

# Venom-SC Quick Reference Guide

Doc D015 V20070306 © 2002 – 2007 Micro-Robotics Ltd.

## ■ Development System

Go to [www.microrobotics.co.uk](http://www.microrobotics.co.uk) and download VenomIDE – the Integrated Development System for Venom-SC.

## ■ Command Line

The Venom command line prompt is:  
-->

Commands for immediate execution are entered at the prompt. The command is executed when you hit *Enter*.

## ■ Procedures

Procedures are normally written in a *.vnm* file and later downloaded. This illustrates the structure of a procedure:

```
TO proc(param1, param2)
  LOCAL local1
  LOCAL local2 := initial_value
  statement1
  statement2
  RETURN local1
END
```

To call a procedure, use its name. Any parameters should follow in parentheses: ( ).

## ■ Statements

A *statement* is a single Venom command, or a set of Venom commands grouped with square brackets: [ ].

Each of these lines is a Venom statement:

```
var := 1
PRINT var
IF var PRINT "YES"
[var := 1 PRINT var]
```

## ■ Names

Names are not case sensitive and are up to 64 characters long. They may contain any letter or numeric character, and underscore, but cannot start with a number.

```
_Fred FRED fred ; all OK
8Fred ; starts with a number!
```

Use meaningful names for readable code.

## ■ Startup

A procedure called *startup* will be run when the controller powers up. A default startup procedure is created by Venom. It calls your procedures *init* and *main*. Type *Run* to run your application as if at power-up.

## ■ Comments

Any text on a line after a ; character is treated as a comment. Comments do not affect the runtime code.

```
var := 1 ; A comment
; This line is pure comment
```

## ■ Variables

Values may be of two numeric types (INT and FLOAT). INTs are 32-bit, and FLOATs are IEEE single precision. Variables also hold procedures, objects, string constants, pointers and macros. The type of a variable is set when the variable is assigned a value. LOCAL declares variables that are local to a procedure.

## ■ Numbers

```
1234 ;Decimal integer
$1AAB ;Hexadecimal integer
%110100 ;Binary integer
12.34 ;Floating point number
12.34E5 ;Exponential notation
```

## ■ Macros

```
#DEFINE any_name any text;comment
```

Macro definitions must be seen before the macro is first used.

## ■ Assignment

Use the *:=* becomes equal to operator.

```
variable := 1.23
```

## ■ Expressions

Venom operators, highest precedence on the top line:

```
. AS INT AS FLOAT Postfix
SQRT EXP LOG SIN COS Prefix...
TAN - ABS NOT ! ? ...
* / DIV MOD ^ Product
+ - Additive
= <> > < >= <= Relational
AND OR EOR Boolean/Bit
```

Parentheses ( ) may be used to change the order of evaluation.

Notes: The Boolean operations actually operate on each binary bit of the data individually, and so double as bit-wise operators. The Venom constant 'TRUE' is defined as all binary 1's, and 'FALSE' as all 0's to fit in with this scheme.

## ■ Decisions

```
IF < condition > THEN
  <statement>
ELSE
  <statement>
```

```
SELECT CASE <expression>
CASE <constant> , ...
  <statement>
CASE <constant> , ...
  <statement>
CASE ELSE
  <statement>
```

## ■ Loops

```
FOREVER
  <statement>
```

```
EVERY <milliseconds>
  <statement>
```

```
REPEAT <nloops>
  <statement>
```

```
WHILE <condition: keep looping?>
  <statement>
```

```
DO
  <statement>
UNTIL < condition: stop looping?>
  (UNTIL... and DO...WHILE may also be used)
```

INDEX & INDEX0 are system variables that give the number of times a loop has executed, starting at 1 and 0 respectively. Use BREAK to exit from a loop prematurely.

## ■ Breaking into a program

Control-C will break into a program.

