PicOS

Seawolf interface

Version 0.2
January 2009

Preamble

This note describes the praxis interface developed for the Seawolf project. It supplements these documents: **LCDGint.pdf** (describing the Nokia LCD display interface, the menu system, and the image handling system), **OEP.pdf** (describing the Object Exchange Protocol module), and **EOL.pdf** (describing the EEPROM object loader). This document comments on the interface specific to the Seawolf praxis, while the modules described in the other documents can be used in other applications in principle unrelated to Seawolf.

Needless to say, everything written below is preliminary. While it refers to existing (already implemented) software, this software is far from being cast in stone.

Building EEPROM images

The present library of functions has been implemented with the intention of facilitating a certain class of applications dealing with people profiles, including pictures, stored in EEPROM (flash memory) at the node. The node is equipped with a Nokia 6100p LCD display, a joystick (4 positions labeled N, E, S, W + push), and two push buttons (labeled B0 and B1).

To simplify the functions (most importantly to reduce the code size), EEPROM has been partitioned into certain zones whose boundaries are static (but can be easily redefined at compile time). Note that the largest objects stored in EEPROM are images – their memory demands by far outweigh the demands of all other objects combined. Consequently, little can be gained by trying to come up with a smart allocation scheme for those other objects. Instead, they can be stored in a fixed (and relatively small) dedicated chunk of EEPROM partitioned in a moderately efficient manner, while the remaining (large) portion of EEPROM is dedicated to images (see **LCDGint.pdf**). This way, even if the non-image objects are not handled extremely efficiently (e.g., incur some fragmentation), the impact of that inefficiency on overall EEPROM usage is negligible.

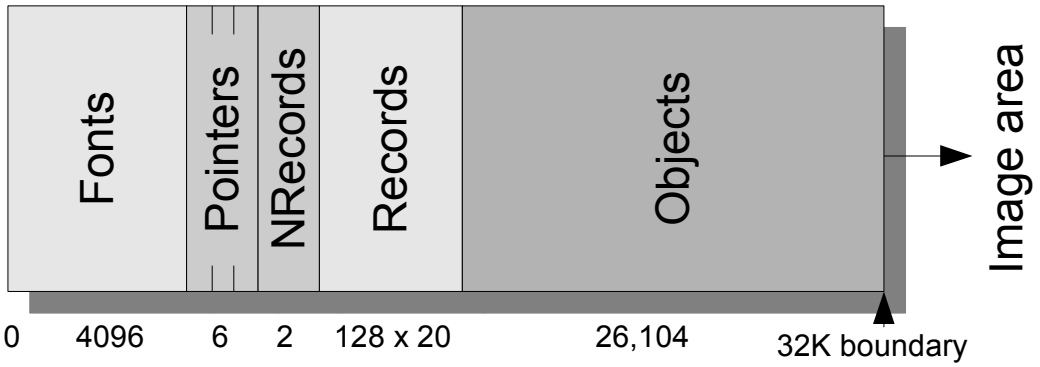

Figure 1: EEPROM layout

The organization of EEPROM is shown in Figure 1. The first 4KB chunk is used by the fonts file needed for rendering texts on the Nokia LCD. At location 4096, there are three word-sized (little-endian) pointers into the Objects area. They point to (in this order):

1. the menu object for the events category list
2. the menu object for my category list
3. the menu object for the records

The respective objects are stored in the Objects area in a way that makes them directly loadable as the corresponding structures of the display manager (see **LCDGint.pdf**). The pointers are relative to the beginning of the Objects area, i.e., each of them must be offset by 6664 to produce the EEPROM address of its object.

The three pointers are followed by the word-sized number of records. This is the number of initial slots in the Records area filled with meaningful data. The Records area is an array of fixed-size entries (20 bytes each) representing the profiles of up to 128 people. This part, in turn, is followed by the Objects area, which is a storage for various loadable objects of the display manager. That area can expand up to the 32K boundary, which begins the Image zone, i.e., the first page of the image zone has number 4 (image pages are 8KB).

The boundaries of EEPROM zones are described in file **sealists.h** (directory **TEST**) as symbolic constants. Note that if these constants should change, you should also update the corresponding definitions in **mkeeprom.tcl** (see below).

