



# LEGO Spybotics

## ROM documentation

### **(Mission engine architecture)**



## Table of contents

- 1 INTRODUCTION.....6**
- 1.1 FIRMWARE LAYER..... 6
- 1.2 ROM ENGINE LAYER..... 7
- 1.3 USING SPYBOT MISSION TEMPLATE.RCX2 ..... 9
- 1.4 SPYBOT BEAD SUBROUTINES..... 10
- 2 SENSORS & PING.....11**
- 2.1 WORLD RELATION TABLE & TARGET ..... 11
- 2.2 TOUCH SENSOR..... 12
- 2.3 LIGHT ..... 12
- 2.4 PING..... 12
- 3 GLOBAL VARIABLES, TIMERS & COUNTERS .....13**
- 3.1 VARIABLES ..... 14
- 3.1.1 nActiveToken ..... 14
- 3.1.2 nAlert ..... 15
- 3.1.3 nBioTick ..... 16
- 3.1.4 nCalibratedLight ..... 17
- 3.1.5 nControllerButton ..... 18
- 3.1.6 nDisplayMode ..... 19
- 3.1.7 nEvents ..... 20
- 3.1.8 nGameWatcher..... 21
- 3.1.9 nGameWatchers ..... 22
- 3.1.10 nGotoSemaphore ..... 23
- 3.1.11 nGotoState..... 24
- 3.1.12 nMessage..... 25
- 3.1.13 nMessageSemaphore ..... 27
- 3.1.14 nMessageToken..... 28
- 3.1.15 nMessageValue ..... 29
- 3.1.16 nSpeed..... 30
- 3.1.17 nState..... 31
- 3.1.18 nStateTargetType..... 32
- 3.1.19 nStateWatcher ..... 33
- 3.1.20 nStateWatchers ..... 34
- 3.1.21 nStatus..... 35
- 3.1.22 nTargetType..... 36
- 3.1.23 nZone ..... 37
- 3.2 TIMERS ..... 38
- 3.2.1 tGameTimer ..... 38
- 3.2.2 tStateTimer..... 39
- 3.2.3 tTickTimer ..... 40
- 3.3 TOKENS ..... 41
- 4 SUBROUTINES .....42**
- 4.1 ENGINE SUBROUTINES..... 42
- 4.1.1 ActivateToken(nTokenType)..... 43
- 4.1.2 AdjustBlink (nLight, nThresholdPercent, nHysteresis, nTime) ..... 44
- 4.1.3 CalibrateLight(var nLight, nTimes)..... 45
- 4.1.4 CheckPost..... 46
- 4.1.5 CheckTokenTick..... 47



- 4.1.6 ClearTokens ..... 48
- 4.1.7 DeactivateToken(nTokenType)..... 49
- 4.1.8 EnterBead ..... 50
- 4.1.9 ExitBead ..... 51
- 4.1.10 FindBeadToken(nTokenIndex, var pToken, var nValue) ..... 52
- 4.1.11 FindFreeToken(var pToken) ..... 53
- 4.1.12 GetGotoSemaphore ..... 54
- 4.1.13 GetMessageSemaphore ..... 55
- 4.1.14 GetStateValue(nIndex, var nValue)..... 56
- 4.1.15 GotoNewState ..... 57
- 4.1.16 IsTokenActive(nTokenType)..... 58
- 4.1.17 MainSub ..... 59
- 4.1.18 RequestGoto(nNewState)..... 60
- 4.1.19 ResetEngine ..... 61
- 4.1.20 ResetMotors ..... 62
- 4.1.21 SelectTarget ..... 63
- 4.1.22 SetDisplay(nMode)..... 64
- 4.1.23 SetMaxPower ..... 65
- 4.1.24 SetPings(nVisibility)..... 66
- 4.1.25 StartStateBeadTask(nNewState, nNewZone)..... 67
- 4.1.26 StopStateBeadTask..... 68
- 4.1.27 TokenTick(nTokenType) ..... 69
- 4.1.28 UpdateDisplayTarget ..... 70
- 4.1.29 UpdateDisplayTick ..... 71
- 4.1.30 WriteToken(pToken, nValue)..... 72
- 4.2 UTILITY SUBROUTINES..... 73
  - 4.2.1 Action(nSound, nDisplay, nMovement, nRepeat, nTime)..... 74
  - 4.2.2 BasicMove(nMove, nTime)..... 75
  - 4.2.3 Disp(nDisplay) ..... 76
  - 4.2.4 FancyMove(nMove, nTime)..... 77
  - 4.2.5 RandomMove(nMove, nTime)..... 78
  - 4.2.6 SlowDownMove(nMove, nTime)..... 79
  - 4.2.7 SpeedUpMove(nMove, nTime)..... 80
  - 4.2.8 Sum2Mem(nMem, nValue) ..... 81
  - 4.2.9 Sum4Mem(nMem, nValue) ..... 82
- 4.3 INTERACTION SUBROUTINES ..... 83
  - 4.3.1 ProcessBlinkEvent ..... 84
  - 4.3.2 ProcessPostEvent..... 85
  - 4.3.3 ProcessVLLEvent ..... 86
  - 4.3.4 ResetMessages ..... 87
  - 4.3.5 SendAllRangeMessage(nMessage, nData)..... 88
  - 4.3.6 SendMessage(nIndex, nCommand, nHiByte, nLoByte)..... 89
  - 4.3.7 SendRCXMessage(nMessage)..... 90
- 4.4 BEAD SUBROUTINES ..... 91
  - 4.4.1 Action\_Bead(nSound, nDisplay, nMovement, nRepeat, nTime)..... 93
  - 4.4.2 AdjustBlink\_Bead(nLight, nThresholdPercent, nHysteresis, nTime)..... 94
  - 4.4.3 AdjustSounds\_Bead(nFrequencyPercent, nTimePercent)..... 95
  - 4.4.4 Advance\_Bead(nExitZone, nTimes)..... 96
  - 4.4.5 Alarm\_Bead(nSound, nYellowWarn, nBlinkRate)..... 97
  - 4.4.6 Alert\_Bead(nSndAlert, nTime, nBlinkRate)..... 98
  - 4.4.7 BasicMovement\_Bead(nMovement, nTime)..... 99
  - 4.4.8 BoostToken\_Bead(nTokenIndex, nMax, pVar, nBoost, nSound)..... 100



- 4.4.9 Bump\_Bead(nTime)..... 101
- 4.4.10 CalibrateLight\_Bead(nSamples)..... 102
- 4.4.11 Countdown\_Bead(nCount, nDir, nStep)..... 103
- 4.4.12 DeactivateToken\_Bead(nTokenType)..... 104
- 4.4.13 Display\_Bead(nDisplay, nTime)..... 105
- 4.4.14 DisplayFor\_Bead(nDisplay, nTicks)..... 106
- 4.4.15 FancyMovement\_Bead(nMovement, nRepeat, nTime)..... 107
- 4.4.16 Fire\_Bead(nMessage, nFireType, pVar, nCost, nStrength, nSound, nWait)..... 108
- 4.4.17 Fx\_Bead(nFx, nTimes)..... 109
- 4.4.18 GameResult\_Bead(nPingGameResult)..... 110
- 4.4.19 Goto\_Bead(nNewState)..... 111
- 4.4.20 LED\_Bead(nLED, nBlink, nInterval, nTime)..... 112
- 4.4.21 MemOp\_Bead(nMem, nOperation, nValue)..... 113
- 4.4.22 MotorTick\_Bead(pVar, nStep, cMotorOnThreshold)..... 114
- 4.4.23 OutOfGame\_Bead..... 115
- 4.4.24 PlayLongTone\_Bead(nTone)..... 116
- 4.4.25 PlaySound\_Bead(nSound, nTime)..... 117
- 4.4.26 PlaySounds\_Bead(nSound1, nTime1, nSound2, nTime2, nSound3, nTime3)..... 118
- 4.4.27 PlayTone\_Bead(nTone)..... 119
- 4.4.28 PointTo\_Bead(nTimes)..... 120
- 4.4.29 PointToLight\_Bead(nSeek, nTime)..... 121
- 4.4.30 PointToward\_Bead(nTimes)..... 122
- 4.4.31 RandomMovement\_Bead(nMovement, nTime)..... 123
- 4.4.32 RestTick\_Bead(pVar, nStep)..... 124
- 4.4.33 Retreat\_Bead(nRetreatZone, nTimes)..... 125
- 4.4.34 Send\_Bead(nTarget, nMessage, nHiByte, nLoByte)..... 126
- 4.4.35 SendAbility\_Bead(nStrength)..... 127
- 4.4.36 SendAllRangeToken\_Bead(nTokenIndex, nMax, nMessage, nStrength, nSound)..... 128
- 4.4.37 SendID\_Bead(nShortID, nMessage, nHiByte, nLoByte)..... 129
- 4.4.38 SendRCX\_Bead(nMessage)..... 130
- 4.4.39 Set\_Bead(nProperty, nValue)..... 131
- 4.4.40 SlidingTone\_Bead(nTone, nStep, nTimes)..... 133
- 4.4.41 SlowDownMovement\_Bead(nMovement, nTime)..... 134
- 4.4.42 SpeedUpMovement\_Bead(nMovement, nTime)..... 135
- 4.4.43 TellToken\_Bead(nTokenIndex, nTarget, nMax, nMessage, nHiByte, nLoByte, nSound)..... 136
- 4.4.44 TimedToken\_Bead(nTokenIndex, nTokenType, nMax, nTime)..... 137
- 4.4.45 TokenMessage\_Bead(nTokenType, nTime)..... 138
- 4.4.46 TurnAway\_Bead(nTimes)..... 139
- 4.4.47 TwoTone\_Bead(nHiTone, nLoTone, nTimes)..... 140
- 4.4.48 VarOp\_Bead(pVar, nOperation, nValue)..... 141
- 4.4.49 VLL\_Bead(nVLL)..... 142
- 4.4.50 Wait\_Bead(nTime)..... 143
- 4.5 USER PROGRAM SUBROUTNES..... 144
  - 4.5.1 sBumpAction..... 145
  - 4.5.2 sExtension(nExtension, var nParam)..... 146
  - 4.5.3 sGetDisplay(var pVar1, var pVar2)..... 147
  - 4.5.4 sGetPoints(var nPoints)..... 148
  - 4.5.5 sInitGame..... 149
  - 4.5.6 sInitState(nState)..... 150
  - 4.5.7 sPollGameWatcher..... 151
  - 4.5.8 sPollStateWatcher..... 152
  - 4.5.9 sProcessBotMessage..... 154



---

Specification LEGO Spybotics ROM documentation	Page Page 5 of 158
---	-----------------------

---

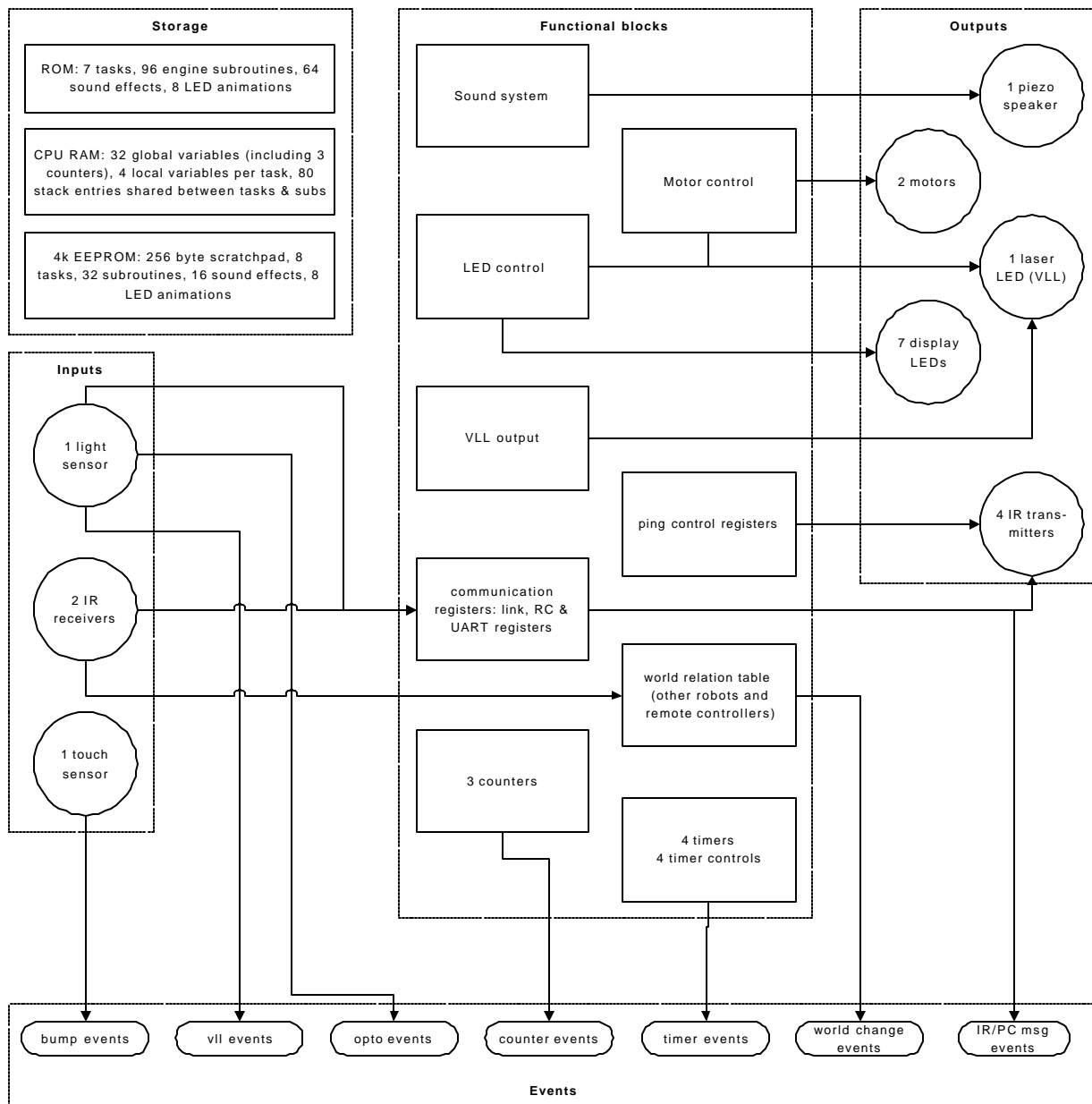
4.5.10	sProcessControllerMessage.....	155
4.5.11	sProcessVLLMessage.....	156
4.5.12	sSpecialAbility .....	157
4.5.13	sUpdateMetabolism.....	158



# 1 Introduction

This document assumes you are familiar with MindScript & Spybotics missions, and wish to use ScriptEd or other programming tool to write & download your own Spybot programs. The Spybotics ROM contains firmware and game engine layers.

