Application Note AN1902 Description of the Application Programming Interface of the HA57 EVO

Exported on 20.02.2020

pei tel Communications GmbH
info@peitel.de
www.peitel.de

Table of Contents

0 History

Date	Revision	Author	Comments
14.06.2018	1.0	BP	First Release
14.02.2019	1.1	RaSc	Review of the release

Table 1: Document History

0.1 Related Documents

No.	Name	Remarks
1	AN1800 HA57 EVO User Manual	Application Note 1800
2	AN1903 AppLoader HA5x SDK & HA57 EVO	Application Note 1903
3	AN1904 LogoLoader for Handsets	Application Note 1904

Table 2: Related Documents

1 Introduction

This document describes the usage of the Software Development Kits (SDK) for the handsets of the HA57 EVO Series by pei tel Communications GmbH. It describes the structure and the handling of the SDK in detail. Using this documentation will enable the user to create his own user applications for the HA57 EVO or to modify the command protocol.

Please ensure that you have the documents listed in 0.1 Related Documents available for reference.

1.1 Glossary

Application	Software to be loaded on the HA57 EVO using the provided firmware as operating system.
Firmware	The system software of the HA57 EVO. It provides all necessary functions and procedures to access the resources of the HA57 EVO.

1.2 Abbreviations

API	Application Programming Interface
RAM	Random Access Memory
SDK	Software Development Kit

2 Prerequisites

The SDK is available on all handsets HA57 EVO. The development of user applications requires a development system for the ARM Cortex M3 processor. Two versions were tested thoroughly and can be recommended:

1. µVision V5.26.2.0 (KEIL™ Software) is a commercial solution used for the firmware
2. Atollic TrueSTUDIO® for STM32 in combination with the GNU-C-Compiler (ARM-GCC) is a freely available solution

3 Software Architecture

The firmware of the HA57 is a tread-based system. The access to the hardware resources must be conducted by driver functions. An uploaded application must be recognized by the system. It is started as a task and gets access to the system functions provided by the firmware. A direct access to the hardware components for the HA57 EVO is not allowed. Due to the CPU architecture, there is no protection from illegal accesses.

If an application creates a detectable hardware error (FAULT), the automatic activation at system start up is removed to ensure the capability for updates of the device.

An application is constructed around an event handler function. This handler receives events from the system and processes them by triggering appropriate reactions. The API is realized in form of a symbol table handed over to the application.

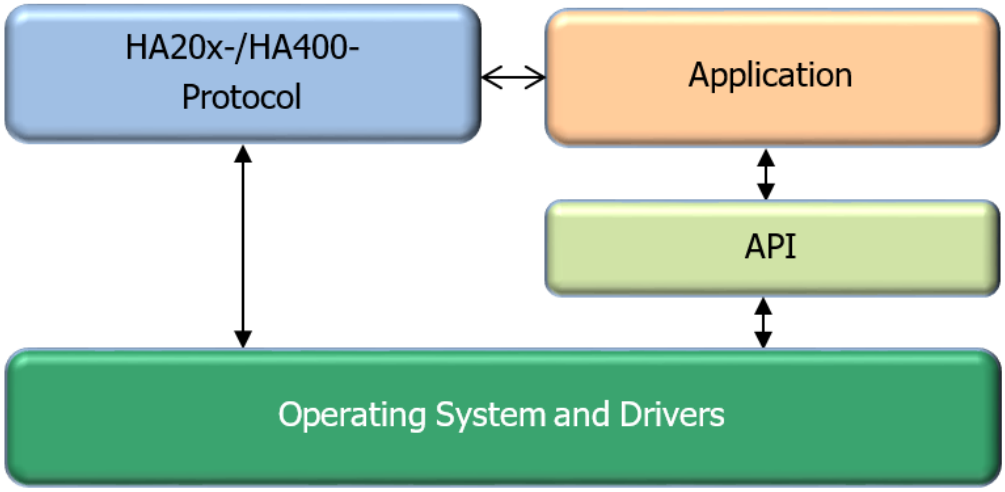

Figure 1: Software Stack

4 Components of the SDK

The SDK consists of this documentation, an application programming tool, project templates for the KEIL™ and GCC tool chains and demo applications.

The development environment can be chosen by the user taking in account the necessary requirements. One of the project templates may be modified accordingly.

5 Reference Applications

A project file with a project template as well as a detailed example application for both application methods of the SDK are provided for both of the common development environments KEIL™ and the STM processor free development environment Atollic TrueSTUDIO® based on gcc.

The example highlights the programming approach and the handling of the API in detail. The project template can be used as basis for own developments. This approach needs to be modified for different development environments accordingly to the IDE in use.

5.1 KEIL™

Both the example application and the project template for KEIL™ are directly available in the "µVision5" workbench.

5.1.1 Components of the Project Template

The project template consists of:

- A project file with guidelines for the development environment
- A folder Source/: 2 c files (entry point and application)
- A folder Include/: 1 h file (application)
- A folder System/: 1 h file (API definition), 1 assembler startup file, (the ld file is not required for the µVision compilation)

5.1.2 Reference for KEIL™

The reference project for KEIL™ contains:

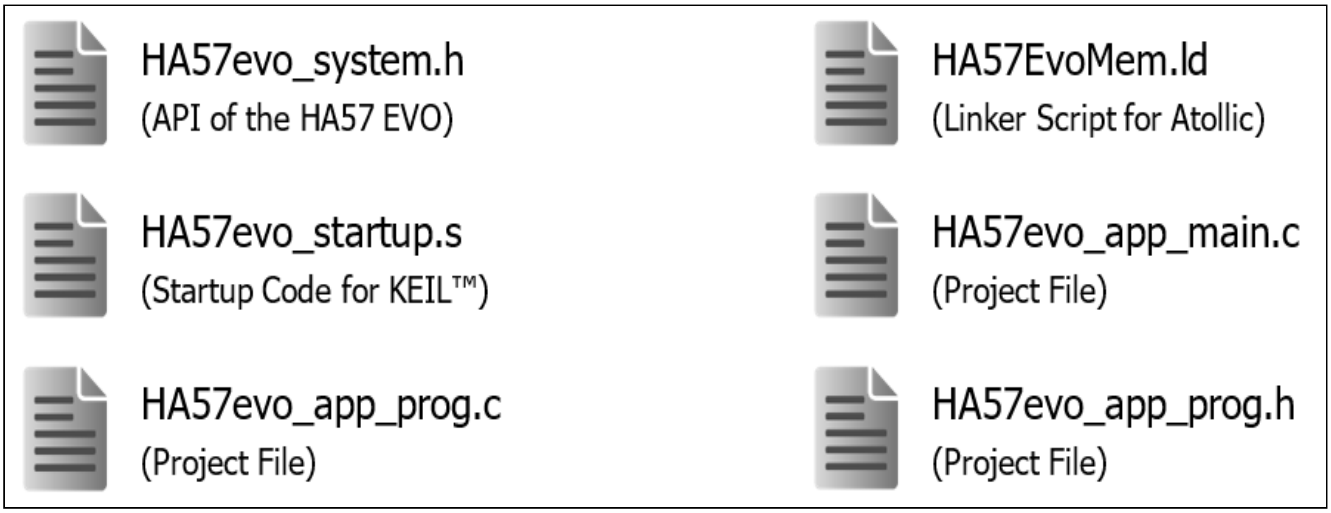

Figure 2: Reference Project for KEIL™

5.1.3 Folder and File Structure KEIL™

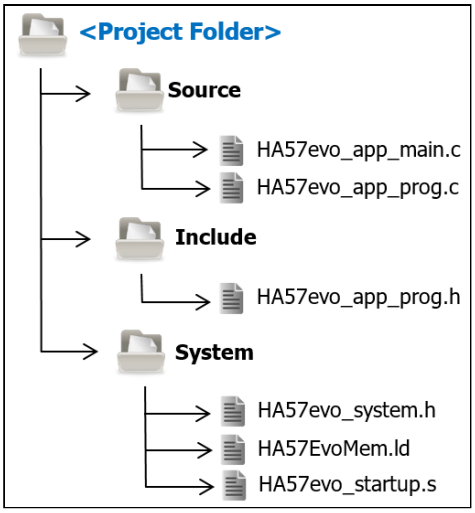

Figure 3: File Structure KEIL™

5.1.4 Suggested Programming Approach

For the implementation of a custom application using the KEIL™ development environment, the following approach is suggested:

1. Copy the project template to the working area.
2. Start the KEIL™ development environment.
3. Open your copy of the project in the development environment.
4. Rename the application files as desired.
5. Compilation. Verify that the compilation runs without errors and warnings!
6. Start to add your own application functionality.

5.2 Atollic TrueSTUDIO®

Both the example application as well as the project template were composed and tested using the tools of Atollic. These tools are available free in the Internet. Both of them are provided with detailed installation instructions.

5.2.1 Components of the Project Template

The project template consists of:

- Project files and settings with guidelines for the development environment
- A folder Source/: 2 c files (entry point and application)
- A folder Include/: 1 h file (application)
- A folder System/: 1 h file (API definition), (1 assembler startup file is not in use), ld file (linker script)

5.2.2 Reference for GCC

The reference project for GCC contains:

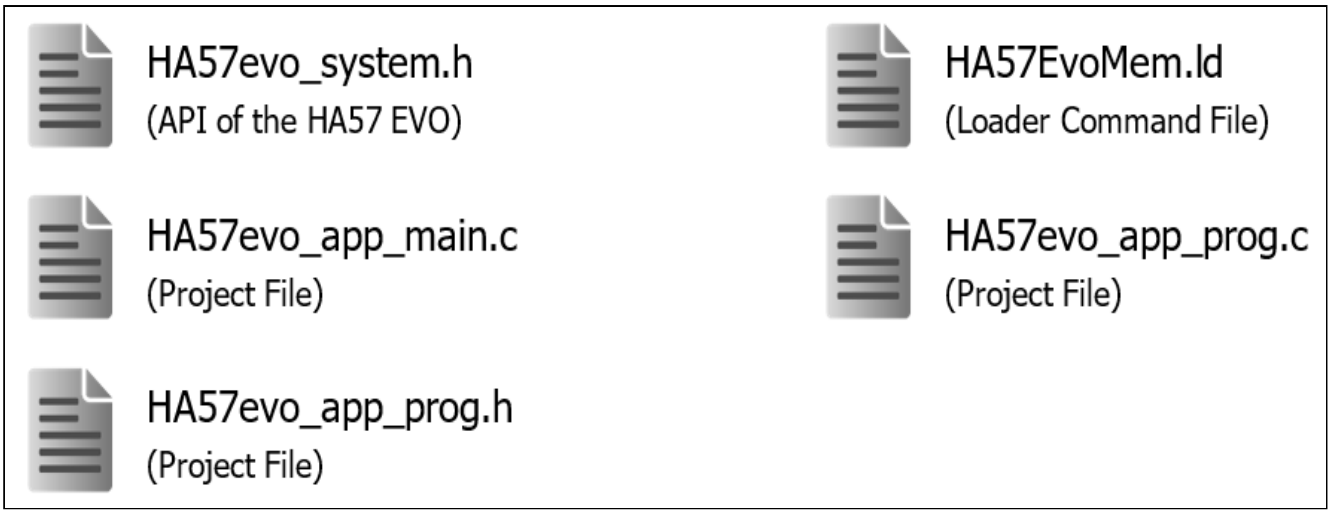

Figure 4: Reference Project for GCC

5.2.3 Folder and File Structure GCC

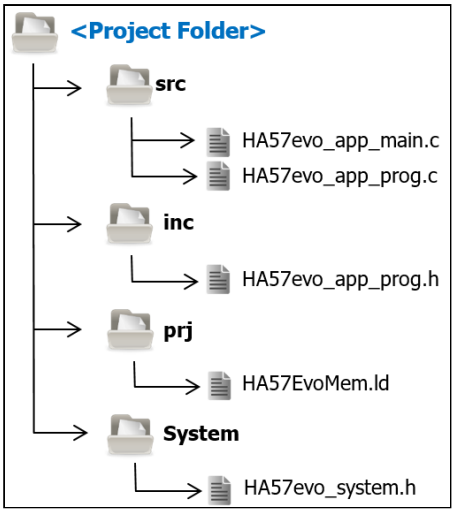

Figure 5: File Structure GCC

5.2.4 Suggested Programming Approach

For the implementation of a custom application using the "Atollic" development suite, the following approach is suggested:

1. Copy the project template to the working area.
2. Start Atollic (Eclipse based).
3. Import/Open your copy of the project in the project explorer.
4. Select the root directory.

5. Compilation. Verify that the compilation runs without errors and warnings!
6. Start to write your own application.

5.3 Alternative Compilation

The following presents a collection of technical details for an alternative compilation, e.g. a cross compilation in Linux with an ARM-GCC. It is necessary to follow these steps to create the hex files needed for the application:

1. Compilation of the c files to objects. Use the relevant include paths of the local implementation (Include/), the SDK API (System/) and the standard Lib for necessary type definitions (that's where stdint.h can be located). The firmware compiles ARM Thumb, but any desired combination of the generation should work as well.
2. Linking is done without standard libraries and the entry point "application" of the main c file. This function is located at the address 0x08050000 (in thumb mode this might be shown at 0x08050001). The firmware jumps to this address, with the symbol table being the first argument when the application is active. There is no initialization like, e.g. a library, which means, global variables/objects of the application code can't be initialized. Therefore, it is not recommended to use C++ or custom-made libraries. The dynamic loader functionality needs to be integrated in "application". For the GCC code, there are concepts, but they extremely depend on the compiler.
    a. For KEIL™, an own .entry area (section) is used for this functionality, which is accordingly placed first in the asm file.
    b. The Atollic project uses a linker script, which moves the respective .entry area to the necessary address.
3. The created binary needs to be transferred into the Intel Hex format, or respectively, some linkers are able to create the format directly. If correctly linked, this file should only contain addresses of the flash area 0x0805000 to 0x8BF800. RAM areas should be without code.

