036) FATAL TAR ERROR = 0x74 (116)

# EZURiO

07F5 ( 2037) FATAL TAR ERROR = 0x75 (117)  
07F6 ( 2038) FATAL TAR ERROR = 0x76 (118)  
07F7 ( 2039) FATAL TAR ERROR = 0x77 (119)  
07F8 ( 2040) FATAL TAR ERROR = 0x78 (120)  
07F9 ( 2041) FATAL TAR ERROR = 0x79 (121)  
07FA ( 2042) FATAL TAR ERROR = 0x7A (122)  
07FB ( 2043) FATAL TAR ERROR = 0x7B (123)  
07FC ( 2044) FATAL TAR ERROR = 0x7C (124)  
07FD ( 2045) FATAL TAR ERROR = 0x7D (125)  
07FE ( 2046) FATAL TAR ERROR = 0x7E (126)  
07FF ( 2047) FATAL TAR ERROR = 0x7F (127)  
1000 ( 4096) CFG\_READ\_ERROR  
1001 ( 4097) CFG\_UNKNOWN\_KEY  
1002 ( 4098) CFG\_RECORD\_TOO\_SMALL  
1201 ( 4609) STR\_SIZE\_LIMIT  
1500 ( 5376) TKN\_UNTERMINATED\_STRING  
1501 ( 5377) TKN\_UNEXPECTED\_WHITESPACE\_IN\_CONST  
1502 ( 5378) TKN\_INVALID\_CONSTANT\_CHAR  
1503 ( 5379) TKN\_CONSTANT\_TOO\_BIG  
1600 ( 5632) UWL\_TYPE\_MISMATCH  
1601 ( 5633) UWL\_UNKNOWN\_VAR\_TYPE  
1602 ( 5634) UWL\_UNKNOWN\_VAR\_SPACE  
1700 ( 5888) IMM\_INCOMPLETE\_COMMAND  
1701 ( 5889) IMM\_PARM\_NOT\_STRING  
1702 ( 5890) IMM\_UNEXPECTED\_ARGS  
1703 ( 5891) IMM\_PARM\_NOT\_INTEGER  
1704 ( 5892) IMM\_UNEXPECTED\_TOKEN  
1705 ( 5893) IMM\_CONSTANT\_NOT\_WORD  
1706 ( 5894) IMM\_OUTPUT\_FILE\_EXISTS  
1707 ( 5895) IMM\_SOURCE\_FILE\_MISSING  
1800 ( 6144) FSA\_FILENAME\_INVALID\_CHR  
1801 ( 6145) FSA\_FILENAME\_EMPTY  
1802 ( 6146) FSA\_FILENAME\_UNTERMINATED  
1803 ( 6147) FSA\_FILENAME\_TOO\_LONG  
1804 ( 6148) FSA\_OPENMODE\_NOTALLOWED  
1805 ( 6149) FSA\_MEDIA\_FULL  
1806 ( 6150) FSA\_OPEN\_FAIL  
1807 ( 6151) FSA\_ERASE\_FAIL  
1808 ( 6152) FSA\_NO\_FREE\_HANDLES  
1809 ( 6153) FSA\_FAIL\_OPENFILE  
1900 ( 6400) NVO\_NO\_ANCHOR  
1901 ( 6401) NVO\_SECTOR\_FULL  
1902 ( 6402) NVO\_INVALID\_SEGMENT  
1903 ( 6403) NVO\_DELETED\_OBJECT\_ACCESS  
1904 ( 6404) NVO\_NO\_MORE\_CHILDREN  
1905 ( 6405) NVO\_NO\_MORE\_SIBLINGS  
1906 ( 6406) NVO\_NO\_MORE\_CHANGED  
1907 ( 6407) NVO\_ORPHANED  
1908 ( 6408) NVO\_SEARCH\_FAIL  
1909 ( 6409) NVO\_LINK\_NOT\_EMPTY  
190A ( 6410) NVO\_FIRST\_FILEDATA\_MISSING  
190B ( 6411) NVO\_CHANGE\_NOT\_ALLOWED  
190C ( 6412) NVO\_WRITEOPEN\_TOOMANY  
190D ( 6413) NVO\_OPEN\_TOOMANY  
190E ( 6414) NVO\_FILE\_EXISTS  
190F ( 6415) NVO\_TAGTYPE\_MISMATCH  
1910 ( 6416) NVO\_FILE\_MISSING

