Todays date: 10-Mar-2004
This is RoBiOS version 6.0

Library of functions build into RoBiOS

The following describes the RoBIOS operating system library routines.
Newer versions of the RoBIOS software may differ from the functionality described below see the latest software documentation. See also timing information on RoBIOS routines.

In application files use:

```
#include "eyebot.h"
```

The following libraries are available in **ROM** for programming in C and are **automatically linked** when calling "gcc68" and the like (using librobi.a).
Note that there are also a number of libraries available which are *not listed here*, since they are not in ROM but in the *EyeBot* distribution (e.g. elaborate image processing library). They can also be linked with an application program, as shown in the demo programs provided.

Return Codes

Unless specifically noted otherwise, all routines return 0 when successfull, or a value !=0 when an error has occurred. Only very few routines support multiple return codes.

Image Processing

Basic image processing functions (library robios):

```
Data Types:
        /* image is 80x60 but has a border of 1 pixel */
        #define imagecolumns 82
        #define imagerows 62
```

```
        typedef BYTE image[imagerows][imagecolumns];
        typedef BYTE colimage[imagerows][imagecoulmns][3];

int IPLaplace (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Laplace operator is applied to the source image
                        and the result is written to the destination image

int IPSobel (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Sobel operator is applied to the source image
                        and the result is written to the destination image

int IPDither (image *src, image *dest);
        Input:          (src) source b/w image
        Output:         (dest) destination b/w image
        Semantics:      The Dithering operator with a 2x2 pattern is applied
                        to the source image and the result is written to the
                        destination image

int IPDiffer (image *current, image *last, image *dest);
        Input:          (current) the current b/w image
                        (last) the last read b/w image
        Output:         (dest) destination b/w image
        Semantics:      Calculate the grey level difference at each pixel
                        position between current and last image, and
                        store the result in destination.

int IPColor2Grey (colimage *src, image *dest);
        Input:          (src) source color image
        Output:         (dest) destination b/w image
        Semantics:      Convert RGB color image given as source to 8-bit
                        grey level image and store the result in
                        destination.
```

Advanced image processing functions are available as library *improc*. For detailed info see Improv web-page.

Key Input

Using the standard Unix "libc" library, it is possible to use standard C "scanf" commands to read key "characters" from the "keyboard".

```
int KEYGetBuf (char *buf);
        Input:          (buf) a pointer to one character
        Output:         (buf) the keycode is written into the buffer
                        Valid keycodes are: KEY1,KEY2,KEY3,KEY4 (keys
                        from left to right)
        Semantics:      Wait for a keypress and store the keycode into the
                        buffer

int KEYGet (void);
        Input:          NONE
        Output:         (return code) the keycode of a pressed key is returned
                        Valid keycodes are: KEY1,KEY2,KEY3,KEY4 (keys
                        from left to right)
        Semantics:      Wait for a keypress and return keycode

int KEYRead (void);
        Input:          NONE
```

```
        Output:         (return code) the keycode of a pressed key is
                        returned or 0 if no key is pressed.
                        Valid keycodes are: KEY1,KEY2,KEY3,KEY4 (keys
                        from left to right) or 0 for no key.
        Semantics:      Read keycode and return it. Function does not wait.

int KEYWait (int excode);
        Input:          (excode) the code of the key expected to be pressed
                        Valid keycodes are: KEY1,KEY2,KEY3,KEY4 (keys
                        from left to right) or ANYKEY.
        Output:         NONE
        Semantics:      Wait for a specific key
```

LCD Output

Using the standard Unix "libc" library, it is possible to use standard C "printf" commands to print on the LCD "screen". E.g. the "hello world" program works:

```
printf("Hello, World!\n");
```