## 1.1 Firmware layer





## 1.2 ROM engine layer

25 global variables (including 3 counters), & 3 timers are allocated as control registers (see **Globals.h**); 7 variables & 1 timer are available for user programs:

<b>Game control</b> game & tick timers, biotick counter, status & event registers, target type, motor speed, opto calibration, display mode & alert registers	<b>State machine control</b> current state & zone, goto state & semaphore, state timer, state target type
<b>State &amp; game watchers</b> game watcher index & limit, state watcher index & limit	<b>Messages</b> command, relation table index & parameters, controller button message, message semaphore
<b>Tokens</b> active token, message token	<b>User</b> 7 global user variables, 1 user timer

Spybot user programs consist of tasks & subroutines defined in MindScript (or LASM), created using any of the following methods:

- Ignore the Spybotics ROM engine completely. These programs may have up to 8 tasks & 32 subroutines, using any of the 32 global variables (including 3 counters) & 4 timers. This document is not necessary – refer to the VPB.hlp file. Optionally use the **Spybot.h** header and no others.
- Only use the Spybotics ROM engine utility subroutines. These programs may also have up to 8 tasks & 32 subroutines, using any of the 32 global variables (including 3 counters) & 4 timers. Use the **Utils.h** header (& optionally the **Spybot.h** header) and no others. Refer to section 4.2 for a description of the utility subroutines & parameters.
- Use **SpybotMissionTemplate.rcx2** to create missions. All ROM subroutines may be used, including the bead subroutines. This document describes the sensors, variables, counters, timers, & subroutines used by the template, together with those user subroutines that are reserved and must be defined in the user program for the engine to function correctly. This document is primarily for programs created using this method.
- Use any of the Spybotics ROM engine subroutines. These programs should use the **Globals.h** header, which defines all the subroutines; together with global constants, variables, timers & counters: these are used by most



of the ROM engine subroutines. Some of the global variables are used as control registers by the ROM engine, and will need to be correctly initialized & controlled during program execution. The **Spybot.h** header may also be used. This level of access to the Spybot system is only for advanced programmers.

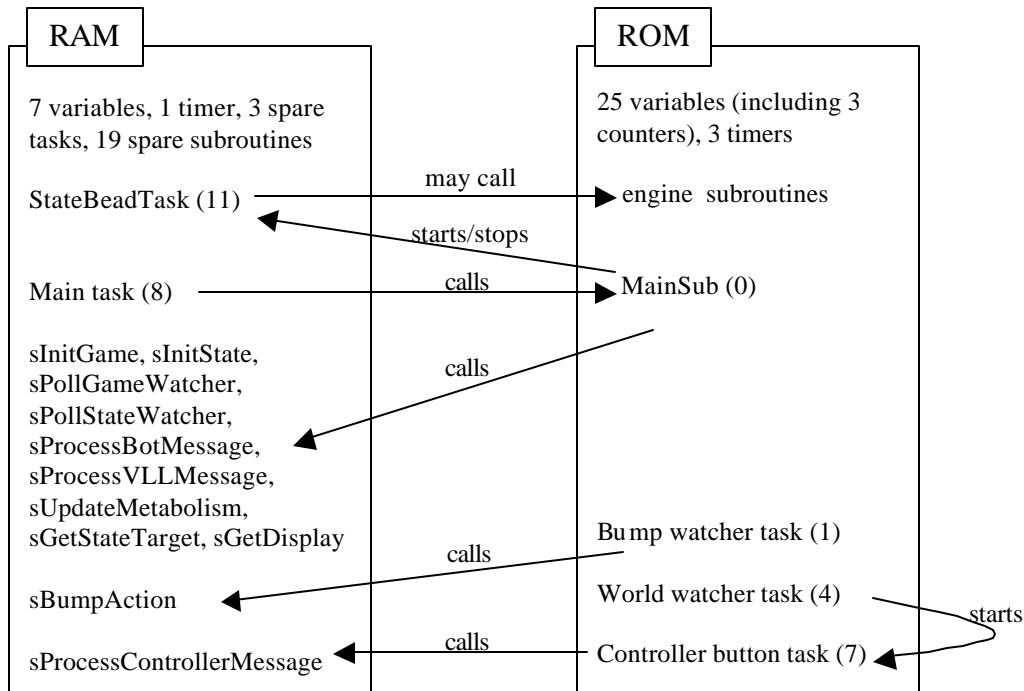
The Spybotics ROM engine is a task and subroutine library that supports the implementation of Spybot missions. User programs may use the library to implement missions or other applications. A Spybotics user program that makes use of the Spybotics ROM library (e.g. by using **SpybotMissionTemplate.rcx2**) runs as a main task, with event watchers and an action task:

- The main task initializes the system, including any user initialization (via `sInitGame` & `sInitState`), then polls each game & state watcher (via `sPollGameWatcher` & `sPollStateWatcher`) in turn; handling buffered messages (via `sProcessBotMessage` & `sProcessVLLMessage`); updating the metabolic system (via `sUpdateMetabolism`) every bio tick (once a second); reacting to world events by selecting a new state and/or zone when appropriate; selects a target according to mission criteria (via `sGetStateTargetType`); and controlling the `StateBeadTask`.
- The `BumpWatcher` ROM task monitors bump events, handling obstacle avoidance.
- The `WorldWatcher` ROM task monitors world enter & target events, messages from other Spybots, controllers & the PC, VLL messages & light blink events, and buffers each event for processing later by the main task (except for controller messages, which are immediately handled by the `ControllerTask`).
- The `ControllerTask` ROM task executes controller button code when controller button messages are received.
- The `StateBeadTask` executes programmed actions depending on the Spybots internal (i.e. global variables & timers) & external (i.e. perceived by sensors) states.

External & internal events are handled by watchers which set flag bits in a global event register if the corresponding bit in the global status register is enabled. The event register is monitored by a forever loop in the main task. Task priority is (highest first): bump, main task, controller, state beads, world watcher.



### 1.3 Using SpybotMissionTemplate.rcx2



The Spybotics ROM engine implements a state machine, where transitions between states are requested by the user program. A program may have one or more states, zero or more game watchers (& each state may have zero or more state watchers). Transitions between states (via the Goto\_Bead subroutine) may be requested by a game or state watcher, a controller button or other message action, bumper or blink action, or by actions in the StateBeadTask. If a state has a target (nStateTargetType <> cNothing), the engine selects a sub-state nZone depending on the state target (i.e. target selection criteria). State actions are implemented by subroutines in the StateBeadTask, which selects the sequence of actions according to the current state & sub-state (nState & nZone global variables). Action subroutines are described as **beads** in this document – a bead is a subroutine that achieves (or tries to achieve) some goal within a limited time – a complete & discrete unit of action.

The main task calls MainSub (the main engine ROM entry point) – this subroutine initializes the Spybot system, and enters a loop which only exits when nState = cGameOverState. After internal initialization, MainSub calls the user subroutine sInitGame to set up game watchers (nGameWatchers = cGameWatchers), display mode, and other user specific globals, then it calls user subroutine sInitState with nState = cStartState. Every time the engine changes state sInitState is called, and should set up the number of state watchers (nStateWatchers) and state target type (nStateTargetType) – for target type options, see **Globals.h**. The engine then starts the world & bump watchers (also defined in ROM), and enters a loop which:

- polls each game & state watcher in turn (calling sPollGameWatcher & sPollStateWatcher –game watchers are only polled if nState > cStartState);
- switches state if a state change has been requested by a user action – which involves stopping the StateBeadTask (waiting for any executing bead to tidy up), updating nState & nZone; then restarting StateBeadTask;
- dispatches messages to the appropriate user message handling subroutine (sProcessControllerMessage, sProcessBotMessage, or sProcessVLLMessage);



- if `nState > cStartState`, calls `sUpdateMetabolism` once per second & updates the display (via `sGetDisplay` if necessary, depending on the display mode);

When `nState = cGameOverState`, `MainSub` turns off the motors, stops all tasks, and exits.

To create a new Spybot mission or other game using the template, first edit the global constants to set the number of game watchers, together with a unique named constant (which must be positive) for each state. Then add code to each user subroutine:

`sInitGame` – set `nGameWatchers`, `nDisplayMode` (& optional user actions)  
`sGetDisplay` – update display variables if necessary (may be omitted)  
`sPollGameWatcher` – implement each game watcher (may be omitted)  
`sPollStateWatcher` – implement each state watcher (may be omitted)  
`sInitState` – sets `nStateWatchers` & `nStateTargetType` (& optional user actions)  
`sProcessControllerMessage` – implement actions for each controller button (may be omitted)  
`sProcessBotMessage` – implement actions for each Spybot message (may be omitted)  
`sProcessVLLMessage` – implement actions for each VLL message (may be omitted)  
`sUpdateMetabolism` – implement actions for metabolic tick (may be omitted)  
`sBumpAction` – implement actions for bump events (may be omitted)

Lastly add appropriate state & zone bead subroutine calls in the `StateBeadTask`. User subroutines, sounds and LED animations may also be defined.

## 1.4 Spybot bead subroutines

Bead action subroutines may be used in the `StateBeadTask`, bump subroutine (`sBumpAction`), message subroutines (`sProcessControllerMessage`, `sProcessBotMessage`, & `sProcessVLLMessage`), state initialization (`sInitState`), and metabolic tick (`sUpdateMetabolism`). There are a number of useful bead subroutines in ROM – additional bead subroutines may be downloaded with a user program, provided:

- each subroutine must start with a call to **EnterBead**, and end with a call to **ExitBead**;
- unless the bead executes quickly, the code body should be protected by a **monitor ExitBeadEvent { } abort on event** block, followed by tidy-up code (e.g. floating the motors);
- access to resources (motors, sound, LEDs, VLL) should be protected by **try { }** blocks.

The sample user program **SpybotMissionDemo.rcx2** shows a simple game with two states, where one autonomous Spybot is programmed to try to bump into another Spybot (under RC control) 3 times within 2 minutes. A few of state & game watchers are used to control the mission, and 2 game variables are defined – one is used to control display of the remaining time, and the other counts the bumps/score. A user defined bead is shown, using **EnterBead**, **ExitBead** and the **monitor ExitBeadEvent { } abort on event**, and access control **try { }** blocks, as described above.

The default mission start sound is also defined – additional user sounds and LED animations may be added, using either the predefined names in **Globals.h**, or new ones.



## 2 Sensors & ping

### 2.1 World relation table & target

The world relation table contains information for up to 16 world (Spybot, controller or PC) objects. Each device can emit a “ping” at regular intervals. A ping is an IR message identifying the sender (a one byte short ID) and containing additional information. Each ping received by a Spybot is used to determine the spatial position (range & direction) of the sender. Whenever a ping is received, the position & device specific ping information in the relation table is updated. Other IR messages from devices are used to update the position information only. The rate at which a Spybot sends pings can be modified by writing to a ping register (see ping[iDuration] below). The information carried by a Spybot’s ping may also be modified by writing to a ping register (see ping[iInfo] below). The ping rate for controllers is fixed.

World relation table entries must be indexed using a global variable (shown as nIndex in the examples below). Game applications should make sure that world[nIndex, iRange] <> cNowhere before assuming other world values are valid.

If a target is selected (target <> cNoTarget), the target values below can be used as a shortcut to access the target world entry, without an index variable. Warning – if a target is not selected, returned values are meaningless. Applications should make sure that target[iRange] <> cNowhere before assuming other target values are valid.

The value of world[nIndex, iShortID] or target[iShortID] is useful when a program wishes to distinguish between Spybots and controllers:

```
world[nIndex, iShortID] > cPCID //item is a Spybot
target[iShortID] > cPCID //target is a Spybot
```

world[nIndex, **iShortID**] – short ID of selected item

target[**iShortID**] – short ID of target

Read only

Possible values: cNoID, cControllerID1 to cControllerID6, cPCID, cMinBotID to cMaxBotID

world[nIndex, **iLinkID**] – link ID of selected item

target[**iLinkID**] – link ID of target

Read only

Possible values: cNoID, cControllerID1 to cControllerID6, cPCID

world[nIndex, **iRange**] – current range of selected item

target[**iRange**] – current range of target

Read only

Possible values: cNowhere, cAnywhere, cThere, cHere

Examples:

```
if (target[iRange] = cHere { /* beads */ }
```

world[nIndex, **iDirection**] – last known direction of selected item

target[**iDirection**] – last known direction of target

Read only

Possible values: cLeft, cLeftOfCentre, cCentre, cRightOfCentre, cRight

Examples:

```
if (target[iDirection] = cCentre { /* beads */ }
```



world[nIndex, **iAspect**] – last known aspect (view) of selected item (Spybots only)

target[i**Aspect**] – last known aspect of target (Spybots only)

Read only

Possible values: cFrontLeft, cFront, cFrontRight, cBackRight, cBack, cBackLeft

Examples:

```
if (target[iAspect] = cFront { /* beads */ }
```

world[nIndex, **iNote**] – game specific note for selected item

target[i**Note**] – game specific note for target

Read/write

Possible values: 0 to 15

target[i**Info**] – game specific ping information for target (Spybots only)

Read only

Possible values: 0 to 255

## 2.2 Touch sensor

The touch sensor is declared as “bumper” in Spybot.h, and is a boolean switch sensor.

## 2.3 Light

The light sensor is declared as “opto” in Spybot.h, and contains the current percentage light reading (0 to 100). The last average light sensor reading is in the global variable “nCalibratedLight”, set by the system when the run button is pressed, and by Beads.CalibrateLight\_Bead.

For example:

```
when opto > nCalibratedLight + 20 { Goto_Bead(gsBright) }
```

## 2.4 Ping

Part of ping[iInfo] may contain the current Spybot mode, which is not stored in any global variable. All ping values are read/write.

ping[i**Duration**] – time between pings (in 10mS steps)

Possible values: 0 (= no ping) to 255

Initial value: 25

Changed by: SetVisibility bead & stealth token

ping[i**Info**] – game specific information for this Spybot

Possible values: bits 0 & 1 game state, bits 2-4 mode (0-7), bits 5-7 state (0-7)

ping extraction masks: cPingGameMask, cPingModeMask, cPingStateMask

Initial value: 0

Changed by: system (in response to changes in state), Beads.GameResult\_Bead, Beads.OutOfGame\_Bead, Beads.Set\_Bead(setGameMode,nValue).



