

# **RobotSee Virtual Machine Specification 0.0.1**

**By Eric Gregori ( [www.EMGRobotics.com](http://www.EMGRobotics.com) )**

**Copyright 2009-2010**

## **Introduction**

RobotSee is a programming language designed to be as simple as BASIC with the power of C. If you remember programming a Commodore 64, AppleII, or any other computer from the 1980's then you have programmed in BASIC. **BASIC** or **B**eginners **A**ll-purpose **S**ymbolic **I**nput **C**ode is a simple language used for years as a entry point into programming. BASIC has been replaced by more advanced but harder to learn languages like C, C++, and JAVA. The purpose of RobotSee is to bridge a gap between a simple entry language like BASIC, and a more powerful harder to learn language like C. RobotSee was designed for the programmer and non-programmer alike, providing a stepping stone to more advanced languages.

Advanced programmers will notice that RobotSee uses contexts similar to C ( including a switch - case ). This allows the advanced programmers to write code using similar techniques as C ( for example using switch - case for state machines ). At the same time, beginning programmers will notice that RobotSee provides intuitive and powerful keywords allowing even a simple program to do advanced function ( like ComputerVision and Speech Recognition ). RobotSee is also a expandable language.

# RobotSee

As simple as BASIC, with the power of C

- RobotSee is a language designed to be a gateway from BASIC to C.
- RobotSee is simple enough for non programmers to use, yet feature rich enough for experienced programmers.

Variables do not require type definition.

Variables are global by default ( simple scope )

No main() required – execution starts from top of file

BASIC



RobotSee



C

Local variables supported  
Parameter passing supported  
Return values supported  
Switch() – case  
C-type arrays

The RobotSee interpreter is released under the GNU GPL ( revision 3 ). You can find the license.txt file in the RobotVisionToolkit directory. For more information goto [www.EMGRobotics.com](http://www.EMGRobotics.com).

Welcome to the EMGRobotics  
Home of the RobotVisionToolkit and RobotSee



[Home](#) | [Computer Vision Training](#) | [Videos](#) | [Downloads](#) | [Robot Gallery](#) | [Linux](#) | [About me](#)