The following routines can be used for specific output functions:

```
int LCDPrintf(const char format[], ...);
        Input:          format string and parameters
        Output:         NONE
        Semantics:      Prints text or numbers or combination of both
                        onto LCD. This is a simplified and smaller
                        version of standard Clib "printf".

int LCDSetPrintf(int row, int column, const char format[], ...);
        Input:          print position and format string with parameters
        Output:         NONE
        Semantics:      Prints text or numbers or combination of both
                        onto LCD at specified location.
                        Identical to calling LCDSetPos followed by LCDPrintf.

int LCDClear(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Clear the LCD

int LCDPutChar (char char);
        Input:          (char) the character to be written
        Output:         NONE
        Semantics:      Write the given character to the current cursor
                        position and increment cursor position

int LCDSetChar (int row,int column,char char);
        Input:          (char) the character to be written
                        (column) the number of the column
                        Valid values are: 0-15
                        (row) the number of the row
                        Valid values are: 0-6
        Output:         NONE
        Semantics:      Write the given character to the given display position

int LCDPutString (char *string);
        Input:          (string) the string to be written
        Output:         NONE
        Semantics:      Write the given string to the current cursor position
                        and increment cursor position
```

```
int LCDSetString (int row,int column,char *string);
        Input:          (string) the string to be written
                        (column) the number of the column
                        Valid values are: 0-15
                        (row) the number of the row
                        Valid values are: 0-6
        Output:         NONE
        Semantics:      Write the given string to the given display position

int LCDPutHex (int val);
        Input:          (val) the number to be written
        Output:         NONE
        Semantics:      Write the given number in hex format at current
                        cursor position

int LCDPutHex1 (int val);
        Input:          (val) the number to be written (single byte 0..255)
        Output:         NONE
        Semantics:      Write the given number as 1 hex-byte at current
                        cursor position

int LCDPutInt (int val);
        Input:          (val) the number to be written
        Output:         NONE
        Semantics:      Write the given number as decimal at current cursor
                        position

int LCDPutIntS (int val, int spaces);
        Input:          (val)    the number to be written
                        (spaces) the minimal number of print spaces
        Output:         NONE
        Semantics:      Write the given number as decimal at current cursor
                        position using extra spaces in front if necessary

int LCDPutFloat (float val);
        Input:          (val) the number to be written
        Output:         NONE
        Semantics:      Write the given number as floating point number
                        at current cursor position

int LCDPutFloatS (float val, int spaces, int decimals);
        Input:          (val)      the number to be written
                        (spaces)   the minimal number of print spaces
                        (decimals) the number of decimals after the point
        Output:         NONE
        Semantics:      Write the given number as a floating point number
                        at current cursor position using extra spaces in
                        front if necessary and with specified number of
                        decimals

int LCDMode (int mode);
        Input:          (mode) the display mode you want
                        Valid values are: (NON)SCROLLING|(NO)CURSOR
        Output:         NONE
        Semantics:      Set the display to the given mode
                        SCROLLING: the display will scroll up one
                                line, when the right bottom corner is
                                reached and the new cursor position
                                will be the first column of the now
                                blank bottom line
                        NONSCROLLING: display output will resume in
                                the top left corner when the bottom
                                right corner is reached
                        NOCURSOR: the blinking hardware cursor is not
```

```
                        displayed at the current cursor position
                CURSOR: the blinking hardware cursor is
                        displayed at the current cursor
                        position

int LCDSetPos (int row, int column);
        Input:          (column) the number of the column
                        Valid values are: 0-15
                        (row) the number of the row
                        Valid values are: 0-6
        Output:         NONE
        Semantics:      Set the cursor to the given position

int LCDGetPos (int *row, int *column);
        Input:          (column) pointer to the storing place for current
                        column.
                        (row) pointer to the storing place for current row.
        Output:         (*column) current column
                        Valid values are: 0-15
                        (row) current row
                        Valid values are: 0-6
        Semantics:      Return the current cursor position

int LCDPutGraphic (image *buf);
        Input:          (buf) pointer to a greyscale image (80*60 pixel)
        Output:         NONE
        Semantics:      Write the given graphic b/w to the display
                        it will be written starting in the top left corner
                        down to the menu line.  Only 80x54 pixels will
                        be written to the LCD, to avoid destroying the
                        menu line.

int LCDPutColorGraphic (colimage *buf);
        Input:          (buf) pointer to a color image (80*60 pixel)
        Output:         NONE
        Semantics:      Write the given graphic b/w to the display
                        it will be written starting in the top left corner
                        down to the menu line.  Only 80x54 pixels will
                        be written to the LCD, to avoid destroying the
                        menu line.  Note: The current implementation
                        destroys the image content.

int LCDPutImage (BYTE *buf);
        Input:          (buf) pointer to a b/w image (128*64 pixel)
        Output:         NONE
        Semantics:      Write the given graphic b/w to the whole display.

int LCDMenu (char *string1, char *string2, char *string3,char *string4);
        Input:          (string1) menu entry above key1
                        (string2) menu entry above key2
                        (string3) menu entry above key3
                        (string4) menu entry above key4
                        Valid Values are:
                          - a string with max 4 characters, which
                            clears the menu entry and writes the new one
                          - "" : leave the menu entry untouched
                          - " " : clear the menu entry
        Output:         NONE
        Semantics:      Fill the menu line with the given menu entries

int LCDMenuI (int pos, char *string);
        Input:          (pos) number of menu entry to be exchanged (1..4)
                        (string) menu entry above key <pos> a string
                        with max 4 characters
```

```
        Output:         NONE
        Semantics:      Overwrite the menu line entry at position pos with
                        the given string

int LCDSetPixel (int row, int col, int val);
        Input:          (val) pixel operation code
                        Valid codes are:  0 = clear pixel
                                          1 = set pixel
                                          2 = invert pixel
                        (column) the number of the column
                        Valid values are: 0-127
                        (row) the number of the row
                        Valid values are: 0-63
        Output:         NONE
        Semantics:      Apply the given operation to the given pixel
                        position.  LCDSetPixel(row, col, 2) is the same
                        as LCDInvertPixel(row, col).

int LCDInvertPixel (int row, int col);
        Input:          (column) the number of the column
                        Valid values are: 0-127
                        (row) the number of the row
                        Valid values are: 0-63
        Output:         NONE
        Semantics:      Invert the pixel at the given pixel position.
                        LCDInvertPixel(row, col) is the same as
                        LCDSetPixel(row, col, 2).

int LCDGetPixel (int row, int col);
        Input:          (column) the number of the column
                        Valid values are: 0-127
                        (row) the number of the row
                        Valid values are: 0-63
        Output:         (return code) the value of the pixel
                        Valid values are:       1 for set pixel
                                                0 for clear pixel
        Semantics:      Return the value of the pixel at the given
                        position

int LCDLine(int x1, int y1, int x2, int y2, int col)
        Input:          (x1,y1) (x2,y2) and color
        Output:         NONE
        Semantics:      Draw a line from (x1,y1) to (x2,y2) using the Bresenham A
                        top    left  is   0, 0
                        bottom right is 127,63
                        color: 0 white
                               1 black
                               2 negate image contents

int LCDArea(int x1, int y1, int x2, int y2, int col)
        Input:          (x1,y1) (x2,y2) and color
        Output:         NONE
        Semantics:      Fill rectangular area from (x1,y1) to (x2,y2)
                        it must hold: x1 < x2 AND y1 < y2
                        top    left  is   0, 0
                        bottom right is 127,63
                        color: 0 white
                               1 black
                               2 negate image contents
```

Camera

The following functions handle initializing and image reading from either grayscale or color camera:

```
int CAMInit (int mode);
        Input:          (mode) camera initialization mode
                        Valid Values are: NORMAL
        Output:         (return code) Cameraversion or Errorcode
                        Valid values: 255   = no camera connected
                                      254   = camera init error
                                      0     = QuickCam V1 grayscale
                                      16    = QuickCam V2 color
                                      17    = EyeCam-1 (6300)
                                      18    = EyeCam-2 (7620)
                                      19    = EyeCam-3 (6620)
        Semantics:      Reset and initialize connected camera
        Note:           [Previously used to set zoom factor on Quickcam: WIDE,NOR

int CAMRelease (void);
        Input:          NONE
        Output:         (return code) 0 = success
                                     -1 = error
        Semantics:      Release all resources allocated using CAMInit().

int CAMGetFrame (image *buf);
        Input:          (buf) a pointer to a grey scale image
        Output:         NONE
        Semantics:      Read an image size 62x82 from grey scale camera.
                        Return 8 bit gray values 0 (black) .. 255 (white)

int CAMGetColFrame (colimage *buf, int convert);
        Input:          (buf) a pointer to a color image
                        (convert) flag if image should be reduced to 8 bit gray
                                  0 = get 24bit color image
                                  1 = get 8bit grayscale image
        Output:         NONE
        Semantics:      Read an image size 82x62 from color cam and reduce it
                        if required to 8 bit gray scale.
        Note:           - buf needs to be a pointer to 'image'
                        - enable conversion like this:
                                image buffer;
                                CAMGetColFrame((colimage*)&buffer, 1);

int CAMGetFrameMono (BYTE *buf);
        Note:           This function works only for EyeCam
        Input:          (buf) pointer to image buffer of full size (use CAMGet)
        Output:         (return code) 0 = success
                                     -1 = error (camera not initialized)
        Semantics:      Reads one full gray scale image (e.g. 82x62, 176x144, 320
                        depending on camera module

int CAMGetFrameRGB (BYTE *buf);
        Note:           This function works only for EyeCam
        Input:          (buf) pointer to image buffer of full size (use CAMGet)
        Output:         (return code) 0 = success
                                     -1 = error (camera not initialized)
        Semantics:      Reads one full color image in RBG format, 3 bytes per pix
                        (e.g. 82x62*3, 176x144*3, 320x240*3) depending on camera

int CAMGetFrameBayer (BYTE *buf);
        Note:           This function works only for EyeCam
        Input:          (buf) pointer to image buffer of full size (use CAMGet)
        Output:         (return code) 0 = success
                                     -1 = error (camera not initialized)
        Semantics:      Reads one full color image in Bayer format, 4 bytes per p
                        (e.g. 82x62*4, 88x72*4, 160x120*4) depending on camera mo

int CAMSet (int para1, int para2, int para3);
```

```
        Note:           parameters have different meanings for different cameras
        Input:QuickCam  (para1) camera brightness
                        (para2) camera offset (b/w camera) / hue (color camera)
                        (para3) contrast (b/w camera) / saturation (color camera)
                        Valid values are: 0-255
              ------------------------------------------------
              EyeCam    (para1) frame rate in frames per second
                        (para2) not used
                        (para3) not used
                        Valid values are: FPS60, FPS30, FPS15,
                         FPS7_5, FPS3_75, FPS1_875, FPS0_9375, and FPS0_46875.
                         For the VV6300/VV6301, the default is FPS7_5.
                         For the OV6620, the default is FPS1_875.
                         For the OV7620, the default is FPS0_48375.
        Output:         NONE
        Semantics:      Set camera parameters

int CAMGet (int *para1, int *para2 ,int *para3);
        Note:           parameters have different meanings for different cameras
        Input:QuickCam  (para1) pointer for camera brightness
                        (para2) pointer for offset (b/w camera) / hue (color came
                        (para3) pointer for contrast (b/w camera) / saturation (c
                        Valid values are: 0-255
              ------------------------------------------------
              EyeCam    (para1) frame rate in frames per second
                        (para2) full image width
                        (para3) full image height
        Output:         NONE
        Semantics:      Get camera hardware parameters

int CAMMode (int mode);
        Input:          (mode) the camera mode you want
                        Valid values are: (NO)AUTOBRIGHTNESS
        Output:         NONE
        Semantics:      Set the display to the given mode
                        AUTOBRIGHTNESS: the brightness value of the
                          camera is automatically adjusted
                        NOAUTOBRIGHTNESS: the brightness value is not
                          automatically adjusted
                        This function is not implemented in the
                        FIFO-enabled EyeCam driver.
```