The above-mentioned sections are filled with code during compilation according to certain rules. If static code is not relocatable, the linker will assign real addresses to the sections. For dynamic libraries, there is relocatable code. This assignment (and as well the dereferencing of references to other areas) occurs during runtime using a dynamic linker. We explicitly do not use this; therefore, a not relocatable, statically linked code needs to be created (option of the compiler/linker).

Common sections are:

- .code/.text: The program code in a read-only section. Sub-sections are possible, which contain fixed stings in the code.
- .rodata: All global/static **constant** objects. These remain in the directly addressable flash.
- .data: The values of all global/static objects placed in the flash which are not pre-initialized with 0 and which are at startup copied from this section into the RAM area. The linker script of the gcc delivers appropriate conditions, so that the "memcpy" in "application" can do the copying. If this copying does not occur (KEIL™), all global variables will not be initialized.
- .bss: A 0-initialized RAM area for 0-initialized objects (these are practically all static, non-initialized variables/ objects). GCC will set this area via memset to 0, KEIL™ is missing this step. Therefore, all static/global variables/objects remain uninitialized.
- .entry: The code area of the function "application" with compiler dependent indication of a section in the C code. This area needs to be set to the address 0x08050000.

Notes

- Test the correct functioning of the printf variants with the variable parameters. At this point, different compilers will produce discrepancies (stack direction and endianness are relevant!).
- In case the compiler removes unused code, mark the "application" function as "used". Otherwise, the created binary will soon be empty.

6 AppLoader HA57 SDK & HA57 EVO

The program "**AppLoader HA57 SDK & HA57 EVO**" is a Windows® application that provides the means for uploading and starting a user application on the HA57 EVO handset. The upload procedure requires a free serial interface to be available on the PC.

Please find a detailed description of the uploader program in the documentation "AN1903 AppLoader HA5x SDK & HA57 EVO".

> **Note**
> Before loading an application with a standard (115200 Baud) baud rate, the internal emulation "pei tel HA20x" or "pei tel HA400" must be activated, otherwise the handset will not be recognized by the AppLoader tool. See also document "AN1800 HA57 EVO User Manual", section 4.4 Emulation.

7 Porting of HA57 SDK

Features and processes are based on the HA57 SDK layout (HA57 SDK: predecessor model). Special attention was payed to the compatibility of the features. Due to new hardware components and technical progress, some changes occurred, therefore, please pay attention when using already available software:

- The display's resolution increased to 240x320 and the colour space to RGB-565. Therefore, absolute coordinates (not relative to display size) need to be doubled in most cases. Colour specifications not based on constants need to be converted. The macro RGB565_COLOR gives a hint, how to do this.
- The SDK API doesn't rely anymore on the fixed addresses of the firmware. Instead, a symbol table is handed over to the entry function of the application. This changes initial procedure; the central event handler is no longer the entry. It is set, or not set, in the structure appl_Info_s together with the font handler. Setting the object appl_Info_s needs to be complete, in accordance with the type definition. It is necessary to match the main file with the new AppEVO_main.c file. There's an advantage in this: After a change in the firmware, the application doesn't need to be recompiled. The entry in the API function is respectively *table->function*. Common function calls are provided by C macros.
- Most of the text functions use now UTF8 instead of the mostly not documented ASCII coding (this isn't valid for text-mode based functions). We assume this will be compatible to most of the usages so far.
- The flash API is missing, as there isn't an external flash hardware for the HA57 EVO. Querying the flash values, will return, as the HA57 without flash did, a 0.
- The HA57 SDK implementation of the cursor in *appl_Info_s* gives non-explicable results. Therefore, this field will be ignored. However, it will remain part of the structure.
- The font names contain the normal/bold properties, therefore, the parameters *appl_letter_size_t* are not needed. However, if the value "bold" is set, a normal font will be displayed in its bold version, if available.
- The function *appl_LCD_setTextOffset* gives, when using non-standard fonts, inexplicable results. Using it creates a risk that elements of the graphic areas overwrite each other.
- Images (bmp) are normally filed in form of a byte array in the C code, adjusted to display resolution and colour coding. Both now changed. To perform a fast translation, put a bmp into the brackets of the function *appl_LCD_setCompatibility*. Then scale the image by factor 2 and adjust the colours to RGB-555. Nevertheless, the coordinates need to be adjusted accordingly.

  > **Note**
  > The memory space needed for double resolution quadruples!

- Bitmaps were offered as a replacement for pictograms or missing fonts. The bitmaps in the protocols are compatible, meaning, scaled by factor 2. The SDK offers the scaling as well. The function *appl_LCD_setCompatibility* can be used for scaling. Otherwise, you can just create new bitmaps. It is recommended, in regard to the new UTF font enhancements, to do without.
- The definition for *appl_hw_description_s* has been changed, as the HA57 EVO has a PPT button, but not unbalanced audio. In addition, the serial interface can only be used as RS232-V.24.

8 Application Programming Interface (API)

The API functions provide the access to the resources of the HA57 EVO. The available functions are declared in the "HA57evo_system.h" header file. This file also contains a documentation in English. Especially, the sample applications for the API tests are extensive applications with all available functions and can be used as additional documentation.

As all functions are conveyed with a symbol table, there is a set of macros, which offer the functions analogue to the previous SDK. Due to the symbol tables, the application is independent of the firmware version. The firmware can add to the API version, which will never change available symbols, but might add new ones.

8.1 Memory Map

The memory areas available for the use by applications are marked in **GREEN**.

System (reserved)	0xFFFF FFFF
	0x2001 8000
RAM (Application)	0x2001 7FFF
	0x2000 8000
RAM (Firmware)	0x2000 8000
	0x2000 0000
Not populated	0x1FFF FFFF
	0x080C 0000
System (reserved)	0x080B FFFF
	0x080B F800
Flash (Application)	0x080B F7FF
	0x0805 0000
Firmware	0x0804 FFFF
	0x0800 0000
System (reserved)	0x07FF FFFF
	0x0000 0000

Figure 6: Memory Map

The memory offers 64 kB RAM for static and global objects or data as well as 446 kB FLASH for code and constant data (fonts, icons, images).

> **Note**
> Stack and Heap (malloc) are part of the firmware, which limits it to 32 kB! The usage of many or large local variables or recursions can lead to a stack overflow.

8.2 Entry Point

The application entry point is fixed to the absolute address **0x0805 0000**.

This address is used to hand over the symbol table and to receive the description of the application in exchange back. After the one-time entry, all other referencing occurs on basis of the handed over objects.

8.2.1 Symbol Table

The structure sdl_symbols_s consists of a version field followed by pointers to functions. Those pointers represent all available entry points in direction of the firmware. The handed over object is stored in the flash and therefore can't be modified. A set of macros is available in order to avoid that functions need to be called in a complex way using symbol table objects. The current solution is compatible to the HA57 SDK.

> **Note**
> Symbol tables type-safely implement all function calls in the firmware as well as in the application. Warnings of the compiler should be paid attention.

After a firmware update on the device, the SDK version might increase and functions might have been added at the end of the structure. This doesn't matter for older implementations.

8.2.2 Versioning

The version concept with a SDK is more complex than a simple firmware of a device. The firmware has its own version, which in most cases consists of a main version and a subversion number, even in some cases of a build number or a patch level. This version is independent of the SDK. The version of the SDK relates to an API version, which ideally shouldn't be changed with the firmware. SDK expansions are allowed, which changes the subversion of the SDK, but not the firmware (only the build number, as firmware functions should remain unchanged). Therefore, SDK versions and firmware versions are independent values.

In addition, there is an SDK version from the point of view of the writer of an application, which is noted in the software (which is installed on the PC) for the SDK (in HA57evo_system.h). This is the version of the used SDK API. The device firmware, where the compilation is loaded to, can offer a different SDK version (existing SDK version). There is no problem if the subversion of this firmware is higher, but, if it is lower, the registration of the application won't work. The entry function sees the current SDK version of the firmware in the system table and needs to write the required version of the compilation into the user info.

8.3 Required Code Skeleton

A specific code skeleton is required to create an application. Compliance with this scheme guaranties a correct integration of the application into the running system and its registration in the Setup for correct (re-/de-) activation. The registration process in the Setup is described in detail in the user manual.

8.3.1 Type Name Definition

The following type names were predefined to the application type:

appl_types_t	typedef enum

8.3.1.1 appl_types_t

This data type provides the application type for the registration.

Syntax

```
typedef enum
{   ATYPE_NON,
    ATYPE_PROGRAM,
    ATYPE_HA20XEXT,
    ATYPE_HA400EXT
}   appl_types_t;
```

Parameter
ATYPE_NONE No application is activated, meaning there is no jump to the callbacks of the application. ATYPE_PROGRAM The application controls the complete HA57 EVO using the API and replaces the protocol handlers of the serial interface. Therefore, loading of applications is not possible for this activation. The serial interface works byte-oriented without command interpreter, binary transfers should work without problems. ATYPE_HA20XEXT The application complements or modifies the HA20x protocol. Some API areas of the PROGRAM mode are not available. The serial interface works command oriented. ATYPE_HA400EXT The application complements or modifies the HA400 protocol. Some API areas of the PROGRAM mode are not available. The serial interface works command oriented.

8.3.2 Struct appl_Info_s

This structure is used to report

- the name of the application
- the SDK version at compilation (can deviate from the firmware version after an update)
- the type of the application
- the event callback
- the font handler

to the system. When, at activation, the SDK version is higher than the version of the current system or even wrong (magic code), the message "**Application Version Mismatch**" is shown on the display for 10 s. In this case, the application is not activated.

Both callback functions are allowed to be NULL, but it does not make sense to load an application without callback function.

The application type indicates the way, how the code of the application is integrated. An application of the type "PROGRAM" takes over total control over the handset after boot up. A handset of the type "EXPANASION" extends one of the protocols of the HA57 EVO.

Definition

```
typedef struct
{   const char *applName;
    #define SDK_MAGIC 0x06022019

    #define SDK_IF_VERSION 1
    #define SDK_VERSION (SDK_MAGIC + SDK_IF_VERSION)
    uint32_t SDK_VersionCode;
    appl_types_t applType;
    void* (*application)(unsigned short event);
    void (*char_info)(uint32_t utf32, appl_typeface_t font,
    appl_char_info_t *info);
}   appl_Info_s;
```

Parameter

```
applName
    Name of the application for menu selection

SDK_MAGIC
    Magic code for structural testing

SDK_IF_VERSION 1
    Requested SDK interface version. Might be lower than the implemented version in
the device.

SDK_VERSION
    Value to be inserted into the field SDK_VERSION code.

SDK_VersionCode
    Handed over value for version testing.

applType
    Requested type of application

void* (*application)(unsigned short event)
    Applications event handler (function pointer) with the argument:
    event: Mask, dependent on the type of application.
    The return type is either NULL, which makes the device reboot in the ATYPE_PROGRAM
mode or a pointer (which might be NULL as well) to a status value for both protocol
extensions.

void (*char_info)(uint32_t utf32, appl_typeface_t font, appl_char_info_t *info)
    Font handler (function pointer) with the arguments:
    utf32           the UTF32 character to be drawn
    font            the font to be used (size and formatting)
    info (in/out)   character object to be pre-filled and continuously refilled.
Expected font, width, height, no bitmap, character width 0.
```

The application function needs as return a static object of this structure, preferably, it can be stored in the flash constantly as shown in the examples.

> **Note**
> This structure was modified slightly compared with the old SDK and the handover was changed.

8.3.3 System Events

The program sequence of an application is controlled by events. Therefore, each action is triggered by an event. The following table shows all available events.

Only use the events that are available for the respective application type.

Event	Description
EV_KEY_INPUT	A key was pressed.
EV_KEY_RELEASED	A key was released.
EV_KEY_TIMEOUT	The KEY timer has expired.
EV_HOOK_INPUT	The state of the HOOK switch changed.
EV_PTT_INPUT	The state of the PTT key changed.
EV_POWER_ON	The event is sent immediately after POWER ON; the application software might react with an initialization of the system (display, configuration).
EV_LOGO_TIMEOUT	The LOGO timer has expired.
EV_BYTE_INPUT	A byte was received at the UART (for programs only).
EV_LIGHT_TIMEOUT	The LIGHT timer has expired.
EV_USER1_TIMEOUT	The custom timer 1 has expired.
EV_USER2_TIMEOUT	The custom timer 2 has expired.
EV_USER3_TIMEOUT	The custom timer 3 has expired.
EV_VERSION_GET	For system information, a version information can be handed over.