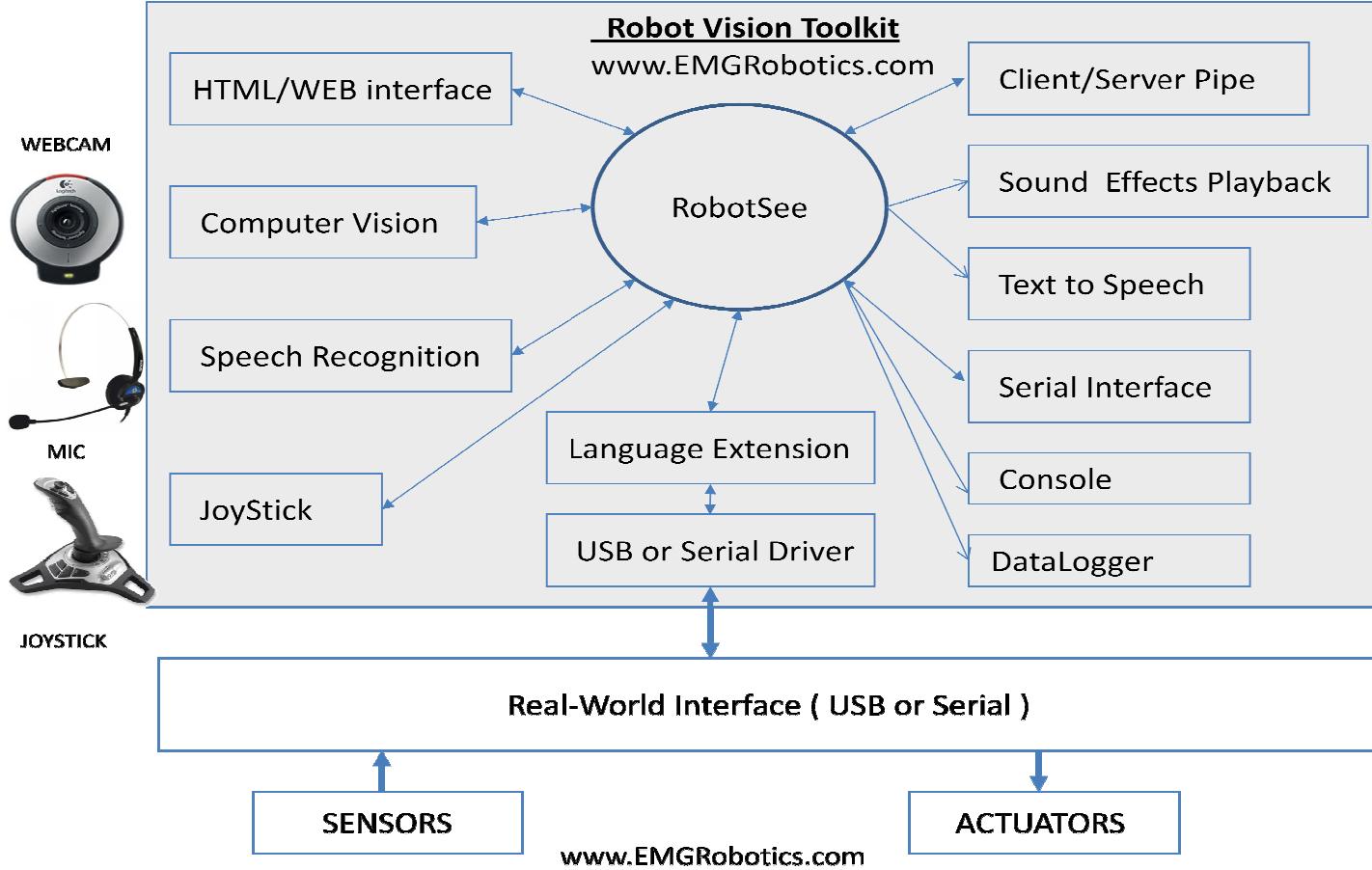
[Click Here to Goto www.emgware.com](#)

[Click Here to Goto Robotics and Software blogspot](#)

[RobotSee Users Group](#)

## RobotVisionToolkit

RobotSee is the language used by the RobotVisionToolkit. The RobotVisionToolkit was originally designed to provide non-programmers a easy method of adding computer vision to their robots. The toolkit has expanded far beyond it's original purpose, supporting a lot more then just computer vision, and being used for a lot more then just robotics.



## RobotSee Virtual Machine

RobotSee was originally implemented as a interpreted language. As it was ported to different systems, it was determined that the interpreted version of the language required too much RAM and had poor performance on smaller systems ( less the 800Mhz ). This was in part due to the interpreter being implemented in C++ ( requires lots of RAM ), and the ascii nature of the interpreter.

A virtual machine solution was chosen to meet the following requirements;

- 1) Fast
- 2) Small Flash and Ram footprint
- 3) Easily ported to various micros and Operating Systems
- 4) Built-in debugging support
- 5) Built-in monitor functions
- 6) Simple to use