## ■ Waiting

To wait for an event, or for a certain time:  
AWAIT <event?> ; ...an event  
WAIT <milliseconds> ;...for a time

## ■ Multi-Tasking

Task swapping is taken care of by the operating system and happens every millisecond or so.

```
START <statement>
tsk := START <statement>
==> ; shows tasks are active
STOP tsk ;stop a task
STOP ALL ;Stop all tasks
LIST TASK ;List all the tasks
```

## ■ Objects

Define objects with MAKE or NEW.  
MAKE <global> <class> (params)  
<var>:= NEW <class> (params)

Use NEW to assign an object to a LOCAL. Send messages to objects with 'dot' (.)  
buf . Put (val)  
val := buf . Element (10)  
buf . Element (INDEX0) := val

## ■ Pointers

The 'form-a-pointer-to' operator is @.

```
Ptr := @var ; Pointer to var
```

The 'de-reference' (or 'follow-pointer') operator is !

```
Print !ptr ; Prints var.
```

To call a procedure pointer from an ARRAY, with parameters:

```
[ ( !tab . (n) ) (params) ]
```

Use [] to delimit a statement starting with '('

## ■ Printing

```
PRINT <item> , <item> ...
```

```
PRINT TO <device> , <item> , <item>
```

Print items:

```
<Expression> , "Quoted string"  
CR , HOME , CLS , FONT <font no.>  
CHR char , GOTOXY (x, y) , BEEP
```

...and many others.

Format information is passed as 'colon' parameters:

```
PRINT 12 : fieldwidth
```

```
PRINT 12.0 : fld_wdth : precision
```

Use ~ and ~~ to print in Hexadecimal and Binary:

```
-->PRINT ~255 , " ", ~~255  
FF 11111111-->
```

## ■ Arrays

Arrays are objects that hold a fixed-size set of data of one type. A 'prototype' parameter is used to define data type. Prototype may be an integer, float, string constant, or pointer. Integer values of 8, 16, 32 give the size of each element in bits.

Arrays may hold constant or variable data.

Arrays of constant data site alongside procedures, in your .vnm file:

```
ARRAY arr (prototype , size)  
<constant data> ,  
<constant data> ,  
...  
END
```

Arrays of variable data are created with MAKE or NEW within a procedure.

```
MAKE a Array(prototype , size,  
data...)
```

Elements of the array are accessed with the Element message.

```
arr . Element (n)
```

```
arr . (n) ; [See 'Shortcuts']
```

## ■ Strings and Characters

```
"A string" ;String constant
```

```
'A' ;Character constant
```

\ is the escape character:

```
\\ → \        \" → "
```

```
\$hh → any ASCII char in hex
```

For text manipulation use *Buffer* or *String* objects.

## ■ Direct Access to Memory

```
? address := val ;write a byte
```

```
val := ? address ;read a byte
```

Use ?? & ???? for 2 and 4-byte accesses into to the memory space.

## ■ Downloading code

Write your program in a .vnm file using VenomIDE. Download a file using the menu **Edit>Download**.

## ■ Programming the Flash

There are commands that copy your application code from RAM into the flash, and also commands to program new flash devices with the 5805 Application board.

```
Protect(0) ; Clear app. area  
Protect(1) ; 'ROM' application  
Copy(0,15) ; copy flash  
Copy(1,flags);download new system
```

## ■ Exception Handling

To define a region of code that may be left easily in an exception, use:

```
Exception := REGION <reg. name>  
<statement: the code>
```

*Exception* is TRUE if EXIT was used.

To exit from a region use:

```
EXIT <region_name>
```

Catching errors. Err is the number of the error that occurred in the code. You can

also catch a specific error by using an error number in place of ALL.

```
Err := CATCH ALL
```

```
<statement: the code>
```

To generate an error deliberately, use:

```
THROW <err_no>
```

## ■ Help

To get help on any Venom keyword, highlight it in VenomIDE and right click for a menu.

To get help on a global variable:

```
HELP <variable>
```

## ■ Useful information

```
PRINT system
```

Reports on the system information:

```
PRINT net
```

Reports on devices attached to an I<sup>2</sup>C Bus.

## ■ Shortcuts

Any message used without an object will be applied to the Operating System object:

```
System . Run ;Run application  
Run ;same as above
```

The *Element* message may be omitted.

```
Buf . Element(n)
```

```
Buf . (n) ;same as above
```

## ■ Common Mistakes

```
IF a AND b
```

This produces unexpected behaviour when, say, a is 1 and b is 2. Better code would be:

```
IF a <> 0 AND b <> 0
```

```
MAKE b Buffer("")
```

```
b := "Some Text"
```

Here we end up losing the buffer. The correct code is:

```
PRINT TO b, "Some Text"
```

# Object Types

See the Venom-SC help file for details of all the objects and the messages they take.