### 3 Global variables, timers & counters

Globals.h declares all the engine global variables, counters & timers. 25 of the 32 global variables & counters are reserved for engine use, the remaining 7 are available for user programs. 3 of the 4 timers are also reserved for engine use – the fourth timer is available for user programs.

The engine also uses 7 events (declared in Events.h); the remaining 9 may be used in user programs, by additional event watchers and/or monitors.



## 3.1 Variables

### 3.1.1 nActiveToken

Type: var

Values: bits 0 - 7 time (0-255), bits 8 - 15 token type (1-255)

Constants: cTokenTimeMask = 0x00ff access to time bits  
cTokenTypeMask = 0xff00 access to token type bits  
cTokenType = 0x0100 type divider (i.e. nActiveToken / cTokenType)

Description: Contains the current active timed token, or zero if none is active. A timed token is activated by TimedToken\_Bead, and deactivated by the ROM Engine.

The low byte is the number of seconds remaining before the token expires. The high byte is the token type – see section 3.1 Tokens for a list of pre-defined token types.

Example: May be used in message handling conditions (e.g. testing for shields when processing a fire message):

```
sub sProcessBotMessage
{
    local nParam1 = nMessageValue
    local nParam2 = nMessageValue

    nParam1 /= 256
    nParam2 &= 0xff

    select nMessage & cMessageCommandMask
    {
        when cCommandFireLaser
        {
            if nActiveToken & cTokenTypeMask <> cShieldsToken
            {
                VarOp_Bead(@nEnergy, varDec, nParam2)
                Fx_Bead(fxShudder, 6)
            }
        }
    }
}
```

See IsTokenActive for a better way of testing for active tokens than above.



### 3.1.2 nAlert

Type: var

Values: cNoAlert (= -1), or a sound, 0 – 79

Description: Used by Alert\_Bead to play an alert sound at random on each subsequent call to Alert\_Bead with the same sound.



### 3.1.3 nBioTick

Type: var (counter)

Values: seconds since game started, 0 - 32767

Description: Incremented by the engine once per second as soon as the engine leaves the cStartState (= 0).

May be used in game & state watcher conditions.



### 3.1.4 nCalibratedLight

Type: var

Values: average opto (light) sensor reading (%), 0 - 100

Description: Initialized when the user program starts by calling ResetEngine, and every time the CalibrateLight\_Bead is executed, this variable holds the average opto sensor light level.

May be used as a reference value in game & state watcher conditions.



### 3.1.5 nControllerButton

Type: var

Values: controller button code, one of:

nMessage Value	Button
cControllerButton1 = 0x0100	1: upper left
cControllerButton2 = 0x0200	2: top center
cControllerButton3 = 0x0001	3: upper right
cControllerButton4 = 0x0101	4: lower left
cControllerButton5 = 0x0201	5: lower right

Description: Set by ProcessPostEvent, this variable holds the controller button code, used by sProcessControllerMessage.



### 3.1.6 nDisplayMode

Type: var

Values: Current LED display mode, one of:

- cDisplayState = 0x01
- cDisplayRandom = 0x02
- cDisplayNothing = 0x03
- cDisplayOneVar = 0x11
- cDisplayTwoVar = 0x12
- cDisplayGameTimeRemaining = 0x13
- cDisplayRadar = 0x21
- cDisplayProximity = 0x22
- cDisplayAnimation = 0x30 - 0x3f

Description: Used by the engine to control LED display updates via UpdateDisplayTick & UpdateDisplayTarget. Initial display mode should be set in sInitGame; subsequently Set\_Bead(setDisplayMode,nMode) & Display\_Bead can be used to change the display mode, and DisplayFor\_Bead to temporarily change the mode.

ResetEngine sets the initial (default) display mode to cDisplayRadar.



### 3.1.7 nEvents

Type: var

Values:

Event bit	Purpose	Use
cWorldEvent = 0x0020	1 = world (enter) event	reserved
cTargetEvent = 0x0040	1 = target event	reserved
cPostEvent = 0x0080	1 = post event	reserved
cBumpEvent = 0x1000	1 = bump event	user

Description: Event bits are set (if the corresponding bit in nStatus is set) when an event occurs. Some of the bits are reserved for engine use; one (cBumpEvent) is available for user programs. Reserved bits are set & cleared by various engine subroutines; the cBumpEvent bit is set by the cBumpWatcher watcher task if the corresponding bit is set in nStatus, but only cleared when the engine changes state. User programs may access and change this bit as required. Execution of sBumpAction always happens on bump events: it is not controlled by nStatus.cBumpEvent.



### 3.1.8 nGameWatcher

Type: var (counter)

Values: Current state watcher to test, 0 – (nGameWatchers – 1)

Description: Used by sPollGameWatcher to select one of the user game condition(s).



### 3.1.9 nGameWatchers

Type: var

Values: Number of game watchers in this user program, 0 - 32767

Description: Set by sInitGame, this variable is used by the main task loop to iterate nGameWatcher from 0 to GameWatchers - 1.



### 3.1.10 nGotoSemaphore

Type: var

Values: 1 = access to nGotoState allowed  
< 1 access to nGotoState blocked

Description: Used by GetGotoSemaphore & GotoNewState to implement a semaphore on the nGotoState variable.



### 3.1.11 nGotoState

Type: var

Values: requested new state, 1 – 7; other possible values are:

cGameOverState = -1  
cOutOfGameState = -2  
0 = no pending goto state request

Description: The nGotoState variable is guarded by nGotoSemaphore & GetGotoSemaphore, and stores a pending goto state request from Goto\_Bead or OutOfGame\_Bead, via RequestGoto. Once a state transition has completed, the engine clears the value of nGotoState to 0.



### 3.1.12 nMessage

Type: var

Values: bits 0 - 7 command, 0 - 255, bits 8 - 15 index, 0 - 15 or one of:  
cLinkcast = 0x40  
cBroadcast = 0x80  
cInvalidRxIndex = 0x20

Constants:  
cCommandTypeMask = 0x00f0 access to command type bits

Description: ProcessPostEvent, ProcessVLLEvent & ProcessBlinkEvent update nMessage & nMessageValue when granted access via the nMessageSemaphore. The low byte is the command, and the high byte is the senders index in the world relation table, or one of the three special values shown above.

Pre-defined message formats are shown below; user applications are free to define any unused messages &/or undefined message parameters:

Command (nMessage & 0xff)	Param high byte (nMessageValue/256)	Param low byte (nMessageValue & 0xff)	nMessageValue
cCommandStart = 0x10			cGameID
cGameCommand = 0x11			undefined
cCommandGetVar = 0x12	0		pVar, 0 - 31
cCommandValue = 0x13			value of pVar
cCommandWin = 0x14			0
cCommandLose = 0x15			0
cCommandBlink = 0x16			0
cCommandGive = 0x17			
cCommandTake = 0x18			
cCommandFireLaser = 0x21	target aspect	strength	
cCommandFireSpinner = 0x22	target aspect	strength	
cCommandFireGrenade = 0x23	target aspect	strength	
cCommandReactToSnaptrax = 0x31	range	strength/time	
cCommandReactToGigaMesh = 0x32	range	strength/time	
cCommandReactToTechnojaw = 0x33	range	strength/time	
cCommandReactToShadowstrike = 0x34	range	strength/time	
cCommandMagnet = 0x41		time	
cCommandRepulse = 0x42		time	
cCommandFlashBlind = 0x43		time	
cCommandFreeze = 0x44		time	
cCommandSlow = 0x45		time	
cCommandReverse = 0x46		time	
cCommandDizzy = 0x47		time	
cCommandTypeUser = 0x50	undefined	undefined	



<b>Command</b> (nMessage & 0xff)	<b>Param high byte</b> (nMessageValue/256)	<b>Param low byte</b> (nMessageValue & 0xff)	<b>nMessageValue</b>
cCommandSmartBomb = 0x61	range	strength/time	
cCommandTypeVLL = 0x70	0		VLL command, 0-127



### 3.1.13 nMessageSemaphore

Type: var

Values: 1 = access to nMessage & nMessageValue allowed  
< 1 access to nMessage & nMessageValue blocked

Description: Used internally by GetMessageSemaphore, MainSub & CheckPost to implement a semaphore on the nMessage & nMessageValue variables.



### 3.1.14 nMessageToken

Type: var

Values: bits 0-7 time (0-255), bits 8-15 token type (1-255)

Constants: cTokenTimeMask = 0x00ff access to time bits  
cTokenTypeMask = 0xff00 access to token type bits  
cTokenType = 0x0100 type divider (i.e. nMessageToken / cTokenType)

Description: Contains the current message timed token, or zero if none is active. A message token is activated by TokenMessage\_Bead, and deactivated by CheckTokenTick.

The low byte is the number of seconds remaining before the token expires. The high byte is the token type – see section 3.3 Tokens for a list of pre-defined token types.



### 3.1.15 nMessageValue

Type: var

Values: low byte: message low byte, 0 – 255; high byte: message high byte, 0 - 255

Description: ProcessPostEvent, ProcessVLLEvent & ProcessBlinkEvent update nMessage & nMessageValue when granted access via the nMessageSemaphore.

Pre-defined message values are shown in the table for nMessage.



### 3.1.16 nSpeed

Type: var

Values: cFastSpeed = 3  
cNormalSpeed = 2  
cSlowSpeed = 1

Description: Used by SetMaxPower to set a global motor speed limit. Initial setting is cNormalSpeed. The speed can be set by Set\_Bead(setSpeed,nValue). It is also modified and restored by the activation and deactivation of cTurboToken and cSlowToken (quicksand).



### 3.1.17 nState

Type: var

Values: user states are (normally) 1 – 7, other possible values are:

cGameOverState = -1  
cOutOfGameState = -2  
cStartState = 0

Description: The current engine state. The user state bead task executes beads in states 1 – 7. State changes can only occur in response to user program execution of Goto\_Bead or OutOfGame\_Bead.

Initially the engine is in the cStartState. Once the program exits the cStartState, transitions between states 1 – 7 are possible while the game is being played. Transition to the cOutOfGameState leaves the engine executing an end of game loop; transition to the cGameOverState (from cStartState, states 1 – 7, or the cOutOfGameState) ends the program.



### 3.1.18 nStateTargetType

Type: var

Values:

Value	Meaning
cNothing = 0	No target
cAnything = 1	Any Spybot or controller
cAnyController = 2	Any controller
cAnyBot = 3	Any Spybot
cMyController = 4	Controller linked to me
cNotMyController = 5	Controller not linked to me
cTargetID = 12	Anything with ID same as hi byte of nTargetType
cTargetTeamID = 13	Anything with ID same as hi byte of nTargetType
cTargetNote = 14	Anything with iNote same as hi byte of nTargetType

Description: If nTargetType <> cStateTarget (= -1), the target selection mechanism ignores nStateTargetType, and uses the current value of nTargetType instead.

Internally, GetStateValue uses the value of nTargetType (and nStateTarget if nTargetType = -1) to calculate the priority of each Spybot & controller in the relation table.



### 3.1.19 nStateWatcher

Type: var (counter)

Values: Current state watcher to test, 0 – (nStateWatchers – 1)

Description: Used by sPollStateWatcher to select one of the user game condition(s) for the current state (in nState).



### 3.1.20 nStateWatchers

Type: var

Values: Number of state watchers in the current state (nState), 0 - 32767

Description: Set by sInitState, this variable is used by the main task loop to iterate nStateWatcher from 0 to nStateWatchers - 1.



### 3.1.21 nStatus

Type: var

Values:

Status bit	Purpose	Default	Use
cMainTaskBead = 0x0001	1 = bead executing in task 0 & 8 (main)	0	reserved
cBumpTaskBead = 0x0002	1 = bead executing in task 1 (cBumpWatcher)	0	reserved
cStateTaskBead = 0x0004	1 = bead executing in task 3 or 11 (cStateBeadTask)	0	reserved
cControllerTaskBead = 0x0008	1 = bead executing in task 7 (cControllerTask)	0	reserved
cTickEvent = 0x0010	1 = tick events enabled	1	reserved
cWorldEvent = 0x0020	1 = world (enter) events enabled	1	user
cTargetEvent = 0x0040	1 = target events enabled	1	user
cPostEvent = 0x0080	1 = post events enabled	1	user
cLockTarget = 0x0100	1 = target locked	0	user
cRunBeads = 0x0200	1 = beads can run, 0 = block bead execution	1	reserved
cExtStateValue = 0x0400	1 = user selection of state zone & target selection (sExtension, extGetStateValue & extSelectTarget)	0	reserved
cEventSounds = 0x0800	1 = sounds on events, 0 = no sounds	0	user
cBumpEvent = 0x1000	1 = register bump events in nEvents, 0 = ignore	1	user
cBlinkEvent = 0x2000	1 = call ProcessBlinkEvent on blink, 0 = ignore	1	user
cDisplayAction = 0x4000	1 = execute UpdateDisplayTarget & UpdateDisplayTick, 0 = skip	1	user
cSuspendStateBeads = 0x8000	1 = suspend normal state bead task execution; 0 = normal state bead task execution	0	reserved

Description: Each bit in the nStatus register is used to enable or disable various engine sub-components. Some can be accessed by user programs via the Set\_Bead, marked 'user' in the table. Others are reserved for use internally.

Bits enabling events correspond to the same bit positions in the nEvents variable. Default values apply before any beads start executing.



### 3.1.22 nTargetType

Type: var

Values:

Value	Meaning
cStateTarget = -1	Use nStateTarget value to select target
cNothing = 0	No target
cAnything = 1	Any Spybot or controller
cAnyController = 2	Any controller
cAnyBot = 3	Any Spybot
cMyController = 4	Controller linked to me
cNotMyController = 5	Controller not linked to me
cTargetID = 12	Spybot with specific short ID (ID is hi byte of nTargetType)
cTargetTeamID = 13	Spybot with specific linkID (ID is hi byte of nTargetType)
cTargetNote = 14	Spybot with specific iNote value (iNote is hi byte of nTargetType)

Description: If nTargetType = cStateTarget (= -1), the target selection mechanism uses nStateTargetType values. Otherwise the nTargetType values are used to select a target.

Internally, GetStateValue uses the value of nTargetType (and nStateTarget if nTargetType = -1) to calculate the priority of each Spybot & controller in the relation table. Some pre-defined timed tokens use nTargetType to temporarily override the state target, which is restored when the token deactivates.