EEPROM layouts are described by XML files. Here is an example file which illustrates all the XML elements that can appear in an EEPROM definition:

```
<sealist>
<eventcategories>
        <meter>[0, 122], 3, 4, 0</meter>
        <menu>
                coordinates:    [10, 10]
                font:           0
                colors:         3, 1
                width/height:   18, 8
        </menu>
        <category>Session on feeding while drinking</category>
        <category>Session on beer brewing</category>
        <category>Keynote talk: how to decorate a pub</category>
        <category>Reception: wine tasting</category>
        <category>Vodka bar</category>
        <category>Greek restaurant</category>
        <category>Irish pub</category>
</eventcategories>
<mycategories>
        <meter>[66, 122], 3, 3, 1</meter>
        <menu>[10, 65] 0, 4, 0, 18, 8</menu>
        <category>Likes to drink</category>
        <category>Pal</category>
        <category>Smokes pot</category>
        <category>Smart</category>
        <category>Money: buys drinks</category>
        <category>Teetotaler</category>
        <category>Boring</category>
        <category>Money: doesn't pay for drinks</category>
        <category>Smells badly</category>
</mycategories>
<people>
        <note>
                extend-up:              [0, 122]
                font:                   0
                colors:                 1, 0
                width (characters):     21
```

```
        height: (missing or zero means extensible)
</note>
<name>[0, 10], 0, 1, 5, 21, 1</name>
<nickname>[0, 26], 0, 2, 7, 21, 1</nickname>
<menu>[48, 26], 1, 2, 7, 10, 12</menu>
<record class="+">
        <id>0xBACA0002</id>
        <name>Pawel Gburzynski</name>
        <!-- insert ".. " -->
        <nickname>Mistrz</nickname>
        <why>0,1,3;4,5,6</why>
        <!-- format the text automatically -->
        <note>Liver is a touchy subject</note>
</record>
<record class="-">
        <id>0xBACA0001</id>
        <name>Wlodek Olesinski</name>
        <nickname>CEO</nickname>
        <why>0,1,3;4,5,6</why>
        <note>Prefers straight vodka with a touch of grapefruit juice</note>
</record>
<record class="+">
        <id>0xBACA0006</id>
        <name>Rob Finnucan</name>
        <nickname>Green Rob</nickname>
        <why>2,3,4;6</why>
        <note>Likes Nortel Sunset: pure spirit on the rocks</note>
</record>
<record class="+">
        <id>0xBACA0003</id>
        <name>Gerry Sendrowicz</name>
        <nickname>Zdzichu</nickname>
        <why>1,2,5;3,5</why>
        <note>Getting older; does not drink as much as he used to</note>
</record>
<record class="+">
        <id>0xBACA0007</id>
        <name>Kitty Ha</name>
        <nickname>Kitty</nickname>
        <why>3,5</why>
        <note>Likes cooking while drinking</note>
</record>
<record class="+">
        <id>0xBABA0004</id>
        <name>Olsonet Comms</name>
        <nickname>Olsonet</nickname>
        <why>1;1,3</why>
        <note>Expert group: Visit events on safe beer and vodka mixing</note>
</record>
<record class="+">
        <id>0xBABA0005</id>
        <name>Combat Networks</name>
        <nickname>Combat</nickname>
        <why>0,3;3,4</why>
        <note>
                Pros league: Listen to fascinating Tales From Two Irish Pubs
```


```
            </note>
        </record>
</people>
</sealist>
```

The <menu> entries appearing in the category lists and in the record list (inside the <peoples> element) describe the parameters of the respective menus. Any fragments of those entries that cannot be interpreted as nonnegative numbers are ignored. The numbers are strictly positional and they stand for (in this order):

1. the X coordinate of the menu's top left corner
2. the Y coordinate of the top left corner
3. the font
4. the background color
5. the foreground color
6. the width
7. the height

Note that they directly correspond to the arguments of the standard menu creation function of the display manager.