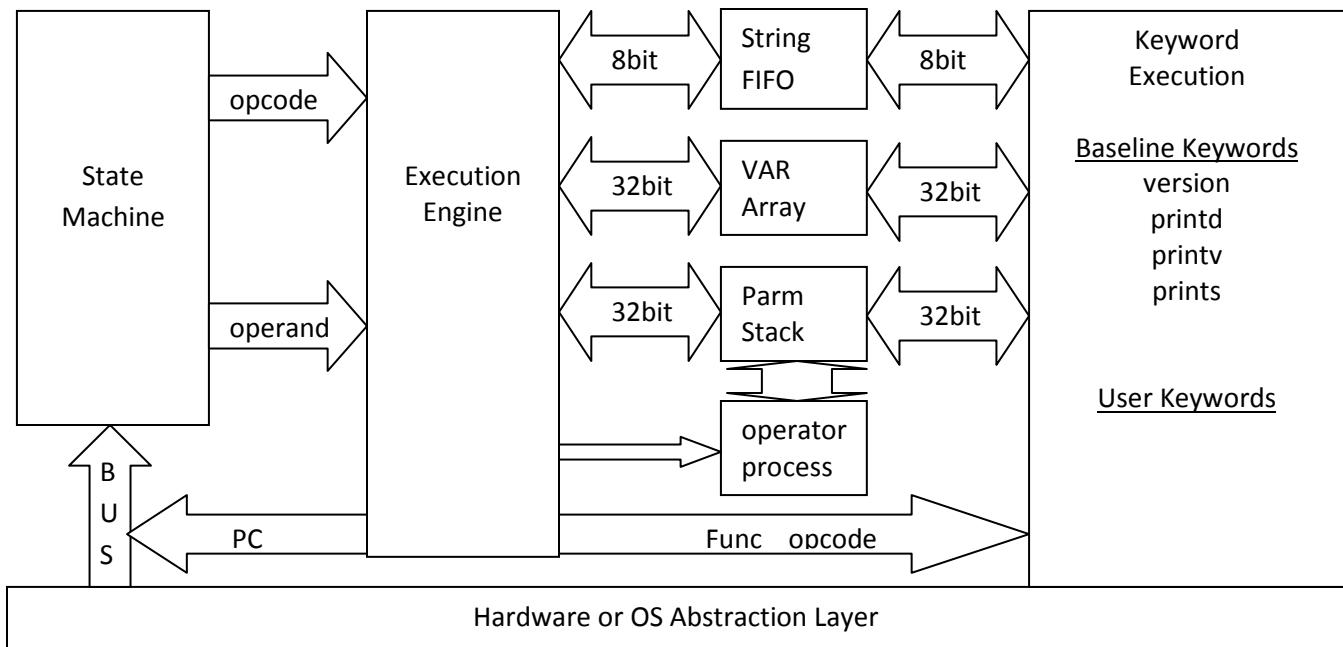
The goal is to port the Virtual Machine to Windows, Linux, MQX, and various microcontrollers as small as 8 bits. The Virtual Machine model provides a thin layer between the hardware and the compiler. This layer provides a natural environment to provide debugging support, and other monitor functions ( program download ).

The Virtual Machine is written in C, making it easily ported to small microcontrollers, and Windows/Linux systems. In addition, the Virtual Machine model isolates the RobotSee compiler from the embedded operating system. This simplifies the overall system design significantly.

The fastest and most memory efficient method would have been compiling to a intermediate language, then using a tool dedicated to the specific microcontroller or OS to complete the build to binary. This would negate the simple to use requirement, and still require a thin layer ( although much thinner ) to implement monitor and debugging functions. The Virtual Machine turns out to be the best compromise to meet all the requirements for RobotSee.

