

# languageONE

## version3

### OVERVIEW

*languageONE* is the both the newest and the oldest of programming languages. By the strictest of definitions it is assembler code, but you would never know that by looking at it. More broadly, it is a set of macros that provide for all the functionality required when programming. And it is more. It provides a framework to support the macros in an assembled program and systematically defines the structures required to manage data and program flow. *languageONE* provides for 64bit coding and can be used "as is" or can be used to facilitate an in depth knowledge of the system, so that the full power of the language can be realised. Unlike some other systems which allow in line assembler *languageONE* is assembler, so if you want to write some assembler code, just do it.

*languageONE* is extensible (able to be extended). Because the language is based on macros you can code a solution, give it a macro name and it then becomes part of the language. Additionally, the existing syntax can be changed to suit your programming style. ie. The delivered syntax allows “\_GT” or “>” in decision making. However, as this is just a predefined token and accessible via an %include file, you are free to modify this to whatever you prefer. More easily, *languageONE* implements the idea of Synonyms.

So..What is a macro

a macro is a single instruction that expands automatically into a set of instructions to perform a particular task. And what makes macros so powerful is that the preprocessor of a macro assembler allows you to code parameters that may differ each time the macro is expanded. Consider the following assembler macro:-

```
%macro $initialise 1-2 ''  
    mov al, %2  
    mov rdi, %1  
    mov rcx,[%1-8]  
    cld  
    rep stosb  
%endmacro
```

this probably doesn't look too familiar but when coded in *languageONE* it would look like:-

*Initialise w\_FirstName*

This is how you might initialise a data item. (example only)

### LINUX/WINDOWS

Version 3.00 separated the Linux and the Windows versions. The Linux version continues to target the NASM assembler, while the Windows version targets MASM.

# languageONE

version3

## THE SYSTEM

*languageONE* is implemented with a number of (o)bject modules that provide functionality to the system. These modules are stored in the languageONE.lib file and must be statically linked with your assembled code. You may think of them as the runtime system for languageONE

## WHERE IS EVERYTHING

### languageONE

pre	commandline pre assemble script [Linux Only]
assemble	commandline assemble script
link	commandline link script
makeONE	commandline make utility
bin	<b>languageONE executables</b>
	[Linux] reWRITER
	[Windows] reWRITER
doc	<b>Documentation</b>
	This manual
	This manual
	This manual
	gnu general public license
	Documentation for the packages system
	Documentation for the packages system
	Documentation for the packages system
	github readme
	Text file of all version history
Html	<b>Html screens/images</b>
include	<b>LanguageONE components</b>
Lib	<b>LanguageONE runtime</b>
Src	<b>LanguageONE source</b>
examples	<b>Example programs</b>
_FLTK	<b>Fltk (gui) programs</b>

# languageONE

## version3

### THE LANGUAGEONE "reWRITER"

#### Overview

*HINT:- It may be more useful to skip this section and come back to it.*

As mentioned above the *languageONE "reWRITER"* is a pre-processor that manipulates text. It is NOT a compiler nor is it an assembler. (A language *rewriter* is usually a program that translates the form of expressions without a change of language). It is in fact written in *languageONE* and can be modified and assembled as any other *languageONE* program can be.

NASM and other assemblers implement a pre-processor that presents certain facilities to make coding easier. As such a program as input to NASM may contain pre-processor statements to direct the pre-processor. ie. %macro and %endmacro are both pre-processor directives. NASM's pre-processor implements some unequalled features but is fixed in the way it works. As an example, NASM uses no brackets other than single line macros. Brackets are almost mandatory in many programming languages and such the *languageONE "reWRITER"* will manipulate their use.

The command line for "*reWRITER*", assembling and linking is as follows:-

makeONE <program name> (no extension – but must be “.ONE”)

The command line for reWriting only is as follows:-

bin/languageONE <program name> (no extension – but must be “.ONE”)

#### "reWRITER" functions:-

Note the terminology used in this manual. Non reWritten code is referred to as 'raw' or 'pure' *languageONE*, while reWritten code is referred to as “cooked” *languageONE*. When describing macros both “raw” and “cooked” syntax will be listed.

#### Continuation Character

NASM/MASM themselves recognise the \ (backslash) as a continuation character and as such is also recognised by *languageONE* (by simply letting it pass through)

#### Line Splitting

NASM/MASM expects one instruction per line. The *languageONE "reWRITER"* will acknowledge the “|” character as a line splitting character and split the lines into their components.

#### Bracketing

All brackets found, other than those on single line macros, will be removed. This gives you a great deal of freedom in the way you wish to code ie.

#### Commas

Commas will be inserted where NASM expects to see them.

# languageONE

## version3

### Synonyms

*languageONE* encompasses the idea of “synonyms”.

Essentially a synonym may be used at a program level to rename any token. This allows you to use the synonym within your program and the "reWRITER" will replace it with it's original value. The synonym directives, BEGIN.SYNONYMS and END.SYNONYMS, and the code they enclose, must be coded as comments (starting the line with a semi-colon). The synonym (coded 1<sup>st</sup>) and the replaced value must be separated by a colon. The structure is as follows:-

```
; BEGIN.SYNONYMS
; Words.          : @WORDS_
; Integers.       : @INTEGERS_
; Numbers.        : @NUMBERS_
; END.SYNONYMS
```

Synonyms may be placed in a separate file and “included” in your code by supplying the path\filename:-

```
; BEGIN.SYNONYMS
; include\LPACK1.SYM
; END.SYNONYMS
```

#### **Note:-**

The file “include\LPACK1.SYM is provided with the languageONE system and is denoted as a language pack. It redefines all the system KEYWORDS to those favoured by the author of languageONE and is coded in all example programs .

### Beginning/End Constructs

BEGIN – END constructs will be counted and reported at the end of the reWriting

[000012] Begin Subs	[000012] End Subs
[000012] Begin Repeats	[000012] End Repeats
[000002] IF's	[000002] End Ifs
[000000] Dot Commands	[000000] Dot Ends
[000000] Begin Test's	[000000] End Tests
[000000] When's	[000000] Wend's
[000008] Open Braces	[000008] Close Braces
[000000] Open SqBrackets	[000000] Close SqBrackets

# languageONE

## version3

### Decimal Places

Where NASM/MASM sees a decimal point it assigns a value based on its assumption that the program will use the Floating Point Unit. *languageONE* does not in fact use the FPU but rather performs fixed point arithmetic. To that end numbers with decimal places will be manipulated by the "reWRITER" to allow for correct processing. Ie:-

@insertnumber Test1,-1.23456,'####,##9.99999-'  
will be transformed into

@insertnumber Test1,-123456,'####,##9.99999-'

@insertnumber Test2,1.234,'##9.99999-'  
will be transformed into

@insertnumber Test2,123400,'##9.99999-'

@insertnumber Test4,12345,'##9.99999-'  
will be transformed into

@insertnumber Test4,1234500000,'##9.99999-'

In addition, where a fixed point number is used as a literal following the begin.instructions directive a picture must be supplied. The "reWRITER" will now provide that picture such that you may code only the fixed point number. So, whereas a "raw" *languageONE* program requires for example

@numbers\_add n\_Destination,{12345,'99.999'}

you may code

@numbers\_add n\_Destination,12.345

### Inference

The "reWRITER" will 'infer' calculations in the following way:-

w\_SomeWord = 'Hello World'

and the "reWRITER" will infer:-

@words\_copy 'Hello World',w\_SomeWord

A = B + 2 \* 80

and the "reWRITER" will infer:-

@integers\_calc A,=,B,+,2,\*,80

A = B + 2.34 \* 67.345

and the "reWRITER" will infer:-

@numbers\_calc A,=,B,+,{234,"9.99"},\*,{67.345,"99.999"}

### Precedence

*languageONE* uses square brackets to denote precedence. You can think of square brackets simply as telling the reWriter to "Do this first"

A = [[[B + 2.34 + 3.45] / 3] + 123.88] \* [5 + [C \* 2.45]]

# *languageONE*

## *version3*

### Table/Array Elements

*languageONE* extends the use of square brackets and in combination with inference can now recognise Table Elements coded in the more traditional format.

`n_Integer[Idx]`

**Note:-** The *languageONE* reWriter cannot infer indexed table elements {ie. `Item[1] = 1`} coded prior to the BIND command.

With the introduction of Arrays, *languageONE* also extends the use of braces and in combination with inference can now recognise Array Elements coded in the more traditional format.

`n_Integer{Idx}`

### WINDOWS

As the raw *languageONE* code for Windows varies in some ways from the raw *languageONE* code for Linux, the reWriter in Version 3.00 has been enhanced in terms of its output. MASM is now the target for Windows, while NASM is retained as the target for Linux. If coding “cooked” code you should include the word “WINDOWS” anywhere in the 1<sup>st</sup> line of your program and MASM syntax code will be produced.

The Synonym language pack provided with release 3.00 aids in maintaining the same syntax for both operating systems when utilising the “cooked” version of each program.

### DATA DEFINITION

Data in *languageONE* is defined in one of two areas. For text strings it is within the Dictionary and for numbers it is within the Matrix.

#### DICTIONARY

The 'dictionary' is the area of the program where words, phrases and sentences may be described. The '**BEGIN.DICTIONARY**' directive is used to inform the system that the dictionary will begin here. The '**END.DICTIONARY**' directive is used to inform the system that the Dictionary is complete.

LINUX	%include "BEGIN.DICTIONARY"
WINDOWS	include <BEGIN.DICTIONARY>

*languageONE* uses the keyword "**@INSERTWORD**" to define strings within the dictionary.

LINUX	@INSERTWORD
WINDOWS	@INSERTWORD

#### example:-

**@Insertword** w\_MyFieldName, 32, 'My Fields Literal'

w\_MyFieldName is the field name used within the *languageONE* program  
32 is the length of the string  
'My Fields Literal' is the initial value given to w\_MyFieldName

**Note:-** that when the given literal is shorter than the defined string length the field will be padded with spaces thus to initialise a blank field you would code:-

**@INSERTWORD** w\_Blanks,64,"" [NASM]  
**@INSERTWORD** w\_Blanks,64," " [MASM] (a space must be coded)

#### alias:-

**@Insertword** {w\_Alias,w\_MyFieldName}, 32, 'My Fields Literal'

An alias may be defined when using the @INSERTWORD macro. Both names may be used throughout your program.

#### Qualifying a string

Wherever a string is used within a program it may be qualified by coding a starting position and the number of characters to reference. It takes the form:-

LINUX	{w_MyFieldName,8,16}
WINDOWS	<<w_MyFieldName,8,16>>

This reads as – starting at character eight take the next 16 characters.

# languageONE

## version3

### MATRIX

The 'matrix' is the area of the program where Numbers may be described. The 'BEGIN.MATRIX' directive is used to inform the system that the Matrix will begin here. The 'END.MATRIX' directive is used to inform the system that the Matrix is complete.

LINUX	%include "BEGIN.MATRIX"
WINDOWS	include <BEGIN.MATRIX>

languageONE uses the keyword "@INSERTNUMBER" to define Numbers within the matrix

LINUX	@INSERTNUMBER
WINDOWS	@INSERTNUMBER

#### example:-

@Insertnumber w\_MyNumber,100

w\_MyNumber      is the field name used within the languageONE program  
100                initialises the number field to 100

@Insertnumber w\_MyNumber,100,'###9-'

w\_MyNumber      is the field name used within the languageONE program  
100                initialises the number field to 100  
'#,#9- '          The numbers 'PICTURE'

The numbers 'picture' describes how the system handles/outputs the number. It comprises a series of characters that define the formatting of the number in a similar way to the BASIC programming language. The following symbols are represented:-

'-' (the minus sign) [Last character]

This defines the number as being signed. The number can represent both positive and negative numbers. It should be the last character of the picture.