Table 3: Event List for Application Type "Program"

Event	Description
EXP_EV_KEY_INPUT	A key was pressed.
EXP_EV_KEY_RELEASED	A key was released.
EXP_EV_KEY_TIMEOUT	The KEY timer has expired.
EXP_EV_HOOK_INPUT	The state of the HOOK switch changed.
EXP_EV_PTT_INPUT	The state of the PTT key changed.
EXP_EV_POWER_ON	The event is sent immediately after POWER ON; the application software might react with an initialization of the system (UART, display, configuration).
EXP_EV_COMMAND_RECEIVED	A command was received.
EXP_EV_TEXT_RECEIVED	A text was received.
EXP_EV_USER_TIMER_1	The custom timer 1 has expired.
EXP_EV_USER_TIMER_2	The custom timer 2 has expired.
EXP_EV_USER_TIMER_3	The custom timer 3 has expired.

Table 4: Event List for Application Type "Expansion"

8.3.4 Function application()

This function is the entry point of the application. The application programmer must ensure that the linker places the implementation of this function at the defined entry point address 0x08050000.

The example applications and the project templates for new projects already contain the necessary linker settings.

KEIL™:

"-entry=application" is added to the linker options. The memory mapping must be defined in the Options.

GCC:

The memory map is defined by respective records in the file HA57EvoMem.ld. These settings will be loaded by the Makefile via LDSCRIPT.

8.3.5 Event Handler

The code of the *default* path of the event selection needs to be kept as described. The system expects, when calling an application without event, the return of the pointer to the structure *appl_Info_s* for a possible update of the name. This approach is compatible with the old SDK.

Example: Application of the type "Program"

```
appl_Info_s  applInfo;

void *application (unsigned short event)
{   int rv = 1;
    void * ptr = &rv;
    if (event & EV_LOGO_TIMEOUT)
    {   /* Timer overflow LOGO_TIMER */
    }
    if (event & EV_KEY_TIMEOUT)
    {   /* Timer overflow KEY_TIMER */
    }
    if (event & EV_LIGHT_TIMEOUT)
    {   /* Timer LIGHT_TIMER has expired */
    }
    if (event & EV_KEY_INPUT)
    {   /* Key pressed */
    applicationKeyHandler();
    }
    if (event & EV_USER_TIMER1)
    {   /* Timer overflow USER_TIMER1 */
    }
    if (event & EV_USER_TIMER2)
    {   /* Timer overflow USER_TIMER2 */
    }
    if (event & EV_USER_TIMER3)
    {   /* Timer overflow USER_TIMER3 */
    }
    if (event & EV_KEY_RELEASED)
    {   /* Key released */
    }
    if (event & EV_HOOK_INPUT)
    {   /* Hook status changed */
    }
    if (event & EV_PTT_INPUT)
    {   /* PTT status changed */
    }
    if (event & EV_VERSION_GET)
    {   /* Application Version request */
        ptr = "1.00.01";
    }
    if (event & EV_POWER_ON)
    {   /* First call after power on */
        applicationInit();
    }
    if (event & EV_BYTE_INPUT)
    {   /* UART has received bytes in byte mode */
    }
    if (event == 0)
    {   /* Name of application as it will appear in setup menu */
        applInfo.applName = "HA57app";
```

```
        /* !!DO NOT MODIFY NEXT LINES!! */
            applInfo.SDK_VersionCode = (u32)&SDK_VERSION;
            applInfo.applType = ATYPE_PROGRAM;
            ptr = &applInfo;
    }
    return ptr;
}
```

Example: Application of the type "Expansion"

```
appl_Info_s  applInfo;

void *application (unsigned short event)
{   int rv = 1;
    void * ptr = &rv;
if (event & EXP_EV_COMMAND_RECEIVED)
    {   /* command received */
    rv = commandReceived();
    }
    if (event & EXP_EV_TEXT_RECEIVED)
    {   /* text received */
        rv = textReceived();
    }
    if (event & EXP_EV_KEY_INPUT)
    {   /* Key pressed */
        rv = applicationKeyHandler();
    }
    if (event & EXP_EV_KEY_RELEASED)
    {   /* Key released */
        rv = applicationKeyHandler();
    }
    if (event & EXP_EV_KEY_TIMEOUT)
    {   /* key timer overflow */
    }
    if (event & EXP_EV_HOOK_INPUT)
    {   /* HOOK status changed */
    }
    if (event & EXP_EV_PTT_INPUT)
    {   /* PTT status changed */
    }
    if (event & EXP_EV_POWER_ON)
    {   /* First call after power on */
        applicationInit();
    }
    if (event == 0)
    {   /* Name of application as it will appear in setup menu */
        applInfo.applName = "HA57app_new_name";
    }
    return ptr;
}
```

The return value of the function "application" is a pointer. This pointer is interpreted differently depending on the type of application.

Program:	Pointer to requested information or pointer to 1. In case of a *NULL* – abortion of the application and restart of the device.
Expansion:	For events that signal an action, the pointer points to 0, when the signaled input was treated conclusively. When the pointer points to 1, the input is still treated. For events that request information, the pointer points to the required information.

8.3.6 Font Handler

The font handler universally replaces the bitmap functions of the old SDK on UTF basis. The handler gets an UTF32 value, the font and a prefilled char structure which contains the expected height and width of the font in pixels. For particular characters, the values might differ. The reference for the placement on the display is always the upper left corner of the character. It is expected that the bitmap of the character and resolution of the bitmap are registered. If not, the compiled font description is used or a replacement character (blank space) is displayed.

In this way, fonts of the HA57 EVO can be extended or replaced.

Definition

```
typedef struct
{   uint8_t font_width;
    uint8_t font_height;
    uint8_t height;
    uint8_t width;
    const uint8_t* bitmap;
}   appl_char_info_t;
```

Parameter

```
uint8_t font_width;
    Font width (for monospace fonts and spaces inclusive blanks)

uint8_t font_height;
    Font height, not to be confused with pixel height

uint8_t height;
    Height in pixel

uint8_t width;
    Width of the bitmap in pixel

const uint8_t* bitmap;
    Height * width / 8 bytes
```

The example **Font extensions** shows the currently in the firmware available default UTF8 implementation.

To consider memory space, the colour depth of the pixel descriptions for fonts is limited to 1.

8.3.7 Special Features

The HA57 EVO API offers no virtual or especially protected environment. The code runs directly in the context of the firmware and especially without the normally usual C startup. Here are the consequences:

- Crashes will also crash the firmware, which will be in most cases able to react per fault handler. When the fault handler locates the crash in the application, the application will be deactivated at next startup and needs to be reactivated by hand. This will avoid boot loops.
- The idle handler of the system offers a simple busy loop detection. When the event handler needs too much time, the device will be rebooted by Watchdog, at this point without deactivation of the user code. A delay function (as well with 0) as argument reboots the Watchdog. Generally, it is recommended to avoid busy loops in the code (see also coding of UART tests).
- When linking the own application with own libraries, the required startup initializations are not called. This is also valid for the 0-initialization of static/global variables, pre-initialized objects and especially the necessary C++ startups (ctor) at C++ usage. Therefore, such code won't work at some places. With special effort, it is possible to implement this at the entry point of the application for the used compiler. However, this is not part of the SDK API! For GCC there is an approach to realize the 0 initialization and pre-initialization of the objects (might not work depending on the compiler). In general, it should be assumed that not all variables are initialized!
- In cases where it isn't possible to enter the config menu of the device after a non-detectable problem in order to deactivate the application, the device needs to be sent in to the manufacturer. Therefore, it is recommended during the development phase to show the start-up logo for a longer amount of time to be able to activate the config menu in this time.
- Event functions are normally called by exactly one thread of the firmware and do not need to be implemented reentrant. Interrupt events (timer, serial data, keys…) are handled per event and respective firmware threads and might be delayed.

8.4 Display

This section describes all functions provided for writing to the display. These functions range from the display of plain text all the way to controlling individual pixels of the display.

8.4.1 Defined Types

The following type has been defined for the use by DISPLAY functions:

appl_display_mode_t	typedef enum

8.4.1.1 appl_display_mode_t

For simple display of fonts, especially for normal protocol functions, pre-defined display profiles for the text area (SECTION B) can be selected. These control implicitly the font size, placement and the mapping of bytes to UTF characters (compatible to the protocol description). Of course, current developments should be based on UTF. When the text functions are used for display modes, the firmware provides a text memory and the necessary refresh of the text is reduced to changes.

Syntax

```
typedef enum
{   DISP_MODE_ASCII = 0,
    DISP_MODE_SMS = 1,
    DISP_MODE_TB = 2,
    DISP_MODE_CYRILL = 3,
    DISP_MODE_BIG_SHIFT = 4,
    DISP_MODE_BIG_CENTER = 13,
    DISP_MODE_UTF8 = 17
}   appl_display_mode_t;
```

Parameter

```
DISP_MODE_ASCII

DISP_MODE_SMS

DISP_MODE_TB

DISP_MODE_CYRILL
    Normal mode, character set according to the HA20x Code Table

DISP_MODE_BIG_SHIFT
    Mixed mode

DISP_MODE_BIG_CENTER
     Centred text

DISP_MODE_UTF8
    Modern UTF-8 coding of the strings for normal display
```

8.4.1.2 appl_section_t

The HA20x defines three display sections:

- A: Symbols
- B: Text
- C: Navigation

Syntax

```
typedef enum
{   SECTION_A = 1,
    SECTION_B = 2,
    SECTION_C = 4,
}   appl_section_t;
```

Parameter

```
SECTION_A
    Section symbol area

SECTION_B
    Section text area

SECTION_B
    Section navigation symbols and softkeys
```

Example

```
appl_section_t  section;
section = SECTION_C;
```

Mixed areas like SECTION_AB, … are defined as well, to ensure a type-save handover for bit masks.

8.4.2 Functions

The following DISPLAY functions are available:

appl_LCD_clear()
appl_LCD_clearCommonScreen()
appl_LCD_clearSection()
appl_LCD_setDisplayMode()
appl_LCD_setBrightness()
appl_LCD_getBrightness()
appl_LCD_setCompatibility()

8.4.2.1 appl_LCD_clear()

This function clears the complete display using the specified colour. The colour can be selected from pre-defined colour values or custom values can be used (see also 8.6.1 Encoding of colours).

Syntax

```
void appl_LCD_clear(u16 color);
```

Parameter

```
color
    16 Bit value in accordance with section 8.6.1 Encoding of colours
```

Reply

None

Example

```
appl_LCD_clear(White);
```

8.4.2.2 appl_LCD_clearCommonScreen()

This function clears the HA20x-typical display. The areas for icons, text and soft keys are filled with their initialized background colours.

Syntax

```
void appl_LCD_clearCommonScreen(void);
```

Parameter

None

Reply

None

Example

```
appl_LCD_clearCommonScreen();
```

8.4.2.3 appl_LCD_clearSection()

This function clears single sections separately or in combination on the HA20x-typical display. The display areas for icons, text and softkeys appear in the initialized background colours.

Syntax

```
void appl_LCD_clearSection( section_t section);
```

Parameter

```
Sections to be deleted
```

Reply

```
None
```

Example

```
appl_LCD_clearSection( SECTION_AC );
```

8.4.2.4 appl_LCD_setDisplayMode()

This function selects the pre-defined display mode. This command resets the associated text buffer and the cursor position.

Syntax

```
void appl_LCD_setDisplayMode(appl_display_mode_t dm);
```

Parameter

```
dm
    Display mode specified as value of type appl_display_mode_t
```

Reply

None

Example

```
appl_LCD_setDisplayMode(DISP_MODE_BIG_SHIFT);
```

8.4.2.5 appl_LCD_setBrightness()

This command sets the display brightness: 21 levels of brightness are available. Values range from 0 to LCD_MAX_BRIGHTNESS (20). The value 0 switches the illumination off.

Syntax

```
void appl_LCD_setBrightness(const u8 brightness);
```

Parameter

```
brightness
    Brightness 0 … 20
```

Reply

None

Example

```
appl_LCD_setBrightness ( 5 );
```

8.4.2.6 appl_LCD_getBrightness()

Query of the display brightness.

Syntax

```
u8 appl_LCD_getBrightness(void);
```

Parameter

None

Reply

```
u8
    Brightness  0… 20
```

Example

```
brightness = appl_LCD_getBrightness ();
```

8.4.2.7 appl_LCD_setCompatibility()

Some functions now interpret the colour space RGB555 instead of RGB565 and the bmp and bitmap functions scale by the factor 2. This helps to port already available HA57 SDK applications faster and as well to avoid memory problems.

Syntax

```
Syntax:    u8 appl_LCD_getBrightness(void);
```

Parameter

```
Boolean
    TRUE    HA57 SDK compatibility mode
    FALSE   normal SDK functions
```

Reply

None

8.5 Text

This section describes the API for the output of text using pre-defined fonts.

8.5.1 Defined Types

The following types have been defined for the TEXT area:

appl_typeface_t	typedef enum
appl_text_section_t	typedef enum
appl_shapes_t	typedef enum
appl_text_description_s	typedef struct
appl_scr_text_s	typedef struct

8.5.1.1 appl_typeface_t

Generally, there are fonts in three sizes on offer and for small and normal fonts the bold varieties. They can be selected using this parameter.

Definition

```
typedef enum
{   FONT_SMALL
    FONT_NORMAL
    FONT_LARGE
    FONT_SMALL_BOLD
    FONT_NORMAL_BOLD
    FONT_NR
}   appl_typeface_t;
```

Parameter

```
Font

FONT_SMALL
    Small font

FONT_NORMAL
    Normal default font

FONT_LARGE
    Big font

FONT_SMALL_BOLD
    Small bold font

FONT_NORMAL_BOLD
    Normal bold font

FONT_NR
    Number of supported fonts
```

8.5.1.2 appl_text_section_t

A separate font type can be configured for each component of the defined text sections.