### 3.1.23 nZone

Type: var

Values: One of cAnywhere = 1, cThere = 2, cHere = 3

Description: Current zone within nState, set by GetStateValue, and used by the cStateBeadTask to select one of the 3 sequences of bead zones to execute within the current state.



## 3.2 Timers

### 3.2.1 tGameTimer

Type: timer

Values: tenths of a second, 0 - 32767

Description: Cleared and started as soon as the engine leaves the cStartState (= 0).

May be used in game & state watcher conditions.



### 3.2.2 tStateTime r

Type: timer

Values: tenths of a second, 0 - 32767

Description: Cleared and started by GotoNewState every time the engine changes to a new state. The tStateTimer also starts when the program run button is pressed and the engine is in the start state.

May be used in game & state watcher conditions.



### 3.2.3 tTickTimer

Type: timer

Values: tenths of a second, 0 - 10

Description: Used by the engine to implement the once per second system update cycle, including incrementing nBioTick.



### 3.3 Tokens

The following table shows the predefined token actions for timed tokens. These tokens can be activated via the TimedToken\_Bead, and also in response to a message via the TokenMessage\_Bead. The cShieldsToken action should be programmed explicitly in Spybot/controller message handlers. In addition to the tick actions shown below, the engine tick decrements the token timer, and deactivates the token when it expires. Blank entries indicate no action.

cTokenType	Activate action	Tick action	Message action	Deactivate action
cShieldsToken = 0x0100	sndActivateShields			sndDeactivateShields
cReflectToken = 0x0200	sndActivateReflect		if cCommandTypeFire, Fire_Bead called, reflecting attack	sndDeactivateReflect
cMagnetToken = 0x0300	suspends bead task, target = anything	sndMagnet, Advance_Bead(cContact, 100)		restarts bead task, target = state target
cCloakToken = 0x0400	sndActivateCloak, pings turned off			sndDeactivateCloak, pings normal
cQuadDamageToken = 0x0500	none	sndQuadDamage		
cRepulseToken = 0x0600	suspends bead task, target = anything	sndMagnet, Retreat_Bead(cContact, 100)		restarts bead task, target = state target
cFlashBlindToken = 0x0700	sndActivateFlashBlind, target type = cNothing			target = state target
cTurboToken = 0x1000	speed = cFastSpeed	sndTurbo		speed = cNormalSpeed
cFreezeToken = 0x1100	global off [left right]	sndFreeze		global on [left right]
cSlowToken = 0x1200	speed = cSlowSpeed	sndSlow		speed = cNormalSpeed
cReverseToken = 0x1300	global reverse [left right]	sndReverse		global reverse [left right]
cDizzyToken = 0x1400	suspends bead task, target = anything	sndDizzy, RandomMovement_Bead(spin/turn,10)		restarts bead task, target = state target



## 4 Subroutines

By default parameters are in only; in/out parameters are prefixed by the “var” keyword (which is not valid MindScript, but only used here for documentation purposes).

### 4.1 Engine subroutines

Apart from MainSub, EnterBead & ExitBead, most of these subroutines are for internal use only.

- ActivateToken(nTokenType)
- AdjustBlink (nLight, nThresholdPercent, nHysteresis, nTime)
- CalibrateLight(var nLight, nTimes)
- CheckPost
- CheckTokenTick
- ClearTokens
- DeactivateToken(nTokenType)
- EnterBead
- ExitBead
- FindBeadToken(nTokenIndex, var pToken, var nValue)
- FindFreeToken(var pToken)
- GetGotoSemaphore
- GetMessageSemaphore
- GetStateValue(nIndex, var nValue)
- GotoNewState
- IsTokenActive(nTokenType)
- MainSub
- RequestGoto(nNewState)
- ResetEngine
- ResetMotors
- SelectTarget
- SetDisplay(nMode)
- SetMaxPower
- SetPings(nVisibility)
- StartStateBeadTask(nNewState, nNewZone)
- StopStateBeadTask
- TokenTick(nTokenType)
- UpdateDisplayTarget
- UpdateDisplayTick
- WriteToken(pToken, nValue)



#### 4.1.1 ActivateToken(nTokenType)

Sub index: 25

Parameters: nTokenType type of token, 1 - 255

Locals: none

Resources: sound, motors

Access control: retry on fail

Calls: StopStateBeadTask, SelectTarget, SetPings, SetPower, sExtension

Description: Implements pre-defined token activation effects, such as playing a token specific sound or controlling motors. Calls sExtension for user-defined activation.

Notes: See section 3.3 Tokens for more details.



### 4.1.2 AdjustBlink (nLight, nThresholdPercent, nHysteresis, nTime)

Sub index: 10

Parameters: nLight Centre value for blink event – usually nCalibratedLight, 0-100  
nThresholdPercent high & low blink threshold percent above/below nLight, 0-100  
nHysteresis blink event hysteresis, 0-100  
nTime blink time in 10 mS steps, 1-32767

Locals: none

Resources: none

Access control: none

Calls: none

Description: Adjusts the blink event thresholds, hysteresis & time.

If nLight is above 80%, sets the high threshold to 95% and the low threshold to 75%.

If nLight is below 20%, sets the high threshold to 25% and the low threshold to 5%.

Warning: if the computed high & low thresholds are the same, the cWorldWatcher task will receive a stream of blink events (if cBlinkEvent is set in nStatus), so that other events monitored by cWorldWatcher (PostEvent, EnterEvent, TargetEvent, VLLEvent) will be missed.

Notes: Default is AdjustBlink(nCalibratedLight,20,2,45), as set by ResetEngine at the start of each program run.



### 4.1.3 CalibrateLight(var nLight, nTimes)

Sub index: 9

Parameters: nLight (out) average opto sensor value (%), 0 - 100  
nTimes number of samples, 1 - 320

Locals: nTemp running total

Resources: none

Access control: none

Calls: none

Description: Takes nTimes samples of the opto sensor, pausing 100 mS between samples, and returning the average light reading in nLight.

Notes:



#### 4.1.4 CheckPost

Sub index: 1

Parameters: none

Locals: nCommand  
nTarget

Resources: any – depending on user program beads

Access control: none

Calls: Fire\_Bead, sProcessControllerMessage, sProcessVLLMessage, sProcessBotMessage, sSpecialAbility

Description: Called by MainSub to dispatch messages from other Spybots, controllers, or VLL. Also when a fire message is received tests for an active reflect token, and bounces the message back.

Notes:



#### 4.1.5 CheckTokenTick

Sub index: 23

Parameters: none

Locals: none

Resources: sound, motors

Access control: none

Calls: TokenTick, DeactivateToken

Description: Called once per second by MainSub, this subroutine decrements any active message or timed tokens, and executes their tick or deactivate action.

Notes:



#### 4.1.6 ClearTokens

Sub index: 20

Parameters: none

Locals: p token eeprom pointer

Resources: none

Access control: none

Calls: none

Description: Called by MainSub at the start of each program to clear all tokens from the eeprom token buffer.

Notes:



#### 4.1.7 DeactivateToken(nTokenType)

Sub index: 27

Parameters: nTokenType type of token, 1 - 255

Locals: none

Resources: sound, motors

Access control: retry on fail

Calls: StartStateBeadTask, SelectTarget, SetPings, SetPower, sExtension

Description: Implements pre-defined token deactivation effects, such as playing a token specific sound or controlling motors. Calls sExtension for user-defined deactivation.

Notes: See section 3.3 Tokens for more details.



#### 4.1.8 EnterBead

Sub index: 40

Parameters: none

Locals: none

Resources: none

Access control: none

Calls: none

Description: Bead entry is blocked if the engine has disabled bead execution, for example when a state switch is pending. The main task (0) is never blocked. Also sets the corresponding task bit in nStatus, showing a bead is executing.

Notes:



#### 4.1.9 ExitBead

Sub index: 41

Parameters: none

Locals: none

Resources: none

Access control: none

Calls: none

Description: Clears the corresponding task bit in nStatus, showing a bead is not executing.

Notes:



**4.1.10 FindBeadToken(nTokenIndex, var pToken, var nValue)**

Sub index: 21

Parameters: nTokenIndex unique token bead index to find, 1 - 1023  
pToken (out) pointer to found token in eeprom (0x48 – 0x7e), -1 if nTokenIndex not found  
nValue (out) found token eeprom value, undefined if pToken = -1

Locals: p eeprom token pointer  
nToken current eeprom token value  
bContinue boolean search flag, 1 = continue, 0 = done

Resources: none

Access control: none

Calls: none

Description: Used by token beads to search the eeprom buffer for a specific token. The token will be found if it has been previously activated during this program. Bits 0 – 5 of nValue contain the remaining token count; if this is zero, the token is exhausted. Bits 6 – 15 are nTokenIndex.

Notes:



#### 4.1.11 FindFreeToken(var pToken)

Sub index: 22

Parameters: pToken (out) pointer to first free token slot in eeprom (0x48 – 0x7e), -1 if no free slot found

Locals: p eeprom token pointer  
nToken current eeprom token value  
bContinue boolean search flag, 1 = continue, 0 = done

Resources: none

Access control: none

Calls: none

Description: Used by token beads to search the eeprom token buffer for a free token slot.

Notes:



#### 4.1.12 GetGotoSemaphore

Sub index: 4

Parameters: none

Locals: bRetry semaphore retry flag, 1 = retry, 0 = exit  
nTries retry counter, 0 – cMaxGotoSemaphoreRetries (=50) + 1

Resources: none

Access control: none

Calls: none

Description: Used by RequestGoto to implement the nGotoSemaphore, guarding access to the nGotoState variable.

Will time out (& grant access) after approximately 25 seconds, preventing lock-up which may be caused by badly behaved user programs.

Notes:



### 4.1.13 GetMessageSemaphore

Sub index: 5

Parameters: none

Locals: bRetry semaphore retry flag, 1 = retry, 0 = exit  
nTries retry counter, 0 – cMaxMessageSemaphoreRetries (=20) + 1

Resources: none

Access control: none

Calls: none

Description: Used by ProcessPostEvent, ProcessVLLEvent & ProcessBlinkEvent to implement the nMessageSemaphore, guarding access to the nMessage & nMessageValue variables.

Will time out (& grant access) after approximately 10 seconds, preventing lock-up which may be caused by badly behaved user programs.

Notes:



#### 4.1.14 GetStateValue(nIndex, var nValue)

Sub index: 12

Parameters: nIndex world relation table index, 0 – 15, or cNoTarget (= 255)  
nValue (out) state value, cNowhere (= 0) – cHere (= 3)

Locals: nShortID short ID of nIndex'd Spybot/controller (1 - 255), or cNoID (= 0)  
nTargetValue nIndex'd Spybot/controller range (1 – 3), or cNowhere (= 0)  
nType target selection criteria: nStateTargetType or nTargetType

Resources: none

Access control: none

Calls: none

Description: Examines the world relation table and returns a priority (ie range) value if the world table entry selected by nIndex matches the target selection criteria. Used by SelectTarget to set the current target, by MainSub & GotoNewState to select the zone within a state.

Notes: See global variables nStateTargetType & nTargetType for target selection types.



#### 4.1.15 GotoNewState

Sub index: 2

Parameters: none

Locals: nNewZone            zone for new state, cNowhere (= 0) – cHere (= 3)

Resources: sound

Access control: none

Calls: StopStateBeadTask, StartStateBeadTask, GetStateValue, SetDisplay, sInitState

Description: Uses the global nGotoState to switch to a new state, by stopping the state bead task, initialising the new state, and restarting the state bead task. Plays sndGoto if event sounds are enabled.

Notes:



#### 4.1.16 IsTokenActive(nTokenType)

Sub index: 24

Parameters: nTokenType type of token, 1 - 255

Locals: bActive boolean token active flag, 1= active, 0 = inactive

Resources: none

Access control: none

Calls: none

Description: Not used by the engine system, this function is provided as a convenience for message and watcher conditions, because it tests both the nActiveToken & nMessageToken globals. For example:

```
when cCommandFireGrenade
{
    if IsTokenActive(cShieldsToken) = 1
    {
        VarOp_Bead(@nEnergy, varDec, nParam2)
        Fx_Bead(fxShocked, 20)
    }
}
```

Notes:



#### 4.1.17 MainSub

Sub index: 0

Parameters: none

Locals: nNewZone            zone for current state, cNowhere (= 0) – cHere (= 3)

Resources: sound, motors, LEDs

Access control: none

Calls: ResetEngine, ClearTokens, ResetMessages, Sum2Mem, GotoNewState, CheckPost, Engine.UpdateEndGame, SelectTarget, CheckTokenTick, GetStateValue, StopStateBeadTask, StartStateBeadTask, SelectTarget, UpdateDisplayTick, UpdateDisplayTarget, sInitGame, sInitState, sPollStateWatcher, sPollGameWatcher, sUpdateMetabolism, sProcessBotMessage

Description: Initialises the engine, then executes the main game & state watcher loop, testing for messages and other system events, updating the display and end game data once per second.

Notes:



#### 4.1.18 RequestGoto(nNewState)

Sub index: 3

Parameters: nNewState new state to change to, usually 1 – 7, may also be one of:  
cGameOverState = -1  
cOutOfGameState = -2

Locals: none

Resources: none

Access control: none

Calls: none

Description: Implements a state change request by getting the nGotoSemaphore, then setting the global nGotoState.

Notes:



#### 4.1.19 ResetEngine

Sub index: 11

Parameters: none

Locals: none

Resources: sound, motors, LEDs, VLL

Access control: none

Calls: CalibrateLight, AdjustBlink, SetMaxPower, SetPings, ResetMotors

Description: Called by MainSub to initialise the motors, engine global variables, clear the display, calibrate the opto sensor, enable standard engine events, and set default ping visibility.

Notes: Because MainSub calls this subroutine before sInitGame, it is possible for user programs to over-ride ResetEngine defaults.



#### 4.1.20 ResetMotors

Sub index: 13

Parameters: none

Locals: nMotorControl      eeprom motor control:  
                                 bits 0-2 normal speed global power (0-7),  
                                 bits 3-5 slow speed global power (0-7),  
                                 bit 6 left motor direction (1=forward, 0=backward),  
                                 bit 7 right motor direction (1=forward, 0=backward)

Resources: motors

Access control: none

Calls: none

Description: Called by ResetEngine to initialise the motors.