System Functions

```
miscellaneous:
--------------
char *OSVersion(void);
        Input:          NONE
        Output:         OS version
        Semantics:      Returns string containing running RoBIOS version.
        Example:        "3.1b"

int OSError(char *msg,int number,BOOL dead);
        Input:          (msg) pointer to message
                        (number) int number
                        (dead) switch to choose deadend or keywait
                        Valid values are:       0 = no deadend
                                                1 = deadend
        Output:         NONE
        Semantics:      Print message and number to display then
                        stop processor (deadend) or wait for key

int OSMachineType(void);
```

```
        Input:          NONE
        Output:         Type of used hardware
            Valid values are:
            VEHICLE, PLATFORM, WALKER
        Semantics:      Inform the user in which environment the program runs.

int OSMachineSpeed(void);
        Input:          NONE
        Output:         actual clockrate of CPU in Hz
        Semantics:      Inform the user how fast the processor runs.

char* OSMachineName(void);
        Input:          NONE
        Output:         Name of actual Eyebot
        Semantics:      Inform the user with which name the Eyebot is
                        titled (entered in HDT).

unsigned char OSMachineID(void);
        Input:          NONE
        Output:         ID of actual Eyebot
        Semantics:      Inform the user with which ID the Eyebot is titled
                        (entered in HDT).

interrupts:
-----------
int OSEnable (void);
        Input:          NONE
        Output:         NONE
        Semantics:      Enable all cpu-interrupts

int OSDisable (void);
        Input:          NONE
        Output:         NONE
        Semantics:      Disable all cpu-interrupts

variable save to tpuram:
------------------------
int OSGetVar(int num);
*)
        Input:          (num) number of tpupram save location
                        Valid values are:       SAVEVAR1-4 for word saving
                                                SAVEVAR1a-4a/1b-4b for byte savin

        Output:         (return code) the value saved
                        Valid values are:       0-65535 for word saving
                                                0-255 for byte saving
        Semantics:      Get the value from the given save location

int OSPutVar(int num, int value);
*)
        Input:          (num) number of tpupram save location
                        Valid values are:       SAVEVAR1-4 for word saving
                                                SAVEVAR1a-4a/1b-4b for byte savin
                        (value) value to be stored
                        Valid values are:       0-65535 for word saving
                                                0-255 for byte saving

        Output:         NONE
        Semantics:      Save the value to the given save location

*) SAVEVAR1-3 already occupied by RoBiOS
```

Multitasking

```
int OSMTInit(BYTE mode);
        Input:          (mode) operation mode
                        Valid values are: COOP=DEFAULT,PREEMPT
        Output:         NONE
        Semantics:      Initialize multithreading environment

struct tcb *OSSpawn (char *name, void (*code)(void), int stksiz, int pri, int uid
        Input:          (name) pointer to thread name
                        (code) thread start address
                        (stksize) size of thread stack
                        (pri) thread priority
                        Valid values are: MINPRI-MAXPRI
                        (uid) thread user id
        Output:         (return code) pointer to initialized thread
                        control block
        Semantics:      Initialize new thread, tcb is initialized and
                        inserted in scheduler queue but not set to
                        READY

int OSMTStatus(void);
        Input:          NONE
        Output:         PREEMPT, COOP, NOTASK
        Semantics:      returns actual multitasking mode (preemptive,
                        cooperative or sequential)

int OSReady(struct tcb *thread);
        Input:          (thread) pointer to thread control block
        Output:         NONE
        Semantics:      Set status of given thread to READY

int OSSuspend(struct tcb *thread);
        Input:          (thread) pointer to thread control block
        Output:         NONE
        Semantics:      Set status of given thread to SUSPEND

int OSReschedule(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Choose new current thread

int OSYield(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Suspend current thread and reschedule

int OSRun(struct tcb *thread);
        Input:          (thread) pointer to thread control block
        Output:         NONE
        Semantics:      READY given thread and reschedule

int OSGetUID(thread);
        Input:          (thread) pointer to thread control block
                                 (tcb *)0 for current thread
        Output:         (return code) UID of thread
        Semantics:      Get the UID of the given thread

int OSKill(struct tcb *thread);
        Input:          (thread) pointer to thread control block
        Output:         NONE
        Semantics:      Remove given thread and reschedule

int OSExit(int code);
        Input:          (code) exit code
        Output:         NONE
```

```
        Semantics:      Kill current thread with given exit code and message

int OSPanic(char *msg);
        Input:          (msg) pointer to message text
        Output:         NONE
        Semantics:      Dead end multithreading error, print message to display
                        and stop processor

int OSSleep(int n)
        Input:          (n) number of 1/100 secs to sleep
        Output:         NONE
        Semantics:      Let current thread sleep for at least n*1/100
                        seconds.  In multithreaded mode, this will
                        reschedule another thread.  Outside
                        multi-threaded mode, it will call OSWait().

int OSForbid(void)
        Input:          NONE
        Output:         NONE
        Semantics:      disable thread switching in preemptive mode

int OSPermit(void)
        Input:          NONE
        Output:         NONE
        Semantics:      enable thread switching in preemptive mode

In the functions described above the parameter "thread" can always be
a pointer to a tcb or 0 for current thread.
```

Semaphores

```
int OSSemInit(struct sem *sem,int val);
        Input:          (sem) pointer to a semaphore
                        (val) start value
        Output:         NONE
        Semantics:      Initialize semaphore with given start value

int OSSemP(struct sem *sem);
        Input:          (sem) pointer to a semaphore
        Output:         NONE
        Semantics:      Do semaphore P (down) operation

int OSSemV(struct sem *sem);
        Input:          (sem) pointer to a semaphore
        Output:         NONE
        Semantics:      Do semaphore V (up) operation
```