Syntax

```
typedef enum
{   TEXT_SECTION,
    SOFTKEY_SECTION
}   appl_text_section_t;
```

Parameter

```
TEXT_SECTION
    Text area

SOFTKEY_SECTION
    Softkey area
```

8.5.1.3 appl_shapes_t

Font display can be modified as follows:

Definition

```
typedef enum
{   SH_ORDINARY     = 0x0000,
    SH_CURSOR       = 0x0001,
    SH_UNDERLINE    = 0x0002,
    SH_TRANSPARENT  = 0x0040,
    SH_INVERS       = 0x0080,
    SH_VARIABLE     = 0x0100
}   appl_shapes_t;
```

Parameter

```
SH_ORDINARY
    Ordinary display: Text color on background color.

SH_INVERS
    Invers: Text in background colour, background in text colour

SH_TRANSPARENT
    Transparent (Text in text colour on existing background)

SH_CURSOR
    Underlined with 3 pixels thickness

SH_UNDERLINE
    Underlined with 2 pixels thickness

SH_VARIABLE
    Variable justification
```

8.5.1.4 appl_text_description_s

This struct collects different aspects of the formatted output of a text.

Syntax

```
typedef struct
{   char *text;
    u16 x;
    u16 y;
    u8 cursor;
    appl_shapes_t shape;
    appl_letter_size size
    appl_typeface_t font;
}   appl_text_description_s;
```

Parameters

```
text
    Pointer to the UTF8 text to be displayed. Must be zero-terminated.

x
    X-coordinate of the start of output (Range: 0 to LCD_MAX_X-1 (0-239)).

y
    Y-coordinate of the start of output (Range: 0 to LCD_MAX_Y-1 (0-319)).

cursor
    Compatibility; parameter is ignored

shape
    Display mode specified as value of type appl_shapes_t

size
    Compatibility; modifying factor for bold, can be set also by using font.

font
    Font type of the text in according with type appl_typeface_t.
```

8.5.1.5 appl_scr_text_s

This struct describes the properties of a text scrolling through a display line.

Syntax

```
typedef struct
{   u8 step;
    appl_typeface_t font;
    u8 line;
    char *text;
}   appl_scr_text_s;
```

Parameter

```
step
    Period of a single scroll step in units of 5ms

font
    Typeface specified as value of type appl_typeface_t

line
    Position of the scrolling text line (24 pixels each);
    Value range: 0 to LCD_MAX_LINES-1 (0-13)

text
    Pointer to the UTF8 text to be displayed. Must be zero-terminated.
```

8.5.2 Functions

The following TEXT functions are provided:

appl_LCD_setTextOffset()
appl_LCD_setTextCursorPos()
appl_LCD_setTextSurface()
appl_LCD_writeText()
appl_LCD_centerText()
appl_LCD_displayDefinedChar()
appl_LCD_displayText()
appl_LCD_enableCursor()
appl_LCD_scrText()
appl_LCD_scrClear()

> **Notice**
> Some text functions support the text modes with text memory, other display the text directly at the given coordinates without buffer.

8.5.2.1 appl_LCD_setTextOffset()

An offset in pixels is defined at start of a text field for text modes (normally 52). This setting influences the text output with *appl_LCD_writeText()*.

> **Notice**
> For the HA57 SDK this parameter was defined quite vaguely. Therefore, the semantic has changed.

Syntax

```
void appl_LCD_setTextOffset(u16 toffs);
```

Parameter

```
toffs
    Distance of the text field from the upper edge of the display in pixels
```

Reply

```
None
```

Example

```
appl_LCD_setTextOffset(16);
```

8.5.2.2 appl_LCD_setTextCursorPos()

This function places the cursor at the specified character position within the text section. The position is counted in characters from the start. The current cursor position affects the text output by *appl_LCD_writeText()*.

Syntax

```
void appl_LCD_setTextCursorPos(u16 tcpos);
```

Parameter

```
tcpos
    New position of the text cursor;
    Range: LCD_MAX_LINES * MAX_CHAR_PRO_LINE_12 -1 (0-223)
```

Reply

```
None
```

Example

```
appl_LCD_setTextCursorPos(0);
```

8.5.2.3 appl_LCD_setTextSurface()

This function configures the typeface to be used for the text output with the functions *appl_LCD_writeText()* or *appl_LCD_showSoftkey()*.

Syntax

```
void appl_LCD_setTextSurface(appl_text_section_t section,
appl_typeface_t font);
```

Parameter

```
section
    Output section specified as value of appl_text_section_t

font
    Typeface specified as value of type appl_typeface_t
```

Reply

None

Example

```
appl_LCD_setTextSurface(TEXT_SECTION, FONT_NORMAL);
appl_LCD_setTextSurface(SOFTKEY_SECTION, FONT_NORMAL_BOLD);
```

8.5.2.4 appl_LCD_writeText()

This function outputs the specified text beginning at the current cursor position into the text field while taking into account the configured display mode. At the end of a line, the text is continued in the next line. The newline-character \n forces a premature line break. Usually, only an update of changed content appears on the display. This very efficient procedure restricts to fixed letter spacing and the display modes available.

Syntax

```
void appl_LCD_writeText(char *text);
```

Parameter

```
text
    Pointer to the zero-terminated string containing the text to display.
```

Reply

None

Example

```
appl_LCD_writeText("Settings");
```

8.5.2.5 appl_LCD_centerText()

This function displays the specified UTF text in the center of the text section. The output can occur in maximum 6 centered lines, the newline-character forces a line break.

Syntax

```
void appl_LCD_centerText(appl_typeface_t font, char *text);
```

Parameter

```
font
    Typeface of the text in accordance with type appl_typeface_t

text
    Pointer to the zero-terminated string containing the text to display.
```

Reply

```
None
```

Example

```
appl_LCD_centerText(FONT_LARGE,"Text");
```

8.5.2.6 appl_LCD_displayDefinedChar()

An ASCII character from the predefined font is written to the specified position. This function exists for compatibility reasons and extremely restricts the normal function of the text output.

Syntax

```
void appl_LCD_displayDefinedChar(u16 X, u16 Y, char ascii, appl_typeface_t font,
appl_status_t cu);
```

Parameter

```
x
    X-coordinate of the start of output (Range: 0 to LCD_MAX_X-1 (0-239)).

y
    Y-coordinate of the start of output (Range: 0 to LCD_MAX_Y-1 (0-319)).

ascii
    ASCII code of the character to display.

font
    Typeface of the text in accordance with type appl_typeface_t

cu
    cursor state in accordance with type appl_status_t
```

Reply

```
None
```

Example

```
appl_LCD_displayDefinedChar(10,10,'A',FONT_NORMAL,ON);
```

8.5.2.7 appl_LCD_displayText()

The text defined in the structure *appl_text_description_s* is directly written at the specified coordinates. The text memory of the text modes remains unchanged. Therefore, the cursor field doesn't make sense and can be ignored. This is the most flexible way of text output; the user himself is responsible for refresh actions.

Syntax

```
void appl_LCD_displayText(appl_text_description_s *td);
```

Parameter

```
td
    Pointer to the structure of the type appl_text_description_s
```

Reply

None

Example

```
text_description_s  text_description;

appl_LCD_clear(White);
text_description.text = "First line";
text_description.X = 5;
text_description.Y = 30;
text_description.cursor = 0;
text_description.shape = SH_ORDINARY;
text_description.font = FONT_NORMAL;

appl_LCD_displayText(&text_description);
text_description.Y += 15;
text_description.text = "Second line";

appl_LCD_displayText (&text_description);
```

8.5.2.8 appl_LCD_enableCursor()

This function turns the display of the text cursor for text modes on or off.

Syntax

```
void appl_LCD_enableCursor(appl_status_t  cue);
```

Parameter

```
cue
    Cursor status specified as value of type appl_status_t
    ON      display cursor
    OFF     display no cursor
```

Reply

None

Example

```
appl_LCD_enableCursor(ON);
```

8.5.2.9 appl_LCD_scrText()

This function makes the specified text scroll through one line of the display according to the parameters that accompany it within the passed struct argument.

Syntax

```
void appl_LCD_scrText(appl_scr_text_s *scrt);
```

Parameter

```
scrt
    Pointer to a appl_scr_text_s struct.
```

Reply

None

Example

```
appl_scr_text_s  scrt;

scrt.step  = 3;                 // Shift Time Step: 15 ms
scrt.font  = FONT_NORMAL;
scrt.line  = 5;                 // Scrolling Text in Line 5
scrt.text  = "Scrolling Text";

appl_LCD_scrText (&scrt);
```

8.5.2.10 appl_LCD_scrClear()

This function stops the output of scrolling text and clears the internal line buffer. The previously occupied display line is now available for other output.

Syntax

```
void appl_LCD_scrClear(void);
```

Parameter

None

Reply

None

Example

```
appl_LCD_scrClear();
```

8.6 Colours

The types and functions described here are used for the colour settings of all parts of the display output.

8.6.1 Encoding of Colours

Colours can be defined with custom values in a colour depth of up to 16 Bit in accordance with the following coding:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	r	r	r	r	r	g	g	g	g	g	g	b	b	b	b	b
	Red					Green						Blue				

Pre-defined constants are available for 10 common standard colors:

Syntax

```
#define White       0xFFFF      // White
#define Black       0x0000      // Black
#define Grey        0xF7DE      // Gray
#define Blue        0x001F      // Blue
#define Blue2       0x051F      // Light Blue
#define Red         0xF800      // Red
#define Magenta     0xF81F      // Magenta
#define Green       0x07E0      // Green
#define Cyan        0x7FFF      // Cyan
#define Yellow      0xFFE0      // Yellow
```

8.6.2 Defined Types

The following types have been defined for the use by color functions:

appl_color_section_t	typedef enum

8.6.2.1 appl_color_section_t

The values of this enum identify all the defined display sections that can be assigned a separate colour. The selection of a colour is available for

- text and its background,
- icons and their background,
- soft key text and its background, and
- graphics output.

Colours are encoded as 16-bit values as specified by 8.6.1 Encoding of Colours.

Syntax

```
typedef enum
{   TEXT_COLOR,
    TEXT_BACKGROUND_COLOR,
    ICON_COLOR,
    ICON_BACKGROUND_COLOR,
    SOFTKEY_COLOR,
    SOFTKEY_BACKGROUND_COLOR,
    GRAPHIC_COLOR
}   appl_color_section_t;
```

Parameters

```
TEXT_COLOR
    Colour used for text output.

TEXT_BACKGROUND_COLOR
    Colour used for the background of text output.

ICON_COLOR
    Colour used for the display of icons.

ICON_BACKGROUND_COLOR
    Colour used for the background of icons.

SOFTKEY_COLOR
    Colour used for the display of soft key labels.

SOFTKEY_BACKGROUND_COLOR
    Colour used for the background of soft key labels.

GRAPHIC_COLOR
    Colour used for graphics output.
```

8.6.3 Functions

The following COLOR functions are provided:

appl_LCD_setTextColor()
appl_LCD_setBackColor()
appl_LCD_Color()

8.6.3.1 appl_LCD_setTextColor()

Configures the colour for pixel-based, direct text output. The colour settings are used immediately. Subsequently output text will be displayed in the configured colour.

Syntax

```
void appl_LCD_setTextColor(u16 color);
```

Parameter

```
color
    16-bit value as specified by 8.6.1 Encoding of Colours.
```

Reply

```
None
```

Example

```
appl_LCD_setTextColor(Black);
```

8.6.3.2 appl_LCD_setBackColor()

This function configures the colour painted in the background of pixel-based text output. Subsequently output text will be displayed on top of a background painted in the configured colour.

Syntax

```
Syntax:    void appl_LCD_setBackColor(u16 color);
```

Parameter

```
color
    16-bit value as specified by 8.6.1 Encoding of Colours.
```

Reply

None

Example

```
appl_LCD_setBackColor(White);
```

8.6.3.3 appl_LCD_Color()

This function re-sets colours for the specified display area and the area in question, with exception of GRAPHIC_COLOR, is updated.

Syntax

```
void appl_LCD_Color(appl_color_section_t section, u16 color);
```

Parameter

```
section
    Display section specified as value of type appl_color_section_t.

color
    16-bit value as specified by 8.6.1 Encoding of Colours.
```

Reply

None

Example

```
appl_LCD_Color(ICON_COLOR, Blue);
```

8.7 LED

This section describes how to control the lighting of the display and the keypad.

8.7.1 Defined Types

The following typenames have been predefined for the LED section:

appl_ledCommand_t	typedef enum

8.7.1.1 appl_ledCommand_t

This enum identifies the commands available for the lighting control.

Syntax

```
typedef enum
{   LED_CMD_OFF,
    LED_CMD_ON,
    LED_CMD_DARK,
    LED_CMD_LIGHT
}   appl_ledCommand_t;
```

Parameter

```
LED_CMD_OFF
    Turn lighting OFF.

LED_CMD_ON
    Turn lighting ON.

LED_CMD_DARK
    Turn lighting ON with reduced brightness of LCD.

LED_CMD_LIGHT
    Turn lighting ON with full brightness of LCD.
```

8.7.2 Functions

The following LED functions are provided:

appl_LED_control()

8.7.2.1 appl_LED_control()

This function switches the LED lighting according to the specified command.