Notes:



### 4.1.21 SelectTarget

Sub index: 16

Parameters: none

Locals: nIndex world relation table index, 0 - 15  
nPriority priority of each nIndex'd Spybot/controller, cNowhere (= 0) – cHere (= 3)  
nNewTarget nIndex of highest priority Spybot/controller in world, 0 - 15  
nStimulus maximum nPriority found, cNowhere (= 0) – cHere (= 3)

Resources: none

Access control: none

Calls: GetStateValue, UpdateDisplayTarget

Description: Selects the current target by searching the world relation table and using the global nStateTarget or nTargetType selection criteria (as evaluated by GetStateValue). If no target is found, clears the current target.

Updates the display if the mode is radar or proximity, improving update speed.

Notes:



#### 4.1.22 SetDisplay(nMode)

Sub index: 18

Parameters: nMode new display mode, one of:

- cDisplayState = 1
- cDisplayRandom = 2
- cDisplayNothing = 3
- cDisplayOneVar = 0x11
- cDisplayTwoVar = 0x12
- cDisplayGameTimeRemaining = 0x13
- cDisplayRadar = 0x21
- cDisplayProximity = 0x22
- cDisplayAnimation = 0x30 - 0x3f

Locals: nTemp LED representation of nState for cDisplayState

Resources: LEDs

Access control: abort on fail

Calls: Utils.Disp

Description: Changes the global LED display mode. For animations & state display, updates the LEDs directly. Otherwise the display is updated once per second by UpdateDisplayTick, or once per second and on target events by UpdateDisplayTarget.

Notes:



### 4.1.23 SetMaxPower

Sub index: 14

Parameters: none

Locals: nPower                    global power, 0 - 7

Resources: motors

Access control: none

Calls: none

Description: Sets the motor global power according to the current value of the global variable nSpeed, which must be one of cFastSpeed (= 3), cNormalSpeed (= 2), or cSlowSpeed (= 1). This limits the Spybots maximum speed.

Notes:



#### 4.1.24 SetPings(nVisibility)

Sub index: 15

Parameters: nVisibility requested Spybot visibility, one of:  
cStealthVisibility = 0  
cLowVisibility = 1  
cNormalVisibility = 2  
cHighVisibility = 3

Locals: none

Resources: none

Access control: none

Calls: none

Description: Sets the interval between high power pings to 100mS (cHighVisibility), 250 mS (cNormalVisibility) or 500 mS (cLowVisibility); cStealthVisibility turns pinging off.

Notes:



#### 4.1.25 StartStateBeadTask(nNewState, nNewZone)

Sub index: 7

Parameters: nNewState new state to start, 1 – 7  
nNewZone new zone to start, cNowhere (= 0) – cHere (= 3)

Locals: none

Resources: LEDs

Access control: none

Calls: SelectTarget, SetDisplay

Description: Puts the new state into the ping, selects a new state target, clears all except post events, starts the state bead task, and updates the display if the display mode is show state.

Notes:



#### 4.1.26 StopStateBeadTask

Sub index: 6

Parameters: none

Locals: nTries iterations of loop waiting for state bead task to stop, 0 – cMaxStopStateBeadTaskRetries (= 500) + 1

Resources: none

Access control: none

Calls: none

Description: Fires the ExitBeadEvent to terminate any executing beads. If the task calling this subroutine is the state bead task, terminates the task immediately. Otherwise loops waiting for the state bead task to terminate. If the task does not terminate after the maximum number of retries, kills the task anyway, to prevent engine lock-up with mis-behaving beads.

Notes:



#### 4.1.27 TokenTick(nTokenType)

Sub index: 26

Parameters: nTokenType type of token, 1 - 255

Locals: nMovement random motion selection for RandomMovement\_Bead, one of:

moveRandomSpinLeft = 0x0203  
moveRandomSpinRight = 0x0204  
moveRandomTurnLeft = 0x0205  
moveRandomTurnRight = 0x0206

Resources: motors, sound

Access control: retry on fail

Calls: Advance\_Bead, Retreat\_Bead, RandomMovement\_Bead, sExtension.

Description: Executed once per second to implement pre-defined token tick effects, such as playing a token specific sound or controlling motors. Calls sExtension for user-defined token ticks.

Notes: See section 3.1 Tokens for more details.



#### 4.1.28 UpdateDisplayTarget

Sub index: 17

Parameters: none

Locals: none

Resources: LEDs

Access control: abort on fail

Calls: none

Description: Implements the radar and proximity display modes, by updating the LEDs to show the direction or range of the current target.

Notes:



### 4.1.29 UpdateDisplayTick

Sub index: 19

Parameters: none

Locals: pVar1 depends on nDisplayMode  
pVar2 depends on nDisplayMode  
nTemp LED display bits

Resources: LEDs, sound

Access control: abort on fail

Calls: SetDisplay, sGetDisplay, sExtension

Description: Implements the one var, two var and game time remaining display modes by updating the LED display with a value calculated from the appropriate global variables. Called once per second by MainSub, this subroutine also decrements the timer set by DisplayFor\_Bead, restoring the display once the time expires. If event sounds are enabled, a sound is made each time the LEDs change.

One var display uses the 3 red & 3 green LEDs as a light bar: when the var is near cMax (=1000), all LEDs illuminate; at 500 (50%) only the red LEDs are on; at 0 all LEDs are off.

Two var display uses 3 red LEDs as a light bar for the first variable, and the green LEDs for the second variable.

Game time display uses all red & green LEDs, like the one var display.

Notes:



### 4.1.30 WriteToken(pToken, nValue)

Sub index: 28

Parameters: pToken pointer to token in eeprom (0x48 – 0x7e)  
nValue token value to write to eeprom, -32768 to 32767

Locals: none

Resources: none

Access control: none

Calls: none

Description: Used by token beads to write new or updated values to the eeprom token buffer.

Notes:



## 4.2 Utility subroutines

These routines may be called from any program, even if they do not use the engine (they are defined in Utils.h & Globals.h – only include one of these headers).

- Action(nSound, nDisplay, nMovement, nRepeat, nTime)
- BasicMove(nMove, nTime)
- Disp(nDisplay)
- FancyMove(nMove, nTime)
- RandomMove(nMove, nTime)
- SlowDownMove(nMove, nTime)
- SpeedUpMove(nMove, nTime)
- Sum2Mem(nMem, nValue)
- Sum4Mem(nMem, nValue)



**4.2.1 Action(nSound, nDisplay, nMovement, nRepeat, nTime)**

Sub index: 44

Parameters: nSound sound to play, 0 – 79; -1 = play no sound  
nDisplay LED animation to play, 0 – 15, -1 = play no LED animation  
nMovement Spybot motion (see BasicMove, FancyMove, RandomMove, SlowDownMove, SpeedUpMove), -1 = none  
nRepeat number of times to repeat motion, 0 - 32767  
nTime time to wait if nMovement = -1, otherwise nTime parameter for movement subs

Locals: none

Resources: Motors, LEDs, sound

Access control: nDisplay: abort on fail; nMovement: retry on fail

Calls: Disp, BasicMove, RandomMove, FancyMove, SlowDownMove, SpeedUpMove

Description: Plays any combination of sound, LED animation, and movement, like a multimedia presentation.

Notes:



### 4.2.2 BasicMove(nMove, nTime)

Sub index: 43

Parameters: nMove type of Spybot motion required, one of:

- moveForward = 0x0101
- moveBackward = 0x0102
- moveSpinLeft = 0x0103
- moveSpinRight = 0x0104
- moveTurnLeft = 0x0105
- moveTurnRight = 0x0106
- moveAvoidLeft = 0x0107
- moveAvoidRight = 0x0108
- moveRest = 0x0109
- moveStop = 0x010a

nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: motors

Access control: none

Calls: none

Description: Performs the requested Spybot motion, for the specified duration. The motors are not floated or braked, and motor power is not restored on exit.

Notes:



### 4.2.3 Disp(nDisplay)

Sub index: 42

Parameters: nDisplayLED animation to display, 0 - 15

Locals: none

Resources: LEDs

Access control: none

Calls: none

Description: Displays one of the built-in animations. Passing an out-of-range value (-1 or > 15) does not turn the display off. Passing an undefined user display value (8-15) will turn the display off.

Notes:



#### 4.2.4 FancyMove(nMove, nTime)

Sub index: 47

Parameters: nMove type of Spybot motion required, one of:

- moveZigZag = 0x0301
- moveShake = 0x0302
- moveScan = 0x0303
- moveStep = 0x0304
- moveStepBack = 0x0305
- moveSearch = 0x0306
- moveFakeLeft = 0x0307
- moveFakeRight = 0x0308
- moveBugForward = 0x0309
- moveLazy = 0x030a
- moveWalk = 0x030b
- moveWalkBack = 0x030c
- moveDance = 0x030d

nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: motors

Access control: none

Calls: none

Description: Performs the requested Spybot motion, for the specified duration. The motors are not floated or braked, and motor power is not restored on exit.

Notes: Fancy moves may not always take the exact time specified.



#### 4.2.5 RandomMove(nMove, nTime)

Sub index: 46

Parameters: nMove type of Spybot motion required, one of:

- moveRandomForward = 0x0201
- moveRandomBackward = 0x0202
- moveRandomSpinLeft = 0x0203
- moveRandomSpinRight = 0x0204
- moveRandomTurnLeft = 0x0205
- moveRandomTurnRight = 0x0206
- moveRandomRest = 0x0207

nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: motors

Access control: none

Calls: none

Description: Performs the requested Spybot motion, for the specified duration. The motors are not floated or braked, and motor power is not restored on exit.

Notes:



#### 4.2.6 SlowDownMove(nMove, nTime)

Sub index: 48

Parameters: nMove type of Spybot motion required, one of:

moveForwardSlowDown = 0x0401  
moveBackwardSlowDown = 0x0402  
moveSpinLeftSlowDown = 0x0403  
moveSpinRightSlowDown = 0x0404

nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: motors

Access control: none

Calls: none

Description: Performs the requested Spybot motion, for the specified duration. The motors are not floated or braked, and motor power is not restored on exit.

Notes:



#### 4.2.7 SpeedUpMove(nMove, nTime)

Sub index: 49

Parameters: nMove type of Spybot motion required, one of:  
moveForwardSpeedUp = 0x0501  
moveBackwardSpeedUp = 0x0502  
moveSpinLeftSpeedUp = 0x0503  
moveSpinRightSpeedUp = 0x0504

nTime time to wait (10mS steps), 0 – 32767

Locals: none

Resources: motors

Access control: none

Calls: none

Description: Performs the requested Spybot motion, for the specified duration. The motors are not floated or braked, and motor power is not restored on exit.

Notes:



#### 4.2.8 Sum2Mem(nMem, nValue)

Sub index: 50

Parameters: nMem eeprom address  
nValue value to add

Locals: nTemp eeprom byte value

Resources: none

Access control: none

Calls: none

Description: Adds nValue to a 2 byte location in eeprom, stored low byte first.

Notes: No overflow checks are performed on the eeprom location.



#### 4.2.9 Sum4Mem(nMem, nValue)

Sub index: 51

Parameters: nMem eeprom address  
nValue value to add

Locals: nTemp eeprom byte value

Resources: none

Access control: none

Calls: none

Description: Adds nValue to a 4 byte location in eeprom, stored lsb byte first.

Notes: No overflow checks are performed on the eeprom location.



### 4.3 Interaction subroutines

These routines are mostly for internal ROM engine use.

- ProcessBlinkEvent
- ProcessPostEvent
- ProcessVLLEvent
- ResetMessages
- SendAllRangeMessage(nMessage, nData)
- SendMessage(nIndex, nCommand, nHiByte, nLoByte)
- SendRCXMessage(nMessage)



### 4.3.1 ProcessBlinkEvent

Sub index: 39

Parameters: none

Locals: none

Resources: sound

Access control: abort on fail

Calls: GetMessageSemaphore

Description: Converts an opto sensor blink event into a cCommandBlink message, for eventual processing by sProcessBotMessage (cCommandBlink is one of the cCommandTypeGame commands).

The message global variable nMessage is set to cCommandBlink; the nMessageValue is always 0.

Plays a short sound if event sounds are enabled.

Notes:



### 4.3.2 ProcessPostEvent

Sub index: 32

Parameters: none

Locals:        nIndex            index of sender in relation table, 0 – 15, or cInvalidRxIndex (= 16)  
                 nCommand        first byte of message: command, 0 - 255  
                 nHiByte         second byte of message, 0 - 255  
                 nLoByte         third byte of message, 0 - 255

Resources:    sound

Access control: abort on fail

Calls:         GetMessageSemaphore

Description:   Processes IR messages from other Spybots or controllers, or from the PC, for eventual processing by sProcessBotMessage or sProcessControllerMessage.

The message global variable nMessage high byte is set to the senders index in the world relation table, and the low byte to the message nCommand; and the nMessageValue to  $256 * nHiByte + nLoByte$ , -32768 - 32767.

Starts (or restarts) the cControllerTask when a controller button press message is received.

Plays a short sound if event sounds are enabled.

Notes:



### 4.3.3 ProcessVLLEvent

Sub index: 36

Parameters: none

Locals: none

Resources: sound

Access control: abort on fail

Calls: GetMessageSemaphore

Description: Processes VLL messages from other, Spybots PBricks or USB IR Transmitters, for eventual processing by sProcessVLLMessage.

The message global variable nMessage is set to cCommandTypeVLL; and the nMessageValue to the VLL data, 0 - 127.

Plays a short sound if event sounds are enabled.

Notes:



#### 4.3.4 ResetMessages

Sub index: 33

Parameters: none

Locals: none

Resources: none

Access control: none

Calls: none

Description: Initialises the Interaction module message handler variables.

Notes:



### 4.3.5 SendAllRangeMessage(nMessage, nData)

Sub index: 38

Parameters: nMessage message to send, 0 - 255  
nData data to send, 0 - 255

Locals: nIndex world relation table index, 0 - 15

Resources: none

Access control: none

Calls: SendMessage

Description: Sends nMessage to all Spybots in the world relation table that are in the cHere, cThere or cAnywhere zones, with the actual Spybot range as the hiByte of each message.

Notes:





### 4.3.7 SendRCXMessage(nMessage)

Sub index: 37

Parameters: nMessage PBMessage to send, 1 - 255

Locals: none

Resources: none

Access control: none

Calls: none

Description: Sends an RCX PBMessage at 2400 baud with biphase encoding & sum checksum, which can be received by an RCX or Scout.