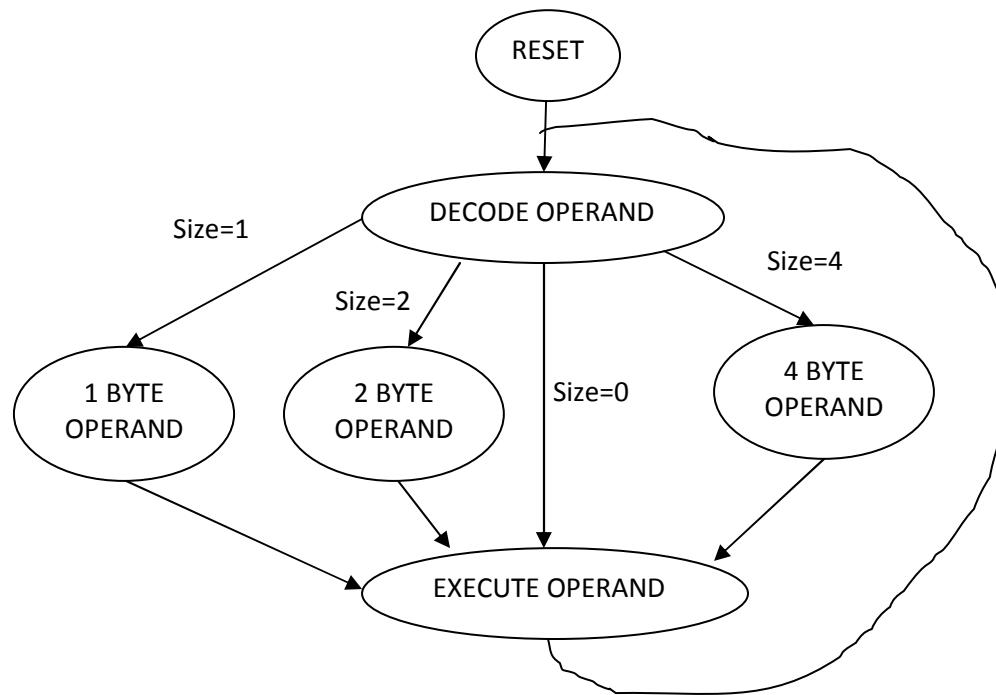
The RobotSee VM is a stack based CISC machine. Opcodes are organized by family, and ID. The high 2 bits of the opcode specify the number of bytes in the operand. The remaining 6 bits specify the family. All opcodes are capable of taking up to 4 bytes of operand, but not all require it. Un-needed bytes are ignored.

## RobotSee Virtual Machine - Block Diagram

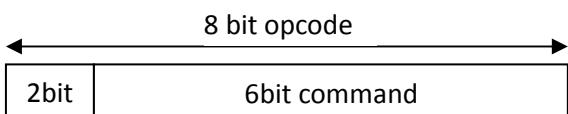


## RobotSee Virtual Machine - State Machine

A state machine is utilized by the virtual machine to facilitate reading the bus, opcode decoding, and opcode execution. Opcodes have a 2 bit size field. The state machine is responsible for decoding the field, and reading in the specified number of bytes.



## RobotSee Virtual Machine - Programmers Model



2bit operand size

00	= 0 byte operand	C
01	= 1 byte operand	C P
10	= 2 byte operand	C PP
11	= 4 byte operand	C PPPP

<b>6 bit opcodeID</b>	<b>Description</b>
('J'-0x20)	Jump Always to address specified in parameter
('?' -0x20)	Jump if stack pop is 0 to address specified in parameter
('N'-0x20)	if stack pop is NOT 0 to address specified in parameter
('F'-0x20)	Call keyword specified in parameter (keyword parameters are on call stack)
('O'-0x20)	Call operator specified in parameter ( basically same as keyword )
('D'-0x20)	Push VALUE of Constant on stack
('V'-0x20)	Push VALUE of variable on stack
('C'-0x20)	Copy top of stack, and push on stack
('S'-0x20)	Push VALUE on string stack
('T'-0x20)	Top of stack into variable referenced by parameter
('P'-0x20)	Pop stack into variable referenced by parameter
('U'-0x20)	Pop stack, dump data
('#'-0x20)	noop, used for Line Number when debugging