Timer

```
int OSSetTime(int hrs,int mins,int secs);
        Input:          (hrs) value for hours
                        (mins) value for minutes
                        (secs) value for seconds
        Output:         NONE
        Semantics:      Set system clock to given time

int OSGetTime(int *hrs,int *mins,int *secs,int *ticks);
        Input:          (hrs) pointer to int for hours
                        (mins) pointer to int for minutes
                        (secs) pointer to int for seconds
                        (ticks) pointer to int for ticks
        Output:         (hrs) value of hours
                        (mins) value of minutes
```

```
                        (secs) value of seconds
                        (ticks) value of ticks
        Semantics:      Get system time, one second has 100 ticks

int OSShowTime(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Print system time to display

int OSGetCount(void);
        Input:          NONE
        Output:         (return code) number of 1/100 seconds since last reset
        Semantics:      Get the number of 1/100 seconds since last reset.
                        Type int is 32 bits, so this value will wrap
                        around after ~248 days.

int OSWait (int n);
        Input:          (n) time to wait
        Output:         NONE
        Semantics:      Busy loop for n*1/100 seconds.


timer-irq:
----------
TimerHandle OSAttachTimer(int scale, TimerFnc function);
        Input:          (scale) prescale value for 100Hz Timer (1 to ...)
                        (TimerFnc) function to be called periodically
        Output:         (TimerHandle) handle to reference the IRQ-slot
                        A value of 0 indicates an error due to a full list(max. 1
        Semantics:      Attach irq-routine (void function(void)) to the irq-list.
                        The scale parameter adjusts the call frequency (100/scale
                        of this routine to allow many different applications.
        Note:           Execution time of any attached routine (and total time of
                        all attached routines) has to be significantly < 10ms.
                        Otherwise timer interrupts will be missed and motor/senso
                        timing gets corrupted.

int OSDetachTimer(TimerHandle handle)
        Input:          (handle) handle of a previous installed timer irq
        Output:         0 = handle not valid
                        1 = function successfully removed from timer irq list
        Semantics:      Detach a previously installed irq-routine from the irq-li
```

Download and RS-232

```
int OSDownload(char *name,int *bytes,int baud,int handshake,int interface); **)
        Input:          (name) pointer to program name array
                               (bytes) pointer to bytes transferred int
                               (baud) baud rate selection
                               Valid values are: SER4800,SER9600,SER19200,SER384
                               SER115200(only SERIAL2-3, SERIAL1 depending on CP
                               (handshake) handshake selection
                               Valid values are: NONE, RTSCTS, IRDA (IRDA only SI
                               (interface): serial interface
                               Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = no error, download incomplete - call again
                        99 = download complete
                         1 = receive timeout error
                         2 = receive status error
                         3 = send timeout error
                         5 = srec checksum error
                         6 = user canceled error
```

```
                         7 = unknown srecord error
                         8 = illegal baud rate error
                         9 = illegal startadr. error
                        10 = illegal interface

        Semantics:      Load user program with the given serial setting
                        and get name of program.  This function must
                        be called in a loop until the return code is
                        !=0. In the loop the bytes that have been
                        transferred already can be calculated from the
                        bytes that have been transferred in this round.

int OSInitRS232(int baud,int handshake,int interface);
        Input:          (baud) baud rate selection
                               Valid values are:
                               SER4800,SER9600,SER19200,SER38400,SER57600,SER115
                        (handshake) handshake selection
                               Valid values are: NONE,RTSCTS, IRDA (IRDA only SE
                        (interface) serial interface
                               Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = ok
                         8 = illegal baud rate error
                        10 = illegal interface
        Semantics:      Initialize rs232 with given setting

int OSSendCharRS232(char chr,int interface);
        Input:          (chr) character to send
                        (interface) serial interface
                                    Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = good
                         3 = send timeout error
                        10 = illegal interface
        Semantics:      Send a character over rs232

int OSSendRS232(char *chr,int interface);
        Input:          (chr) pointer to character to send
                        (interface) serial interface
                                    Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = good
                         3 = send timeout error
                        10 = illegal interface
        Semantics:      Send a character over rs232.  Use OSSendCharRS232()
                        instead.  This function will be removed in the future.


int OSRecvRS232(char *buf,int interface);
        Input:          (buf) pointer to a character array
                               (interface) serial interface
                               Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = good
                         1 = receive timeout error
                         2 = receive status error
                        10 = illegal interface
        Semantics:      Receive a character over rs232

int OSFlushInRS232(int interface);
        Input:          (interface) serial interface
                               Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = good
```

```
                        10 = illegal interface
        Semantics:      resets status of receiver and flushes its
                        FIFO. Very useful in NOHANDSHAKE-mode to bring
                        the FIFO in a defined condition before
                        starting to receive

int OSFlushOutRS232(int interface);
        Input:          (interface) serial interface
                                Valid values are: SERIAL1-3
        Output:         (return code)
                         0 = good
                        10 = illegal interface
        Semantics:      flushes the transmitter-FIFO.  Very useful to abort
                        current transmission to host (ex: in the case
                        of a not responding host)

int OSCheckInRS232(int interface);
        Input:          (interface) serial interface
                                Valid values are: SERIAL1-3
        Output:         (return code) >0 : the number of chars currently availabl
                                    <0 : 0xffffff02 receive status error (no cha
                                         0xffffff0a illegal interface
        Semantics:      useful to read out only packages of a certain size

int OSCheckOutRS232(int interface);
        Input:          (interface) serial interface
                                Valid values are: SERIAL1-3
        Output:         (return code) >0 : the number of chars currently waiting
                                    <0 : 0xffffff0a illegal interface
        Semantics:      useful to test if the host is receiving properly
                                or to time transmission of packages in the speed
                                host can keep up with

int USRStart(void);                                                          **)
        Input:          NONE
        Output:         NONE
        Semantics:      Start loaded user program.

int USRResident(char *name, BOOL mode);                                      **)
        Input:          (name) pointer to name array
                        (mode) mode
                        Valid values are: SET,GET
        Output:         NONE
        Semantics:      Make loaded user program reset resistant
                        SET     save startaddress and program name.
                        GET     restore startaddress and program name.

**) this function must not be used in user programs !!!!
```

Audio

Sampleformat: WAV or AU/SND (8bit, pwm or mulaw)
Samplerate: 5461, 6553, 8192, 10922, 16384, 32768 (Hz)
Tonerange: 65 Hz to 21000 Hz
Tonelength: 1 msec to 65535 msecs

```
int AUPlaySample(char* sample);
        Input:          (sample) pointer to sample data
        Output:         (return code) playfrequency for given sample
                                0 if unsupported sampletype
        Semantics:      Plays a given sample (nonblocking)
                        supported formats are:
```

```
                        WAV or AU/SND  (8bit, pwm or mulaw)
                        5461, 6553, 8192, 10922, 16384, 32768 (Hz)

int AUCheckSample(void);
        Input:          NONE
        Output:         FALSE while sample is playing
        Semantics:      nonblocking test for sampleend

int AUTone(int freq, int msec);
        Input:          (freq) tone frequency
                        (msecs) tone length
        Output:         NONE
        Semantics:      Plays tone with given frequency for the given
                        time (nonblocking)
                        supported formats are:
                        freq = 65 Hz to 21000 Hz
                        msecs = 1 msec to 65535 msecs

int AUCheckTone(void);
        Input:          NONE
        Output:         FALSE while tone is playing
        Semantics:      nonblocking test for toneend

int AUBeep(void);
        Input:          NONE
        Output:         NONE
        Semantics:      BEEP!

int AURecordSample(BYTE* buf, long len, long freq);
        Input:          (buf) pointer to buffer
                        (len) bytes to sample + 28 bytes header
                        (freq) desired samplefrequency
        Output:         (return code) real samplefrequency
        Semantics:      Samples from microphone into buffer with given
                        frequency (nonblocking)
                        Recordformat: AU/SND (pwm) with unsigned 8bit samples

int AUCheckRecord(void);
        Input:          NONE
        Output:         FALSE while recording
        Semantics:      nonblocking test for recordend


int AUCaptureMic(void);
        Input:          NONE
        Output:         (return code) microphone value (10bit)
        Semantics:      Get microphone input value
```