Notes:



## 4.4 Bead subroutines

Bead subroutines all have the “\_Bead” suffix, and are intended to support the easy implementation of user interface beads. For many user interface beads there will be a one-to-one mapping, with appropriate parameters. All bead subroutines call EnterBead on entry, and ExitBead on exit. User beads in eeprom should follow the same convention. Bead subroutines that take a long time to execute (typically those trying to achieve some sort of Spybot goal, eg pointing to another Spybot) should have their code guarded by an ExitBeadEvent monitor.

Action\_Bead(nSound, nDisplay, nMovement, nRepeat, nTime)  
AdjustBlink\_Bead(nLight, nThresholdPercent, nHysteresis, nTime)  
AdjustSounds\_Bead(nFrequencyPercent, nTimePercent)  
Advance\_Bead(nExitZone, nTimes)  
Alarm\_Bead(nSound, nYellowWarn, nBlinkRate)  
Alert\_Bead(nSndAlert, nTime, nBlinkRate)  
BasicMovement\_Bead(nMovement, nTime)  
BoostToken\_Bead(nTokenIndex, nMax, pVar, nBoost, nSound)  
Bump\_Bead(nTime)  
CalibrateLight\_Bead(nSamples)  
CountDown\_Bead(nCount, nDir, nStep)  
DeactivateToken\_Bead(nTokenType)  
Display\_Bead(nDisplay, nTime)  
DisplayFor\_Bead(nDisplay, nTicks)  
FancyMovement\_Bead(nMovement, nRepeat, nTime)  
Fire\_Bead(nMessage, nFireType, pVar, nCost, nStrength, nSound, nWait)  
Fx\_Bead(nFx, nTimes)  
GameResult\_Bead(nPingGameResult)  
Goto\_Bead(nNewState)  
LED\_Bead(nLED, nBlink, nInterval, nTime)  
MemOp\_Bead(nMem, nOperation, nValue)  
MotorTick\_Bead(pVar, nStep, cMotorOnThreshold)  
OutOfGame\_Bead  
PlayLongTone\_Bead(nTone)  
PlaySound\_Bead(nSound, nTime)  
PlaySounds\_Bead(nSound1, nTime1, nSound2, nTime2, nSound3, nTime3)  
PlayTone\_Bead(nTone)  
PointTo\_Bead(nTimes)  
PointToLight\_Bead(nSeek, nTime)  
PointToward\_Bead(nTimes)  
RandomMovement\_Bead(nMovement, nTime)  
RestTick\_Bead(pVar, nStep)  
Retreat\_Bead(nRetreatZone, nTimes)  
Send\_Bead(nTarget, nMessage, nHiByte, nLoByte)  
SendAbility\_Bead(nStrength)  
SendAllRangeToken\_Bead(nTokenIndex, nMax, nMessage, nStrength, nSound)  
SendID\_Bead(nShortID, nMessage, nHiByte, nLoByte)  
SendRCX\_Bead(nMessage)  
Set\_Bead(nProperty, nValue)  
SlidingTone\_Bead(nTone, nStep, nTimes)  
SlowDownMovement\_Bead(nMovement, nTime)  
SpeedUpMovement\_Bead(nMovement, nTime)  
TellToken\_Bead(nTokenIndex, nTarget, nMax, nMessage, nHiByte, nLoByte, nSound)  
TimedToken\_Bead(nTokenIndex, nTokenType, nMax, nTime)



TokenMessage\_Bead(nTokenType, nTime)  
TurnAway\_Bead(nTimes)  
TwoTone\_Bead(nHiTone, nLoTone, nTimes)  
VarOp\_Bead(pVar, nOperation, nValue)  
VLL\_Bead(nVLL)  
Wait\_Bead(nTime)



#### 4.4.1 Action\_Bead(nSound, nDisplay, nMovement, nRepeat, nTime)

Sub index: 64

Parameters: nSound sound to play, 0 – 79; -1 = play no sound  
nDisplay LED animation to play, 0 – 15, -1 = play no LED animation  
nMovement Spybot motion (see Utility subroutines), -1 = none  
nRepeat number of times to repeat motion, 0 - 32767  
nTime time to wait if nMovement = -1, otherwise nTime parameter for movement subs

Locals: none

Resources: motors, LEDs, sound

Access control: in Action

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, Action

Description: Plays any combination of sound, LED animation, and movement, like a multimedia presentation. See Action for additional documentation. If a movement is specified, floats the motors on exit.

Notes:



#### 4.4.2 AdjustBlink\_Bead(nLight, nThresholdPercent, nHysteresis, nTime)

Sub index: 79

Parameters: nLight blink event centre value– usually nCalibratedLight, 0-100  
nThresholdPercent high & low blink threshold percentage above & below nLight, 0-100  
nHysteresis blink event hysteresis, 0-100  
nTime blink time in 10 mS steps, 1-32767

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, AdjustBlink

Description: Adjusts the blink event thresholds, hysteresis & time. See Engine.AdjustBlink for additional documentation.

Notes: Default set by ResetEngine at the start of each program run is  
AdjustBlink\_Bead(nCalibratedLight,20,2,45):

optoEvent.high = nCalibratedLight + 20%  
optoEvent.low = nCalibratedLight - 20%  
optoEvent.hysteresis = 2%  
optoEvent.time = 50 – 500 (=50 + 45 \* 10) mS



### 4.4.3 AdjustSounds\_Bead(nFrequencyPercent, nTimePercent)

Sub index: 104

Parameters: nFrequencyPercent frequency shift: see LASM PBSoundCtrl\_FreqScale  
nTimePercent playback speed: see LASM PBSoundCtrl\_TimeScale

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: Shifts the frequency of all non-fixed tones in sounds up or down, and changes the playback speed.  
AdjustSounds\_Bead(100,100) sets the frequency shift and playback speed back to normal.

Notes: nFrequencyPercent = 50% lowers all non-fixed tones by an octave; nTimePercent = 200% doubles the playback time.



#### 4.4.4 Advance\_Bead(nExitZone, nTimes)

Sub index: 84

Parameters: nExitZone Goal range for target, 3 = cHere, 4 = cContact  
nTimes Number of iterations to make of the Advance loop, 1-32767

Locals: none

Resources: Motors

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Tries to move a Spybot towards the current target. If there is no target, or the target is in the anywhere zone, the motors float. If the target is in or nearer than the specified nExitZone, the bead exits. Otherwise the motors drive the Spybot towards the target. The subroutine repeats the Advance loop nTimes.

Notes: Use nExitZone = cContact to make the Advance\_Bead continue when the target is in the here zone.



#### 4.4.5 Alarm\_Bead(nSound, nYellowWarn, nBlinkRate)

Sub index: 92

Parameters: nSound sound to play, 0 – 79; -1 = play no sound  
nYellowWarn 0 = yellow LED off, 1 = yellowLED on  
nBlinkRate 0 = no blinking, 1-255 = blink rate (10mS steps)

Locals: none

Resources: LEDs, sound

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: A one-shot bead that plays a sound and uses the yellow LED to show an alarm state (optionally blinking).

Notes: Use Alarm\_Bead(-1,0,0) to turn the yellow LED alarm off.



#### 4.4.6 Alert\_Bead(nSndAlert, nTime, nBlinkRate)

Sub index: 90

Parameters: nSndAlert sound to play, 0 – 79; -1 = play no sound  
nTime time to wait for nSndAlert to play  
nBlinkRate 0 = no blinking, 1-255 = blink rate (10mS steps)

Locals: none

Resources: LEDs, sound

Access control: none

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays nSndAlert, waits for the specified time, then turns the yellow LED on, with the specified blink rate. If the subroutine is called again with the same sound index (eg because a game/state watcher is repeatedly triggered), it randomly plays sndGeneralAlert followed by nSndAlert, on average one time in 10,.

Notes: Use Alert\_Bead(-1,0,0) to clear the alert and turns the yellow off.



#### 4.4.7 BasicMovement\_Bead(nMovement, nTime)

Sub index: 87

Parameters: nMovement type of Spybot motion: see Utils.BasicMove  
nTime time for motion, 0 – 32767 (10mS steps)

Locals: none

Resources: Motors

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, BasicMove

Description: Monitors ExitBeadEvent; floats the motors on exit. Exits if another task with higher priority is already controlling the motors.

Notes:



#### 4.4.8 BoostToken\_Bead(nTokenIndex, nMax, pVar, nBoost, nSound)

Sub index: 74

Parameters: nTokenIndex unique token ID  
nMax number of times this token bead can be used  
pVar pointer to global var to adjust (0-31), -1 if none  
nBoost amount to add to pVar (-32768 – 32767)  
nSound sound to play, 0 – 79; -1 = play no sound

Locals: pToken pointer to token in eeprom  
nTemp token value  
nValue pVar value

Resources: sound

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, FindBeadToken, FindFreeToken, WriteToken

Description: Boosts a variable by a fixed amount, a limited number of times. The sound only plays when the boost is successful, otherwise this bead makes a small click sound

Notes: pVar is clamped to a value from cMin (= 0) to cMax (=1000).



#### 4.4.9 Bump\_Bead(nTime)

Sub index: 100

Parameters: nTime time before giving up 0 – 32767 (in 10mS steps)

Locals: none

Resources: motors

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Drives the Spybot forward until it hits something, or the specified time elapses.

Notes:



#### 4.4.10 CalibrateLight\_Bead(nSamples)

Sub index: 78

Parameters: nSamples      number of samples to read from the opto sensor, 1 - 32767

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, CalibrateLight

Description: Recalibrates the global variable nCalibratedLight by taking nSamples from the opto sensor.

Notes: It may be a good idea to call AdjustBlink\_Bead with the newly calibrated value if blink events are used.



#### 4.4.11 Countdown\_Bead(nCount, nDir, nStep)

Sub index: 97

Parameters: nCount number to count down/up, 1 - 7  
nDir count direction: countDirDown = 0, countDirUp = 1  
nStep time to wait between counts (10 mS steps), 0 - 32767

Locals: nCounter

Resources: LEDs, sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Uses the 7 LEDs to count down from nCount to 0, or up from 0 to nCount, displaying the count as a bar of LEDs.

Notes:



#### 4.4.12 DeactivateToken\_Bead(nTokenType)

Sub index: 99

Parameters: nTokenType one of the built-in or user timed tokens, previously activated using TimedToken\_Bead (see globals.h for valid values)

Locals: none

Resources: sound

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, DeactivateToken

Description: Deactivates a token. Plays the deactivate sound (if any).

Notes: Cannot be used to deactivate a message token.



#### 4.4.13 Display\_Bead(nDisplay, nTime)

Sub index: 96

Parameters: nDisplay LED animation to play, 0 – 15  
nTime time to wait (10 mS steps), 0 - 32767

Locals: none

Resources: LEDs

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, Disp

Description: Plays one of the ROM or user programmed animations. The LED display is not cleared on exit, so animations that play in a forever loop will continue.

Notes: Access to the LEDs from other tasks will stop any animation started by Display\_Bead, unless the nWiat value is long enough for the display to complete, and the task calling Display\_Bead has a higher priority.



#### 4.4.14 DisplayFor\_Bead(nDisplay, nTicks)

Sub index: 106

Parameters: nDisplay LED animation to play, 0 – 15  
nTicks time to play the animation (seconds), 1 - 255

Locals: none

Resources: LEDs

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: Starts the engine playing a LED animation. After nTicks seconds the display reverts to what it was before calling DisplayFor\_Bead.

Notes: The global variable nDisplayMode is used to store the old display mode (in the high byte), and the number of remaining ticks (in the low byte).



#### 4.4.15 FancyMovement\_Bead(nMovement, nRepeat, nTime)

Sub index: 109

Parameters: nMovement type of Spybot motion: see Utils.FancyMove  
nRepeat times to repeat the fancy move, 1-32767  
nTime time for motion, 0 – 32767 (10mS steps)

Locals: none

Resources: Motors

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, FancyMove

Description: Monitors ExitBeadEvent; floats the motors on exit. Exits if another task with higher priority is already controlling, or takes control of, the motors.

Notes:

**4.4.16 Fire\_Bead(nMessage, nFireType, pVar, nCost, nStrength, nSound, nWait)**

Sub index: 77

Parameters:

nMessage	message to send to target Spybot, usually one of 0x20 – 0x2f (typically cCommandFireLaser, cCommandFireSpinner or cCommandFireGrenade), 1 - 255
nFireType	weapon range & spread, one of: fireTypeAnywhere = 0 fireTypeThereWide = 1 fireTypeThereNarrow = 2 fireTypeHereWide = 3 fireTypeHereNarrow = 4
pVar	pointer to global var to be decremented by nCost, 0-31
nCost	cost of firing weapon, -32768 - 32767
nStrength	strength of fire message, sent as lo byte, 0 - 255
nSound	sound to play, 0 – 79; -1 = play no sound
nWait	time to wait for sound to play before sending fire message (10mS steps), 0 -32767

Locals:

nTemp	pVar value
bFire	0 = target out of range or spread, 1 = send fire message

Resources: VLL, sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, SendMessage

Description: For nMessage = cCommandFireLaser, plays a fire sound while flashing the VLL, otherwise plays the specified nSound and waits for nWait. If the target is within the range & spread specified by nFireType, sends nMessage to the target Spybot, with the target aspect (ie front, back, front left,...) in the hi byte and nStrength in the low byte.

Notes: When this bead is called with nMessage = cCommandFireLaser, the nWait parameter is 5 times the number of times to fire the laser (eg nWait = 50 fires the laser 5 times).



#### 4.4.17 Fx\_Bead(nFx, nTimes)

Sub index: 68

Parameters: nFx special effect, one of:  
fxShudder = 0  
fxShocked = 1  
fxFireAtBot = 2  
fxTwitter = 3  
nTimes number of times to repeat the special effect, 0 - 32767

Locals: none

Resources: motors, sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays one of the Spybot special motion effects. For fxShudder, no sound is played. For fxShocked, sndShocked is played while the Spybot moves. For fxFireAtBot, plays a series of laser firing tones. For fxTwitter, makes a random twittering sound.

Notes:



#### 4.4.18 GameResult\_Bead(nPingGameResult)

Sub index: 67

Parameters: nPingGameResult game result, one of:  
cPingNotPlayingGame = 0  
cPingWonGame = 2  
cPingLostGame = 3

Locals: nTemp ping info

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SetPings, Sum2Mem, Sum4Mem, sGetPoints

Description: Changes the Spybots ping info field (bits 0 & 1) to show that the Spybot is out of the game, usually because it won or lost. Also restores Spybots visibility (so other Spybots can see this one lost or won), updates eeprom play seconds, total play time, number of wins & losses, total game points, and maximum number of Spybots encountered.