'9' This defines a place value for the number.

If there is a zero in this position it will be displayed

'#' This also defines a place value for the number

However, a leading zero in this position will be displayed as a space

',' This defines a thousands indicator and will be displayed when appropriate.

'.' This defines the number as a Fixed Point number.

**Note:-** If a number has no picture, a default 26 character picture will be used. If a number without picture is accepted as input, languageONE will assign a picture developed from that input.

ie. accept=1,234.56- Assigned Picture= 9,999.99-



# languageONE

## version3

### Integers

Unsigned: *All numbers will be considered as signed*

Signed: From -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807  
can be represented as integers.

### Fixed point numbers

Fixed Point numbers are used in *languageONE*. This is different from many programming languages that use floating point numbers. Floating point numbers require a FPU and are not entirely accurate. (Ever had that experience with a calculator where you enter something like  $2 * 2$  and get an answer of 3.999999999)

Fixed point numbers in *languageONE* are integers with an 'implied' decimal place. This is implemented via the picture clause of the insertnumber keyword. By defining a picture like '999.99', you are asking for a number that contains 2 decimal places. A picture of say '#,###,##9.9999-' defines a number with 4 decimal places with a sign displayed (remembering all numbers are treated as signed).

Because all numbers are actually 64 bit integers the following restrictions apply to Fixed Point numbers. Using *languageONE*'s largest integer (9,223,372,036,854,775,807) the following range is encapsulated

.9223372036854775807

This number can define the size of something smaller than an electron

922337203685477580.7

This number can count, in seconds, from the big bang until now and be only half filled. It follows that the largest number that contains the largest no of decimal places is 9999999999.999999999 or 999999999.999999999

### Defining Fixed Point Numbers

Because fixed point numbers are held as an integer with a implied decimal place you must acknowledge the number of decimal places in a given value. That is, given a picture of '999.99' and a required value of 123 you must code 12300,'999.99'. Coding 123,'999.99' implies the number is 1.23

### Literals

If a fixed point number is required for a literal then a picture **MUST** be coded following that literal. ie: {123,'9.99'} equates to 1.23

**Note:-** *this may be handled by the "reWRITER". A pure Fixed Point number may be defined ie A = 1.23 – be aware though that the reWritten version of your program will of had the number converted to its implied Fixed Point value)*

### Setting precedence

square brackets denote precedents for number and these are handled by the "reWRITER". Ie

$A = [ [ [B + 2.34 + 3.45] / 3 ] + 123.88 ] * [ 5 + [C * 2.45] ]$

### alias:-

@Insertnumber {w\_Alias,w\_MyNumber},100,'# ,##9.9'

An alias may be defined when using the @INSERTNUMBER macro. Both names may be used through your program.

### FILES

The 'FILES' section is the area of the program where Files may be described. The 'BEGIN.FILES' directive is used to inform the system that file descriptions will begin here. The 'END.FILES' directive is used to inform the system that the file section is complete.

LINUX	%include "BEGIN.FILES"
WINDOWS	Include <BEGIN.FILES>

Each entry begins with the 'keyword' @INSERTFILE, followed by a file type indicator, a name you choose to refer to the file within the program, and the external name of the file.

LINUX	@INSERTFILE
WINDOWS	@INSERTFILE

#### example:-

Insertfile Delimiter, I-Name, './X-Name'

where Delimiter may be

L,  
c\_LF,  
c\_CSV,  
'?'  
c\_RECORD,  
c\_RANDOM,  
c\_INDEXED,  
c\_DIRECTORY,  
'Your Own Delimiter'

where I-Name is the name used within the program to reference the file  
where './X-Name' is the external name of the file

#### DELIMITERS

##### c\_NULL

Indicates that a NULL value (0x00) is used to delimit logical records. *languageONE* will add this character when writing to the file and strip this character when reading from the file.

##### c\_LF

Indicates that a LINEFEED value (0x0A) or (0x0A,0x0D) is used to delimit logical records. *languageONE* will add this character when writing to the file and strip this character when reading from the file.

##### c\_CSV

Indicates a standard comma delimited file will be used. Alphanumerics will be enclosed in quotation marks, fields will be separated by commas and a LINEFEED character will delimit the logical record

##### '?' Your own character

Any single character enclosed in quotes

### DELIMITERS...cont

#### *c\_RECORD*

Indicates that a RECORD will be used to read from and write to the file (refer next section)

#### *c\_RANDOM*

Indicates that the file can be accessed by record no. RECORDS are implicit in random files.

#### *c\_INDEXED*

Indicates that the file is optimised for use with the Xtables module. Its format echo's the Xtable format in memory, ie there is an 8 byte index field and a 1 byte status field that precedes each record. It is accessed by record no. RECORDS are implicit in indexed files.

#### *c\_DIRECTORY*

Indicates that it is a directory that is being read.

The external name may be replaced by a dictionary word and can be allocated dynamically by simply setting it equal to name. ie.

[in the file section] @Insertfile c\_LF,IN\_File

[in the code section] IN\_File = "External File Name"

### File Status

When accessing files *languageONE* provides a 'STATUS' field. It is defined as FileName\_STATUS. Thus the status field for the file A01\_File would be *A01\_File\_STATUS*. This field can be checked each time a file is accessed.

Present values are:-

Constant	Value	Meaning
EOF	10	End Of File
INVALIDKEY	23	Invalid Key
DUPLICATE	22	More than 1 record exists for this key
LOCKED	90	Record or File is locked

### File Size

Additionally, when a file has been opened, *languageONE* provides a 'SIZE' field. It is defined as FileName\_SIZE. Thus the size field for the file A01\_File would be *A01\_File\_SIZE*.

# languageONE

## version3

### RECORDS

A record is a collection of elements, typically in fixed number and sequence and typically indexed by serial numbers or identity numbers. The elements of records may also be called fields or members.

Records provide for structuring data in a logical way. They are usually, but not always, used to describe data in a file and thus provide clearer code when defined with the file in question. Each entry in a file is described as a record. Records are also mandatory in *languageONE* tables. (refer next section)

Each record entry starts with **@BEGIN\_RECORD** keyword. It is followed by a numeric value defining the length of the record, and then the record name. Each record terminates with the **@END\_RECORD** keyword followed by the record name. Fields can then be defined within the BEGIN and END of the record by using **@Insertword** and **@insertnumber**.

LINUX	@BEGIN_RECORD
WINDOWS	@BEGIN_RECORD

#### Record Description

##### **@begin\_record** record\_length,A01\_Record

```
@insertnumber A01_No1,      00, '99999'  
@insertnumber A01_No2,      00, '99999'  
@insertnumber A01_No3,      00, '99999'  
@insertnumber A01_Word1,    10, ' '  
@insertnumber A01_Word2,    10, ' '
```

##### **@end\_record** A01\_Record

The files module must be informed of the record length and it is Parameter 1 or the **@BEGIN\_RECORD** keyword that accomplishes this.

#### Record No

When a record is used for File access, *languageONE* provides a Record Number which may be used to access records in a random manner rather than sequentially [one after the other]. This access method is termed 'relative' or 'random'. The Record Number is the record name followed by '\_NO', so in the case above the record no field will be **A01\_Record\_NO**.

### TABLES

Tables are defined in *languageONE* as structured internal storage. They are treated very similarly to Files but unlike files must ALWAYS have a record associated with them and thus are almost synonymous with random access files. This allows file records to be read into storage quite efficiently. You insert a table by defining its size (Record Length) times No Of Records that you require.

The 'TABLES' section is the area of the program where tables may be described. The '**BEGIN.TABLES**' directive is used to inform the system that Table descriptions will begin here. The '**END.TABLES**' directive is used to inform the system that the definitions are complete.

LINUX	%include "BEGIN.TABLES"
WINDOWS	Include <BEGIN.TABLES>

#### Insert Table

@Inserttable WorkTable, RecordLength \* NoOfRecords

Inserting a table defines the table name and the overall size of the table, defined as the record length times the number of records. Before a table can be used it must be bound.

#### Bind Table

LINUX	@TABLES_BIND
WINDOWS	@TABLES_BIND

@Tables\_bind WorkTable,A01\_Record,NoOfRecords

While the @INSERTTABLE keyword tells *languageONE* about the overall size of the table, @TABLES\_bind, binds a record to the table and defines the dimensions. It is coded within the body of the program (following the %include BEGIN.INSTRUCTIONS)

Of course Tables/Arrays can have more than 2 dimensions. *languageONE* allows up to 9 dimensions (you count the record as the 1st dimension.). They are defined by first INSERTing a TABLE with the dimensions detailing the size and then by binding a record to the table.

ie @TABLES\_bind WorkTable2,A01\_Record,3,4

If you use an analogy like pages in a book you could say that the record defines the words across the page, the 3 defines the lines down the page and the 4 defines the pages. You can expand this analogy with an extra dimension defining books. So the index 3,4,5 would define 3 books, 4 pages, and 5 lines leaving the record to define the words across the line.

@Inserttable WorkTable2,RecLength\*3\*4\*5

@Tables\_Bind WorkTable,A01\_Record,3,4,5

Tables are read or written by providing the required index(s) numbers in the get and put operations. Refer further ahead.

**Note:-** The *languageONE* reWriter cannot infer indexed table elements {ie. Item[1] = 1} coded following to the BIND command.

### XTABLES

XTables are large tables that have been optimised for use with indexed files, they may however be used for any other purpose. They are single dimension tables with memory being allocated at runtime.

#### InsertXTable

@InsertXTable LargeTable

Inserting an XTable defines the table name. Before a table can be used it must be bound.

#### Bind Xtable

LINUX	@XTABLES_BIND
WINDOWS	@XTABLES_BIND

@XTables\_bind LargeTable,Record,Size

While the @INSERTXTABLE keyword tells *languageONE* the xtables name, @XTABLES\_bind, binds a record to the table and defines the tables size. It is coded within the body of the program (following the BEGIN.INSTRUCTIONS)

**NOTE:-** An XTABLE contains an extra 9 bytes at the front of each record to hold a Delete Flag and an Index and must be taken into account when defining the xTable size.

Size = NoOfRecords\*(RecordLength + 9)

**Note:-** The *languageONE* reWriter cannot infer indexed table elements {ie. Item[1] = 1} coded prior to the BIND command.

### ARRAYS (Integer)

An Array – introduced in Version V2.10 - is a contiguous no of integers (all numbers are represented as qwords in languageONE) with memory being allocated at runtime. They do not have the control block normally associated with a languageONE data structure and therefore must be accessed via the [ARRAYS.LIB](#) package. As of version V3.01 multi-dimension arrays may be coded.

A new Quick Sort algorithm has been coded for Arrays in order to optimise the sort speed of integers. Only single-dimension arrays may be sorted. The '[TABLES](#)' section is the area of the program where Arrays may be described.

#### InsertArray

LINUX	<a href="#">@InsertArray</a>
WINDOWS	<a href="#">@InsertArray</a>

[@InsertArray](#) *ArrayName,6,12,16*

### SIMPLE INTEGER ARITHMETIC

There are 6 simple arithmetic commands in *languageONE* that are not part of the *NUMBERS.o* object package but available to every *languageONE* program. It must be noted though that they have an almost 1 to 1 relationship to their underlying assembler equivalents and are not error checked in anyway.