The <meter> elements describe the display parameters of the “meters”, which are associated with records and represent graphically their association with category groups. Essentially, a meter is a special case of a text object (see below) containing exactly 16 characters in a special font. The numerical parameters describe:

1. the X coordinate of the meter (the left side)
2. the Y coordinate of the meter (the upper side)
3. the font
4. the background color
5. the foreground color

The element <note>, <name>, and <nickname> describe the ways of rendering the respective textual components of a record on the screen. All these components are turned into text objects of the display manager. The numerical parameters have the following meaning:

1. the X coordinate of the text area (the left side)
2. the Y coordinate of the text area (as explained below)
3. the font
4. the background color
5. the foreground color
6. the width in characters
7. the height in characters

The last number is optional: if missing (or zero), the height is determined based on the actual length of the character string.

Normally, the Y coordinate is interpreted as the position of the top line of text (i.e., the upper edge of the object's bounding box). If the string “extend-up” appears within the element's body (as it happens for <note> in the above example), the Y coordinate refers to the position of the last line (the bottom edge of the area's bounding box). In such a case, if the height of the object is unspecified (flexible), the text will extend up.

The numbering of categories is determined by the ordering of the items; thus, for example, *Session on feeding while drinking* has number 0 (within event categories) and

Teetotaler is number 5 (within my categories). The maximum number of categories in each group is 16. The category sets are stored as complete (loadable) menu objects in the Objects area.

The Records area is an array of fixed-length records with this layout:

network ID
event categories
my categories
class
event categories meter
my categories meter
note
image
name

Except for the network ID, which is a **`lword`** (4 bytes), each attribute takes exactly 2 bytes (a **`word`**); thus, the record size amounts to 20 bytes. The two categories words are bit maps reflecting the category membership. The rightmost bit corresponds to category number zero (the first one on the respective list), and 1 stands for "present".

Only the lower byte of the class word is used at present: it stores the character specified with the *class* attribute of the <record>.

The remaining words of the structure are pointers into the Objects area (relative to the origin of that area). All of them, except for *image,* point to text objects. Thus, for example, *event categories meter,* points to the text object representing the record's meter for event categories. The image word points to the image object (note that this is a displayable object of the display manager, not the image itself). Also note that the nickname is not stored within the record: it is present in the record menu pointed to by the third word from the Pointers area.

When a nickname is transformed into a menu line (for the object menu), these three characters: "**`..`** " (i.e., two dots and a blank) are applied as a prefix. This prefix provides an area to accommodate the record status, which is dynamically displayed in the menu. Also, category strings appearing in the categories menus are automatically prefixed with "**`Ex.`**" or "**`Mx.`**", where ***`x`*** is the category number.

Here is how you can generate EEPROM files:

```
mkeeprom.tcl fontfile xmldata output
```

where ***`fontfile`*** is a file containing fonts for the LCDG driver, ***`xmldata`*** is an XML file with the layout described above, and ***`output`*** is the target file where the resulting EEPROM image is to be written. The way to prepare a font file is to go to directory **`TEST/FONTS`** and do this:

```
gcc fontfile.c
a.exe > ../fonts.bin
```

The present set of fonts (which will be written to **`fonts.bin`** with the above sequence) consists of three "standard" fonts (numbered 0-2) + one special font (number 3) used for rendering the "meters" representing graphically the category membership for a record.

The script will build the EEPROM image by placing the contents of the fonts file at location zero, and then building the Pointers, Records, and Objects zones, based on the

XML description. When processing a record, the scripts will check whether the current directory includes a file named ***image_xxxxxxxx.nok***, where **xxxxxxxx** is the lower-case hex digit content of the network ID attribute. For example, if the ID (as specified in the record's <id> element) is **0xBACA0001**, the file name will be **image_baca0001.nok**. If such a file is found, it is interpreted as the image file (picture) to be associated with the record. Otherwise, the record will have no associated picture (its *image* word will be WNONE). The way to convert pictures to the format acceptable by **mkeeprom.tcl** is decribed in **LCDGint.pdf**.

It is possible to indicate to the script that the picture files should be sought in a specific directory (not necessarily the current one), e.g.,

```
mkeeprom.tcl -i MY/PICS FONTS/fonts.bin data.xml eeprom.bin
```

The **-i** parameter must be followed by a directory path and, if present, must precede all other arguments.