Syntax

```
void appl_LED_control(appl_ledCommand_t cmd);
```

Parameter

```
cmd
    Command specified as value of type appl_ledCommand_t
```

Reply

```
None
```

Example

```
appl_LED_control(LED_CMD_OFF);
```

8.8 Graphics

This section describes the provided means for graphics output.

8.8.1 Defined Types

The following types have been defined for the use by graphics functions:

appl_direction_t	typedef enum

8.8.1.1 appl_direction_t

Values of this enum specify the direction of lines drawn in parallel to one of the axes.

Syntax

```
typedef enum
{   DIR_HORIZONTAL,
    DIR_VERTICAL
}   appl_direction_t;
```

Parameter

```
DIR_HORIZONTAL
    Horizontal (X) direction, from left to right.

DIR_VERTICAL
    Vertical (Y) direction, from top to bottom.
```

8.8.2 Functions

The following GRAPHICS functions are provided:

appl_LCD_drawRect()
appl_LCD_DrawCircle()
appl_LCD_DrawLine()
appl_LCD_writePixelPrepare()
appl_LCD_writePixel()
appl_LCD_setPixelPosition()

8.8.2.1 appl_LCD_drawRect()

This function draws a rectangle. It is specified by the coordinates of its top left corner.

Syntax

```
void appl_LCD_drawRect(u16 x, u16 y, u16 height, u16 width);
```

Parameter

```
x
    X-coordinate of the top left corner point; Range: 0 to LCD_MAX_X-1 (0-239)

y
    Y-coordinate of the top left corner point; Range: 0 to LCD_MAX_Y-1 (0-319)

height
    Height of the rectangle in pixels; Range: 1 to LCD_MAX_Y (1-320)

width
    Width of the rectangle in pixels; Range: 1 to LCD_MAX_X (1-240)
```

Reply

None

Example

```
appl_LCD_drawRect(10, 10, 30, 20);
```

8.8.2.2 appl_LCD_DrawCircle()

This function draws a circle specified by its center and its radius.

Syntax

```
void appl_LCD_DrawCircle(u16 x, u16 y, u16 radius);
```

Parameter

```
X
    X-coordinate of the center point; Range: 0 to LCD_MAX_X-1 (0-239)

Y
    Y-coordinate of the center point; Range: 0 to LCD_MAX_Y-1 (0-319)

radius
    Radius of the circle in pixels; Range: 1 to LCD_MAX_X (1-240)
```

Reply

None

Example

```
Example:   appl_LCD_DrawCircle(50, 50, 20);
```

8.8.2.3 appl_LCD_DrawLine()

This function draws a line from the specified starting point and of the specified length. The coordinates refer to the start point on the left or on top.

Syntax

```
void appl_LCD_DrawLine(u16 x, u16 y, u16 length, appl_direction_t direction);
```

Parameter

```
x
    X-coordinate of the starting point; Range: 0 to LCD_MAX_X-1

y
    Y-coordinate of the starting point; Range: 0 to LCD_MAX_Y-1

length
    Length of the line in pixels; Range: 1 to LCD_MAX_Y (1-320)

direction
    The direction of the line to be drawn specified as value of type appl_direction_t
```

Reply

None

Example

```
appl_LCD_DrawLine(10, 10, 30, DIR_HORIZONTAL);
```

8.8.2.4 appl_LCD_writePixelPrepare()

This function prepares the display for the output of pixel data. Technically, the display is put into data mode and then interprets the next data as colours.

Syntax

```
void appl_LCD_writePixelPrepare(void);
```

Parameter

None

Reply

None

Example

```
appl_LCD_writePixelPrepare();
```

8.8.2.5 appl_LCD_writePixel()

This function outputs a single pixel in the specified color. This only works when the display is in data mode. It is recommended to avoid writing many single pixels.

Syntax

```
void appl_LCD_writePixel(u16 color);
```

Parameter

```
color
    16-bit value as specified by 8.6.1 Encoding of Colours.
```

Reply

None

Example

```
appl_LCD_writePixel(Green);
```

8.8.2.6 appl_LCD_setPixelPosition()

This function places the pixel cursor at the specified position so that the next output of a pixel will occur there. Technically, a data window, with the length of one pixel and the width of one pixel is created. This is an extremely inefficient process!

Syntax

```
void appl_LCD_setPixelPosition(u16 x, u16 y);
```

Parameter

x
    New x-coordinate of the pixel cursor; Range: 0 to LCD_MAX_X-1 (0-239)

y
    New y-coordinate of the pixel cursor; Range: 0 to LCD_MAX_Y-1 (0-319)

Reply

None

Example

```
appl_LCD_setPixelPosition(10, 10);
```

8.9 Soft Keys

This section describes the control of the text fields used for soft keys.

8.9.1 Defined Types

The following types have been defined for the use by soft key functions:

appl_sk_side_t	typedef enum
appl_sk_property_t	typedef enum

8.9.1.1 appl_sk_side_t

The values of this enum identify one of the two text fields available for soft key labels.

Syntax

```
typedef enum
{   SK_LEFT,
    SK_RIGHT
}   appl_sk_side_t;
```

Parameter

```
SK_LEFT
    Label of left soft key.

SK_RIGHT
    Label of right soft key.
```

8.9.1.2 appl_sk_property_t

The values of this enum distinguish the static and the blinking display of the label text.

Syntax

```
typedef enum
{   SK_STATIC,
SK_BLINKING
}   appl_sk_property_t;
```

Parameter

```
SK_STATIC
    Static label text.

SK_BLINKING
    Blinking label text.
```

8.9 Soft Keys

This section describes the control of the text fields used for soft keys.

8.9.1 Defined Types

The following types have been defined for the use by soft key functions:

appl_sk_side_t	typedef enum
appl_sk_property_t	typedef enum

8.9.1.1 appl_sk_side_t

The values of this enum identify one of the two text fields available for soft key labels.

Syntax

```
typedef enum
{   SK_LEFT,
    SK_RIGHT
}   appl_sk_side_t;
```

Parameter

```
SK_LEFT
    Label of left soft key.

SK_RIGHT
    Label of right soft key.
```

8.9.1.2 appl_sk_property_t

The values of this enum distinguish the static and the blinking display of the label text.

Syntax

```
typedef enum
{   SK_STATIC,
    SK_BLINKING
}   appl_sk_property_t;
```

Parameter

```
SK_STATIC
    Static label text.

SK_BLINKING
    Blinking label text.
```

8.9.2 Functions

The following SOFTKEY functions are provided:

appl_LCD_showSoftkey()
appl_LCD_setSoftkeyProperty()

8.9.2.1 appl_LCD_showSoftkey()

This function updates the text of the specified soft key label.

Syntax

```
void appl_LCD_showSoftkey(appl_sk_side_t side, char *text);
```

Parameter

```
side
    Selects the soft key label to update as value of type appl_sk_side_t.

text
    Points to the zero-terminated string containing the new soft key label.
    Maximum length: 7 characters.
```

Reply

None

Example

```
Example:   appl_LCD_showSoftkey(SK_LEFT, "OK");
```

8.9.2.2 appl_LCD_setSoftkeyProperty()

This function updates how the specified soft key label is displayed.

Syntax

```
void appl_LCD_setSoftkeyProperty(appl_sk_side_t side, appl_sk_property_t pr);
```

Parameter

```
side
    Selects the soft key label to update as value of type appl_sk_side_t

pr
    Specifies how to display the label as value of type appl_sk_property_t
```

Reply

```
None
```

Example

```
appl_LCD_setSoftkeyProperty(SK_RIGHT, SK_STATIC);
```

8.10 Icons

The upper part of the display of the HA57 EVO can be used for the display of pre-defined icons. This section describes the handling of such icons.

8.10.1 Defined Types

The following types have been defined for the use by icon functions:

appl_icon_t	typedef enum
appl_rocker_mode_t	typedef enum

8.10.1.1 appl_icon_t

The values of this enum identify several system-defined icons:

- Field strength (signal strength)
- Roaming
- Numeric digit
- SMS
- Volume and
- Mute

Syntax

```
typedef enum
{   ICON_SIGNALSTRENGTH,
    ICON_ROAMING,
    ICON_MCALLS,
    ICON_SMS,
    ICON_VOLUME_HF,
    ICON_VOLUME_PR,
    ICON_MICRO_MUTE,
    ICON_AUDIO_MODE
}   appl_icon_t;
```

Parameter

```
ICON_SIGNALSTRENGTH
    Icon for the strength of the GSM signal.

ICON_ROAMING
    Roaming icon.

ICON_MCALLS
    Single numeric digit.

ICON_SMS
    SMS icon (envelope symbol).

ICON_VOLUME_HF
    Icon for the volume of the hands-free set.

ICON_VOLUME_PR
    Icon for the volume of the handset.

ICON_MICRO_MUTE
    Icon for the microphone mute.

ICON_AUDIO_MODE
    Icon for the audiomode.
```

The graphic representation of these icons is shown in the User Manual.

8.10.1.2 appl_rocker_mode_t

An icon displaying arrows for the rocker navigation may be shown between the text fields of the soft key labels. The values of this enum identify the available symbols.

Syntax

```
typedef enum
{   ROCKER_OFF,
    ROCKER_DOWN,
    ROCKER_UP,
    ROCKER_UP_DOWN,
    ROCKER_LEFT,
    ROCKER_RIGHT,
    ROCKER_LEFT_RIGHT,
    ROCKER_ALL
}   appl_rocker_mode_t;
```

Parameter

```
ROCKER_OFF
    No rocker navigation.

ROCKER_DOWN
    Downwards navigation only.

ROCKER_UP
    Upwards navigation only.

ROCKER_UP_DOWN
    Down- and upwards navigation.

ROCKER_LEFT
    Leftwards navigation only.

ROCKER_RIGHT
    Rightwards navigation only.

ROCKER_LEFT_RIGHT
    Left- and rightwards navigation.

ROCKER_ALL
    Complete rocker navigation (up, down, left, right).
```

The graphical representation of these navigation icons is shown in the User Manual.

8.10.2 Functions

The following ICON functions are provided:

appl_LCD_setIcon()
appl_LCD_drawRocker()

8.10.2.1 appl_LCD_setIcon()

This function displays a pre-defined icon.

Syntax

```
Syntax:    void appl_LCD_setIcon(appl_icon_t icon, u8 value);
```

Parameter

```
icon
    Specifies the icon to display as value of type appl_icon_t

value
    Value to be assigned to the icon. The valid range depends on the selected icon.
Detailed information is available in the User Manual.
```

Reply

```
None
```

Example

```
appl_LCD_setIcon(ICON_ROAMING, 1);
```

8.10.2.2 appl_LCD_drawRocker()

This function updates the icon displayed for the rocker navigation.

Syntax

```
void appl_LCD_drawRocker(appl_rocker_mode_t mode);
```

Parameter

```
mode
    The rocker navigation to offer specified as a value of type appl_rocker_mode_t
```

Reply

```
None
```

Example

```
appl_LCD_drawRocker(ROCKER_ALL);
```

8.11 Images and Bitmaps

The handset is capable of displaying images. It can handle color images in BMP format as well as monochrome bitmaps. A logo is a special BMP image, which is provided by the system.

8.11.1 Defined Types

The following types have been defined for the use by image and bitmap functions:

logo_t	typedef enum
appl_bmp_description_s	typedef struct
appl_bm_description_s	typedef struct

8.11.1.1 logo_t

The values of this enum select the logo to display. The default logo is the one of pei tel. A custom logo may also be available. If no custom logo has been stored to the device flash, the pei tel logo will be displayed regardless of the selection.

Syntax

```
typedef enum
{   DEFAULT_LOGO,
    CUSTOMER_LOGO
}   logo_t;
```

Parameter

```
DEFAULT_LOGO
    Select the pei tel logo.

CUSTOMER_LOGO
    Select the custom logo.
```

8.11.1.2 appl_bmp_description_s

This struct summarizes the properties of a stored image (BMP file). As usual for BMP images, the coordinate origin is the lower left corner. But, the position of the image is given in relation to [0,0] at the left upper corner of the display.

Syntax

```
typedef struct
{   u16 * bmpAddress;
    u8 xPos;
    u8 yPos;
    u8 xDots;
    u8 yDots;
    bool byteOrder;
}   appl_bmp_description_s;
```

Parameter

```
bmpAddress
    Pointer to 16-bit encoded BMP data.

xPos
    X-coordinate of image origin; Range: 0 to LCD_MAX_X-1 (0-239)

yPos
    Y-coordinate of image origin; Range: 0 to LCD_MAX_Y-1 (0-319)

xDots
    Width of image in pixels; Range: 1 to LCD_MAX_X (1-240)

yDots
    Height of image in pixels; Range: 1 to LCD_MAX_Y (1-320)

byteOrder
    Alinement of colour values: if set to TRUE, the MSB and the LSB of the 16-bit
values will be exchanged.
```

8.11.1.3 appl_bm_description_s

This function is kept for compatibility reasons. Modern system would use UTF symbols and set respective text. The UTF bitmaps of the device can be extended as desired.

This structure describes the properties of a user-defined bitmap image. The image is encoded line by line and bit by bit within a line. The individual lines are joined without gaps to form a compact bitmap representation. The bit stream is stored as a vector of 32-bit values.

Example: Definition of a new symbol of 6 by 5 pixels.