If you add 1 to 9,223,372,036,854,775,807 you will get -9,223,372,036,854,775,808.

However, if you are careful, they are the best way to use integers

#### Equals

LINUX	@integers_eq w_MyNumber,123
WINDOWS	@integers_eq w_MyNumber,123
COOKED	w_MyNumber = 123

#### Addition

LINUX	@integers_add w_MyNumber,123
WINDOWS	@integers_add w_MyNumber,123
COOKED	w_MyNumber = w_MyNumber + 123
COOKED(Shorthand)	w_MyNumber += 123

#### Subtraction

LINUX	@integers_sub w_MyNumber,123
WINDOWS	@integers_sub w_MyNumber,123
COOKED	w_MyNumber = w_MyNumber - 123
COOKED(Shorthand)	w_MyNumber -= 123

#### Multiplication

LINUX	@integers_mul w_MyNumber,123
WINDOWS	@integers_mul w_MyNumber,123
COOKED	w_MyNumber = w_MyNumber * 123
COOKED(Shorthand)	w_MyNumber *= 123

#### Division

LINUX	@integers_div w_MyNumber,123
WINDOWS	@integers_div w_MyNumber,123
COOKED	w_MyNumber = w_MyNumber / 123
COOKED(Shorthand)	w_MyNumber /= 123

**Note:-** that for this integer function, the remainder will be lost.



# languageONE

## version3

### Calculation

LINUX	@integers_calc w_MyNumber,=,128,*,64,-,32
WINDOWS	@integers_calc w_MyNumber,=,128,*,64,-,32
COOKED	w_MyNumber = 128 * 64 - 32

Applies each operation to the destination field working from left to right of the source integers.

### Setting precedents

The order of an integer operation may be directed by way of square brackets. As an example:-  $A = [(8 + B) * 3] + 5$  will direct languageONE to evaluate the  $[8 + B]$  1<sup>st</sup>, multiply the result by 3 2<sup>nd</sup> and finally add 5 to the answer. The "reWRITER" will enable this by developing the operation one by one within a macro and apply those values to the final calculation. The original line of code will be commented and replaced by the system developed macro name.

In essence square brackets say to languageONE, do this 1<sup>st</sup>.

### LOGICAL EXPRESSIONS

There are 3 logical expressions in *languageONE* that are not part of the *NUMBERS.o* object package but available to every *languageONE* program. It must be noted though that they have an almost 1 to 1 relationship to their underlying assembler equivalents and are not error checked in anyway.

#### Logical And

LINUX	@integers_and w_MyNumber,Number/Literal
WINDOWS	@integers_and w_MyNumber,Number/Literal

#### Logical Or

LINUX	@integers_or w_MyNumber,Number/Literal
WINDOWS	@integers_or w_MyNumber,Number/Literal

#### Logical Xor

LINUX	@integers_xor w_MyNumber,Number/Literal
WINDOWS	@integers_xor w_MyNumber,Number/Literal

# languageONE

## version3

### CONSTANTS

are a way of giving a name to a value. It is good programming practise and will save you a lot of headaches further on. As an example a record length may be used in several places and so by replacing the value with the constant `c_RecordLength` the record length can be changed in one spot (at its definition) and be correct in every use of the constant.

### Linux

Constants are defined by the use of the `%define` directive.

#### examples:-

```
%define c_RecordLength 47
%define c_NoOfRecords 1024
%define c_Size 47*1024
%define c_Size (47 * 1024) + 88
%define c_Size c_RecordLength*c_NoOfRecords
```

### Windows

Define constants with the `EQU` and precede the constant with a “%” (percentage sign) when used following its definition.

#### examples:-

```
c_RecordLength EQU 47
c_NoOfRecords EQU 1024
c_Size EQU 47 * 1024
c_Size EQU (47 * 1024) + 88
c_Size EQU %c_RecordLength
c_Size EQU %(c_RecordLength*c_NoOfRecords)
```

**note:-** When 2 constants are used together bracket them and use only a single % sign

### Inline Assembler

There is no package required to include assembler code within a *languageONE* program. As noted *languageONE* IS assembler, but for the reWRITER to ignore assembler code in should be enclosed within “@BEGIN\_ASSEMBLER” and “@END\_ASSEMBLER” keywords.

```
@begin_assembler (BeginAssembler with language Pack 1)
push rax
pop rbx
@end_assembler (EndAssembler with language Pack 1)
```

may be coded anywhere following the `%include 'BEGIN.INSTRUCTIONS'` directive. Data should be coded following the correct use of section directives.

LINUX	Section .data @insertword w_Hello, 5,"Hello" Section .text
WINDOWS	.data @insertword w_Hello, 5,"Hello" .code

# *languageONE*

*version3*

## LANGUAGEONE RUNTIME SYSTEM

### Introduction

*languageONE* delivers functionality via keywords. Each keyword is an assembler macro that is used to manage the parameters and make a call to one of the *languageONE* object modules. (In this way, if a particular object module is not required you will not have to link it in). Keywords are case-Insensitive such that @CURSOR, @cursor or even @CuRsOr are acceptable forms of coding.

ARRAYS.LIB	Array processing services
COMMON.LIB	Common/General processing services
DECISIONS.LIB	Decision making services
FILES.LIB	File Handling Services
NUMBERS.LIB	Fixed point number services
STDIO.LIB	Standard Console services
TABLES.LIB	Table processing services
WORDS.LIB	AlphaNumeric processing services
WWW.LIB	Network processing services
XTABLES.LIB	Xtable processing services

**Note:-** the languageONE object modules are delivered in a library name [languageONE.lib](#) and it is this that is linked to your program.

# languageONE

## version3

### COMMON.LIB

The **COMMON.LIB** module provides generalised routines for *languageONE*.

#### Get the current date

LINUX	@Date_Get
WINDOWS	@Date_Get

@date\_get noofdays,datestring

Returns the current number of days and date

noofdays is an integer value defined as the number of days from the linux epoch (1970/01/01).  
datestring is a 10 character word containing a date formatted as CCYY/MM/DD

Number of Days is useful in the processing of past or future dates. A @date\_get will return the no of days associated with the current date. By adding/subtracting a numeric value to the no of days and then performing a @date\_datefromdays a future or past date can be calculated.

#### Date from Days

LINUX	@Date_DateFromDays
WINDOWS	@Date_DateFromDays

@date\_datefromdays noofdays,datestring

Returns the date associated with the No Of Days passed.

#### Days from Date

LINUX	@Date_DaysFromDate
WINDOWS	@Date_DaysFromDate

@date\_dayefromdate noofdays,datestring

Returns the No Of Days associated with the Date passed.

This function should be used when validating a date. If the date is invalid then the system field ERROR\_CODE will be set to 1

# languageONE

## version3

### Seconds

LINUX	@Date_Seconds
WINDOWS	@Date_Seconds

@date\_seconds seconds

@InsertNumber Seconds, 0,'999999999999999999999999'

Returns the no of seconds from the linux epoch (1970/01/01).

@date\_seconds is useful in performing “stop-watch” functions. Store the no of seconds prior to starting a process and subtract it from the no of seconds at the process end to produce an elapsed time.

### Timer

LINUX	@Date_Timer
WINDOWS	@Date_Timer

@date\_timer seconds,milliseconds

@InsertNumber seconds, 0,'999999999999999999'

@InsertNumber milliseconds, 0,'.999999'

@date\_timer is useful in performing “stop-watch” functions that require more accuracy than @date\_seconds

### Run (Batched)

LINUX	@Run
WINDOWS	@Run

@Run Command,ResponseFileHandle

This keywords allows a program to run another program. Filename must be either a binary executable, or a script. Note that in this release a fully qualified name ie “path/program” must be provided. Ie “/usr/bin/nasm”

The macro returns a system variable, CHILD\_PID defining the new running process's 'Process Identification Number'

The macro returns a FileHandle that duplicates the child process's STDOUT and STDERR. This file should not be opened by the calling process but must be closed by it. Read this file to collect the called programs output

# languageONE

## version3

### Run (Interactive)

LINUX	@Run
WINDOWS	@Run

This keywords allows a program to run an interactive program (generally a terminal). Note that in this release a fully qualified name ie “path/program” must be provided. Ie “/usr/bin/xterm”

**NOTE:-** languageONE will decide whether a BATCH or an INTERACTIVE call is being made by noting the number of parameters that are passed.

### Wait

LINUX	@Wait
WINDOWS	@Wait

#### @Wait ProcessId

This keyword, followed by a Process Identification Number, will suspend your program until the program identified by integer has completed

#### [WINDOWS]

When running a program on a windows system, it has been found that buffers for StdOut are NOT cleared when StdOut is closed or the sub-ordinate program terminates. In the absence of a setbuf or fflush call being available the following work around may be used:-

Define a file

@insertfile c\_LF,STDOUT

Set the handle to StdOut (Note that the file does not have to be opened)

@begin\_assembler

push qword[StdOutHandle]

pop qword[STDOUT\_HANDLE]

@end\_assembler

Write to this file rather than using display statements

@files\_write STDOUT,'Your text goes here'

### STDIO.LIB

The **STDIO.LIB** object module provides input/output routines for the Linux terminal. Fields used in a STDIO call may be literals, words from the dictionary and numbers from the matrix. The following keywords are supported:-

#### Cursor

LINUX	@Cursor
WINDOWS	@Cursor

**@Cursor** row,column

positions the cursor on the screen detailed by the row and column parameters. ie.

@Cursor 03,05

**Note:-** cursor sets the values of the system variable c\_Cursor. You must perform a “@display c\_Cursor” for the cursor to be positioned.

#### Display

LINUX	@Display
WINDOWS	@Display

**@Display** field,field,field,field,....

Displays the listed fields on the screen. Any number of fields may be passed. A useful field to use here could be the predefined field LF. This is the linefeed character and when used as the last field of the display keyword will trigger a linefeed following the last field listed.

#### Display Line

LINUX	@Display_Line
WINDOWS	@Display_Line

**@Display** field,field,field,field,....

Displays the listed fields on the screen and follows each one with the linefeed character. It is the equivalent of coding “display field,LF,field,LF,field,LF...”

# languageONE

## version3

### Display at

LINUX	@Display_At
WINDOWS	@Display_At

@Display\_At 05,06,"Hello World"

Displays the listed field on the screen at the specified row and column position. It is equivalent of coding: "cursor 03,02" followed by "display c\_Cursor,field". **Note:-** it makes no sense to display more than one field

### Acceptline

LINUX	@Acceptline
WINDOWS	@Acceptline

@Acceptline w\_Input

Accepts the listed field on the screen from the current cursor position. Only a single field at a time can be accepted. The field is terminated when the <ENTER> key is pressed.

### Acceptline.at

LINUX	@Acceptline_at
WINDOWS	@Acceptline_at

@Acceptline\_At 05,06,w\_Input

Accepts the listed field on the screen at the specified row and column position. It is equivalent of coding: "cursor 03,02" followed by "acceptline c\_Cursor ,field"

### Accept at

LINUX	@Acceptt_at
WINDOWS	@Accept_at

@Accept\_At 05,06,w\_Input

Accepts the listed field on the screen at the specified row and column position. This differs from the other accept keywords in that it restricts the number of characters entered to the size of the receiving field. It also reports any function key that has been pressed. It is useful when full screen applications are being developed.