If the picture directory (or the current directory, if no picture directory has been explicitly specified) contains an image file named **wallpaper.nok**, this picture will be included in the EEPROM file (at the very beginning of the Image area). The label of that picture should look like the ASCII string *"wallpaper"* terminated with a NULL byte. Such a picture will be used as the background by the LCDG display manager.

The output file generated by **mkeeprom.tcl** stores the absolute content of EEPROM starting at location 0. Its length depends on the amount of EEPROM filled by the data, most notably, on the number of images. There is an alternative (more compact) way of representing the EEPROM contents as a list of chunks. This feature has been programmed into the script, but is disabled at present.

Extra functions provided in **sealists.c**

By including **sealists.h** in the praxis's header, you get access to a few functions that will allow you to extract data from EEPROM set up according to the layout described in the previous section. These functions are described below. To make sense of them, you have to read **LCDGint.pdf** first.

```
lcdg_dm_obj_t *seal_mkcmenu (Boolean ev);
```

The function builds a a categories menu (by loading the respective object from EEPROM). This is either the event categories menu (if **ev** is YES) or my categories menu (if **ev** is NO).

Note that the menu object (whose pointer is returned by the function), as well as its **Lines** attribute, are allocated by a single call to **umalloc**. Consequently, the proper way to deallocate this object (when it is no longer needed) is to deallocate just the object (without worrying about the lines). It is recommended to use the display manager's function **lcdg_dm_free** for this purpose.

The function may fail, in which case it will return NULL. The reason for failure can be determined by examining **LCDG_DM_STATUS**, which, in this case, can receive one of these values:

LCDG_DMERR_GARBAGE	the corresponding categories list is empty
LCDG_DMERR_NOMEM	**umalloc** failure (not enough memory)

If the symbolic constants describing the parameters of the categories menu (in **sealists.h**) have malicious values (or/and the font file in EEPROM is broken), additional error codes are possible, i.e., those corresponding to the possible failures of **lcdg_dm_newmenu**.

```
lcdg_dm_obj_t *seal_mkrmenu ();
```

The function builds a records menu object. Its behavior is analogous to **seal_mkcmenu**.

```
sea_rec_t *seal_getrec (word num);
```

The function allows you to get hold of the complete record number **num** (assuming that the first record in the Records area has number 0). The function allocates (and returns a pointer to) the following structure (declared in **sealists.h**):

```
typedef struct {
      lword NID;
      word  ECats,
            MCats,
            CL,
            ME,
            MY,
            NT,
            IM,
            NM;
} sea_rec_t;
```

The attributes are: **NID** – the network ID, **ECats**, **MCats** – the category membership flags (events, my), **CL** – the class character, **ME** – pointer to the event categories meter, **MY** – pointer to the my categories meter, **NT** – pointer to the note string object, **IM** – pointer to the image object, **NM** – pointer to the name object.

The function will fail, returning NULL and setting **LCDG_DM_STATUS** to nonzero in the following circumstances:

- The EEPROM area supposed to contain the record menu is corrupted; **LCDG_DM_STATUS** returns **LCDG_DMERR_GARBAGE**.
- There is not enough memory to allocate **sea_rec_t**; **LCDG_DM_STATUS** returns **LCDG_DMERR_NOMEM**.

You should remember to deallocate the structure created by **seal_getrec** when it is no longer needed. The recommended way to do this is to call **lcdg_dm_free**.

In order to get hold of the object represented by a pointer, the pointer must be transformed into the EEPROM address of the object. This macro, defined in **sealists.h**, can be used for this purpose:

```
#define seal_objaddr(ptr) ((lword)((ptr)+SEA_EOFF_TXT))
```

where the constant **SEA_EOFF_TXT** determines the origin of the Objects area. Here is another macro (defined in the same file):

```
#define seal_gettext(off) lcdg_dm_newtext_e (seal_objaddr (off))
```


which transforms a text object pointer into the object.

The only pointer that can be legitimately void is the image object pointer (**IM**). If no picture is associated with the record, that pointer contains WNONE.

```
word seal_findrec (lword ID);
```

This function searches the Records zone for a record whose network ID matches the specified value. If such a record is found, the function returns its number (starting from zero); otherwise, the function returns WNONE.

```
void seal_disprec (sea_rec_t *rec);
```

This function displays information pertaining to the record whose pointer it receives as the argument. Specifically, it performs these actions:

If the image object pointer (**IM**) is not WNONE, the function displays the image associated with the record.