					Coding:
					1 1 1 1 1 b
					1 0 0 0 1 b
					1 0 0 0 1 b
					1 0 0 0 1 b
					1 0 0 0 1 b
					1 1 1 1 1 b

Resulting compact bitmap representation:

1	1	1	1	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	1	1	1	x	x
F				C				6				3				1				8				F				C			
0xFC								0x63								0x18								0xFC							

Table 5: Encoding of a Bitmap

x = don’t care

Bitmap: u32 square[1] = {0xFC6318FC};

The encoding of the image starts at the upper left corner. A set bit [1] represents a pixel of the text colour, a cleared bit [0] represents a pixel to be drawn in the background colour. The bitmap data must be padded to the next byte boundary. The values of possibly added bits are irrelevant.

Syntax

```
typedef struct
{   u32 *bitmap;
    u8 width;
    u8 height;
    u16 x;
    u16 y;
    u16 pixel_color;
    u16 back_color;
    appl_shapes_t shape;
}   appl_bm_description_s;
```

Parameter

```
bitmap
    Pointer to the compact bitmap encoding of the image.

width
    Width of the image in pixels; Range: 1 to LCD_MAX_X (1-240)

height
    Height of the image in pixels; Range: 1 to LCD_MAX_Y (1-320)

x
    X-coordinate of image origin; Range: 0 to LCD_MAX_X-1 (0-239)

y
    Y-coordinate of image origin; Range: 0 to LCD_MAX_Y-1 (0-319)

pixel_color
    Colour of active pixels (foreground)

back_color
    Background colour for non-active pixels

shape
    Display mode specified as value of type appl_shapes_t
```

8.11.2 Functions

The following IMAGE and BITMAP functions are provided:

appl_LCD_writeBMP()
appl_LCD_writeBitmap()
appl_LCD_writeLogo()

8.11.2.1 appl_LCD_writeBMP()

This function draws the BMP image as specified by the passed appl_bmp_description_s parameter. The BMP data has to be passed in "little endian" format resulting. As the storage is done mostly in "big endian", *byteOrder* must be set to TRUE. The colour coding RGB565 is mandatory and incompatible with the HA57 SDK (RGB555) meaning the images need to be recreated with adjusted colour space and resolution.

Syntax

```
Syntax:    void appl_LCD_writeBMP(appl_bmp_description_s *bmd);
```

Parameter

bmd
    Pointer to an appl_bmp_description_s struct.

Reply

None

Example

```
appl_bmp_description_s bmp_description;

bmp_description.bmpAddress = &pct;
bmp_description.xPos = 0;
bmp_description.yPos = 0;
bmp_description.xDots = LCD_MAX_X;
bmp_description.yDots = LCD_MAX_Y;
bmp_description.byteOrder = TRUE;

appl_LCD_writeBMP(&bmp_description);
```

8.11.2.2 appl_LCD_writeBitmap()

This function draws a bitmap as specified by the passed *appl_bm_description_s* parameter.

Syntax

```
void appl_LCD_writeBitmap(appl_bm_description_s *bm);
```

Parameter

```
bm
    Pointer to the bitmap see section 8.6.1 Encoding of Colours
```

Reply

```
None
```

Example

```
appl_bm_description_s  bm_description;
u8 square[4] = {0xFC, 0x63, 0x18, 0xFC};

bm_description.bitmap = square;
bm_description.width = 5;
bm_description.height = 6;
bm_description.x = 20;
bm_description.y = 30;
bm_description.shape = SH_ORDINARY;

appl_LCD_writeBitmap(&bm_description);
```

8.11.2.3 appl_LCD_writeLogo()

This function displays the specified logo.

Syntax

```
void appl_LCD_writeLogo(logo_t logo);
```

Parameter

```
logo
    The logo to display specified as value of type logo_t
```

Reply

None

Example

```
appl_LCD_writeLogo(CUSTOMER_LOGO);
```

8.12 System Information

This section describes how to retrieve system information and relevant hardware addresses. The returned values reflect the equipment of the hosting device and may differ on non-standard models.

8.12.1 Defined Types

The following types have been defined for the use by system information functions:

sys_parameter_t	typedef enum

8.12.1.1 sys_parameter_t

This struct contains the basic addresses and capacity figures of a device.

Syntax

```
typedef enum
{   RAM_BASE_ADDRESS,
    FLASH_BASE_ADDRESS,
    RAM_SIZE,
    FLASH_SIZE,
    FLASH_SECTOR_SIZE,
    FLASH_BLOCK_SIZE
}   sys_parameter_t;
```

Parameter

```
RAM_BASE_ADDRESS
    Base address of the random-access memory (RAM) for applications.

FLASH_BASE_ADDRESS
    Base address of the program flash memory for applications.

RAM_SIZE
    Size of the available RAM for applications.

FLASH_SIZE
    Size of a separate flash memory (not included) ->0

FLASH_SECTOR_SIZE
    Sector size of the flash memory -> 0

FLASH_BLOCK_SIZE
    Block size of the flash memory -> 0
```

8.12.2 Functions

The following SYSTEM INFORMATION functions are provided:

appl_System_getParameter()

8.12.2.1 appl_System_getParameter()

This function retrieves the basic system parameters.

Syntax

```
u32 appl_System_getParameter(sys_parameter_t par);
```

Parameter

```
par
    Pointer to the sys_parameter_t struct, which is to be filled with the system
parameters.
```

Reply

None

Example

```
sys_parameter_t sys_parameter;
appl_System_getParameter(&sys_parameter);
```

8.13 Timer

This section describes how timers can be utilized in the application design.

8.13.1 Defined Types

The following types have been defined for the use by timer functions:

appl_system_timer_t	typedef enum

8.13.1.1 appl_system_timer_t

Six (6) system timers are provided for the use by an application. The KEY_TIMER should be used to control the repeat rate of the keypad; the LOGO_TIMER should control the display of the logo; and the LIGHT_TIMER should control the lighting. The three USER_TIMERx are available for free usage.

Syntax

```
typedef enum
{   KEY_TIMER,
    LIGHT_TIMER,
    LOGO_TIMER,
    USER_TIMER1,
    USER_TIMER2,
    USER_TIMER3
}   appl_system_timer_t;
```

Parameter

```
KEY_TIMER
    Timer for controlling the key repeat rate.

LIGHT_TIMER
    Timer for controlling the lighting.

LOGO_TIMER
    Timer for controlling the display of the logo.

USER_TIMER1
    Freely available Timer1

USER_TIMER2
    Freely available Timer2

USER_TIMER3
    Freely available Timer3
```

8.13.2 Functions

The following TIMER functions are provided:

appl_Timer_delay()
appl_Timer_start()
appl_Timer_stop()
appl_Timer_is()

8.13.2.1 appl_Timer_delay()

This function suspends the user application for the specified number of system ticks. The duration of a system tick is 10 ms. This function especially serves the Watchdog for freeze detection.

Syntax

```
void appl_Timer_delay(unsigned int tick);
```

Parameter

```
tick
    Number of system ticks.
```

Reply

None

Example

```
appl_Timer_delay(100);

// Wait for 1 s (= 100 * 10 ms)
```

8.13.2.2 appl_Timer_start()

This function starts the specified timer with the specified target running time. The running time is specified in timer ticks. The duration of a timer tick is 100 ms for the timers *KEY_TIMER*, *LIGHT_TIMER*, *LOGO_TIMER*, *USER_TIMER1*, and *USER_TIMER2*. It is 10 ms for the timer *USER_TIMER3*.

Syntax

```
void appl_Timer_start(unsigned int tick, appl_system_timer_t tm);
```

Parameter

```
tick
    Number of timer ticks.

tm
    Identifies the timer to use by a value of type appl_system_timer_t.
```

Reply

None

Example

```
appl_Timer_start(30, LOGO_TIMER);

// Time out in 3 s (= 30 * 100 ms)
```

8.13.2.3 appl_Timer_stop()

This function stops the specified timer. The stopped timer will not generate a timeout event. Already expired timers do not need to be stopped explicitly, which means, timers are never cyclic.

Syntax

```
void appl_Timer_stop(appl_system_timer_t tm);
```

Parameter

tm
Identifies the timer to stop by a value of type appl_system_timer_t

Reply

None

Example

```
appl_Timer_stop(USER_TIMER1);
```

8.13.2.4 appl_Timer_is()

This function determines whether the specified timer is currently running (TRUE) or not (FALSE).

Syntax

```
bool appl_Timer_is(appl_system_timer_t tm);
```

Parameter

```
tm
    Identifies the timer to query by a value of type appl_system_timer_t
```

Reply

```
bool
    Timer activity: TRUE - running or FALSE - not running.
```

Example

```
if (appl_Timer_is(USER_TIMER1))
{    appl_Timer_stop(USER_TIMER1);
}
```

8.14 UART

UART connection serves as the central control interface to the handset. The initialization and utilization of this interface are described in this section.

The standard initialization is 115200/8/N/1 and always active together with the start event. Reading and writing is buffered (range is 1-2 kByte, the value might be changed for firmware updates).

8.14.1 Defined Types

The following types have been defined for the use by UART functions:

UART_Parity_t	typedef enum
UART_StopBits_t	typedef enum
UART_WordLength_t	typedef enum

8.14.1.1 UART_Parity_t

The values of this enum identify the parity setting of the UART transmission.

Syntax

```
typedef enum
{   UART_NO_PARITY,
    UART_EVEN_PARITY,
    UART_ODD_PARAITY
}   UART_Parity_t;
```

Parameter

```
UART_NO_PARITY
    No parity checking.

UART_EVEN_PARITY
    Even parity.

UART_ODD_PARITY
    Odd parity.
```

8.14.1.2 UART_StopBits_t

The values of this enum identify the duration of the stop bit.

Syntax

```
typedef enum
{   UART_0_5_STOPBIT,
    UART_1_STOPBIT,
    UART_1_5_STOPBIT,
    UART_2_STOPBIT
}   UART_StopBits_t;
```

Parameter

```
UART_0_5_STOPBIT
    Duration of the stop bit ½ bit time.

UART_1_STOPBIT
    Duration of the stop bit 1 bit time.

UART_1_5_STOPBIT
    Duration of the stop bit 1.5 bit times.

UART_2_STOPBIT
    Duration of the stop bit 2 bit times.
```

8.14.1.3 UART_WordLength_t

This enum identifies the number of bits used for the transfer of one character (data byte).

Syntax

```
typedef enum
{   UART_WORDLENGTH_8B,
    UART_WORDLENGTH_9B
}   UART_WordLength_t;
```

Parameter

```
UART_WORDLENGTH_8B
    A transmitted character consists of 8 bits.

UART_WORLDLENGTH_9B
    A transmitted character consists of 9 bits.
```

8.14.2 Functions

The following UART functions are provided:

appl_UART_sendString()
appl_UART_write()
appl_UART_clear()

appl_UART_bytesToRead()
appl_UART_init()
appl_UART_setByteState()
appl_UART_getByte()
appl_UART_read()

8.14.2.1 appl_UART_sendString()

This function transmits the specified zero-terminated string across the UART. This function is blocked until string output, when the output buffer is full.

Syntax

```
Syntax:    void appl_UART_sendString(char *strData);
```

Parameter

```
strData
    Pointer to the zero-terminated string to be transmitted.
```

Reply

```
None
```

Example

```
appl_UART_sendString("Init");
```

8.14.2.2 appl_UART_write()

This function transmits *n* bytes across the UART. This function serves the transmit buffer. The transmission happens asynchronously, sometimes much later, and with full transmit buffer, the number of stored bytes might be smaller than necessary.

Syntax

```
int appl_UART_write u8 *data, int n);
```

Parameter

```
data
    Pointer to a vector of bytes.

n
    Number of characters to be transmitted.
```

Reply

```
int
    The actual number of transferred bytes.
```

Example

```
u8 buffer[5];
buffer[0] = 'O';
buffer[1] = 'K';
appl_UART_write(buffer, 2);
```

8.14.2.3 appl_UART_clear()

This function clears the receive buffer of the UART, it is recommended for the case of receive errors.

Syntax

```
Syntax:    void appl_UART_clear(void);
```

Parameter

```
None
```

Reply

None

Example

```
appl_UART_clear();
```

8.14.2.4 appl_UART_bytesToRead()

This function retrieves the number of unread bytes in the receive buffer of the UART.

Syntax

```
u16 appl_UART_bytesToRead(void);
```

Parameter

None

Reply

```
u16
    Number of unread bytes in the buffer.
```

Example

```
u16 n;
n = appl_bytesToRead();
```

8.14.2.5 appl_UART_init()

This function initializes the UART using the specified parameters.

Syntax

```
void appl_UART_init(u32 br, UART_Parity_t par,
UART_StopBits_t  stb, UART_WordLength_t wl);
```

Parameter

```
br
    Baud rate.

par
    Parity specified as value of type UART_Parity_t.

stb
    Stop bits specified as value of type UART_StopBits_t.

wl
    Bit count of a character specified as value of type UART_WORDLength_t.
```

Reply

None

Example

```
appl_UART_init(115200, UART_NO_PARITY, UART_1_STOPBIT,
UART_WORDLENGTH_8B);
```

8.14.2.6 appl_UART_getByte()

This function retrieves the next unread byte from the receive buffer. As the reception of data will generate an event, polling loops are not needed! This procedure is inefficient and should be avoided for larger amounts of data. Please take into account, that binary data can't be distinguished from error codes.

Syntax

```
Syntax:    u8 appl_UART_getByte(void);
```

Parameter

None

Reply

u8
    Next byte retrieved from the receive buffer.