### Screen Attributes

The following screen attributes may be “display”ed to manage a terminal.

#### Terminal Colours

##### Foreground Background

v_BlackFG	v_WhiteFG		v_BlackBG	v_WhiteBG
v_DarkGreyFG	v_LightGreyFG		v_DarkGreyBG	v_LightGreyBG
v_RedFG	v_LightRedFG		v_RedBG	v_LightRedBG
v_GreenFG	v_LightGreenFG		v_GreenBG	v_LightGreenBG
v_YellowFG	v_LightYellowFG		v_YellowBG	v_LightYellowBG
v_BlueFG	v_LightBlueFG		v_BlueBG	v_LightBlueBG
v_MagentaFG	v_LightMagentaFG		v_MagentaBG	v_LightMagentaBG
v_CyanFG	v_LightCyanFG		v_CyanBG	v_LightCyanBG
v_DefaultFG			v_DefaultBG	

##### Attributes Graphics Characters

v_Bold	v_ResetBold		v_BottomRight	v_TopRight
v_Dim	v_ResetDim		v_BottomLeft	v_TopLeft
v_Underlined	v_ResetUnderlined		v_LeftMiddle	v_RightMiddle
v_Blink	v_ResetBlink		v_BottomMiddle	v_TopMiddle
v_Reverse	v_ResetReverse		v_Cross	v_Line
v_Hidden	v_ResetHidden		v_Bar	
v_ResetAll	v_ClearScreen			

# *languageONE*

## *version3*

Returned by @Accept=At

Constant	Value
c_RETURN	000
c_FUNCTIONKEY1	001
c_FUNCTIONKEY2	002
c_FUNCTIONKEY3	003
c_FUNCTIONKEY4	004
c_FUNCTIONKEY5	005
c_FUNCTIONKEY6	006
c_FUNCTIONKEY7	007
c_FUNCTIONKEY8	008
c_FUNCTIONKEY9	009
c_FUNCTIONKEY10	010
c_FUNCTIONKEY11	011
c_FUNCTIONKEY12	012
c_ALT	013
c_ARROWUP	014
c_ARROWDOWN	015
c_ARROWRIGHT	016
c_ARROWLEFT	017
c_END	018
c_HOME	019
c_INSERT	020
c_ENDOFFIELD	021
c_ESCAPE	027
c_BACKSPACE	127
c_NONE	999

### WORDS.LIB

The **WORDS.LIB** module provides string handling routines. They function from left (source) to right (destination). Although there is “\_copy” keyword, a copy is always done when a second field is coded, such that you may code- “words.uppercase w\_Name” to change the characters in w\_Name to upper case and you may also code “words.uppercase w\_Name1,w\_Name2 to copy the w\_Name1 to w\_Name2 and change w\_Name2 to upper case. In this example w\_Name1 would be left unchanged. The following keywords are supported:-

#### Copy

LINUX	@Words_Copy
WINDOWS	@Words_Copy

#### @Words\_Copy Source,Destination

This will copy the number of characters held in Sourcefield to Destinationfield. Reminder: If the Sourcefield is shorter than the Destination field, NO padding will take place. Note that although *languageONE* will accept “@words\_copy w\_Name”, it does not make sense to do so.

#### Pad

LINUX	@Words_Pad
WINDOWS	@Words_Pad
COOKED	Word1 = Word2

#### @Words\_Pad Source,Destination

Performs the same function as COPY but will pad a longer Destinationfield with spaces.

The sourcefield may be a numeric field and it is this routine that may be used to convert numbers to alphanumerics.

#### Uppercase

LINUX	@Words_Uppercase
WINDOWS	@Words_Uppercase

#### @Words\_Uppercase Source,Destination(Optional)

Will copy the Sourcefield to the Destination field and convert the Destinationfield to all upper case characters. Coding “@Words\_uppercase Sourcefield” will convert the Sourcefield to all upper case characters

### Lowercase

LINUX	@Words_Lowercase
WINDOWS	@Words_Lowercase

@Words\_Lowercase Source, Destination(Optional)

Will copy the Sourcefield to the Destination field and convert the Destination field to all Lower case characters. Coding "@Words\_Lowercase Sourcefield" will convert the Sourcefield to all Lower case characters

### Insert

LINUX	@Words_Insert
WINDOWS	@Words_Insert

@Words\_Insert Source, Destination

Inserts the Sourcefield into the Destinationfield by moving Destination characters to the right. This works well when qualified. ie- @words\_insert "RET",{Destinationfield,5} says insert the word "RET" into the Destination field starting at the 5th character

### Find

LINUX	@Words_Find {"text",Counter},"Biblical Text"
WINDOWS	@Words_Find <<"text",Counter>>,"Biblical Text"

Locates the phrase "text" in the literal and returns the character position in counter

### Replace

LINUX	@Words_Replace {"ret","rog"},Word1
WINDOWS	@Words_Replace <<"ret","rog">>,Word1

Locates all occurrences of "RET" in word1 and replaces them with "ROG"

### Environment

LINUX	@Words_Environment
WINDOWS	@Words_Environment

@Words\_Environment DestinationField

A special WORDS function that will return the command line parameters and store them in Destination field

### Length

LINUX	@Words_Length
WINDOWS	@Words_Length

@Words\_Length Word,Length

A special WORDS function that will return the length of a word

### String to record

LINUX	@Words_StringToRecord
WINDOWS	@Words_StringToRecord

@Words\_StringToRecord String, Record

A WORDS function that will populate a record from a string. This is required because fields are managed by a control structure and records being a collection of fields are managed by multiple control structures

### Record to string

LINUX	@Words_RecordToString
WINDOWS	@Words_RecordToString

@Words\_RecordToString Record,String

A WORDS function that will populate a string from a record. This is required because fields are managed by a control structure and records being a collection of fields are managed by multiple control structures

### NUMBERS.LIB

The **NUMBERS.LIB** module provides fixed point number handling routines. They function from right (source) to left (destination). Note that although **NUMBERS.LIB** can handle integers, the 6 inbuilt integer functions (**@integer\_eq**, **@integer\_add**, **@integer\_sub**, **@integer\_mul**, **@integer\_div**, **@integer\_calc**) are more efficient. Remembering the **integer** functions have no error correction it is best to use **NUMBERS.LIB** for integers when range boundaries are expected to be crossed.

Unsigned: All numbers will be considered as signed

Signed: From -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807

#### Equals

LINUX	@Numbers_eq n_No, {123,"9.99}
WINDOWS	@Numbers_eq n_No, <<123,"9.99">>
COOKED	n_No = {123,"9.99}

@Numbers\_eq n\_Dest,n\_Src

Sets the value of Destinationfield to Sourcefield. The sourcefield in this case may be an alphanumeric field and it is this routine that may be used to covert alphanumerics to numbers

#### Addition

LINUX	@Numbers_add n_No, {123,"9.99}
WINDOWS	@Numbers_add n_No, <<123,"9.99">>
COOKED	n_No = n_No + {123,"9.99}

@Numbers\_add n\_Dest,n\_Src

Adds the value of Sourcefield to Destinationfield

#### Subtraction

LINUX	@Numbers_sub n_No, {123,"9.99}
WINDOWS	@Numbers_sub n_No, <<123,"9.99">>
COOKED	n_No = n_No - {123,"9.99}

Subtracts the value of Sourcefield from Destinationfield

#### Multiplication

LINUX	@Numbers_mul n_No, {123,"9.99}
WINDOWS	@Numbers_mul n_No, <<123,"9.99">>
COOKED	n_No = n_No * {123,"9.99}

Multiplies the Destinationfield by the Sourcefield

# languageONE

## version3

### Division

LINUX	@Numbers_div n_No, {123,"9.99}
WINDOWS	@Numbers_div n_No, <<123,"9.99">>
COOKED	n_No = n_No / {123,"9.99}

Divides the Destinationfield by the Sourcefield

### Overloading

A feature of the NUMBERS.LIB package is that these macros may be “overloaded”, meaning that more than 1 Sourcefield may be coded. ie:-

Numbers.add Destinationfield,2,4,16,32

This would add 2 then 4 then 16 and then 32 (Total of 54) to the Destinationfield

This can be a useful in many ways. Take for example cubing a number.

With the following defined in the matrix

Insertnumber w\_no1, 3

Insertnumber w\_no2, 3

You can overload the multiplication to effect a cubing of the number

numbers.mul (w\_no1,w\_no2,w\_No2)

### Calculation

LINUX	@Numbers_calc n_No,={123,"9.99"},+,1.23,-,n_No2
WINDOWS	@Numbers_calc n_No,=<<123,"9.99">>,+,1.23,-,n_No2
COOKED	n_No = n_No + {123,"9.99"} + 1.23 - n_No2

Applies each operation to the destination field working from left to right of the source numbers (Number precedence will be followed if square brackets are used).

### Random

LINUX	@Numbers_Random n_No
WINDOWS	@Numbers_Random n_No

Returns a random number in the range 0 thru 65,536.

**Note:** *that this function is not intended to return a random number that can be relied upon for robust applications. It merely takes the number of nano-seconds, splits it into 2 16bit words, xor's the 2 and delivers the result. By definition it has the range of 0 thru 65,536. You may restrict its range by ANDing it with a mask ie: to obtain a number between 0 and 6 (ie a throw of a dice) ADD the returned value by 0110B. If a more “industrial strength” random no is required it can be obtained in:*

*Linux - investigate the /dev/<u>random file system*

*Windows - refer CryptoAPI*

# languageONE

## version3

### DECISIONS.LIB

The **DECISIONS.LIB** module provides for decision making in *languageONE*. In reality it performs one function, that of a compare. The many manifestations of that are handled by the macros themselves.

The following table describes operation equivalences. Only Constants are allowed in Windows but the **reWRITER** will handle conversion from the literal [=,!>, etc] to the Constant [EQ,\_NEQ,etc]

Equals	_EQ	=
isNOTEqualTo	_NEQ	!=
isLessThan	_LT	<
isNOTLessThan	_NLT	!<
isGreaterThan	_GT	>
isNOTGreaterThan	_NGT	!>
In	_IN	
Not In	_NIN	

You are free to add to this table by editing **DECISIONS.INC** and describing your own operations name

**Note:** for the decisions module:-

RAW	As Shown
COOKED	No Commas required

### IF statements

**@IF** w\_no1,=,w\_no2

Do Something

**@ELSE**

Do Something Else

**@END.IF**

This is a fairly standard If statement and is no different from most languages. There is no elseif statement in languageONE because it is easily handled. ie.

**@IF** w\_no\_1,=,w\_no2

Do Something

**@ELSE**

**@IF** w\_Alpha,=,"A"

Do Something

**@ELSE**

Do Something Else

**@END.IF**

**@END.IF**



### IF statements...cont

Equally, you could use the line splitting character

```
@IF w_no_1 = w_no2
    Do Something
@ELSE | @IF w_Alpha = "A"
    Do Something
@ELSE
    Do Something Else
@END.IF
@END.IF
```

### @\_IF, @\_OR, @\_AND, @\_END

*Compound Ifs are defined as those that contain '.or' or '.and' statements. They must begin with a '.if' and be terminated with a '.end'*

In order to understand the .AND and the .OR statement you must consider how *languageONE* functions. An IF statement will set a flag as either TRUE or FALSE. Subsequently an .OR statement will proceed if the flag is so to FALSE (no need to do anything if the previous IF returned a TRUE) and set the same flag to either TRUE or FALSE. Similarly an .AND statement will proceed if the flag is set to TRUE (no need to do anything if the previous IF returned a FALSE) and set the same flag to either TRUE or FALSE.