32 bit wide Parameter Stack

32 bit wide Variable array

8 bit wide String FIFO

## RobotSee Virtual Machine – Assembler Macros

X	Macro	Value
0	PARM0	0x00
1	PARM1	0x40
2	PARM2	0x80
4	PARM4	0xC0

MACRO (see x above )	6 bit opcodeID	Description
JUMPx	('J'-0x20)	Jump Always to address specified in parameter
JUMPZx	('?' -0x20)	Jump if stack pop is 0 to address specified in parameter
JUMPNZx	('N' -0x20)	if stack pop is NOT 0 to address specified in parameter
FUNCx	('F' -0x20)	Call keyword specified in parameter (keyword parameters are on call stack)
OPERx	('O' -0x20)	Call operator specified in parameter ( basically same as keyword )
PUSHCx	('D' -0x20)	Push VALUE of Constant on stack
PUSHVx	('V' -0x20)	Push VALUE of variable on stack
COPYx	('C' -0x20)	Copy top of stack, and push on stack
STRING	('S' -0x20)	Push VALUE on string stack
TOPx	('T' -0x20)	Top of stack into variable referenced by parameter
POPVx	('P' -0x20)	Pop stack into variable referenced by parameter
DUMP	('U' -0x20)	Pop stack, dump data
LINEx	('#' -0x20)	noop, used for Line Number when debugging

## RobotSee Virtual Machine – Variable Array

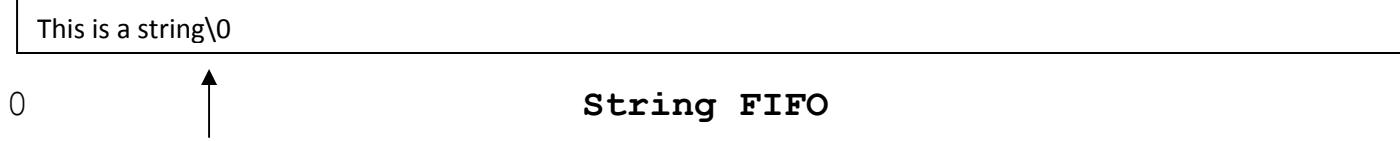
RobotSee supports both global and local variable scope. Both local and global variables are stored in the same region of memory. Global variable space is reserved in the beginning of the region, while local variables are stored at the end. The size of the region is statically allocated when the virtual machine is instantiated.

All RobotSee variables are 32bit unsigned integers. The variable region is broken up into a single linear variable array of a size configured at virtual machine initialization. Arrays are treated as multiple variables.

32 bit unsigned global variables	32 bit unsigned local vars
----------------------------------	----------------------------

## RobotSee Virtual Machine - String FIFO

To facilitate string operation, the RobotSee VM uses a dedicated single character wide FIFO. The FIFO is filled from the front(0) to the back(n) one character at a time using the STRING opcode. The VM automatically NULL terminates the strings. The VM returns a string pointer to any keywords requiring the string data. Upon reading the pointer, the string index is reset back to 0. The String FIFO size is allocated statically when the VM is instantiated.



## RobotSee Virtual Machine – Operators

The Virtual Machine currently supports the following operators via the operator ( OPERx ) opcode. Where x is the ID of the operator.

Operator	ID	Description
+	0x002B	Addition
-	0x002D	Subtraction
/	0x002F	Division
*	0x002A	Multiplication
%	0x0025	Modulus
&	0x0026	Bitwise AND
	0x007C	Bitwise OR
~	0x007E	Bitwise Invert
>>	0x3E3E	Bitwise Right Shift
<<	0x3C3C	Bitwise Left Shift
=	0x003D	Assignment
+=	0x2B3D	A = A + B
-=	0x2D3D	A = A - B
/=	0x2F3D	A = A / B
*=	0x2A3D	A = A * B
%=	0x253D	A = A % B
&=	0x263D	A = A & B
=	0x7C3D	A = A   B
==	0x3D3D	Equals ( Results in a 0 or 1 )
!=	0x213D	Not Equals ( Results in a 0 or 1 )
>=	0x3E3D	Greater then or Equals ( Results in a 0 or 1 )
<=	0x3C3D	Less then or Equals ( Results in a 0 or 1 )
>	0x003E	Greater ( Results in a 0 or 1 )
<	0x003C	Less Then ( Results in a 0 or 1 )
	0x7C7C	Logical OR
&&	0x2626	Logical AND
!	0x0021	Logical NOT

## **RobotSee Virtual Machine – Calling Keywords ( Parameter Stack )**

The RobotSee language is a keyword based language. All the real work of the language is done via keywords. Since keywords are implemented in the native machine language, keywords run at full machine speed. Keywords should not be confused with subroutines. Subroutines are created at runtime, while keywords are creates at VM compile time.