PSD Sensors

Position Sensitive Devices (PSDs) use infrared beams to measure distance. The accuracy varies from sensor to sensor, and they need to be calibrated in the HDT to get correct distance readings.

```
PSDHandle PSDInit(DeviceSemantics semantics);
        Input:          (semantics) unique definition for desired PSD (see hdt.h)
        Output:         (return code) unique handle for all further operations
        Semantics:      Initialize single PSD with given semantics
                        Up to 8 PSDs can be initialized

int PSDRelease(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Stops all measurings and releases all initialized
```

```
                        PSDs

int PSDStart(PSDHandle bitmask, BOOL cycle);

        Input:          (bitmask) bitwise-or of all handles to which parallel
                                  measuring should be applied
                        (cycle)   TRUE  = continuous measuring
                                  FALSE = single measuring
        Output:         (return code) status of start-request
                                  -1 = error (false handle)
                                   0 = ok
                                   1 = busy (another measuring blocks driver)
        Semantics:      Starts a single/continuous PSD-measuring.  Continuous
                        gives new measurement ca. every 60ms.

int PSDStop(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Stops actual continuous PSD-measuring after
                        completion of the current shot

BOOL PSDCheck(void);
        Input:          NONE
        Output:         (return code) TRUE if a valid result is available
        Semantics:      nonblocking test if a valid PSD-result is available

int PSDGet(PSDHandle handle);
        Input:          (handle) handle of the desired PSD
                                 0 for timestamp of actual measure-cycle
        Output:         (return code) actual distance in mm (converted through
                        internal table)
        Semantics:      Delivers actual timestamp or distance measured by
                        the selected PSD.  If the raw reading is out of
                        range for the given sensor, PSD_OUT_OF_RANGE(=9999)
                        is returned.

int PSDGetRaw(PSDHandle handle);
        Input:          (handle) handle of the desired PSD
                                 0 for timestamp of actual measure-cycle
        Output:         (return code) actual raw-data (not converted)
        Semantics:      Delivers actual timestamp or raw-data measured by
                        the selected PSD
```

Servos and Motors

```
ServoHandle SERVOInit(DeviceSemantics semantics);
        Input:          (semantics) semantic (see hdt.h)
        Output:         (return code) ServoHandle
        Semantics:      Initialize given servo

int SERVORelease (ServoHandle handle)
        Input:          (handle) bitwise-or of all ServoHandles which should be r
        Output:         (return code) 0 = ok
                                      errors (nothing is released):
                                       0x11110000 = totally wrong handle
                                       0x0000xxxx = the handle parameter in which
                                         remained set that are connected to a rele
        Semantics:      Release given servos

int SERVOSet (ServoHandle handle,int angle);
        Input:          (handle) bitwise-or of all ServoHandles which should be s
                        (angle)  servo angle
                                 valid values: 0-255
```

```
        Output:         (return code) 0 = ok
                                      -1 = error wrong handle
        Semantics:      Set the given servos to the same given angle

MotorHandle MOTORInit(DeviceSemantics semantics);
        Input:          (semantics) semantic (see hdt.h)
        Output:         (return code) MotorHandle
        Semantics:      Initialize given motor

int MOTORRelease (MotorHandle handle)
        Input:          (handle) logical-or of all MotorHandles which should be r
        Output:         (return code) 0 = ok
                                     errors (nothing is released):
                                      0x11110000 = totally wrong handle
                                      0x0000xxxx = the handle parameter in which
                                      remained set that are connected to a releas
        Semantics:      Release given motor

int MOTORDrive (MotorHandle handle,int speed);
        Input:          (handle) logical-or of all MotorHandles which should be d
                        (speed) motor speed in percent
                        Valid values: -100 - 100 (full backward to full forward)
                                       0 for full stop
        Output:         (return code)  0 = ok
                                      -1 = error wrong handle
        Semantics:      Set the given motors to the same given speed


QuadHandle QUADInit(DeviceSemantics semantics);
        Input:          (semantics) semantic
        Output:         (return code) QuadHandle or 0 for error
        Semantics:      Initialize given Quadrature-Decoder (up to 8 decoders are

int QUADRelease(QuadHandle handle);
        Input:          (handle) logical-or of decoder-handles to be released
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Release one or more Quadrature-Decoder

int QUADReset(QuadHandle handle);
        Input:          (handle) logical-or of decoder-handles to be reseted
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Reset one or more Quadrature-Decoder

int QUADRead(QuadHandle handle);
        Input:          (handle) ONE decoder-handle
        Output:         32bit counter-value (-2^31 .. 2^31-1)
        Semantics:      Read actual Quadrature-Decoder counter, initially zero.
                        Note: A wrong handle will ALSO result in a 0 counter valu

DeviceSemantics QUADGetMotor(DeviceSemantics semantics);
        Input:          (handle) ONE decoder-handle
        Output:         semantic of the corresponding motor
                        0 = wrong handle
        Semantics:      Get the semantic of the corresponding motor

float QUADODORead(QuadHandle handle);
        Input:          (handle) ONE decoder-handle
        Output:         meters since last odometer-reset
        Semantics:      Get the distance from the last reset point of a single mo
                        This is not the overall distance driven since the last re
                        but the distance to the start point.
```

```
int QUADODOReset(QuadHandle handle);
        Input:          (handle) logical-or of decoder-handles to be reseted
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Resets the simple odometer(s) to define the startpoint
```