You must be careful with this as *languageONE* has no way (at present) of bracketing OR's and AND's (remember brackets are simply removed) and cannot evaluate something like:-  
IF (A=B AND C=D) OR (A=C AND (B=D OR E=F))

The following will achieve the same thing. It's just a little bit cumbersome at present. This will be addressed in a later release.

```
@_IF A = B
@_AND C = D
@_END
    Do Something
@END.IF

@_IF B = D
@_OR E = F
@_AND A = C
@_END
    Do the same Thing
@END.IF
```

# languageONE

## version3

### the **=IN** operator

The **\_IN** operator can be used to interrogate lists. Although it is simply a compound “if” it allows for perhaps cleaner coding. Note though that fixed point numbers cannot be coded with the **\_IN** operator as this would result in a double set of braces.

```
@IF w_No1 _IN {1,2,4,8,16,31,64} [YES]  
@IF w_No1 _IN {{123,'9.99},{456,'9.99}} [NO]
```

### the **=NIN** operator

The **\_NIN** operator can also be used to interrogate lists. It does in fact invoke the **\_IN** decision and then negates the answer. ie.

```
@IF w_No1 _NIN {1,2,4,8,16,31,64}
```

### TEST statements

This is a fairly standard statement seen in most languages.

In COBOL it is EVALUATE,

In PASCAL it is CASE OF,

In C it is SWITCH/CASE,

In BASIC it is SELECT CASE etc etc.

It takes the following forms in *languageONE*.

```
@BEGIN.TEST w_No1  
  @WHEN < 1  
    Do Something  
  @WEND  
  
  @WHEN < 2  
    Do Something  
  @WEND  
  
  @OTHERWISE  
    Do Something Else  
@END.TEST
```

```
@BEGIN.TEST c_TRUE  
  @WHEN A = B  
    Do Something  
  @WEND  
  
  @WHEN C > D  
    Do Something  
  @WEND  
  
  @WHEN E != D  
    Do Something  
  @WEND  
@END.TEST
```

# languageONE

## version3

### TEST statements...cont

#### @\_WHEN, @\_OR, @\_AND, @\_END

*Compound Whens* are defined as those that contain '.or' or '.and' statements. They must begin with a '.when' and be terminated with a '.wend'

#### @BEGIN.TEST c\_TRUE

@\_WHEN A = B

@\_AND X = Y

@\_END

Do Something

@WEND

@\_WHEN C > D

@\_AND J = K

@\_END

Do Something

@WEND

@\_WHEN E != D

@\_OR Z = W

@\_END

Do Something

@WEND

@OTHERWISE

Do Something else

#### @END.TEST

\_IN and \_NIN may be used when testing for TRUE

#### @BEGIN.TEST c\_TRUE

@WHEN \_IN {1,2,3,4,5}

Do Something

@WEND

@WHEN \_NIN {1,2,3,4,5}

Do Something Else

@WEND

#### @END.TEST

# languageONE

## version3

### LOOPS

There is no module providing functionality for looping. It is built into the macros themselves as they are only a comparative jump statement. They take 2 forms:-

**@repeat\_if**

**@repeat\_while**

It should be noted that **@repeat\_if** and **@repeat\_while** are functionally equivalent. They are both provided to suit programming style.

**@Repeat\_if** <condition>

Do Something

**@end\_repeat**

A **@repeat\_if** statement has a built in condition which is evaluated *at the top of the loop*.

#### NOTE:-

*When multiple conditions are needed you may use the system supplied **exitRepeat** field. This flag is always tested at the top of the loop and an exit performed if it is found to be true. Note that any code following the exit will still be executed so it is advisable to code an else to allow your instructions to execute correctly.*

*Note that **@\_OR** & **@\_AND** are not suitable for loops. (This will be revisited in a later version)*

#### NOTE:-

the **@EXIT\_REPEAT** macro has been modified to set the exitRepeat flag to true.

**@repeat\_while** 1 = 1

**@if** B = 2

**@EXIT\_REPEAT**

**@else**

Do Something

**@end.if**

**@end\_repeat**

### LOOPS....cont

#### @repeat\_for

```
@Repeat_for <counter>,<start>,<stop>,<step>
Do Something
@end_repeat
```

A @repeat\_for statement has a built in count function which is evaluated at the top of the loop. You provide the counter, the starting value, the ending value and the step (if other than 1)

When multiple conditions are needed you may use the system supplied `exitRepeat` field. This flag is always tested at the top of the loop and an exit performed if it is found to be true. Note that any code following the exit will still be executed so it is advisable to code an else to allow your instructions to execute correctly. *Note that .OR & .AND are not suitable for loops. (This will be revisited in a later version)*

```
@Repeat_for Ctr,1,128,2
  @if A = 1
    @EXIT_REPEAT
  @else
    Do Something
  @end.if
@end_repeat
```

#### **Note:-**

when the start value is less than the stop value then the loop will subtract the step value from the counter. You will be counting backwards.

When the start value is equal to the stop value the loop will add to the step value unless a -ve step value is coded

# languageONE

## version3

### SUB ROUTINES

#### Overview

Using Sub Routines is the common programming practice of breaking up programs into smaller and more manageable tasks. They start with the 'keyword' **@BEGIN\_SUB** followed by a unique name that you choose. They end with the 'keyword' **@\_END\_SUB** followed by the same unique name you have chosen. They may also contain an **@EXIT\_SUB** (followed by the same unique name) to exit the sub routine depending on your coding decisions. To invoke a Sub Routine you use the 'keyword' **@CALL** followed by the Sub Routines name. Sub Routines may invoke other Sub Routines.

When running under a Linux operating system, selecting the debug option for the reWriter will cause the program to output a message to STDOUT when entering or exiting a subroutine.

#### Passing parameters

Version 3.01 developed the use of numeric parameters for Subroutines and Functions. It is implemented by the **@CALL** and the **@USING** macros. The Call is coded as follows:-

RAW	@Call A-SubRoutine,n_Num,3
COOKED	@Call A-SubRoutine n_Num,3

Note that in the raw version of *languageONE* a comma is required after the Subroutine name.

The **@CALL** macro must be matched by a **@USING** macro immediately following the **@BEGIN\_SUB** macro, as follows:-

```
@BEGIN_SUB A-SubRoutine
  @USING {_arg1,'99.99'},{_arg2,'9'}
```

each argument should be enclosed in braces and include a picture definition. As of version 3.01 the datanames are NOT local to the subroutine and must be unique within the program.

# languageONE

## version3

### FUNCTIONS

#### Boolean Functions

Boolean Functions are available in *languageONE*. They start with the 'keyword' `@BEGIN_FUNCTION` followed by a unique name that you choose. They end with the 'keyword' `@END_FUNCTION` followed by the same unique name you have chosen. They may also contain an `@EXIT_FUNCTION` (followed by the same unique name) to exit the function depending on your coded decisions.

Boolean functions are automatically set to return `FALSE` while a `TRUE` result may be indicated by setting `RETURN_CODE` to `TRUE`.

They maybe coded after the following keywords:-

`@IF – @OR – @AND – @REPEAT_IF (@REPEAT_WHILE) – @WHEN`

ie:-

```
@if valid_date
  display “A valid date has been entered”
@end.if
```

When running under a Linux operating system, selecting the debug option for the reWriter will cause the program to output a message to STDOUT when entering or exiting a subroutine.

#### Numeric Functions

Version 3.01 expanded functions to return numeric values and to be passed parameters (refer to SubRoutines above). When coding a numeric function you should include a picture following the function name:-

```
f_Squared,'999999'
```

In this way a field called `df_Squared`, using the picture that has been coded, will be created. For examples:-

```
@BEGIN_FUNCTION f_Squared,'999999'
  @USING {_arg1,'999'}
  @INTEGERS_EQ df_Squared,_arg1
  @INTEGERS_MUL df_Squared,_arg1
@END_FUNCTION
```

RAW	@FUNCTION I,f_Squared,{3,'9'}
COOKED	I = f_Squared(3)

Note that I will be populated with `df_Squared` in the above example

RAW	@IF {f_IsNumeric,3}
COOKED	@IF {f_IsNumeric,3}

Note that in an IF/AND/OR statement the function should set `RETURN_CODE` to true or false

### FILES.LIB

The **FILES.LIB** module provides file access routines for languageONE. It manages both sequential, random, indexed and directory access. The following keywords are supported:-

#### Open

A file must be opened before it is able to be accessed. (and closed after it is not needed). This is performed by the keyword

**@files\_open I\_Name, Action** is used to perform this action, where Actions consist of:-

- \$read** The file will be available for reading only
- \$write** The file will be available for writing to only
- \$readwrite** The file will be available for both reading and writing

When a file is opened write or readwrite you may select starting the write operations from the beginning of the file (thereby overwriting the file) or at the end of the file (appending records to the file). This is achieved by adding (with a + sign) the keywords “**\$beginning**” or “**\$end**”. This would look like:-

**@files\_open WorkFile1,\$write+\$beginning**

A file may be opened for exclusive use. This is done by adding (with a + sign) the keyword “**\$lock**”. This would look like

**@files\_open WorkFile1,\$write+\$beginning+\$lock**

#### Read

There are two forms of the read keyword depending on if you are reading into a record or a series of fields. They are:-

- **@files\_read I\_Name, record, <optional record no(for files of records)>**
  - ie: **@files\_read File, Record,Record\_No** will yield the same result
  - as: **@integers\_eq Record\_No,1**
  - @files\_read File,Record**
- **@files \$read, I\_Name, field, field, field.....**

#### Write

There are two forms of the write keyword depending on if you are writing a record or writing a series of fields. They are:-

- **@files\_write I\_Name, record, <optional record no(for files of records)>**
  - ie: **@files\_write File, Record,Record\_No** will yield the same result
  - as: **@integers\_eq Record\_No,1**
  - @files\_write File,Record**
- **@files \$write, I\_Name, field, field, field.....**

#### Delete

Deletion may only take place when files are accessed randomly (with a record). The command takes the



# languageONE

## version3

form:-

@Files\_delete I\_Name, record,<optional record no(for files of records)>

and is managed by virtue of the Record No associated with the record. I\_Name\_NO

Although *languageONE* will delete the record regardless, it is always good practise to perform a read operation first. This make it easier to assist any user who's intention it is to delete the record.

ie: @files\_delete File, Record, Record\_No will yield the same result

as: @integers\_eq Record\_No,1

@files\_delete File, Record

**NOTE:-** Records are never physically deleted from a file. *languageONE* reserves the 1<sup>st</sup> character in a randomly organised file and maintains its value as either:-

- 0x01 represents a “live” record
- 0x00 represents a deleted record

These records may be restored by editing the file and altering this value to 0x01

**NOTE:-** Xtables contains a **LOAD** and **UNLOAD** function which makes simple the process of purging deleted records.

### Close

Files must be closed when your program terminates. The command takes the form:

@files\_close I\_Name

### Copy

The files module provides a very thin wrapper that allows in kernel copying of files. This is a more efficient method of copying entire files as it is all done within the kernel (as opposed to user space)

@files\_copy ExternalNameFrom, ExternalNameTo

**Note:-** Use Dictionary words.

@insertword FileName1, 09,{ 'External Name',00h}

@insertword FileName2, 09,{ 'External Name',00h}

Replcae braces in the above if coding in raw Windows format

### Rename

The files module provides a very thin wrapper that allows for file renaming.