Another powerful feature of RobotSee is the ability for end users to add their own keywords. In this regard, you can think of RobotSee as a user expandable language. With the interpreted version of RobotSee, keywords were added via DLL's.

The Virtual Machine version of RobotSee is provided as full source. This allows users to add their own keywords simply by inserting their own code into the keyword.c file. The user must also insert the keyword ID into the RobotSee compiler. This is done using a simple text file. The ability to easily add keywords to expand the language makes the RobotSeeVM a powerful BasicStamp™ upgrade.

The parameter stack is the mechanism for sending data to, and getting data from both keywords and operators. The RobotSeeVM is a stack based machine, this means all data passes through the stack. The stack is 32 bits wide, with a depth determined statically at VM instantiation. As the name implies, the stack if a FILO.

## RobotSee Virtual Machine - Debugger

## RobotSee Virtual Machine - Code Modules

<b>Module Name</b>	<b>Description</b>
RobotSeeVM.h	System constants
ExecutionEngine.c	State Machine and execution engine
CallStack.c	Call stack
RamVar.c	Variable interface to ram
StringFifo.c	String FIFO
Operator.c	Operators
Keyword.c	Keywords - You would add your keywords here
InitHardware.c	Low level Hardware init for embedded systems
ports.c	Port to various RTOS's and processors
RobotSeeVM_Debugger.c	Debugger
assembler.h	Assembler MACRO's, primarily for testing
Keyword.h	Keyword ID's
ports.h	Port to various RTOS's and processors
TestAsm.h	A simple test program

## RobotSee Virtual Machine - Example Assembly Program

```
start:  
PUSHC    100  
POPV     0          // var0 = 100  
//-----  
PUSHC    7  
POPV     1          // var1 = 7  
//-----  
FUNC     KEYWORD_VERSION,      // version()  
DUMP  
//-----  
loop:  
PUSHC    0x01FF        //  
FUNC     KEYWORD_DELAY      // Delay( 0x01FF )  
DUMP  
//-----  
STRING   'H'  
STRING   'E'  
STRING   'L'  
STRING   'L'  
STRING   'O'  
STRING   0x0D  
STRING   0x0A  
STRING   0x00  
FUNC     KEYWORD_PRINTS      // print( "HELLO\n" )  
DUMP  
//-----  
PUSHV    0  
FUNC     KEYWORD_PRINTVD      // print( var0 )  
DUMP,  
//-----  
PUSHC    10          // value2 = 10  
PUSHV    0          // value1 = var0  
PUSHC    1          // reference to var1  
FUNC     KEYWORD_TEST       // KEYWORD_TEST( reference, value1, value2 ) returns sum  
POPV     1          // var1 =  
//-----  
PUSHC    1  
PUSHV    0  
OPER     '='          // var0=1  
POPV     0          // var0 =  
//-----  
PUSHC    50  
PUSHV    0  
OPER     '=='         // if( var0 == 50 ) push (1)  
JUMPZ   11          // goto( loop )  
JUMP    0           // goto( start )  
//-----
```

### RobotSee program

```
start:  
var0 = 100  
var1 = 7  
version()  
  
loop:  
delay( 0x01FF )  
print( "HELLO\n" ); print( var0 )  
var1 = test( 10, var0, var1 )  
var0 -= 1  
if( var0 == 50 ); goto( start ); endif()  
goto( loop )
```