Example

```
u8 c;
    c = appl_UART_getByte();
```

8.14.2.7 appl_UART_read()

This function retrieves the specified number of bytes (nbr) of the receive buffer. If successful, the return value equals the parameter "nbr". If less bytes are waiting in the buffer, the return value equals the actual amount of read bytes. It is not recommended to use this function for protocol extensions!

Syntax

```
int appl_UART_read(u8 * buffer, u16 nbr);
```

Parameter

buffer
    Pointer to the buffer.

nbr
    Number of bytes to be read.

Reply

```
int
    Value >= 0  Number of bytes read
    Value = -1  UART synchronization error
    Value = -2  Overflow receive buffer
    Value = -3  UART format error
```

Example

```
u8 buffer[10];
int retVal;

retVal = appl_UART_read( buffer, 10);
if ( retVal < 0 )
    {    // Error handling    }
```

8.14.2.8 appl_UART_getBuffer()

The function retrieves the specified number of bytes (nbr) from the decoder buffer. If successful, the return value equals the parameter "nbr". If less bytes are waiting in the buffer, the return value equals the actual amount of read bytes. The decoder buffer is occupied when a command or text was received and the reception was reported by a respective event. This function can be respectively used for protocol extensions only.

Syntax

```
int appl_UART_getBuffer(u8 * buffer, u16 nbr);
```

Parameter

```
buffer
    Pointer to the buffer.

nbr
    Number of bytes to be read.
```

Reply

```
int
    Value >= 0      Number of bytes read
```

Example

```
u8 buffer[10];
int lng;
lng = appl_UART_getBuffer( buffer, 10);
```

8.14.2.9 appl_UART_useCommand()

The emulation extensions for HA20x or HA400 can feed commands to the decoder. These fed commands are interpreted as if they were received by the UART.

Syntax

```
void appl_UART_useCommand(u8 * command );
```

Parameter

```
command
    Pointer to a zero-terminated command string.
```

Reply

```
None
```

Example

```
#define ESC  0x1B
u8 buffer[10];
sprintf(buffer,"%cIN: %u\r",ESC,5);
appl_UART_useCommand( buffer );
```

8.15 Keypad

The main user input interface is the keypad with 21 individual keys (22 with PTT). This section describes the different input options available for this keypad. From the system's perspective, the keys are numbered in a straight sequence. The layout of the keypad is shown in the User Manual.

Key Number	Key Label
0	- no key pressed -
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	*
12	#
13	Right Soft Key
14	Left Soft Key
15	Right Function Key (ON/OFF)
16	Left Function Key
17	Arrow Up

Key Number	Key Label
18	Arrow Down
19	Arrow Right
20	Arrow Left
24	PTT Key

Table 6: Key Codes

8.15.1 Functions

The following KEYPAD functions are provided:

appl_Key_getCode()
appl_Key_getHook()
app_Key_setBrightness()
app_Key_getBrightness()

8.15.1.1 appl_Key_getCode()

This function retrieves the key code of the currently pressed key. It doesn't work when receiving the KEY_RELEASED even tent returns 0!

Syntax

```
u16 appl_Key_getCode(void);
```

Parameter

None

Reply

```
u16
    Key code of the pressed key according to Table 6: Key Codes.
```

Example

```
u16 keyNumber;
keyNumber = appl_Key_getCode();
```

8.15.1.2 appl_Key_getHook()

This function retrieves the status of the hook switch. An ON status will be returned if the handset is mounted to the cradle, an OFF status will be returned otherwise.

Syntax

```
appl_status_t appl_Key_getHook(void);
```

Parameter

None

Reply

```
appl_status_t
    Status of the hook switch specified as value of type appl_status_t
```

Example

```
appl_status_t hookStatus;
hookStatus = appl_Key_getHook();
```

The result of this query is independent from the settings of the hardware signals.

8.15.1.3 appl_Key_setBrightness()

The brightness of the key pad illumination is set: There are 21 levels of brightness available. The values range from 0 to LCD_MAX_BRIGHTNESS (20). The value 0 switches the illumination off.

Syntax

```
void appl_Key_setBrightness(const u8 brightness);
```

Parameter

```
brightness
    Brightness 0 … 20
```

Reply

```
None
```

Example

```
appl_Key_setBrightness ( 5 );
```

8.15.1.4 appl_Key_getBrightness()

The brightness of the keypad illumination is queried.

Syntax

```
u8 appl_Key_getBrightness(void);
```

Parameter

```
None
```

Reply

```
u8
    Brightness 0 … 20
```

Example

```
brightness = appl_Key_getBrightness ();
```

8.16 Object Flash Memory

This section describes the non-volatile flash memory available for the persistent storage of parameters and configuration data. This data objects are managed through indexes ("*objIdent*"). The number of required data objects must be initialized by the application through *appl_Object_subscribe*. Each data object is uniquely identified by its index out of the range 0 to object_count-1. The maximum supported object count is 256.

The object flash memory is not suited for frequently changing or large data objects.

8.16.1 Defined Types

The following types have been defined for the use by OBJECT-FLASH functions:

object_status_t	typedef enum

8.16.1.1 object_status_t

The values of this enum identify the return status of an operation on the object flash memory.

Syntax

```
typedef enum
{   OBJECT_OK,
OBJECT_EMPTY,
OBJECT_ERROR,
OBJECT_LENGTH
}   object_status_t;
```

Parameter

```
OBJECT_OK
    The data object is available.

OBJECT_EMPTY
    The object is not used yet and can be written to.

OBJECT_ERROR
    An error occurred during the creation of the object.

OBJECT_LENGTH
    A writing operation exceeds the length of the object.
```

8.16.2 Functions

The following OBJECT-FLASH functions are provided:

appl_Object_subscribe()
appl_Object_read()
appl_Object_write()
appl_Object_is()

8.16.2.1 appl_Object_subscribe()

This function initializes the object flash and configures the number of required data objects. This function must only be called once during the initialization of the user application.

Syntax

```
bool appl_Object_subscribe(u8 numberObjects);
```

Parameter

```
numberObjects
    Number of required data objects.
```

Reply

```
bool
    TRUE is returned upon a successful initialization, FALSE otherwise.
```

Example

```
appl_Object_subscribe(5);
```

8.16.2.2 appl_Object_read()

This function reads the specified data object and copies its content to the specified data buffer. The size of the data buffer must match the size of the data object. The object identifier *objIdent* must be strictly smaller than the configured number of objects.

Syntax

```
object_status_t appl_Object_read(u16 objIdent, u8 *data, u8 length);
```

Parameter

```
objIdent
    Identifies the object to read.

data
    Pointer to data buffer receiving the copy of the object contents.

length
    Count of bytes to copy from the data object to the data buffer.
```

Reply

```
object_status_t
    Success status as value of type object_status_t
```

Example

```
appl_hw_description_s   hwd;
Object_status_t        os;

os = appl_Object_read(0, (u8*)&hwd, sizeof(appl_hw_description_s));
```

8.16.2.3 appl_Object_write()

The content of the provided data buffer is copied into the specified object. The previous contents of the data object are silently overwritten.

Syntax

```
object_status_t appl_Object_write(u16 objIdent, u8 *data, u8 length);
```

Parameter

```
objIdent
    Identifies the object to write to.

data
    Pointer to the data buffer providing the data contents.

length
    Count of bytes to copy from the data buffer into the data object.
```

Reply

```
object_status_t
    Success status as value of type object_status_t
```

Example

```
appl_hw_description_s   hwd;
Object_status_t        os;

os = appl_Object_write(0, (u8*)&hwd, sizeof(appl_hw_description_s));
```

8.16.2.4 appl_Object_is()

This function queries the status of a flash memory object.

Syntax

```
object_status_t appl_Object_is(u16 objIdent, u8 length);
```

Parameter

```
objIdent
    Identifies the object to query.

length
    Expected size of the specified data object or memory area in bytes.
```

Reply

```
object_status_t
    Success status as value of type object_status_t
```

Example

```
char buffer[15];

appl_Object_subscribe(6);
if  (appl_Object_is(3, 15) == OBJECT_EMPTY)
{   appl_Object_write(3, "ABCDEFGHIJKLMNO", 15);
}
else
{   appl_Object_read(3, buffer, 15);
}
```

8.17 Hardware

This section describes the functions provided for the control of the device hardware (speaker, microphone, audio amplifier) and the hook and PTT signals. A detailed block diagram is provided in the User Manual.

8.17.1 Defined Types

The following types have been defined for the use by HARDWARE functions:

appl_status_t	typedef enum
appl_hw_description_s	typedef struct

8.17.1.1 appl_status_t

The enum *appl_status_t* is defined for the handware states ON and OFF and can be used as a parameter in function and other structures.

Syntax

```
typedef enum
{   OFF,
    ON
}   appl_status_t;
```

Parameter

```
OFF
    Switch OFF.

ON
    Switch ON.
```

8.17.1.2 appl_hw_description_s

This struct summarizes the hardware settings.

Syntax

```
typedef struct
{   appl_status_t audio;
    appl_status_t microphone;
    appl_status_t speaker;
appl_signal_t signalHookPTT;
}   appl_hw_description_s;
```

Parameter

```
audio
    State of the audio amplifier as value of type appl_status_t

microphone
    State of the microphone as value of type appl_status_t

speaker
    State of the speaker as value of type appl_status_t

signalHookPTT
    Output of the PTT signals to the HW interface as value of type appl_status_t:
    ON:     Signal follows PTT key.
    OFF:    Signal follows hook switch.
```

8.17.1.3 appl_hw_reset_mode_t

This type sets the boot mode.

Syntax

```
typedef enum
{   RESET_MODE_HA5x_SDK,
    RESET_MODE_HA20X,
    RESET_MODE_HA400
}   appl_hw_reset_mode_t;
```

Parameter

```
RESET_MODE_HA5x_SDK
    Reboot of the current application.

RESET_MODE_HA20X
    Reboot as HA20x emulation

RESET_MODE_HA400
    Reboot as HA400 emulation
```

8.17.2 Functions

The following HARDWARE functions are provided:

appl_HW_switchMicro()
appl_HW_switchLP()
appl_HW_switchAudio()
appl_HW_setConfig()
appl_HW_getConfig()
appl_HW_setVolLP()
appl_HW_getVolLP()
appl_HW_setGainMic()
appl_HW_getGainMic()
appl_HW_reset()
appl_HW_getSerialNumber()
appl_HW_setFactoryResetMode()

8.17.2.1 appl_HW_switchMicro()

This function turns the microphone ON or OFF.

Syntax

```
void appl_HW_switchMicro(appl_status_t sw);
```

Parameter

```
sw
    State of the microphone as value of appl_status_t.
```

Reply

None

Example

```
appl_HW_switchMicro(ON);
```

8.17.2.2 appl_HW_switchLP()

This function turns the speaker (earphone) ON or OFF.

Syntax

```
void appl_HW_switchLP(appl_status_t sw);
```

Parameter

sw
    State of the speaker as value of typeappl_status_t.

Reply

None

Example

```
appl_HW_switchLP(OFF);
```

8.17.2.3 appl_HW_switchAudio()

Both, microphone and speaker are switched ON or OFF.

Syntax

```
void appl_HW_switchAudio(appl_status_t sw);
```

Parameter

sw
    State of the audio amplifier as value of appl_status_t.

Reply

None

Example

```
appl_HW_switchAudio(ON);
```

8.17.2.4 appl_HW_setConfig()

This function sets the hardware configuration. Typically, this function is invoked only once during the initialization of the application.

Syntax

```
void appl_HW_setConfig(appl_hw_description_s *hwc);
```

Parameter

hwc
    Pointer to the configuration struct appl_hw_description_s.

Reply

None

Example

```
appl_hw_description_s hwDescr;

hwDescr.audio = OFF;
hwDescr.microphone = OFF;
hwDescr.speaker = OFF;

appl_HW_setConfig(&hwDescr);
```

8.17.2.5 appl_HW_getConfig()

This function queries the hardware settings of the handset.

Syntax

```
void appl_HW_getConfig(appl_hw_description_s *hwc);
```

Parameter

```
hwc
    Pointer to the configuration struct appl_hw_description_s to be filled in.
```

Reply

```
None
```

Example

```
appl_hw_description_s hwDescr;
appl_HW_getConfig(&hwDescr);
```

8.17.2.6 appl_HW_setVolLP()

This function sets the speaker volume. Nine (9) volume levels are available with a range of 0 to MAX_VOLIDX_LP (8). The actual amplification associated with each volume level is documented in the User Manual.

Syntax

```
void appl_HW_setVolLP(u8 volIndex);
```

Parameter

```
volIndex
    Index of the volume level in a range of:
    0               Minimum value
    MAX_VOLIDX_LP   Maximum value
```

Reply

None

Example

```
appl_HW_setVolLP(5);
```

8.17.2.7 appl_HW_getVolLP()

This function queries the current volume level. The value returned is a level index from the range 0 to MAX_VOLIDX_LP (8). The actual amplification associated with each volume level is documented in the User Manual.

Syntax

```
u8 appl_HW_getVolLP(void);
```

Parameter

None

Reply

```
u8
    Current index of the volume level in a range of:
    0               Minimum value
    MAX_VOLIDX_LP   Maximum value
```

Example

```
u8 volume;
volume = appl_HW_getVolLP();
```

8.17.2.8 appl_HW_setGainMic()

This function configures the microphone gain. Ten (10) amplification levels are available with a range of 0 to MAX_GAINIDX_MIC (9). The actual amplification associated with each gain index is documented in the User Manual.