@files\_rename ExternalNameFrom, ExternalNameTo

**Note:-** Use Dictionary words.

@insertword FileName1, 09,{ 'External Name',00h}

@insertword FileName2, 09,{ 'External Name',00h}

Replcae braces in the above if coding in raw Windows format

# languageONE

## version3

### Remove

The files module provides a very thin wrapper that allows for file deletion.

`@files_remove ExternalName`

**Note:-** Use Dictionary words.

`@insertword FileName, 09, {'External Name', 00h}`

Replcae braces in the above if coding in raw Windows format

### Chdir

This changes the current directory to that of DirectoryName. Relative or Absolute.

`@files_chdir DirectoryName`

**Note:-** Use Dictionary words.

`@insertword DirectoryName, ??, {'External Name', 00h}`

Replcae braces in the above if coding in raw Windows format

### Getcwd

`@files_getcwd DirectoryName`

Returns an absolute path of the current work directory

## RANDOM ACCESS FILES

The following are commands that are available to random and indexed access files only.

### Start

Because files may contains deleted records and holes a start command will return the 1<sup>st</sup> valid record and its record number. By initially providing the record number you will determine the starting position. The command takes the form:

`@files_start I_Name, record, <optional record no>`

is managed by virtue of the Record No associated with the record. `I_Name_NO`

ie: `@files_start File, Record, Record_No` will yield the same result

as: `@integers_eq Record_No, 1`

`@files_start File, Record`

**NOTE:-** If you create a random access file with 2 records (1 & 3) then a “Hole” will exist in the file. That is a vacant spot in the file containing a record of nulls.

### Next

Following a start, the next routine will return the next valid record (excluding deleted records and holes). The command takes the form:

`@files_next I_Name, record`

### **DIRECTORIES**

The following are commands that are available when reading a directory.

#### **Open**

A directory is given in the insertfile macro or may be dynamically assigned during program procedures.

#### **Read**

Directories are read sequentially from start to end. Note that the order of files/directories returned may not be consistent or as you may expect. The following [RETURN\\_CODE](#) may be interrogated following this call

- when = 00 : [Linux only] Unknown Entry
- when = 01 : [Linux only] FIFO Entry
- when = 02 : [Linux only] Character Device
- when = 04 : Directory
- when = 06 : [Linux only] Block Device
- when = 08 : File
- when = 10 : [Linux only] Symbolic Link
- when = 12 : [Linux only] Socket
- when = 14 : [Linux only] WhiteOut

#### **Close**

### **RECORD LOCKING**

Record locking is performed slightly differently in the Windows version of [languageONE](#) as opposed to the Linux version.

[Linux](#) : Advisory file and record locking is used to coordinate independent processes. Files and records are not actually locked but there is an agreement between processes that each process will adhere to the protocol. When a process wishes to write a record it is obliged to read the record, lock it, do the write and then unlock it. A second process that attempts to lock the record will receive a warning from the lock function, however reads and writes will continue to function. It is the application logic that supports the locking function.

[Windows](#) : Mandatory locking is where the I/O subsystem enforces the locking protocol. A locked record will return a file status of LOCKED whenever an application attempts to read or write that record.

It would have been possible to replicate mandatory locking within [languageONE](#) for Linux but advisory locking is the preferred protocol. It allows read access where required while leaving the write access to the application. The result of the implementation of locking within [languageONE](#) means that a Windows program would need to be coded slightly differently from a Linux version, however it is possible to do the coding such that the Windows version would perform as expected on a Linux system (but not visa-versa).

### **Lock**

Only files of records can be locked.

[@Files\\_lock I\\_Name, record, <optional record no>](#)

is managed by virtue of the Record No associated with the record. [I\\_Name\\_NO](#)

ie: [@files\\_lock File, Record,Record\\_No](#) will yield the same result

as: [@integers\\_eq Record\\_No,1](#)

[@files\\_lock File, Record](#)

### **Unlock**

Only files of records can be unlocked.

[@Files\\_unlock I\\_Name, record, <optional record no>](#)

is managed by virtue of the Record No associated with the record. [I\\_Name\\_NO](#)

ie: [@files\\_unlock File, Record,Record\\_No](#) will yield the same result

as: [@integers\\_eq Record\\_No,1](#)

[@files\\_unlock File, Record](#)

### TABLES.LIB

Tables, as described in previous parts of this manual, could be visualised as internal representations of random files (although they are more than that). They are however, apart from a need to open or close files, similar in the fact that they must be constructed of records and these records are read (rget) and written (rput). The commands take the following form:

#### Bind

`@tables_bind WorkTable, A01_Record, Index1, Index2, Index3....`

this keyword has been described previously in this manual

#### Rget

`@tables_rget WorkTable, Record_No/Index`

loads the record defined by the record no/index from the table

#### Rput

`@tables_put WorkTable, Record_No/Index`

places the record defined by the record no/index into the table

#### Fget

RAW	<code>@tables_fget WorkTable,FieldNo,FieldName,Index,Index</code>
COOKED	<code>A = FieldName[Index]</code>

loads the field defined by the field No from the table, where

- WorkTable is the name of the table
- FieldNo is the field position in the record
- FieldName is the name of the recipient field. It does NOT have to be the corresponding field in the record but must have identical properties.
- Index,Index,Index...the indexes that define the records (containing the field) location in the table

#### Fput

RAW	<code>@tables_fput WorkTable,FieldNo,FieldName,Index,Index</code>
COOKED	<code>FieldName[Index] = A</code>

stores the field defined by the field No from the table, where

- WorkTable is the name of the table
- FieldNo is the field position in the record
- FieldName is the name of the sending field. It does NOT have to be the corresponding field in the record but must have identical properties. Literals are acceptable as long as they also have identical properties.
  - Numeric literals. You need to provide a picture over-ride to match the receiving fields size. ie. {123,'999'}
  - Alpha fields. Field must be the same size as that defined by the record.
- Index,Index,, the indexes that define the records (containing the field) location in the table

# languageONE

## version3

### Sort

@tables\_sort WorkTable, StartPosition, NoOfCharacters

**NOTE:** Only single dimension tables may be sorted

**NOTE:** A sorted table that is not entirely populated may not return the desired result when searched.

### Search

@tables\_search WorkTable, StartOfKey, EndOfKey, KEY, Index

returns the record associated with the provided key along with the Index used to locate the record. If more than one record exists, the 1<sup>st</sup> record will be returned and the **STATUS** field will be set to **DUPLICATES**

**NOTE:** Only single dimension tables may be searched

**NOTE:** A sorted table that is not entirely populated may not return the desired result when searched.

Like files, when accessing tables *languageONE* provides a 'STATUS' field. It is defined as **TableName\_STATUS**. Thus the status field for the table **A01\_Table** would be **A01\_Table\_STATUS**. This field can be checked each time a file is accessed.

Present values are:-

22(DUPLICATES) = More than one table element exists for this key

23(INVALIDKEY) = Invalid Key

*languageONE* maintains a field named **TableName\_UBOUND** which stores the location of the highest slot within a table. It may be accessed but it must be remembered that **UBOUND** works in only a single dimension. That is, if you have a table indexed as 2,2,2 then if the table is full, **UBOUND** would contain a value of 8.

### XTABLES.LIB

Xtables, are tables on steroids, or more precisely, a optimised facility to work with very large files. On a functional level, they may be used in place of an ISAM (Indexed Sequential Access Manager). By sorting on a particular key and searching for values on that key, they are fast enough to enable applications to be written without an **ISAM**.

A new file type has been defined, that of **c\_INDEXED**, to host xtables on disk. The Indexed file is a direct representation of an **XTABLE** in memory.

**NOTE:-** That XTABLES, being bent more toward file access, are only single dimension

**NOTE:-** tables. This does not mean, however, that they cannot be used for other purposes.

While Tables are defined at the time of assembly, Xtables are dynamic, that is, memory is allocated at runtime

#### Bind

@xtables\_bind LargeTable, LargeRecord, Size

must be used prior to any other xtable operation. It allocates memory for the table as defined by the Size parameter and associates a record with the table.

*NOTE: Remember that an XTABLE contains an extra 9 bytes at the front of each record to hold a Delete Flag and an Index and must be taken into account when defining the xTable size.*

*Size = NoOfRecords\*(RecordLength + 9)*

#### Rget

@xtables\_rget argeTable, Record\_No

loads the record defined by the record no from the table

*NOTE: If a sort has been performed on the table, obtaining a record based solely on its record number will most times not get the record you are expecting. In order to identify your records location in the table, a search must be performed to return the correct record no.*

#### Rput

@xtables\_rput LargeTable, Record\_No)

places the record defined by the record no into the table.

*NOTE: If the table is being loaded manually, it is your responsibility to ensure that the tables upper boundary (tablename UBOUND) is large enough to encompass the record no being added.*

*NOTE: If a sort has been performed on the table, the record no used to replace the table record must be the no returned from a search operation.*

### Fget

RAW	@xtables_fget WorkTable,FieldNo,FieldName,Index
COOKED	A = FieldName[Index]

loads the field defined by the field No from the table, where

- LargeTable is the name of the table
- FieldNo is the field position in the record
- FieldName is the name of the recipient field. It does NOT have to be the corresponding field in the record but must have identical properties.
- Index is the index that defines the records (containing the field) location in the table

### Fput

RAW	@xtables_fput WorkTable,FieldNo,FieldName,Index
COOKED	FieldName[Index] = A

stores the field defined by the field No from the table, where

- LargeTable is the name of the table
- FieldNo is the field position in the record
- FieldName is the name of the sending field. It does NOT have to be the corresponding field in the record but must have identical properties. Literals are acceptable as long as they also have identical properties.
  - Numeric literals. You may need to provide a picture over-ride to match the receiving fields size. ie. {123,'999'}
  - Alpha fields. Field must be the same size as that defined by the record.
- Index is the index that defines the records (containing the field) location in the table

### Delete

@xtables\_delete LargeTable, Record\_No, Initialise

marks the record associated with the record number for deletion. Setting the Initialise field to TRUE will fill the record with NULLS while FALSE will leave the record intact.

It is dependent on the application to handle these deleted records. ie. set the IncludeDeletedRecords to c\_FALSE on any unload or handle what is written to file within your code.

**NOTE:-** If a deleted record is initialised you will not be able successfully search the xtable being that it will contain search keys of 00h



# languageONE

## version3

### Load

@xtables\_load LargeTable, LargeFile, IncludeDeletedRecords

*Although load may be used with random access files,* it has been optimised to perform best with indexed files. As indexed files are a direct representation of an xtable, *languageONE* can “swallow” the entire file with a single read. This is opposed to the reading of each record when coupled with a random file. Note though that when loading an Indexed file, the IncludeDeletedRecords indicator is ignored.

### Unload

@xtables\_unload LargeTable, LargeFile, IncludeDeletedRecords)

Unlike the load function, unload must write each record individually in order to maintain any sort that may have been preformed during processing of the table. IncludeDeletedRecords functions as would be expected for unload.

### Sort

@xtables\_sort LargeTable, StartPosition, NoOfCharacters, Divisor

xtables uses an optimised Quick Sort.