V-Omega Driving Interface

This is a high level wheel control API using the motor and quad primitives to drive the robot.

```
Data Types:
        typedef float meterPerSec;
        typedef float radPerSec;
        typedef float meter;
        typedef float radians;

        typedef struct
        { meter x;
          meter y;
          radians phi;
        } PositionType;

        typedef struct
        { meterPerSec v;
          radPerSec w;
        } SpeedType;

VWHandle VWInit(DeviceSemantics semantics, int Timescale);
        Input:          (semantics) semantic
                        (Timescale) prescale value for 100Hz IRQ (1 to ...)
        Output:         (return code) VWHandle or 0 for error
        Semantics:      Initialize given VW-Driver (only 1 can be initialized!)
                        The motors and encoders are automatically reserved!!
                        The Timescale allows to adjust the tradeoff between
                        accuracy (scale=1, update at 100Hz) and speed(scale>1, up
                        at 100/scale Hz).

int VWRelease(VWHandle handle);
        Input:          (handle) VWHandle to be released
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Release VW-Driver, stop motors

int VWSetSpeed(VWHandle handle, meterPerSec v, radPerSec w);
        Input:          (handle) ONE VWHandle
                        (v) new linear speed
                        (w) new rotation speed
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Set the new speed: v(m/s) and w(rad/s not degree/s)

int VWGetSpeed(VWHandle handle, SpeedType* vw);
        Input:          (handle) ONE VWHandle
                        (vw) pointer to record to store actual v, w values
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Get the actual speed: v(m/s) and w(rad/s not degree/s)

int VWSetPosition(VWHandle handle, meter x, meter y, radians phi);
        Input:          (handle) ONE VWHandle
                        (x) new x-position
                        (y) new y-position
                        (phi) new heading
```

```
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Set the new position: x(m), y(m) phi(rad not degree)

int VWGetPosition(VWHandle handle, PositionType* pos);
        Input:          (handle) ONE VWHandle
                        (pos) pointer to record to store actual position (x,y,phi
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Get the actual position: x(m), y(m) phi(rad not degree)

int VWStartControl(VWHandle handle, float Vv, float Tv, float Vw, float Tw);
        Input:          (handle) ONE VWHandle
                        (Vv) the parameter for the proportional component of the
                        (Tv) the parameter for the integrating component of the v
                        (Vw) the parameter for the proportional component of the
                        (Tv) the parameter for the integrating component of the w
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Enable the PI-controller for the vw-interface and set the
                        As default the PI-controller is deactivated when the vw-i
                        initialized. The controller tries to keep the desired spe
                        VWSetSpeed) stable by adapting the energy of the involved
                        The parameters for the controller have to be choosen care
                        The formula for the controller is:
                                                    t
                        new(t) = V*(diff(t) + 1/T * Int( diff(t)dt )
                                                    0
                        Recommended setting:  VWStartControl(vw, 7.0, 0.3, 7.0, 0
                        V: a value typically around 7.0
                        T: a value typically between 0 and 1.0
                        After enabling the controller the last set speed (VWSetSp
                        taken as the speed to be held stable.

int VWStopControl(VWHandle handle);
        Input:          (handle) ONE VWHandle
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Disable the controller immediately. The vw-interface cont
                        with the last valid speed of the controller.

int VWDriveStraight(VWHandle handle, meter delta, meterpersec v)
        Input:          (handle) ONE VWHandle
                        (delta)  distance to drive in m (pos. -> forward)
                                                        (neg. -> backward)
                        (v)      speed to drive with (always positive!)
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Drives distance "delta" with speed v straight ahead (forw
                        any subsequent call of VWDriveStraight, -Turn, -Curve or
                        while this one is still being executed, results in an imm
                        of this command

int VWDriveTurn(VWHandle handle, radians delta, radPerSec w)
        Input:          (handle) ONE VWHandle
                        (delta)  degree to turn in radians (pos. -> counter-clock
                                                           (neg. -> clockwise)
                        (w)      speed to turn with (always positive!)
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Turns about "delta" with speed w on the spot (clockwise o
                        any subsequent call of VWDriveStraight, -Turn, -Curve or
                        while this one is still being executed, results in an imm
                        of this command
```

```
int VWDriveCurve(VWHandle handle, meter delta_l, radians delta_phi, meterpersec v
        Input:          (handle)    ONE VWHandle
                        (delta_l)   length of curve_segment to drive in m (pos. -
                                                                          (neg. -
                        (delta_phi) degree to turn in radians (pos. -> counter-cl
                                                              (neg. -> clockwise)
                        (v)         speed to drive with (always positive!)
        Output:          0 = ok
                        -1 = error wrong handle
        Semantics:      Drives a curve segment of length "delta_l" with overall v
                        "delta_phi" with speed v (forw. or backw. / clockw. or co
                        any subsequent call of VWDriveStraight, -Turn, -Curve or
                        while this one is still being executed, results in an imm
                        of this command

float VWDriveRemain(VWHandle handle)
        Input:          (handle) ONE VWHandle
        Output:          0.0 = previous VWDriveX command has been completed
                         any other value = remaining distance to goal
        Semantics:      Remaining distance to goal set by VWDriveStraight, -Turn
                        (for -Curve only the remaining part of delta_l is reporte

int VWDriveDone(VWHandle handle)
        Input:          (handle) ONE VWHandle
        Output:         -1 = error wrong handle
                         0 = vehicle is still in motion
                         1 = previous VWDriveX command has been completed
        Semantics:      Checks if previous VWDriveX() command has been completed

int VWDriveWait(VWHandle handle)
        Input:          (handle) ONE VWHandle
        Output:         -1 = error wrong handle
                         0 = previous VWDriveX command has been completed
        Semantics:      Blocks the calling process until the previous VWDriveX()

int VWStalled(VWHandle handle)
        Input:          (handle) ONE VWHandle
        Output:         -1 = error wrong handle
                         0 = vehicle is still in motion or no motion command is a
                         1 = at least one vehicle motor is stalled during VW driv
        Semantics:      Checks if at least one of the vehicle's motors is stalled
```

Bumper / Infrared Sensors

This is for obsolete sensors only available on older Robot models.

```
BumpHandle BUMPInit(DeviceSemantics semantics);
        Input:          (semantics) semantic
        Output:         (return code) BumpHandle or 0 for error
        Semantics:      Initialize given bumper (up to 16 bumpers are possible)

int BUMPRelease(BumpHandle handle);
        Input:          (handle) logical-or of bumper-handles to be released
        Output:         (return code)
                        0 = ok
                        errors (nothing is released):
                         0x11110000 = totally wrong handle
                         0x0000xxxx = the handle parameter in which only those bi
                                      set that are connected to a releasable TPU-
        Semantics:      Release one or more bumper
```

```
int BUMPCheck(BumpHandle handle, int* timestamp);
        Input:          (handle) ONE bumper-handle
                        (timestamp) pointer to an int where the timestamp is plac
        Output:         (return code)
                         0 = bump occurred, in *timestamp is now a valid stamp
                        -1 = no bump occurred or wrong handle, *timestamp is clea
        Semantics:      Check occurrence of a single bump and return the timestam
                        The first bump is recorded and held until BUMPCheck is ca

IRHandle IRInit(DeviceSemantics semantics);
        Input:          (semantics)   semantics
        Output:         (return code) IRHandle or 0 for error
        Semantics:      Initialize given IR-sensor (up to 16 sensors are possible

int IRRelease(IRHandle handle);
        Input:          (handle) logical-or of IR-handles to be released
        Output:         (return code) 0 = ok
                                     errors (nothing is released):
                                      0x11110000 = totally wrong handle
                                      0x0000xxxx = the handle parameter in which
                                      remained set that are connected to a releas
        Semantics:      Release one or more IR-sensors

int IRRead(IRHandle handle);
        Input:          (handle) ONE IR-handle
        Output:         (return code) 0/1 = actual pinstate of the TPU-channel
                                      -1  = wrong handle
        Semantics:      Read actual state of the IR-sensor
```