Notes:





#### 4.4.20 LED\_Bead(nLED, nBlink, nInterval, nTime)

Sub index: 95

Parameters: nLED bit pattern for LED display, 0 - 127  
nBlink bit pattern for blinking LEDs, 0 - 127  
nInterval blink rate (10mS steps), 0 - 255  
nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: LEDs

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Sets the display LEDs to a specific bit pattern, optionally blinking at a specific rate. Any active display animation will be terminated. Waits for the specified time before exiting. Once the bead exits, the global nDisplayMode will usually update the display as the engine continues. To prevent this, set nDisplayMode = cDisplayNothing, either in sInitGame, or by calling Set\_Bead(setDisplayMode, cDisplayNothing).

Notes: Blinking LED's will continue to blink if the display mode is not cDisplayNothing; it may be necessary to stop blinking by calling LED\_Bead(0,0,0,0) to restore normal display mode.



#### 4.4.21 MemOp\_Bead(nMem, nOperation, nValue)

Sub index: 70

Parameters: nMem eeprom memory address, 0 - 255  
nOperation operation to perform, one of:  
memSet = 0  
memInc = 1  
memDec = 2  
memMul = 3  
memDiv = 4  
memRnd = 5  
nValue operand, range depends on operation:  
memSet, memInc, memDec, memMul, memDiv: 0 – 255  
memRnd: 2 - 160

Locals: nTemp eeprom value

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: Performs the specified operation on one of the eeprom memory locations. For the memSet operation, the eeprom location is set randomly to a value from 1 to nValue.

Notes: No checks are made for valid (user) eeprom addresses.  
Eeprom locations are one byte: no checks are performed for over- or under-flow.



#### 4.4.22 MotorTick\_Bead(pVar, nStep, cMotorOnThreshold)

Sub index: 71

Parameters: pVar pointer to global variable to decrement, 0 - 31  
nStep amount to decrement pVar by for each motor on with a power level above cMotorOnThreshold, cMin = 0 – cMax = 1000  
cMotorOnThreshold motor power threshold for decrementing pVar, 0 - 7

Locals: nTemp motor status  
nValue pVar value

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: Reads the instantaneous left & right motor status registers and decrements pVar by nStep for each motor if the power level is above the specified threshold. If both motors are on at a power level above the threshold, pVar is decremented by 2\*nStep. pVar is clamped so it cannot go below cMin (=0).

Notes: This bead is intended for use in metabolic games (eg with a user global energy variable) within the sUpdateMetabolism user program subroutine, which is executed once per second by the engine.



#### 4.4.23 OutOfGame\_Bead

Sub index: 76

Parameters: none

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SetPings, RequestGoto

Description: Request the engine to switch to the cOutOfGameState. The Spybot ping is updated to playing, state 0 (0b000xxx01), and the ping visibility restored to normal (so other Spybots can read the ping info). Game and state timers are stopped.

In the cOutOfGameState, the engine only responds to game watchers (via sPollGameWatcher) and game messages from other Spybots (via sProcessBotMessage). UpdateEndGame is also called in the out-of-game loop. The only way for a Spybot to win or lose is for a game watcher or Spybot message to run a win or lose bead sequence terminated by a Goto\_Bead(cGameOverState).

Notes: This bead should usually be the last in a conditional bead sequence. It is a programming error to call the OutOfGame\_Bead or Goto\_Bead more than once in any bead sequence, and in particular in task 0, which will block on the second call. However, GetGotoSemaphore will timeout after approximately 25 seconds, then the main task will unblock and continue.



#### 4.4.24 PlayLongTone\_Bead(nTone)

Sub index: 105

Parameters: nTone frequency to play (Hz), see LASM playt

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays a single tone for 750 mS, and waits until it has completed playing.

Notes:



#### 4.4.25 PlaySound\_Bead(nSound, nTime)

Sub index: 94

Parameters: nSound sound to play, 0 – 79; -1 = play no sound  
nTime time to wait for sound to play (10mS steps), 0 -32767

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays one of the built-in ROM sounds (0-63) or eeprom sounds (64 – 79). If nSound = sndNone (-1), stops any sound that is currently playing.

Notes:



#### 4.4.26 PlaySounds\_Bead(nSound1, nTime1, nSound2, nTime2, nSound3, nTime3)

Sub index: 102

Parameters: nSound1 sound to play, 0 – 79; -1 = play no sound  
nTime1 time to wait for first sound to play (10mS steps), 0 -32767  
nSound2 sound to play, 0 – 79; -1 = play no sound  
nTime2 time to wait for second sound to play (10mS steps), 0 -32767  
nSound3 sound to play, 0 – 79; -1 = play no sound  
nTime3 time to wait for thrid sound to play (10mS steps), 0 -32767

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays up to three of the built-in ROM sounds (0-63) or eeprom sounds (64 – 79), waiting between each one.

Notes:



#### 4.4.27 PlayTone\_Bead(nTone)

Sub index: 93

Parameters: nTone frequency to play (Hz), see LASM playt

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays a single tone for 100 mS, and waits until it has completed playing.

Notes:



#### 4.4.28 PointTo\_Bead(nTimes)

Sub index: 82

Parameters: nTimes Number of iterations to make of the PointTo loop, 1-32767

Locals: none

Resources: motors, sound

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Tries to point a Spybot at the current target. If there is no target, or the target comes into the centre in the here or there zone the bead exits. If the target is in the anywhere zone, the Spybot spins (left), scanning for the target. Otherwise the motors spin the Spybot left or right, depending on the target bearing. The subroutine repeats the PointTo loop nTimes.

On exit, if the target is in the centre and event sounds are enabled, plays sndPointTo.

Notes:



#### 4.4.29 PointToLight\_Bead(nSeek, nTime)

Sub index: 80

Parameters: nSeek light criteria, one of: seekDark = 0 or seekBright = 1  
nTime time for a complete scan (10 mS steps), 20 - 32767

Locals: nStep step counter  
nPeakStep peak light position so far  
nOldLight previous step light sensor value  
nNewLight new light sensor value

Resources: motors, sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Spins the Spybot in 20 steps, looking for the brightest (or darkest) light value, as detected by the opto sensor, then spins the Spybot back to the brightest/darkest position.

Notes: This algorithm assumes the light sensor is pointing parallel to the floor.



#### 4.4.30 PointToward\_Bead(nTimes)

Sub index: 81

Parameters: nTimes Number of iterations to make of the PointToward loop, 1-32767

Locals: none

Resources: motors, sound

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Tries to move a Spybot so it is pointing at the current target. If there is no target, or the target comes into the centre in the here or there zone the bead exits. If the target is in the anywhere zone, the Spybot spins (left), scanning for the target. Otherwise the motors drive the Spybot forward left or right, depending on the target bearing. The subroutine repeats the PointToward loop nTimes.

On exit, if the target is in the centre and event sounds are enabled, plays sndPointTo.

Notes:



#### 4.4.31 RandomMovement\_Bead(nMovement, nTime)

Sub index: 108

Parameters: nMovement type of Spybot motion: see Utils.RandomMove  
nTime time for motion, 0 – 32767 (10mS steps)

Locals: none

Resources: Motors

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, RandomMove

Description: Monitors ExitBeadEvent; floats the motors on exit. Exits if another task with higher priority is already controlling the motors.

Notes:



#### 4.4.32 RestTick\_Bead(pVar, nStep)

Sub index: 72

Parameters: pVar pointer to global variable to decrement, 0 - 31  
nStep amount to increment pVar by for each motor off, 1- 1000

Locals: nTemp motor status  
nValue pVar value

Resources: none

Access control: none

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Reads the instantaneous left & right motor status registers and increments pVar by nStep for each motor not turned on. If both motors are off (brake or float), pVar is incremented by 2\*nStep. pVar is clamped so it cannot go below cMax (=1000).

Notes: This bead is intended for use in metabolic games or simulations (eg with a user global energy variable) within the sUpdateMetabolism user program subroutine, which is executed once per second by the engine.



#### 4.4.33 Retreat\_Bead(nRetreatZone, nTimes)

Sub index: 85

Parameters: nRetreatZone Goal range for target, 1 = cAnywhere, 2 = cThere  
nTimes Number of iterations to make of the retreat loop, 1-32767

Locals: none

Resources: Motors

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Tries to move a Spybot away from the current target. If there is no target the motors float. If the target becomes outside the nRetreatZone, the bead exits. Otherwise the motors drive the Spybot backwards away from the target. The subroutine repeats the Retreat loop nTimes.

Notes:



#### 4.4.34 Send\_Bead(nTarget, nMessage, nHiByte, nLoByte)

Sub index: 66

Parameters: nTarget Spybot index in world table for message, 0 - 15, 0x40 or 0x80  
nMessage message to send, 0 - 255  
nHiByte first byte of message data, 0 - 255  
nLoByte second byte of message data, 0 - 255

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SendMessage

Description: Sends a message to the specified Spybot. See SendMessage for more details.

Notes:



#### 4.4.35 SendAbility\_Bead(nStrength)

Sub index: 103

Parameters: nStrength strength of special ability message (sent as lo byte), 0- 255

Locals: nMessage message to send

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SendAllRangeMessage

Description: Sends one of the species specific cCommandTypeReactTo messages to all Spybots in the world. See SendAllRangeMessage for more details.

Notes:



#### 4.4.36 SendAllRangeToken\_Bead(nTokenIndex, nMax, nMessage, nStrength, nSound)

Sub index: 98

Parameters: nTokenIndex unique token bead index, 1 - 1023  
nMax number of times this token bead can be used  
nMessage message to send, 0 - 255  
nStrength strength of message (sent in param2), 0 - 255  
nSound sound to play, 0 - 79; -1 = play no sound

Locals: pToken pointer to token in eeprom  
nTemp token value

Resources: sound

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, FindBeadToken, FindFreeToken, WriteToken, SendAllRangeMessage

Description: Sends a range message a limited number of times (nMax) to all Spybots in the world. See SendAllRangeMessage for more details. If the number of token shots has been used up, makes a small clicksound.

Notes:



#### 4.4.37 SendID\_Bead(nShortID, nMessage, nHiByte, nLoByte)

Sub index: 107

Parameters: nShortID Spybot ID to search world for, 8 - 255  
nMessage message to send, 0 - 255  
nHiByte first byte of message data, 0 - 255  
nLoByte second byte of message data, 0 - 255

Locals: nIndex world table index

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, EnterBead, ExitBead SendMessage

Description: Searches the world table for Spybot with nShortID, and sends nMessage with nHiByte, nLoByte parameters if it is found. See SendMessage for more details.

Notes:



#### 4.4.38 SendRCX\_Bead(nMessage)

Sub index: 112

Parameters: nMessage PBMessage to send to RCX, 1 -255

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SendRCXMessage

Description: Sends an RCX PBMessage. See SendRCXMessage for more details.

Notes:

**4.4.39 Set\_Bead(nProperty, nValue)**

Sub index: 88

Parameters: nProperty property to set – see table below  
nValue property parameter – see table below

Locals: none

Resources: Motors, LEDs

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, SetMaxPower, SetPings, SelectTarget, StopStateBeadTask, StartStateBeadTask, SetDisplay

<b>nProperty</b>	<b>Nvalue</b>
setGameMode = 0	Sets the ping mode (bits 2 –4 ) to nValue, 0 – 7
setSpeed = 1	Sets the maximum Spybot motor speed to one of: cSlowSpeed = 1, cNormalSpeed = 2, cFastSpeed = 3
setVisibility = 2	Sets the Spybots visibility (ping rate) to one of: cHighVisibility = 3, cNormalVisibility = 2 cLowVisibility = 1, cStealthVisibility = 0
setTargetType = 3	Sets the Spybot target type to one of: cStateTarget = -1 cNothing = 0                      cAnything = 1 cAnyController = 2              cAnyBot = 3 cMyController = 4                cNotMyController = 5
setTargetTypeID = 4	Sets the Spybot target type to cTargetID = 12; nValue is the shortID to match (0-255), and is placed in high byte of nTargetType
setTargetTeamTypeID = 5	Sets the Spybot target type to cTargetTeamID = 13, nValue is the linkID to match (0-7), and is placed in the high byte of nTargetType
setTargetTypeNote = 6	Sets the Spybot target type to cTargetNote = 14, nValue is the iNote value to match (0-15), and is placed in the high byte of nTargetType
setEnabledStatus = 7	Sets the bit(s) in nValue in nStatus. See Globals.h for nStatus bit allocations.
setDisableStatus = 8	Clears the bit(s) in nValue from nStatus. See Globals.h for nStatus bit allocations.
setSuspendStateBeads = 9	Suspends engine execution of state beads. nValue ignored.
setResumeStateBeads = 10	Restarts engine execution of state beads. nValue ignored.
setDisplayMode = 11	Sets the display mode to nValue: cDisplayState = 1 cDisplayRandom = 2 cDisplayNothing = 3 cDisplayOneVar = 0x11 cDisplayTwoVar = 0x12 cDisplayGameTimeRemaining = 0x13 cDisplayRadar = 0x21



<b>nProperty</b>	<b>Nvalue</b>
	cDisplayProximity = 0x22 cDisplayAnimation = 0x30 - 0x3f
setRCDisable = 12	nValue = 1, disables RC channel; nValue = 0 restores RC channel
setRCChannel = 13	Sets RC channel to nValue (0-3)
setTargetNote = 14	Sets the iNote value of the current target in the world relation table to 0-255.

Notes:



#### 4.4.40 SlidingTone\_Bead(nTone, nStep, nTimes)

Sub index: 91

Parameters: nTone frequency to play (Hz), see LASM play  
nStep change to nTone per loop iteration, -32768 - 32767  
nTimes loop iterations to perform, 1 - 32767

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Plays nTone for 50 mS, waits 40 mS, adjusts nTone by nStep, and repeat this nTimes. The effect is a gliding tone, starting at nTone, lasting approximately nStep \* 100 mS, finishing at nTone + nTimes \* nStep Hz.

Notes:



#### 4.4.41 SlowDownMovement\_Bead(nMovement, nTime)

Sub index: 110

Parameters: nMovement type of Spybot motion: see SlowDownMove  
nTime time for motion, 0 – 32767 (10mS steps)

Locals: none

Resources: Motors

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, SlowDownMove

Description: Monitors ExitBeadEvent; floats the motors on exit. Exits if another task with higher priority is already controlling the motors.