### Search

@xtables\_search LargeTable, StartOfKey, EndOfKey, KEY, Idx)

returns the record associated with the provided key along with the Index used to locate the record. If more than one record exists, the 1<sup>st</sup> record will be returned and the **STATUS** field will be set to **DUPLICATES**

### General

Like files, when accessing xtables *languageONE* provides a 'STATUS' field. It is defined as **TableName\_STATUS**. Thus the status field for the table A01\_Table would be **A01\_Table\_STATUS**. This field can be checked each time a file is accessed.

Present values are:-

22(**DUPLICATES**) = More than one table element exists for this key

23(**INVALIDKEY**) = Invalid Key

*languageONE* maintains a field named **TableName\_UBOUND** which stores the location of the highest slot within a table. It may be accessed directly.

### ARRAYS.LIB

Arrays, as described in previous parts of this manual, are a contiguous run of qwords (8 bytes). They do not have the normal control block that other languageONE data items have and therefore data must be transferred to and from the Array. Arrays are useful when speed is required and to this end an optimised Quick Sort has been implimented.

#### Get

RAW	@Arrays_Get ArrayName, Idx,Idx, Value
COOKED	Value = ArrayName{Idx,Idx}

returns the array element - located by the Index's - to Value

#### Put

RAW	@Arrays_Put ArrayName,Idx,Idx, Value
COOKED	ArrayName{Idx,Idx} = Value

stores the data – value - in the Array element located by Index's

#### Eq

RAW	@Arrays_Eq ArrayName1,Idx,Idx,ArrayName2,Idx,Idx
COOKED	ArrayName1{Idx,Idx} = ArrayName1{Idx,Idx}

stores the data from array2 element in the array1 element

#### Swap – Note the double equals

RAW	@Arrays_Eq ArrayName1,Idx,Idx,ArrayName2,Idx,Idx
COOKED	ArrayName1{Idx,Idx} == ArrayName1{Idx,Idx}

swaps the data between array2 element and array1 element

#### Add

RAW	@Arrays_Add ArrayName1,Idx1,Idx2,ArrayName2,Idx1,Idx2 @Arrays_Add ArrayName1,Idx1,Idx2,Dataname @Arrays_Add ArrayName1,Idx1,Idx2,Numeric Literal
COOKED	ArrayName1{Idx,Idx} += ArrayName2{Idx,Idx} ArrayName1{Idx,Idx} += Dataname ArrayName1{Idx,Idx} += Numeric Literal

add the value to array1 element

# languageONE

## version3

### Sub

RAW	@Arrays_Sub ArrayName1,Idx1,Idx2,ArrayName2,Idx1,Idx2 @Arrays_Sub ArrayName1,Idx1,Idx2,Dataname @Arrays_Sub ArrayName1,Idx1,Idx2,Numeric Literal
COOKED	ArrayName1 {Idx,Idx} -= ArrayName2 {Idx,Idx} ArrayName1 {Idx,Idx} -= Dataname ArrayName1 {Idx,Idx} -= Numeric Literal

subtracts the value from array1 element

### Mul

RAW	@Arrays_Mul ArrayName1,Idx1,Idx2,ArrayName2,Idx1,Idx2 @Arrays_Mul ArrayName1,Idx1,Idx2,Dataname @Arrays_Mul ArrayName1,Idx1,Idx2,Numeric Literal
COOKED	ArrayName1 {Idx,Idx} *= ArrayName2 {Idx,Idx} ArrayName1 {Idx,Idx} *= Dataname ArrayName1 {Idx,Idx} *= Numeric Literal

Multiplies array1 element by the value

### Div

RAW	@Arrays_Div ArrayName1,Idx1,Idx2,ArrayName2,Idx1,Idx2 @Arrays_Div ArrayName1,Idx1,Idx2,Dataname @Arrays_Div ArrayName1,Idx1,Idx2,Numeric Literal
COOKED	ArrayName1 {Idx,Idx} /= ArrayName2 {Idx,Idx} ArrayName1 {Idx,Idx} /= Dataname ArrayName1 {Idx,Idx} /= Numeric Literal

Divides array1 element by the value

### If

RAW	@Arrays_IF ArrayName1,Idx1,Idx2,=,ArrayName2,Idx1,Idx2 @Arrays_IF ArrayName1,Idx1,Idx2,=,Dataname @Arrays_IF ArrayName1,Idx1,Idx2,=,Numeric Literal
COOKED	IF ArrayName1 {Idx,Idx} = ArrayName2 {Idx,Idx} IF ArrayName1 {Idx,Idx} = Dataname IF ArrayName1 {Idx,Idx} = Numeric Literal

Compares array1 element to the value

### **Sort**

[@Arrays\\_sort ArrayName](#)

An optimised Quick Sort of the Array.

Only single dimension Arrays can be sorted correctly

# languageONE

version3

## WWW.LIB

*languageONE* manages a graphical user interface by, dare I say, “leverageing” web technology. In fact a *languageONE* program supplying a graphical experience to a user is a simple HTML server. In this way it may be accessed locally thru a browser (URL being localhost:portNo) or across either a local or remote network (URL being IPAddress:portno).

Socket technology is also used for (I)nter (P)rocess (C)ommunication. This will be described in the following section.

The program may be defined in 3 simple statements.

[@www\\_open PortNo](#)

[@www\\_process screen, responsefield, GET\\_Subroutine, POST\\_Subroutine](#)

[@www\\_close PortNo](#)

and that is it, in its entirety. In fact a complete web site may be managed by the above 3 statements. Of course a great deal of the work is done by the front end that is described by the HTML and perhaps Javascript that may or may not be coded.

V1.15.ONE and onwards will demonstrate the way that *languageONE* can implement a graphical user interface, while V2.02 demonstrates a full blown application, V2.03 demonstrates that same application with record locking and V2.05 demonstrates the same application but built with a separate “file server” program.

The *languageONE* program takes a slightly different take on CGI.

Wiki describes CGI as:-

*In computing, Common Gateway Interface (CGI) offers a standard protocol for web servers to interface with executable programs running on a server that generate web pages dynamically. Such programs are known as CGI scripts or simply as CGIs; though usually written in a scripting language, they can be written in any programming language.*

In regard to the above statement *languageONE* nominates the executable program as taking the lead and supplies the HTML server as a secondary process. By supplying the [POST](#) and [GET](#) call back address's in the [@www\\_process](#) call the *languageONE* program can manage the interface from the back end.

It is important to understand the *languageONE* demonstration programs, V1.15.ONE and V1.16.ONE in addition to the HTML and Javascript associated with these programs, to fully appreciate the end to end process that has been adopted by *languageONE*. It is also taken for granted that everyone nowadays understands web technologies.

**NOTE:-** WWW.LIB provides a “*passthru*” where a ResponseField populated by your application will be returned directly to a browser. This mechanism is controlled by the [RETURN\\_CODE](#) field. Initialising it with the length of the ResponseField tells [WWW.LIB](#) to do the passthru as opposed to the more likely returning of a html document.

V2.02.ONE, V2.03.ONE and V2.05.ONE demonstrate this process.

# *languageONE*

## *version3*

### **Open**

@www\_open portno

This is the 1st call required in a *languageONE* GUI program. This call establishes a connection between a *languageONE* program and the outside world (be it a local connection via a "localhost:portno" URL or a "IPaddress:portno" URL across a network (local or otherwise). It is the first call that a *languageONE* GUI program makes in order to interface with a user in a graphical environment.

### **Process**

@www\_process Screen,ResponseField, GET\_SubRoutineName, POST\_SubRoutineName

This is the 2<sup>nd</sup> call required in a *languageONE* GUI program. It defines the screen name that begins a conversation, a response field that will be passed back to a calling program, the name of a sub routine that will be called by **WWW** when a **GET** is received from the front end and the name of a sub routine that will be called by **WWW** when a **POST** is received from the front end.

It is the *languageONE* application program that interprets the response field and manages the process.

**NOTE:-** At the minimum, the GET Routine should initialise *RETURN\_CODE* & *ERROR\_CODE*

BEGIN.SUB GET\_SubRoutineName

RETURN\_CODE = 0 ; > zero tells the server this program provides the response

ERROR\_CODE = 0 ; Only set ERROR\_CODE if you want the server to terminate

END.SUB GET\_SubRoutineName

### **Close**

@www\_close (portno)

This is the 3<sup>rd</sup> and final call required in a *languageONE* GUI program. This call closes the previously established connection and terminates the process.

### **(I)nter (P)rocess (C)ommunication**

Inter-process communication(IPC) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing.

This is managed in *languageONE* by the [WWW.LIB](#) module. Messages can be sent between processes running on a single machine or multiple machines running on a network (including the internet).

#### **Client**

[@www\\_sendmsg](#)

[@www\\_SendMsg w\\_IP\\_Address, w\\_PortNo, w\\_Msg](#)  
[@www\\_SendMsg "127.0.0.1", "1024", "Hello World"](#)

This command sends a message to the machine described by the socket <IP/Port>.

If your application requires a response it is the applications responsibility to enable that. One way of doing this may be to define a response socket to the server within you message. Ie [www.SendMsg w\\_IP\\_Address, w\\_PortNo, "192.168.0.4 1025 Hello World"](#)

#### **Server**

[@www\\_rcvmsg](#)

[@www\\_RecvMsg w\\_IP\\_Address, w\\_PortNo, w\\_Msg](#)  
[@www\\_RecvMsg "127.0.0.1", "1024", w\\_Msg](#)

The receive message call sets the server into a waiting state where it will listen for any incoming messages. Once it has received one it will return the message to the application program. Note that the server listens for ALL messages on the described port, NOT just one from your application.

### APPENDIX A.

#### How to implement a list in languageONE.

Lists are not particularly part of the language but can be implemented as follows.