Then the function displays the two meters and the note.

The function may tacitly fail (displaying only a subset of the four objects, or none at all), if EEPROM is corrupted or there is no memory to allocate the object structures. Those structures are only needed intermittently, and they are deallocated before the function returns. The displayed objects are not included in the display manager's active object list.

The TEST praxis

This is the description of my TEST praxis. I have removed the old TEST subdirectory of Apps/SEAWOLF and put my old TEST program into app.c_test in directory YEP (containing Wlodek's test praxis). I have modified Wlodek's praxis to compile and work (AFAICT), but I am not going to try to document it.

The praxis, in collaboration with the **oss.tcl** script, illustrates the usage of most of the library functions provided by these modules: **lcdg_images.[ch]**, **lcdg_dispman.[ch]** (see **LCDGInt.pdf**), **sealists.[ch]**, as well as **oep.[ch]** (see **OEP.pdf**), **ab.[ch]** (see **Serial.pdf**) and (implicitly) **ee_ol.[ch]** (see **EOL.pdf**). It also provides a way for downloading files (e.g., generated by **mkeeprom.tcl** described above) to EEPROM. Here is its functional description.

The praxis starts by initializing the UART PHY for the so-called External Reliable Scheme (XRS) described in **Serial.pdf**. Then, it initializes the image handler (**lcdg_im_init**) and the OEP module (**oep_init**). The UART will be switched between XRS (providing for standard command-line communication with the OSS program) and OEP (during transmission of data chunks, e.g. EEPROM content).

Having started the praxis on a node connected to the PC via a UART, you should execute on the PC the **oss.tcl** script present in the **TEST** directory specifying the UART device as the only argument. It is enough to provide the **COM** number (or **ttyUSB** number on Linux systems), although the full device name (e.g., **COM8:** or **/dev/ttyUSB0**) will work as well. The remaining parameters required by XRS are hardwired into the script and match those of the praxis (bit rate = 115200 bps and

maximum packet length = 60 bytes). The latter equals the maximum packet length (excluding the checksum) of CC1100, which is the same as the maximum packet length used by OEP.

Note: at the end of the **oss.tcl** script you will see this line:

```
u_settrace 7 dump.txt
```

In the above form, it will dump all packets exchanged by the script and the praxis to file **dump.txt** in the current (**TEST**) directory. You can remove this line (or change the argument to 0) to switch this off. If you want to see those packets as things proceed, open a separate window and execute

```
tail -f dump.txt
```

in it.

Normally, any line you type as input to the script will be transformed into an XRS packet and send to the praxis a command, which you would traditionally type into a terminal emulator. A line starting with **!** (the exclamation mark) represents something that should be processed by the script itself. This may be a command initiating an OEP exchange between the script and the praxis, e.g., a transfer of EEPROM chunk or a picture. **!!** provides a shortcut for the last issued command that was rejected. A rejection may happen when the previous command hasn't been confirmed yet by the praxis (no XRS ACK has been received for it yet). You will see the message “board busy” in such circumstances, and the command will have to be issued again, e.g., by entering **!!**. While the script could easily buffer commands and issue them automatically as previous commands are confirmed, that could lead to a confusion, especially as some of those commands may result in XRS being temporarily switched off and OEP taking over. Consequently, at most one command can be outstanding at a time.

Here is the list of direct commands (interpreted directly by the praxis). Remember that each of them must fit into one line, with the line length limited (truncated) to 56 characters.

```
off
on [u]
or
```

These commands affect the LCD: turn off, turn on, and refresh. For **on**, the optional argument specifies the contrast. By refresh, we mean redisplaying the current hierarchy of objects by calling **lcdg_dm_refresh**.

```
ci n h [x [y]]
cm n nl fo bg fg x y w h
ct n fo bg fg x y w
```

These commands directly create displayable objects: image, menu, text, from manually entered data. In all cases, the first argument is a number (index) under which the object can be accessed by subsequent commands. The index value must be between 0 and 31, inclusively. If an index is already taken by some object, the old object is removed (its memory freed).