Syntax

```
void appl_HW_setGainMic(u8 gainIndex);
```

Parameter

```
gainIndex
    Index of amplification in a range of:
    0                   Minimum value
    MAX_GAINIDX_MIC     Maximum value
```

Reply

```
None
```

Example

```
appl_HW_setGainMic(3);
```

8.17.2.9 appl_HW_getGainMic()

This function queries the current microphone amplification as a gain index from the range 0 to MAX_GAINIDX_MIC (9). The actual amplification associated with each gain index is documented in the User Manual.

Syntax

```
u8 appl_HW_getGainMic(void);
```

Parameter

None

Reply

```
u8
    Gain index of current mic amplification in a range of:
    0                   Minimum value
    MAX_GAINIDX_MIC     Maximum value
```

Example

```
u8 gain;
gain = appl_HW_getGainMic();
```

8.17.2.10 appl_HW_reset()

This function forces a hardware reset. The handset acts like after switching on the operating voltage.

Syntax

```
void appl_HW_reset(void);
```

Parameter

None

Reply

None

Example

```
appl_HW_reset();
```

8.17.2.11 appl_HW_getSerialNumber()

This function returns a pointer to the serial number of the HA57. The serial number is represented by an ASCII string of maximum 12 characters.

Syntax

```
char * appl_HW_getSerialNumber(void);
```

Parameter

None

Reply

```
char *
    Pointer to the serial number
```

Example

```
char serialNumber[15];
strcpy(serialNumber, appl_HW_getSerialNumber());
```

8.17.2.12 appl_HW_setFactoryResetMode()

This function defines how the handsets reboots after a factory reset conducted from the menu. The function makes sense in the init function of an application.

Syntax

```
void appl_HW_setFactoryResetMode( appl_hw_reset_mode_t mode);
```

Parameter

```
appl_hw_reset_mode_t
    Boot mode
```

Reply

```
None
```

Example

```
appl_HW_setFactoryResetMode( RESET_MODE_HA5x_SDK );
```

8.19 Functions of the C Standard Library

The runtime system provides implementations of many functions of the C standard library. Using these implementations avoids the linking of separate library implementations into the application, which keeps the overall memory footprint of the application small.

In general, the provided functions match the behavior of their corresponding standard implementations. Therefore, this section will only detail deviating specifics. For a general reference of the C library functions (parameters and examples), refer to the literature about the C programming language.

8.19.1 appl_c_itoa()

This function converts an integer into a zero-terminated string representation. This function is not based on an equivalent standard C library function.

Syntax

```
char *appl_c_itoa(int n, char *valueStr);
```

Parameter

```
n
    Integer to convert into a string representation.

valueStr
    Pointer to the result buffer.
```

Reply

```
char *
    A pointer to the result buffer for a successful conversion, NULL otherwise.
```

Example

```
char buffer[16];
int  n = 20;

appl_c_itoa(n, buffer);
```

8.19.2 appl_c_malloc()

This function allocates memory dynamically. This function matches the standard C function *malloc()*.

> **Notice**
> This is firmware memory, which is limited to 32 kB in the heap/stack. It is not recommended to use this possibility!

Syntax

```
void *appl_c_malloc(u32 s);
```

8.19.3 appl_c_free()

This function releases memory previously allocated dynamically. This function matches the standard function *free()*.

Syntax

```
void appl_c_free(void *);
```

8.19.4 appl_c_atoi()

This function converts a numerical string into an integer. This function matches the standard C function *atoi()*.

Syntax

```
int appl_c_atoi(const char *valueStr);
```

8.19.5 appl_c_hexToBin()

This function converts a hexadecimal digit into its numeric value.

Syntax

```
u8 appl_c_hexToBin(char h);
```

Parameter

h
    ASCII character representing a hexadecimal digit ('0'…'9', 'a'…'f', 'A'…'F')

Reply

u8
    Numeric value 0x0…0xF of successfully converted hex digit, 0x80 otherwise.

Example

```
char c = 'E';
u8   b;

b = appl_c_hexToBin(c);

// Result:  b = 0x0E
```

8.19.6 appl_c_strlen()

This function determines the length of a string. This function matches the standard C function *strlen()*.

Syntax

```
unsigned int appl_c_strlen(char *str);
```

8.19.7 appl_c_strncmp()

This function compares two strings with limitation of length. This function matches the standard C function *strncmp()*.

Syntax

```
int appl_c_strncmp(const char *s1, const char *s2, unsigned int n);
```

8.19.8 appl_c_strcmp()

This function compares two strings. This function matches the standard C function *strcmp()*.

Syntax

```
int appl_c_strcmp(const char *s1, const char *s2);
```

8.19.9 appl_c_strcpy()

This function copies a string into another string. This function matches the standard C function *strcpy()*.

Syntax

```
char *appl_c_strcpy(char *s1, const char *s2);
```

8.19.10 appl_c_strncpy()

This function copies two strings with limitation of length. This function matches the standard C function *strncpy()*.

Syntax

```
char *appl_c_strncpy(char *s1, const char *s2, unsigned int n);
```

8.19.11 appl_c_strcat()

This function appends a string to another one. This function matches the standard C function *strcat()*.

Syntax

```
char *appl_c_strcat(char *s1, const char *s2);
```

8.19.12 appl_c_strchr()

This function determines the position of a character within a string. This function matches the standard C function *strchr()*.

Syntax

```
char *appl_c_strchr(const char *s, char c);
```

8.19.13 appl_c_memcmp()

This function compares two memory areas. This function matches the standard C function *memcmp()*.

Syntax

```
int appl_c_memcmp(const void *s1, const void *s2, unsigned int n);
```

8.19.14 appl_c_memcpy()

This function copies the content of a memory area. This function matches the standard C function *memcpy()*.

Syntax

```
void *appl_c_memcpy(void *s1, const void *s2, unsigned int n);
```

8.19.15 appl_c_memchr()

This function searches the first *n* Bytes of a buffer *s* for the value *val*. This function matches the standard C function *memchr()*.

Syntax

```
void *appl_c_memchr(void *s,  unsigned char val, unsigned int n);
```

8.19.16 appl_c_memmove()

This function moves memory areas. This function matches the standard C function *memmove()*.

Syntax

```
void *appl_c_memmove(void *s1, const void *s2, unsigned int n);
```

8.19.17 appl_c_memset()

This function initializes a memory area with a defined byte value. This function matches the standard C function *memset()*.

Syntax

```
void *appl_c_memset(void *s, unsigned char val, unsigned int n);
```

8.19.18 appl_c_sprintf()

This function performs formatted output to a string buffer. This function is a limited derivation of the standard function *sprintf()*.

Syntax

```
int appl_c_sprintf(char *buf, const char *fmt, ...);
```

Parameter

```
buf
    Pointer to a buffer where the resulting string is stored.

fmt
    Format string specified the same way as for the standard function.

…
    Arguments of the format string. This implementation accepts at most two arguments.
```

Reply

```
int
    As defined for the standard function.
```

Example

```
char buffer[16];
int  n = 20;

appl_c_printf(buffer,"Value n: %d\r\n",n);
```

8.19.19 appl_c_snprintf()

Output formatting. This function is a derivation of *sprintf()*.

Syntax

```
int appl_c_snprintf(char *buf, size_t n, const char *fmt, ...);
```

Parameter

```
buf
    Points to a buffer where the resulting string is saved.

n
    Length of the output field.

fmt
    Format string, equals the standard function.

…
Arguments of the format string. In this case, only two arguments are permitted.
```

Reply

```
int
    The return value equals the standard function.
```

Example

```
char buffer[16];
int  n = 20;

appl_c_printf(buffer,16,"Value n: %d\r\n",n);
```

8.19.20 appl_c_vsnprintf()

Output formatting. This function is a derivation of *sprintf()*.

Syntax

```
int appl_c_vsnprintf(char *buf, size_t n, const char *fmt, va_list ap);
```

Parameter

buf
    Pointer to the buffer in which the resulting string is stored.

n
    Length of the output field.

fmt
    Format string, equals the standard function.

ap
    Argument list.

Reply

int
    The return value equals the standard function.

Example

```
char buffer[16];
int  n = 20;
va_list args;
va_start ( args, format );

appl_c_vsnprintf(buffer,16,"Value n: %d\r\n",args);
va_end (args );
```

8.20 Software

8.20.1 appl_SW_getVersion()

This function returns the software version as a string. For versions tests, it is recommended only to relate to the main version, if necessary, as well to the subversion, but never relate to the build (patch) version.

Syntax

```
Syntax:    char *appl_SW_getVersion( void );
```

Parameter

None

Reply

```
char *
    Points to the version string in major.minor.build format.
        Major: main version
        Minor: subversion
        Build: a build-calculator depending consecutive number.
```

Example

```
char buffer[6];

appl_c_strcpy ( buffer,appl_SW_getVersion());
```

8.21 Menu Control

The following functions control some specific menu items:

appl_MENU_setEmulation()
appl_MENU_setFactoryReset()
appl_MENU_setSetupMode()

8.21.1 appl_MENU_setEmulation()

The menu item "Emulation" of the HA57 EVO setup is displayed or hidden. The default setting is TRUE.

Syntax

```
void appl_MENU_setEmulation( bool enable );
```

Parameter

```
bool
    TRUE  the menu item is displayed
    FALSE the menu item is hidden
```

Reply

None

Example

```
appl_MENU_setEmulation(FALSE);
```

8.21.2 appl_MENU_setFactoryReset()

The menu item "factory reset" of the HA57 EVO setup is displayed or hidden. The default setting is TRUE.

Syntax

```
void appl_MENU_setFactoryReset ( bool enable );
```

Parameter

```
bool
    TRUE  the menu item is displayed
    FALSE the menu item is hidden
```

Reply

None

Example

```
appl_MENU_setFactoryReset (FALSE);
```

8.21.3 appl_MENU_setSetupMode()

Use this function to prevent or allow the access to the setup menu of the HA57 EVO. The default setting is TRUE.

Syntax

```
Syntax:    void appl_MENU_setSetupMode ( const bool enable );
```

Parameter

```
bool
    TRUE   the setup menu is available
    FALSE accessing the setup menu is not possible
```

Reply

None

Example

```
appl_MENU_setSetupMode (FALSE);
```

8.22 Animations

For displaying activities, two animations are available.

appl_drawAnimation()
appl_drawProgress()

8.22.1 appl_drawAnimation()

A cyclic pulsating circle, intended as loading animation is displayed. The animation overwrites the available display content and needs a stop command to be overwritten.

Syntax

```
void appl_drawAnimation( appl_status_t status );
```

Parameter

```
status
    OFF     the animation stops
    ON      the animation starts
```

Reply

None

Example

```
appl_drawAnimation(ON);
```

8.22.2 appl_drawProgress()

A progress bar is displayed which can be used for visualizing finite processes. The progress bar overwrites available display content. The outline of the bar is displayed with a degree of filling ranging from 0 to 100 %. To remove the progress bar, the display needs to be cleared.

Syntax

```
void appl_drawProgress( unsigned int level );
```

Parameter

```
level
    0 … 100
```

Reply

```
None
```

Example

```
appl_drawProgress( 50 );
```

8.23 Font Extension

The display of texts of the HA57 EVO bases on the freely available Google fonts RobotoMono (https://fonts.google.com/specimen/Roboto+Mono). The font description was translated with a converter (https://littlevgl.com/ttf-font-to-c-array) into C code and integrated into the firmware. For each character to be shown in the display, the character is translated into a UTF-32 character. If active, the font extension of the application is called. Without a result, the internal bitmap is determined and respectively shown. If the font extension has a result, it is used. In this way, applications can extend internal character sets as well as overwrite them.

The font extension function

Syntax

```
void char_info(uint32_t utf32, appl_typeface_t font, appl_char_info_t *info);
```

has the UTF-32 character as parameter, the font to be displayed and a pre-filled (font_width and font_height, rest 0) character information. The struct must be completed, if width isn't 0, the character is displayed as given. The missing bitmap realizes a blank of the given width.

Syntax

```
typedef struct
{   uint8_t font_width;
    uint8_t font_height;
    uint8_t height;
    uint8_t width;
    const uint8_t* bitmap;
}   appl_char_info_t;
```

Parameters

```
font_width:
    Number of horizontal pixels for a character, standard for monospace fonts.

font_height:
    Number of vertical pixels of a font.

height:
    Height of the bitmap in pixels.

width:
    Width of a given bitmap in pixels (left aligned rounded to byte).

bitmap:
    The byte array of the bitmap for the character (can be 0).
```

It is permitted that height and width differ from specifications. The character is always displayed completely with the given bitmap at the respective position, meaning, it might overwrite neighboring characters. In reverse, characters that are too wide can be overwritten later with other characters. Normally, this can only be noticed with monospace fonts, which are used quite intensely by normal protocol applications.

The SDK of the HA57 EVO now offers the possibility of variable character placing (SH_*VARIABLE*) where strings are set with the true width of the character, but ligatures are not implemented. For memory reasons, 2- or 4-bit pixel variations (aliasing) are not realized; the bitmaps are always 1-pixel maps.

The provided example demonstrates the firmware-internal implemented pixel fonts. If the available bitmaps are removed and extended by, for example, the Greek language, European languages are covert. If just an extension of fonts is asked for, a protocol extension without event handler may be created for the intended protocol.

If UTF-8 is used in custom applications, it is recommended to use a UTF-8 capable IDE or editor (gvim) to force, especially for Windows, the UTF-8 usage and storage.

END OF DOCUMENT