Latches

Latches are low-level IO buffers.

```
BYTE OSReadInLatch(int latchnr);
        Input:          (latchnr) number of desired Inlatch (range: 0..3)
        Output:         actual state of this inlatch
        Semantics:      Reads contents of selected inlatch

BYTE OSWriteOutLatch(int latchnr, BYTE mask, BYTE value);
        Input:          (latchnr) number of desired Outlatch (range: 0..3)
                        (mask)    and-bitmask of pins which should be cleared
                        (inverse!)
                        (value)   or-bitmask of pins which should be set
        Output:         previous state of this outlatch
        Semantics:      Modifies an outlatch and keeps global state consistent
                        example: OSWriteOutLatch(0, 0xF7, 0x08); sets bit4
                        example: OSWriteOutLatch(0, 0xF7, 0x00); clears bit4

BYTE OSReadOutLatch(int latchnr);
        Input:          (latchnr) number of desired Outlatch (range: 0..3)
        Output:         actual state of this outlatch
        Semantics:      Reads global copy of outlatch
```

Parallel Port

```
BYTE OSReadParData(void);
        Input:          NONE
        Output:         actual state of the 8bit dataport
        Semantics:      Reads contents of parallelport (active high)

void OSWriteParData(BYTE value);
        Input:          (value) new output-data
        Output:         NONE
        Semantics:      Writes out new data to parallelport (active high)

BYTE OSReadParSR(void);
        Input:          NONE
        Output:         actual state of the 5 status pins
        Semantics:      Reads state of the 5 status pins (active-high!)
                        BUSY(4), ACK(3), PE(2), SLCT(1), ERROR(0)

void OSWriteParCTRL(BYTE value);
        Input:          (value) new ctrl-pin-output (4bits)
        Output:         NONE
        Semantics:      Writes out new ctrl-pin states (active high!)
                        SLCTIN(3), INT(2), AUTOFDXT(1), STROBE(0)

BYTE OSReadParCTRL(void);
        Input:          NONE
        Output:         actual state of the 4 ctrl-pins
        Semantics:      Reads state of the 4 ctrl-pins (active-high!)
                        SLCTIN(3), INT(2), AUTOFDXT(1), STROBE(0)
```

Analog-Digital Converter

```
int OSGetAD(int channel);
        Input:          (channel)    desired AD-channel range: 0..15
        Output:         (return code) 10 bit sampled value
        Semantics:      Captures one single 10bit value from specified
                        AD-channel. The return value is stored in the least
                        significant bits of the 32 bit return value.

int OSOffAD(int mode);
        Input:          (mode) 0 = full powerdown
                               1 = fast powerdown
        Output:         NONE
        Semantics:      Powers down the 2 AD-converters (saves energy).
                        A call of OSGetAD awakens the AD-converter again
```

Radio Communication

Note: Additional hardware and software (*Radio-Key*) are required to use these library routines.

"EyeNet" network among arbitrary number of EyeBots and optional workstation host. Network operates as virtual token ring and has fault tolerant aspects. A net Master is negotiated autonomously, new EyeBots will automatically be integrated into the net by "wildcard" messages, and dropped out EyeBots will be eliminated from the network. This network uses a RS232 interface and can be run over cable or wireless.

The communication is 8-bit clean and all packets are sent with checksums to detect transmittion

errors. The communication is unreliable, meaning there is no retransmit on error and delivery of packets are not guaranteed.

```
int RADIOInit(void);
        Input:          NONE
        Output:         (return code) 0 = OK
        Semantics:      Initializes and starts the radio communication.

int RADIOTerm(void);
        Input:          NONE
        Output:         (return code) 0 = OK
        Semantics:      Terminate network operation.

int RADIOSend(BYTE id, int byteCount, BYTE* buffer);
        Input:          (id)        the EyeBot ID number of the message destinati
                        (byteCount) message length
                        (buffer)    message contents
        Output:         (return code) 0 = OK
                                      1 = send buffer is full or message is too l
        Semantics:      Send message to another EyeBot. Send is buffered,
                        so the sending process can continue while the
                        message is sent in the background.  Message
                        length must be below or equal to MSGMAXLEN.
                        Messages are broadcasted by sending them to
                        the special id BROADCAST.

int RADIOCheck(void);
        Input:          NONE
        Output:         returns the number of user messages in the buffer
        Semantics:      Function returns the number of buffered messages.
                        This function should be called before
                        receiving, if blocking is to be avoided.

int RADIORecv(BYTE* id, int* bytesReceived, BYTE* buffer);
        Input:          NONE
        Output:         (id)            EyeBot ID number of the message source
                        (bytesReceived) message length
                        (buffer)        message contents
        Semantics:      Returns the next message buffered. Messages are
                        returned in the order they are
                        received. Receive will block the calling
                        process if no message has been received until
                        the next one comes in.  The buffer must have
                        room for MSGMAXLEN bytes.


Data Types:
        struct RadioIOParameters {
          int interface;      /* SERIAL1, SERIAL2 or SERIAL3 */
          int speed;          /* SER4800,SER9600,SER19200,SER38400,SER57600,SER11
          int id;             /* machine id */
          int remoteOn;       /* non-zero if remote control is active */
          int imageTransfer;  /* if remote on: 0 off, 2 full, 1 reduced */
          int debug;          /* 0 off, 1..100 level of debugging spew */
        };

        struct RadioStatus {
          BYTE master;        /* EyeBot ID */
          BOOL active[MAXEYE];/* shows who is active at the moment */
        };

void RADIOGetIoctl(RadioIOParameters* radioParams);
        Input:          NONE
        Output:         (radioParams) current radio parameter settings
```

```
        Semantics:      Reads out current radio parameter settings.

void RADIOSetIoctl(RadioIOParameters* radioParams);
        Input:          (radioParams) new radio parameter settings
        Output:         NONE
        Semantics:      Changes radio parameter settings.  This should
                        be done before calling RADIOInit().

int RADIOGetStatus(RadioStatus *status);
        Input:          NONE
        Output:         (status) current radio communication status.
        Semantics:      Return current status info from RADIO communication.
```

Compass

These routines provide an interface to a digital compass.

Sample HDT Setting

```
compass_type compass = {0,13,(void*)OutBase, 5,(void*)OutBase, 6, (BYTE*)InBase,

HDT_entry_type HDT[] =
{ ...
  {COMPASS,COMPASS,"COMPAS",(void *)&compass},
  ...
};
```