# EZURiO

1911 ( 6417) NVO\_INVALID\_FILE\_HANDLE  
1912 ( 6418) NVO\_FILE\_NOTOPEN  
1913 ( 6419) NVO\_FILE\_READOPEN  
1914 ( 6420) NVO\_FILE\_POS\_ERROR  
1915 ( 6421) NVO\_FILE\_TOO\_BIG  
1916 ( 6422) NVO\_FILE\_CORRUPT  
1917 ( 6423) NVO\_OPEN\_FOR\_WRITE  
1918 ( 6424) NVO\_OBJECT\_MISSING  
1919 ( 6425) NVO\_END\_OF\_OBJECT  
191A ( 6426) NVO\_INV\_CONFIG\_ITEM\_ID  
191B ( 6427) NVO\_NO\_CONFIG\_ITEM\_ID  
191C ( 6428) NVO\_UNEXPECTED\_TAGID  
191D ( 6429) NVO\_CONFIG\_ITEM\_CREATE\_FAIL  
191E ( 6430) NVO\_CONFIG\_ITEM\_TOO\_BIG  
191F ( 6431) NVO\_SEGMENT\_CORRUPT  
1920 ( 6432) NVO\_SEGMENT\_FULL  
1921 ( 6433) NVO\_LOC\_NOT\_FREE  
1922 ( 6434) NVO\_NOT\_FIRST\_CHILD  
1A00 ( 6656) FDV\_NOT\_ACCESSIBLE  
1A01 ( 6657) FDV\_WRITE\_FAIL  
1A02 ( 6658) FDV\_WRITE\_BLOCK\_BOUND  
1A03 ( 6659) FDV\_READ\_BLOCK\_BOUND  
1A04 ( 6660) FDV\_SECTOR\_ERASE\_FAIL  
1A05 ( 6661) FDV\_WRITE\_TO\_INV\_ADDR  
1A06 ( 6662) FDV\_READ\_FROM\_INV\_ADDR  
E001 (57345) PARM\_OUT\_OF\_RANGE  
E002 (57346) UNEXPECTED\_PARM  
E003 (57347) NO\_NETWORK\_FOUND  
E004 (57348) NAMED\_NETWORK\_NOT\_FOUND  
E005 (57349) SYNTAX\_ERROR  
E006 (57350) ATTACH\_FAILURE  
E007 (57351) UNKNOWN\_COMMAND  
E008 (57352) NOTHING\_TO\_DETACH  
E009 (57353) INVALID\_FILENAME  
E00A (57354) GENERIC\_ERROR  
E00B (57355) EMPTY\_FILENAME  
E00C (57356) FILENAME\_TOO\_LONG  
E00D (57357) UNKNOWN\_SUBCOMMAND  
E00E (57358) INCORRECT\_MODE  
E00F (57359) ALREADY\_COMPILING  
E010 (57360) MEDIA\_FULL  
E011 (57361) OPEN\_FAIL  
E012 (57362) FILE\_EXISTS  
E013 (57363) ERASE\_FAIL  
E014 (57364) TOKENISE\_FAIL  
E015 (57365) NOT\_COMPILING  
E016 (57366) UNTERMINATED\_ROUTINE  
E017 (57367) UNTERMINATED\_BLOCK  
E018 (57368) FILE\_MISSING  
E019 (57369) OUT\_OF\_HEAPMEM  
E01A (57370) CORRUPT\_FILE  
E01B (57371) NOT\_DIRECT\_ADDRESSABLE  
E01C (57372) FILE\_READ\_ERROR  
E01D (57373) NULL\_POINTER  
E01E (57374) INTEGER\_EXPECTED  
E01F (57375) DBKEY\_UNKNOWN  
E020 (57376) DBKEY\_TRUNCATED

# EZURiO

E021 (57377) CONST\_TRUNCATION  
E022 (57378) CONFIG\_FAIL  
E023 (57379) VAR\_EXPECTED  
E024 (57380) INDEX\_MISSING  
E025 (57381) INDEX\_INVALID  
E026 (57382) VALUE\_EXPECTED  
E027 (57383) TYPE\_MISMATCH  
E028 (57384) VALUE\_OVERFLOW  
E029 (57385) CANNOT\_RESUME  
E02A (57386) NO\_FACTORY\_DEF  
E02B (57387) ABORTED\_FILE  
E02C (57388) FAIL\_OPENFILE  
E02D (57389) UNKNOWN\_PARM  
E02E (57390) CMDBUF\_OVERFLOW  
E02F (57391) STOPPED  
E030 (57392) RUNTIME\_ERROR  
E031 (57393) DISK\_FULL  
E032 (57394) RTN\_NAME\_EXPECTED  
E033 (57395) MISSING\_ARGS  
E034 (57396) ARG\_ERROR  
E035 (57397) REF\_EXPECTED  
E036 (57398) NO\_SCRIPT\_CONTEXT