Notes:



#### 4.4.42 SpeedUpMovement\_Bead(nMovement, nTime)

Sub index: 111

Parameters: nMovement type of Spybot motion: see SpeedUpMove  
nTime time for motion, 0 – 32767 (10mS steps)

Locals: none

Resources: Motors

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead, SpeedUpMove

Description: Monitors ExitBeadEvent; floats the motors on exit. Exits if another task with higher priority is already controlling the motors.

Notes:



#### 4.4.43 TellToken\_Bead(nTokenIndex, nTarget, nMax, nMessage, nHiByte, nLoByte, nSound)

Sub index: 75

Parameters: nTokenIndex unique token bead index, 1 - 1023  
nTarget Spybot index in world table for message, 0 – 15, 0x40 or 0x80  
nMax number of times this token bead can be used  
nMessage message to send, 0 - 255  
nHiByte first byte of message data, 0 - 255  
nLoByte second byte of message data, 0 - 255  
nSound sound to play, 0 – 79; -1 = play no sound

Locals: pToken pointer to token in eeprom  
nTemp token value

Resources: sound

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, FindBeadToken, FindFreeToken, WriteToken, SendMessage

Description: Sends a message a limited number of times (nMax) to the specified Spybot. See SendMessage for more details. If the number of token shots has been used up, makes a small click sound.

Notes: In addition to sending a message to a specific Spybot, nTarget may also be cLinkcast (= 0x40) or cBroadcast (= 0x80).

**4.4.44 TimedToken\_Bead(nTokenIndex, nTokenType, nMax, nTime)**

Sub index: 73

Parameters: nTokenIndex unique token bead index, 1 – 1023  
nTokenType type of token, 1 - 255  
nMax number of times this token bead can be used, 1-63  
nTime time each token use lasts (seconds), 1 - 255

Locals: pToken pointer to token in eeprom  
nTemp token value

Resources: sound, motors

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, FindBeadToken, FindFreeToken, WriteToken, ActivateToken

Description: Each TimedToken\_Bead carries a limited number (nMax) of token uses; each time the token bead is used, the count is decremented (tokens are stored in an eeprom buffer). When there are no more token 'shots' left, the TimedToken\_Bead makes a small click sound. When a token is successfully activated, it lasts for a pre-defined time (nTime), after which it is automatically deactivated by the engine.

Only one token may be active at a time (stored in the global variable nActiveToken); if a new token is activated while another is already active, the new token overwrites the old one. Message and timed tokens may both be active at the same time.

Notes: See section 3.3 Tokens for pre-defined token action details.



#### 4.4.45 TokenMessage\_Bead(nTokenType , nTime)

Sub index: 114

Parameters: nTokenType type of token, 1 - 255  
nTime time each token use lasts (seconds), 1 - 255

Locals: none

Resources: sound, motors

Access control: none

Monitors: none

Calls: EnterBead, ExitBead, FindBeadToken, FindFreeToken, WriteToken, ActivateToken

Description: The TokenMessage\_Bead activates a message token in response to a controller or Spybot message. When a message token is activated, it lasts for a pre-defined time (nTime), after which it is automatically deactivated by the engine.

Only one message token may be active at a time (stored in the global variable nMessageToken); if a new token is activated while another is already active, the new token overwrites the old one.

Notes: See section 3.3 Tokens for pre-defined token action details.



#### 4.4.46 TurnAway\_Bead(nTimes)

Sub index: 83

Parameters: nTimes

Locals: none

Resources: motors

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Tries to turn the Spybot away from the target. If there is no target, or if the target is in the anywhere zone, the bead exits immediately. Otherwise the bead loops, turning the Spybot to the left or right until the target can no longer be seen in the here or there zones. The subroutine repeats the TurnAway loop nTimes.

Notes:



#### 4.4.47 TwoTone\_Bead(nHiTone, nLoTone, nTimes)

Sub index: 89

Parameters: nHiTone frequency to play (Hz), see LASM playt  
nLoTonefrequency to play (Hz), see LASM playt  
nTimes loop iterations to perform, 1 - 32767

Locals: none

Resources: sound

Access control: abort on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Repeats an alternating tone sequence (like an alarm) nTimes by playing nHiTone for 300 mS, waits 400 mS, plays nLoTone for 300 mS, waits 300 mS.

Notes:



#### 4.4.48 VarOp\_Bead(pVar, nOperation, nValue)

Sub index: 69

Parameters: pVar pointer to global variable to update, 0 - 31  
nOperation operation to perform, one of:  
varSet = 0  
varInc = 1  
varDec = 2  
varMul = 3  
varDiv = 4  
varRnd = 5  
nValue operand, range depends on operation:  
varSet, varInc, varDec, varMul, varDiv: -32768 - 32767  
varRnd: 2 - 160

Locals: none

Resources: none

Access control: none

Monitors: none

Calls: EnterBead, ExitBead

Description: Performs the specified operation on one of the global variables. For the varSet operation, the global variable is set randomly to a value from 1 to nValue.

Notes: No check is made for under- or over-flow, or for a valid global variable pointer.



#### 4.4.49 VLL\_Bead(nVLL)

Sub index: 101

Parameters: nVLL VLL command byte, 0 - 127

Locals: none

Resources: vll

Access control: retry on fail

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Sends a VLL command out from the Spybot's VLL diode.

Notes: Meaning of the VLL commands depends on the target, which could, for example, be a microScout or another Spybot.



#### 4.4.50 Wait\_Bead(nTime)

Sub index: 113

Parameters: nTime time to wait (10mS steps), 0 - 32767

Locals: none

Resources: none

Access control: none

Monitors: ExitBeadEvent - abort on event

Calls: EnterBead, ExitBead

Description: Waits for the specified time. No motor commands are issued.

Notes:



#### 4.5 User program subroutines

User program subroutines are generated from the UI and downloaded to eeprom. When the program starts, the engine calls these subroutines as necessary. Applications generating program subroutines should be careful to use a minimum of program stack space, and ensure the subroutines execute as quickly as possible (since eeprom is slow).

- sBumpAction
- sExtension(nExtension, var nParam)
- sGetDisplay(var pVar1, var pVar2)
- sGetPoints(var nPoints)
- sInitGame
- sInitState(nState)
- sPollGameWatcher
- sPollStateWatcher
- sProcessBotMessage
- sProcessControllerMessage
- sProcessVLLMessage
- sSpecialAbility
- sUpdateMetabolism



### 4.5.1 sBumpAction

Sub index: 140

Parameters: none

Locals: none

Description: Called by the bump event watcher in response to a bumper event, this subroutine typically implements a sequence of beads that will result in the Spybot avoiding obstacles.

Example:

```
sub sBumpAction
{
    BasicMovement_Bead(moveBackward, 60)
    BasicMovement_Bead(moveSpinLeft, 100)
}
```

**4.5.2 sExtension(nExtension, var nParam)**

Sub index: 137

Parameters: nExtension reason for call, 1 – 4 (see table below)  
nParam depends on nExtension, -32768 - 32767

Locals: none

Description: Called when one of the engine subroutines encounters an unknown (user) value, in order to provide a ROM extension mechanism.

nExtension	nParam	Calling subroutine	Description
extActivateToken = 1	(in) nTokenType	ActivateToken	unknown token activated
extTokenTick = 2	(in) nTokenType	TokenTick	unknown token tick
extDeactivateToken = 3	(in) nTokenType	DeactivateToken	unknown token deactiv ated
extDisplay = 4	always 0	UpdateDisplayTick	user display mode

```
Example: sub sExtension(nExtension, nParam)
        {
            if nExtension = extDisplay
            {
                LED[iDisplay] = random 63
            }
        }
```



### 4.5.3 sGetDisplay(var pVar1, var pVar2)

Sub index: 129

Parameters: pVar1 (out) depends on nDisplayMode  
pVar2 (out) depends on nDisplayMode

Locals: none

Description: Called by UpdateDisplayTick to get variable(s) or values used to control the LED display.

nDisplayMode	pVar1	pVar2
cDisplayOneVar = 0x11	pointer to global var, 0 - 31	not used
cDisplayTwoVar = 0x12	pointer to global var, 0 - 31	pointer to global var, 0 - 31
cDisplayGameTimeRemaining = 0x13	allocated game time (seconds) , 1 - 32767	not used

```

Example:  sub sGetDisplay(pVar1, pVar2)
           {
             pVar1 = @nEnergy
             pVar2 = -1
           }

```



#### 4.5.4 sGetPoints(var nPoints)

Sub index: 138

Parameters: nPoints (out) points scored for winning this game, 0 - 32767

Locals: none

Description: Called by GameResult\_Bead(cPingWonGame) when the Spybot has won a game, so that the number of game points can be calculated (depending, for example, on user program global variables). The result will be added to the eeprom ePoints value.

Example:

```
sub sGetPoints(nPoints)
{
    nPoints = 250
    nPoints -= tGameTimer
    nPoints *= nEnergy/200
}
```



#### 4.5.5 sInitGame

Sub index: 128

Parameters: none

Locals: none

Description: Called by MainSub before entering main engine loop, at a minimum this subroutine must initialise the global variables for the number of game watchers, and the display mode. Optionally other game specific initialisation may be performed here too.

Example: 

```
sub sInitGame
{
    nGameWatchers = cGameWatchers
    nDisplayMode = cDisplayOneVar
}
```



### 4.5.6 sInitState(nState)

Sub index: 132

Parameters: nState new state, usually 1 – 7, may also be one of:  
cGameOverState = -1 (may be ignored)  
cOutOfGameState = -2 (may be ignored)  
cStartState = 0

Locals: none

Description: Called by MainSub before entering main engine loop and by GotoNewState each time the engine changes state, at a minimum this subroutine must initialise the global variables for the number of state watchers and the state target type.

```
Example: sub sInitState(nState)
{
    select nState
    {
        when cStartState
        {
            nStateTargetType = cNothing
            nStateWatchers = 1
        }
        when gsAttack
        {
            nStateTargetType = cAnyBot
            nStateWatchers = 1
        }
        when gsRest
        {
            nStateTargetType = cAnyBot
            nStateWatchers = 1
        }
    }
}
```



#### 4.5.7 sPollGameWatcher

Sub index: 130

Parameters: none

Locals: none

Description: Called by MainSub in order to execute one of the game watchers. This subroutine should use the current value of the global game watcher variable (nGameWatcher) to select and execute user specified (conditional) beads.

```
Example: sub sPollGameWatcher
{
    select nGameWatcher
    {
        when gwab_Win?
        {
            if nEndGame = cAllBotsLost
            {
                GameResult_Bead(cPingWonGame)
                PlayLongTone_Bead(3200)
                Goto_Bead(cGameOverState)
            }
        }
        when gwab_Lose?
        {
            if nEnergy < 50
            {
                GameResult_Bead(cPingLostGame)
                PlayLongTone_Bead(1200)
                Goto_Bead(cGameOverState)
            }
        }
    }
}
```



#### 4.5.8 sPollStateWatcher

Sub index: 131

Parameters: none

Locals: none

Description: Called by MainSub in order to execute one of the state watchers. This subroutine should use the current value of the global state and state watcher variables (nState & nStateWatcher) to select and execute user specified (conditional) beads.

```
Example: sub sPollStateWatcher
{
    select nState
    {
        when cStartState
        {
            select nStateWatcher
            {
                when gwab_Start?
                {
                    if nEvents & cBumpEvent <> 0
                    {
                        Goto_Bead(gsAttack)
                    }
                }
            }
        }

        when gsAttack
        {
            select nStateWatcher
            {
                when gsAttack_gwab_Tired?
                {
                    if nEnergy < cStamina
                    {
                        Goto_Bead(gsRest)
                    }
                }
            }
        }

        when gsRest
        {
            select nStateWatcher
            {
                when gsRest_gwab_Recover?
                {
                    if nEnergy > cRecovery
```





#### 4.5.9 sProcessBotMessage

Sub index: 135

Parameters: none

Locals: nParam1 message hi byte value, 0 - 255  
nParam2 message lo byte value, 0 - 255

Description: Called by CheckPost (and MainSub when in the cOutOfGameState state) so that the user program can process messages from other Spybots. The message command is in the low byte of the global nMessage, the sender index in the world relation table is in the high byte of nMessage. The message value is in the global nMessageValue, which is pre-calculated into the locals nParam1 & nParam2.

Some messages and data values are predefined (in messages.h), others may be defined by user applications.

```
Example: sub sProcessBotMessage
        {
            local nParam1 = nMessageValue
            local nParam2 = nMessageValue

            nParam1 /= 256
            nParam2 &= 0xff

            select nMessage & cMessageCommandMask
            {
                when cCommandFireLaser
                {
                    if nActiveToken & cTokenTypeMask <> cShieldsToken
                    {
                        VarOp_Bead(@nEnergy, varDec, nParam2)
                        Fx_Bead(fxShudder, 6)
                    }
                }
            }
        }
```



#### 4.5.10 sProcessControllerMessage

Sub index: 133

Parameters: none

Locals: none

Description: Called by the engine ROM cControllerTask so that the user program can process messages from controllers. The controller button value is in the global nControllerButton.

Example:

```
sub sProcessControllerMessage
{
    select nControllerButton
    {
        when cControllerButton1
        {
            TimedToken_Bead(102, cShieldsToken, 3, 5)
        }
        when cControllerButton4
        {
            BoostToken_Bead(101, 3, @nEnergy, 200)
        }
    }
}
```



#### 4.5.11 sProcessVLLMessage

Sub index: 134

Parameters: none

Locals: none

Description: Called by CheckPost so that the user program can process VLL messages from other Spybots, PBricks or USB IR Transmitters. The VLL message is in the global nMessageValue (0 – 127).

Example:

```
sub sProcessVLLMessage
{
    select nMessageValue
    {
        when 1
        {
            PlayTone_Bead(880)
        }
    }
}
```



#### 4.5.12 sSpecialAbility

Sub index: 139

Parameters: none

Locals: none

Description: Called by CheckPost in response to a special ability message from another Spybot.

Example:

```
sub sSpecialAbility
{
    select nMessageValue /256 //param1 is range
    {
        when cHere
        {
            Fx_Bead(fxShocked, 40)
        }
        when cThere
        {
            Fx_Bead(fxShocked, 20)
        }
        when cAnywhere
        {
            Fx_Bead(fxShocked, 10)
        }
    }
}
```



### 4.5.13 sUpdateMetabolism

Sub index: 136

Parameters: none

Locals: none

Description: Called by MainSub once every second, this subroutine gives the user program an opportunity to adjust global metabolic variables. It may be empty or undefined.

Example: 

```
sub sUpdateMetabolism
{
    RestTick_Bead(@nEnergy, 2)
}
```