Note that the ordering of parameters for **lcdg_dm_newtext** and **lcdg_dm_newmenu** has changed (see **LCDG_int.pdf**). The test praxis uses the old order (I didn't consider it critical to change that).

For **ci**, the remaining arguments specify the image handle (the number of the image's starting page, which can be listed with **li** – see below) and the two coordinates of the image's left top corner on the screen. They are both optional and default to 0, 0.

For **cm**, the parameters describe: the number of lines in the menu, the font number (between 0 and 2), the background color number (0-10), the foreground color, the coordinates of the top left corner, the width in characters, and the height in lines. You will be asked to enter the specified number of lines, which can be any strings up to 56 characters each.

For **ct**, the parameters are: the font number, the background color, the foreground color, the coordinates of the top left corner, and the width in characters. See **LCDGint.pdf** for details. You will be asked to enter one line to complete this command.

lo

The command lists the current set of displayable objects along with their indexes.

li

The command lists all pictures that it can find in EEPROM along with their dimensions, labels, and handles.

da ***n***
dd ***n***

These commands add/delete the specified displayable object (identified by its index) to/from the display list. The addition is carried out by calling **lcdg_dm_newtop**, i.e., the added object is put on the top and is automatically displayed (assuming the LCD is on). The deletion is done with **lcdg_dm_remove**. You should refresh the display (with **or**) to see its effect on the screen.

ee [*from* [*upto*]]

This command erases the EEPROM. The arguments, which both default to zero, are directly passed as arguments to **ee_erase**. Thus, the argument-less variant erases the entire EEPROM. The command takes a while to complete during which the board appears busy to new commands.

ei ***h***

This command erases the image with the specified handle.

m

The command produces three numbers related to memory statistics: current free, minimum free, and minimum stack free (all in words).

fc n [*ev*]

The command creates a category list (a menu object). If ***ev*** is zero (or absent), the list is that of my categories, otherwise, it is event categories. The first argument is the object index, as for **ci**, **cm**, **ct**.

fr n [*c*]

The command creates a record list (a menu object). The second argument is the class identifier (0 = “ignore”, 1 = “yes”, 2 = “no”). The default is 0.

When a menu object is the top displayed object on the scree, the joystick can be used to navigate through the menu. If that menu is a record menu, by pushing the bottom button (B1) you will display the image associated with the currently selected record (if there is one) as well as the meters indicating its category membership. Note that these objects are displayed outside the hierarchy, and **or** will remove them.

Four more command: **si**, **se**, **ri**, **re** are in principle available in the same way as the ones described above, but you shouldn't execute them directly (you can try, but they won't amount to much). These commands provide a handshake for special commands implemented in the script whose purpose is to transfer chunks of EEPROM and images (pictures) via OEP.

To write a chunk of EEPROM on the device, you can use this script command:

!eeput *fwa filename*
!eeput *fwa list_of_values*

The first variant writes to the EEPROM all the bytes of the specified file starting at EEPROM location ***fwa***. The second variant can be used to directly modify a few bytes by hand. The ***list_of_values*** should consist of a sequence of integer (possibly hexadecimal) numbers separated by spaces that will be interpreted as bytes to be stored in EEPROM starting at location ***fwa***.

To dump a chunk of EEPROM, execute this:

!eeget fwa len [*filename*]

The first argument is the starting address in EEPROM, and the second one is the length of the area to be dumped in bytes. If a filename is specified, the EEPROM fragment will be written to that file, otherwise, the bytes will be shown on the screen as hexadecimal values.

Here is how to download a picture:

!imput *filename*

where filename should point to a file containing a properly formatted image (see **LCDGInt.pdf**). To see the handle of a newly downloaded image, you can use the **li** command (see above).

With this command:

!imget h *filename*

you can upload the image with the specified handle and store it to the indicated file.

Note: to make the test praxis usable without the OSS script, I have included a feature whereby pressing the joystick button with the empty object list will:

1. Create three objects: the two categories menus + the record menu, assign them numbers 0, 1, and 3, and put them on the display list.

2. Switch on the display.