Functions

```
int COMPASSInit(DeviceSemantics semantics);
        Input:          Unique definition for desired COMPASS (see hdt.h)
        Output:         (return code) 0 = OK
                                      1 = error
        Semantics:      Initialize digital compass device

int COMPASSStart(BOOL cycle);
        Input:          (cycle) 1 for cyclic mode
                                0 for single measurement
        Output:         (return code) 1 = module has already been started
                                      0 = OK
        Semantics:      This function starts the measurement of the actual headin
                        The cycle parameter chooses the operation mode of the com
                        In cyclic mode (1), the compass delivers as fast as possi
                        actual heading without pause. In normal mode (0) a single
                        is requested and allows the module to go back to sleep mo

int COMPASSCheck();
        Input:          NONE
        Output:         (return code) 1 = result is ready
                                      0 = result is not yet ready
        Semantics:      If a single shot was requested this function allows to ch
                        is already available. In the cyclic mode this function is
                        always indicates 'busy'. Usually a user uses a loop to wa
                          int heading;
                          COMPASSStart(FALSE);
                          while(!COMPASSCheck());  //In single tasking! Otherwise
                          heading = COMPASSGet();
```

```
int COMPASSStop();
        Input:          NONE
        Output:         (return code) 0 = OK
                                      1 = error
        Semantics:      To stop the initiated cyclic measurement this function WA
                        measurement to be finished and stops the module. This fun
                        return after 100msec at latest or will deadlock if no com
                        connected to the EyeBot!

int COMPASSRelease();
        Input:          NONE
        Output:         (return code) 0 = OK
                                      1 = error
        Semantics:      This function shuts down the driver and aborts any ongoin
                        directly.

int COMPASSGet();
        Input:          NONE
        Output:         (return code) Compass heading data: [0..359]
                                      -1 = no heading has been calculated yet
                                           (wait after initializing).
        Semantics:      This function delivers the actual compass heading.

int COMPASSCalibrate(int mode);
        Input:          (mode) 0 to reset calibration data of compass module (req
                               1 to perform normal calibration.
        Output:         (return code) 0 = OK
                                      1 = error
        Semantics:      This function has two tasks. With mode=0 it resets the ca
                        of the compass module. With mode=1 the normal calibration
                        It has to be called twice (first at any position, second
                        to the first position).
                        Normally you will perform the following steps:
                          COMPASSCalibrate(1);
                          VWDriveTurn(VWHandle handle, M_PI, speed);  // turn Eye
                          COMPASSCalibrate(1);
```

IR Remote Control

These commands allow sending commands to an EyeBot via a standard TV remote.

Include

```
#include "irtv.h"  /* only required for HDT files */
#include "IRnokia.h"
```

Sample HDT Setting

```
/* infrared remote control on Servo S10 (TPU11)*/
irtv_type irtv = {1, 11, TPU_HIGH_PRIO, REMOTE_ON, SPACE_CODE, 15,
0x0000, 0x03FF, DEFAULT_MODE, 1, -1, RC_RED, RC_GREEN, RC_YELLOW, RC_BLUE};

HDT_entry_type HDT[] =
{ ...
  {IRTV,IRTV,"IRTV",(void *)&irtv},
  ...
};
```

Functions

```
int IRTVInitHDT(DeviceSemantics semantics);
        Input:          (semantics) unique definition for desired IRTV (see hdt.h
        Output:         (return code) 0 = ok
                                      1 = illegal type or mode (in HDT IRTV entry
                                      2 = invalid or missing "IRTV" HDT entry for
        Semantics:      Initializes the IR remote control decoder by calling IRTV
                        parameters found in the corresponding HDT entry. Using th
                        applications are independant of the used remote control s
                        defining parameters are located in the HDT.

int IRTVInit(int type, int length, int tog_mask, int inv_mask, int mode, int bufs
        Input:          (type)     the used code type
                                   Valid values are:
                                   SPACE_CODE, PULSE_CODE, MANCHESTER_CODE, RAW_C
                        (length)   code length (number of bits)
                        (tog_mask) the bitmask that selects the "toggle bits" in
                                   (bits that change when the same key is pressed
                        (inv_mask) the bitmask that selects the inverted bits in
                                   (for remote controls with alternating codes)
                        (mode)     operation mode
                                   Valid values are: DEFAULT_MODE, SLOPPY_MODE, R
                        (bufsize)  size of the internal code buffer
                        Valid values are: 1-4
                        (delay)    key repetition delay
                                   >0: number of 1/100 sec (should be >20)
                                   -1: no repetition
        Output:         (return code) 0 = ok
                                      1 = illegal type or mode
                                      2 = invalid or missing "IRTV" HDT entry
        Semantics:      Initializes the IR remote control decoder.
                        To find out the correct values for the "type", "length",
                        "inv_mask" and "mode" parameters, use the IR remote contr
                        program (IRCA).
                        SLOPPY_MODE can be used as an alternative to DEFAULT_MODE
                        In default mode, at least two consecutive identical code
                        must be received before the code becomes valid. When usin
                        mode, no error check is performed, and every code becomes
                        immediately. This reduces the delay between pressing the
                        the reaction.
                        With remote controls that use a special repetition coding
                        must be used (as suggested by the analyzer).

                        Typical parameters | Nokia (VCN 620)          | RC5 (Phil
                        -------------------+--------------------------+----------
                        type               | SPACE_CODE               | MANCHESTE
                        length             | 15                       | 14
                        tog_mask           | 0                        | 0x800
                        inv_mask           | 0x3FF                    | 0
                        mode               | DEFAULT_MODE/SLOPPY_MODE | DEFAULT_M

                        The type setting RAW_CODE is intended for code analysis o
                        is specified, all of the other parameters should be set t
                        must be handled by using the IRTVGetRaw and IRTVDecodeRaw

void IRTVTerm(void);
        Input:          NONE
        Output:         NONE
        Semantics:      Terminates the remote control decoder and releases the
                        occupied TPU channel.

int IRTVPressed(void);
        Input:          NONE
        Output:         (return code) Code of the remote key that is currently be
                                      0 = no key
```

```
        Semantics:      Directly reads the current remote key code. Does not
                        touch the code buffer. Does not wait.

int IRTVRead(void);
        Input:          NONE
        Output:         (return code) Next code from the buffer
                                      0 = no key
        Semantics:      Reads and removes the next key code from the code buffer.
                        Does not wait.

int IRTVGet(void);
        Input:          NONE
        Output:         (return code) Next code from the buffer (!=0)
        Semantics:      Reads and removes the next key code from the code buffer.
                        If the buffer is empty, the function waits until a remote
                        key is pressed.

void IRTVFlush(void);
        Input:          NONE
        Output:         NONE
        Semantics:      The code buffer is emptied.

void IRTVGetRaw(int bits[2], int *count, int *duration, int *id, int *clock);
        Input:          NONE
        Output:         (bits)     contains the raw code
                                   bit #0 in bits[0] represents the 1st pulse in
                                   bit #0 in bits[1] represents the 1st space
                                   bit #1 in bits[0] represents the 2nd pulse
                                   bit #1 in bits[1] represents the 2nd space
                                   ...
                                   A cleared bit stands for a short signal,
                                   a set bit for a long signal.
                        (count)    the number of signals (= pulses + spaces) rece
                        (duration) the logical duration of the code sequence
                                   duration = (number of short signals) + 2 * (nu
                        (id)       a unique ID for the current code (incremented
                        (clock)    the time when the code was received
        Semantics:      Returns information about the last received raw code.
                        Works only if type setting == RAW_CODE.

int IRTVDecodeRaw(const int bits[2], int count, int type);
        Input:          (bits)  raw code to be decoded (see IRTVGetRaw)
                        (count) number of signals (= pulses + spaces) in raw code
                        (type)  the decoding method
                                Valid values are: SPACE_CODE, PULSE_CODE, MANCHES
        Output:         (return code) The decoded value (0 on an illegal Manchest
        Semantics:      Decodes the raw code using the given method.
```