In the DICTIONARY, insert a word:

```
@insertword wordlist,17*9,
{
    'FIRST LIST ITEM ' ,      \
    'SECOND LIST ITEM ' ,     \
    'THIRD LIST ITEM ' ,      \
    'FORTH LIST ITEM ' ,      \
    'FIFTH LIST ITEM ' ,      \
    'SIXTH LIST ITEM ' ,      \
    'SEVENTH LIST ITEM' ,     \
    'EIGHTH LIST ITEM ' ,     \
    'NINTH LIST ITEM ' ,      \
}
```

- The length [17\*9] defines the entire word (ie:- Length of each item times the number of items.
- The list must be enclosed in braces “{ }” (Linux) or “<>” (Windows) with each item separated by a comma.
- Each item in the list must be of equal size. (you must define trailing spaces.)
- The list may split across lines by using the “\” (continuation character)

wordlist may then be addressed in the program body in the following manner:-

```
@repeat_for (I,1,17*8,17)
    @words_pad {wordlist,I,17},wordlistItem
...
...
end.repeat
```



# languageONE

version3

## APPENDIX B. KEYWORDS (macros)

### Integers

@INTEGERS_EQ	@INTEGERS_ADD	@INTEGERS_SUB
@INTEGERS_MUL	@INTEGERS_DIV	@INTEGERS_CALC
@INTEGERS_AND	@INTEGERS_OR	@INTEGERS_XOR

### Sub Routines

@BEGIN_SUB	@END_SUB	@EXIT_SUB
@USING		

### Functions

@BEGIN_FUNCTION	@END_FUNCTION	@EXIT_FUNCTION
@USING		

### Screen IO

@CURSOR	@ACCEPTLINE	@ACCEPTLINE_AT
@ACCEPT_AT	@DISPLAY	@DISPLAYLINE
@DISPLAY_AT		

### Strings

@INSERTWORD	@WORDS_COPY	@WORDS_PAD
@WORDS_UPPERCASE	@WORDS_LOWERCASE	@WORDS_INSERT
@WORDS_FIND	@WORDS_REPLACE	@WORDS_ENVIRONMENT
@WORDS_LENGTH	@WORDS_STRINGTORECORD	@WORDS_RECORDTOSTRING

### Numbers

@INSERTNUMBER	@NUMBERS_EQ	@NUMBERS_ADD
@NUMBERS_SUB	@NUMBERS_MUL	@NUMBERS_DIV
@NUMBERS_CALC		

### Decisions

@IF	@_IF	@_OR
@_AND	@_END	
@ELSE	@END_IF	@BEGIN_TEST
@WHEN	@_WHEN	@WEND
@END_TEST		

# languageONE

## version3

### Loops

@REPEAT_IF	@REPEAT_WHILE	@REPEAT_FOR
@EXIT_REPEAT	@END_REPEAT	

### Files

@FILES		
@INSERTFILE	@BEGIN_RECORD	@END_RECORD
@FILES_OPEN	@FILES_READ	@FILES_WRITE
@FILES_START	@FILES_NEXT	@FILES_DELETE
@FILES_CLOSE	@FILES_CHDIR	@FILES_GETCWD
@FILES_COPY	@FILES_RENAME	@FILES_REMOVE

### Tables

@INSERTTABLE	@TABLES_BIND	@TABLES_RPUT
@TABLES_RGET	@TABLES_FPUT	@TABLES_FGET
@TABLES_SORT	@TABLES_SEARCH	

### Xtables

@INSERTXTABLE	@XTABLES_BIND	@XTABLES_RPUT
@XTABLES_RGET	@XTABLES_FPUT	@XTABLES_FGET
@XTABLES_SORT	@TABLES_SEARCH	@XTABLES_SEARCH
@XTABLES_LOAD	@XTABLES_UNLOAD	

### Arrays

@INSERTARRAY	@ARRAYS_GET	@ARRAYS_PUT
@ARRAYS_ADD	@ARRAYS_SUB	@ARRAYS_MUL
@ARRAYS_DIV	@ARRAYS_SWAP	@ARRAYS_EQ
@ARRAYS_SORT	@ARRAYS_IF	

### Date

@DATE_SECONDS	@DATE_TIMER	
@DATE_GET	@DATE_DATEFROMDAYS	@DATE_DAYSFROMDATE

### WWW

@WWW_OPEN	@WWW_PROCESS	@WWW_CLOSE
@WWW_SENDMSG	@WWW_RECVMSG	

### Miscellaneous

@INTEGERS_TOGGLE	@RUN	@WAIT (CHILD_PID)
@TERMINATE (ERROR_CODE)		

### APPENDIX C.

Because languageONE is a set of runtime libraries and access to those libraries is via a set of text based macro, it is possible to alter the syntax of languageONE to suit your own taste. It can be done in 2 ways.

#### Macros

include/LMACROS.CPY contains all the keyword macros that comprise *languageONE*. You may edit these Macros and rename any of them

#### Synonyms

A less permanent way of altering the syntax of *languageONE* is via the Synonyms that are available to the **reWRITER**. These Synonyms tell the **reWRITER** to replace any token with a substitute. *Synonyms are coded as comments*, prior to their use in an application program. They should begin with the **BEGIN.SYNONYMS** directive and end with the **END.SYNONYMS** directive.

```
; BEGIN.SYNONYMS *****
;      InsertWord      :      @INSERTWORD
;      InsertNumber    :      @INSERTNUMBER
; END.SYNONYMS *****
```

The above example directs the **reWRITER** to replace every occurrence of **INSERTWORD** with **@INSERTWORD** and every occurrence of **INSERTNUMBER** with **@INSERTNUMBER**

You may use “include” files in the Synonym section of your program by coding the path/file name of the file containing the synonyms.

```
; BEGIN.SYNONYMS *****
;      include/LPACK1.SYN
; END.SYNONYMS *****
```

**NOTE:-** *languageONE* comes with what is termed a “language Pack” (**LPACK1.SYN**) being a collection of synonyms that have been defined to modify *languageONE* to my particular taste.

### APPENDIX D.

#### System Supplied Fields

LIBRARY	NAME	STRUCTURE	VALUE	ACCESS
STDIO	LF	<i>languageONE</i>	0x0A/0x0D0A	Read
	v_Cursor	<i>languageONE</i>	0,0	Write
COMMON	c_FALSE	Constant	0	
	c_TRUE	Constant	1	
	ERROR_CODE	<i>languageONE</i>	0	Read/Write
	RETURN_CODE	<i>languageONE</i>	0	Read/Write
	CHILD_PID	<i>languageONE</i>	0	Read Only
	exitRepeat	<i>languageONE</i>	c_TRUE/C_FALSE	Read/Write
FILES	c_NULL	Constant	1	
	c_LF	Constant	2	
	c_CSV	Constant	4	
	c_RECORD	Constant	8	
	c_RANDOM	Constant	16	
	c_INDEXED	Constant	32	
	c_DIRECTORY	Constant	64	
	<filename>_READLENGTH	<i>languageONE</i>	0	Read Only
	<filename>_STATUS	<i>languageONE</i>	0	Read Only
	<filename>_SIZE	<i>languageONE</i>	0	Read Only
	<filename>_HANDLE	<i>languageONE</i>	0	Read/Write
	EOF	<i>languageONE</i>	10	
	INVALIDKEY	<i>languageONE</i>	23	
	<recordname>_NO	<i>languageONE</i>	0	
TABLES	<tablename>_STATUS	<i>languageONE</i>	0	Read Only
XTABLES	<tablename>_STATUS	<i>languageONE</i>	0	Read Only
	<tablename>_UBOUND	<i>languageONE</i>	0	
WORDS	w_CommandLine	256 bytes	Program parameters	Read Only
	w_Spaces	128 bytes	{w_spaces,1,128}	
ARRAYS	d(functionname)	As per picture	0	Read/Write

# languageONE

version3

## APPENDIX E.

### Basic Debugging in languageONE

Under a Linux operating system, if the debug switch is set when reWriting the program languageONE will display an Entering/Exiting message on the console when calling Subroutines and Functions.

### Windows Debuggers

languageONE V3.00 was developed because of the need for a useful debugger on the Windows platform. With VisualStudio being the preferred Microsoft option this release brings with it the successful use of that platform. A languageONE programmer needs to instal MASM on their development system and it can be made available to VisualStudio by right clicking on your project, selecting "Build Dependencies" followed by "Build Customizations" and checking MASM as the target.

### Linux Debuggers

Linux systems come with GDB, the GNU Debugger. It is a command line debugger and the back end to several GUI front ends such as DDD, XXGDB and INSIGHT. I personally have become familiar with DDD and it is what I would recommend.

### Linux Debugging

One difficulty in debugging a *languageONE* program under the Linux operating system is the a debuggers ability to align the source with the running code.

The solution to the problem is to produce a source file such that the debugger is able to follow the code line by line. The *languageONE* "reWRITER" has been modified for the purpose and includes a -d option. 'makeONE -d V3.00' will produce a file that is formatted as follows.

The keyword 'DEBUG' will be inserted prior to every languageONE keyword. Ie

```
display('Hello World')
DEBUG
display('What is your name')
DEBUG
acceptline w_UserName
```

DEBUG itself is a single line macro that contains the code

```
mov qword[STOP],1
```

in this way, a watch point may be established in the debugger and set to halt the program whenever STOP = 1. When entering a debug session, issue the following command:-

```
watch STOP if STOP==1
```

All going well, the debugger will then stop at each occurrence of the DEBUG macro, or more to the point, before each line of code to be debugged.

*AMENDMENT:- As of Version 3.05 the DEBUG macro will not be written by the reWriter, rather it has been built into each macro. This is a trial as the source may not align as well as required but it will be monitored until it proves itself. Note also that NASM Version 2.14.02 has been used for testing and higher versions of NASM< will definitely fail to align to the source code properly.*

### APPENDIX F.

#### Macros

Because *languageONE* is actual assembler code, the ability to code macros is a manifestation of the assembler you are using - (Windows) MASM and the (Linux) NASM. Macros written for either of these platforms will function happily within a *languageONE* program however macros written in raw code may sit anywhere in the program while cooked code must follow the `@BEGIN_INSTRUCTIONS` macro.

### APPENDIX G.

#### Assembler Directives

Assembler directives are available to a *languageONE* program and mainly take the form of constants.

Linux	Windows
<code>%define c_YES 1</code>	<code>c_YES EQU 1</code>
<code>%define c_NO 0</code>	<code>c_NO EQU 0</code>

#### LanguageONE Directives

This directive gives *languageONE* a default picture size when creating work fields. It may be used more than once and maintains its value until the next directive is coded.

`%define FP_DefaultPicture '9999.999999'`

An example of a default picture size is the use of fixed point precedence such that:-

`A = [[B * 1.23] / 2.34]` directs *languageONE* to

create work fields as it works thru the precedence levels. Note that a divide can create any number of decimal place (ie `22 / 7`) so it may help to set the no of decimal places with this directive.

### APPENDIX G.

#### languageONE Structures

##### @INSERTWORD

SIZE	VALUE	DEFINITION
DQ	0	Not Used
DQ	0	Not Used
DB	X	Define it as a string
DQ	0	Defines the length
DB	text	String Value

##### @INSERTNUMBER

SIZE	VALUE	DEFINITION
DB	0	No Of Digits
DB	0	No Of Places
DW	0	Reserved
DD	0	Reserved
DQ	1	Scaling for Fixed Point
DB	9	Defines it as a number
DQ	0	Defines the length
DQ	0	Value
DB	"9.9"	Editing Picture

# languageONE

version3

## APPENDIX H.

### (F)ast (L)ight (T)ool (K)it

languageONE V3.04 develops the interface required to link to (Fast) (L)ight (T)ool (K)it. Here's what FLTK has to say on their website ([www.fltk.org](http://www.fltk.org))

*FLTK (pronounced "fulltick") is a cross-platform C++ GUI toolkit for Linux® and Microsoft® Windows. It provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL and its built-in GLUT emulation. FLTK is designed to be small and modular enough to be statically linked, but works fine as a shared library. FLTK also includes an excellent UI builder called FLUID that can be used to create applications in minutes. FLTK is provided under the terms of the GNU Library Public License, Version 2 with exceptions that allow for static linking.*

A search of the web, when looking for GUI toolkits, can return articles that list things like "36 best GUI toolkits". I have investigated them all (believe me) and FLTK is by far the most suitable for languageONE. Although it is written in C++ it can be treated as a "screen painter" without having a full understanding of the base language. fV3.04 is a demonstration program that, if used as a template, should smooth the way into creating GUI's for languageONE. Each click can invoke a "callback" and it is this mechanism that languageONE uses to interface to the running C++ program. Although languageONE takes a subordinate role (ie the languageONE program and languageONE.lib are statically linked to the FLTK program) it can provide all the functionality of a back end server in a similar way that languageONE interfaces and uses html as a web based front end.

Note;- There is no additional functionality build into languageONE to support FLTK, you simply tell your program it is to be linked to a fltk program (Linux->(%define fltk 1) – Windows->(fltk TEXTEQU)

